Pebble Automata for Data Languages: Separation, Decidability, and Undecidability

Tony Tan
Pebble Automata for Data Languages: Separation, Decidability, and Undecidability

Research Thesis

Submitted in partial fulfillment of the requirements for the degree of Doctor of Philosophy

Tony Tan

Submitted to the Senate of the Technion — Israel Institute of Technology
Sivan 5769 Haifa June 2009
The research thesis was done under the supervision of Assoc. Prof. Michael Kaminski in the Computer Science Department.

The generous financial support of the Technion and Raphael and Miriam Mishan fund is gratefully acknowledged.
## Contents

Abstract 1  
Abbreviations and Notations 2  

1 Introduction 3  

2 Models of computation 9  
  2.1 Finite-memory automata (FMA) 10  
  2.2 Pebble automata (PA) 11  

3 Logic for data languages 16  
  3.1 First-order and monadic second-order logic 16  
  3.2 Linear temporal logic with freeze quantifier 18  

4 Robustness of PA languages 21  
  4.1 Two-way alternating finite state automata 21  
  4.2 Converting an alternating strong PA into a deterministic one 24  
    4.2.1 Determinizing pebble 1 26  
    4.2.2 Determinizing pebble $i$ 36  
    4.2.3 Determinizing the automaton $A$ 43  
  4.3 Converting an alternating weak PA into a deterministic one 44  
    4.3.1 From nondeterministic to deterministic 44  
    4.3.2 From alternating to nondeterministic 46  

5 Hierarchy of PA languages 48  
  5.1 Words of $\mathcal{D}^*$ as graphs 50  
  5.2 Sketch of the proof of Lemma 16 55  
  5.3 Proof of Lemma 16 61
List of Figures

4.1 Sets of states resulting in lifting pebble 1. 33
4.2 The conjunctive branching of pebble 2. 34
4.3 The sequence of moves of $\mathcal{A}$ on $\langle a_1 \rangle\langle a_2 \rangle\langle a_3 \rangle\langle a_4 \rangle$. 37

5.1 The graph induced by $G \in \mathcal{G}_{k,m}$. 57
5.2 When pebble 3 is above $A$ and $D$, we apply the induction hypothesis, according to which pebble 2 exits $G_1$ and $\overline{G}_1$ in the same state. 59
5.3 When pebble 3 is above $C$ and $E$, we apply the induction hypothesis, according to which pebble 2 exits $G_2$ and $\overline{G}_2$ in the same state. 60
5.4 Positions of $x_{j-1}^{i-1}, y_{j-1}^{i-1}, x_{j+p-1}^{i-1}$ and $y_{j+p-1}, x_{j+p-1}^{i-1}$. 68
5.5 When pebble $(i+1)$ is above $A$ and $D$, we apply the induction hypothesis, according to which pebble $i$ exits $G_i$ and $\overline{G}_i$ in the same state. 74
5.6 When pebble $(i+1)$ is above $C$ and $E$, we apply the induction hypothesis according to which pebble $i$ exits $G_i$ and $\overline{G}_i$ in the same state. 76

7.1 The tree representation of a run of $\mathcal{A}$ on $w = \langle d_1 \rangle\cdots\langle d_n \rangle$. 92
7.2 The corresponding run of $\mathcal{A}'$ to the one in Figure 7.1. 93
Abstract

In this thesis we continue the study of pebble automata for data languages, which were first introduced by Neven, Schwentick and Vianu in 2001. In essence, pebble automata (PA) are finite state automata with a finite number of pebbles. The pebbles are placed on the input word in the stack discipline: first in last out, to mark the position in the input word. One pebble can only mark one position and the last pebble placed serves as the header. The automaton moves from one state to another depending on the label of the position and the equality tests among data values marked by the pebbles. As studied by Neven, Schwentick and Vianu, PA languages have desirable closure properties; but undesirable in terms of application: its emptiness problem in general is undecidable.

We present the boundary of the subclass of PA languages of which the emptiness problem is decidable. We call this subclass top view weak PA. We show this subclass inherit most of the desirable qualities of PA such as robustness: equivalence among alternating, nondeterministic and deterministic versions; and logical closure properties. It is also strong enough to contain the languages defined by Linear Temporal Logic augmented by one register freeze quantifier.

In addition, we also establish a number of separation results. In particular, we establish the hierarchy of PA languages based on the number of pebbles and separation of Monadic Second Order logic from PA languages.
Abbreviations and Notations

$\Sigma$ — A finite alphabet of labels
$\mathcal{D}$ — A infinite set of data values
FMA — Finite-memory automaton
$k$-FMA — $k$ register finite-memory automaton
PA — Pebble automata
$k$-PA — $k$ pebble automaton
$\text{FO}[\Sigma, +1, <, \sim]$ — First-order logic for finite data words
$\text{MSO}[\Sigma, +1, <, \sim]$ — Monadic second order logic for finite data words
$LTL_1(\Sigma, x, u)$ — One-way linear temporal logic with one register freeze quantifier
Chapter 1

Introduction

Regular languages are clearly one of the most important concepts in computer science. They have applications in basically all branches of computer science. It can be argued that the following properties contributed to their success.

1. Expressiveness: In many settings regular languages are powerful enough to capture the kinds of patterns that have to be described.

2. Decidability: Unlike many general computational models, the mechanisms associated with regular languages allow one to perform automated semantic analysis.

3. Efficiency: The model checking problem, that is, testing whether a given string is accepted by a given automaton can be solved in polynomial time.

4. Closure properties: The regular languages possess all important closure properties.

5. Robustness: The class of regular languages has many characterizations. For example, various extensions like nondeterminism and alternation do not add any expressive power. Another characterizations include regular expressions, monoids and monadic second-order logic.

Moreover, similar notion of regularity has been successfully generalized to other kind of structures, including infinite strings and finite or infinite,
ranked or unranked, trees. Most recent applications of regular languages (on infinite strings and finite, unranked trees, respectively) are in model checking and XML processing.

- In model checking a system is a finite state one and properties are specified in a logic like LTL. Satisfiability of a formula in a system is checked on the structure that is the product of the system automaton and an automaton corresponding to the formula. The step from the “real” system to its finite state representation usually involves many abstractions, especially with respect to data values (variables, process numbers, etc.). Often their range is restricted to a finite domain.

Even though this approach has been successful and found its way into large scale industrial applications, the finite abstractions have some inherent shortcomings. As an example, \(n\) identical processes with \(m\) states each give rise to an overall model of size \(m^n\). If the number of processes is unbounded or unknown in advance, the finite state approach fails. Previous work has shown that even in such setting decidability can be obtained by restricting the problem in various ways [1, 9].

- In XML document processing, regular concepts occur in various contexts. First, most applications restrict the structure of the allowed documents to conform to a certain specification (DTD or XML Schema), which can be modeled as a regular tree language. Second, navigation (XPath) and transformation (XSLT) languages are tightly connected to various tree automata models and other regular description mechanisms, see, for example, [20].

All these approaches concentrate on the structure of the XML documents and ignore the attribute and text values. From a database point of view, this is not completely satisfactory, because a schema should allow one not only to describe the structure of the data, but also to define restrictions on the data values via integrity constraints such as key or inclusion constraints. There exists a work addressing this problem [3], but like in the case of model checking, the methods heavily rely on a case-to-case analysis.

So, in the above settings, the finite state abstraction leads to interesting results, but does not address all problems arising in applications. In both
cases, it would be already a big advance, if each position, in either a string or a tree, could carry a data value in addition to its label.

This thesis is part of a broader research program which aims at studying such extensions in a systematic way. As any kind of operations on the infinite domain quickly leads to undecidability of basic processing tasks (even a linear order on the domain is harmful), we concentrate on the setting, where data values can only be tested for equality. Furthermore, in this thesis we only consider finite data strings, that is, finite strings, where each position carries a label from a finite alphabet and a data value from an infinite domain. Recently, there has been a significant amount of work in this direction, see [4, 5, 8, 13, 14, 21, 25].

Roughly speaking, there are two approaches to studying data languages: logic and automata. We will give a brief survey on both approaches. For a more comprehensive survey, we refer the reader to [25]. The study of data languages, which can also be viewed as languages over infinite alphabets, starts with the introduction of finite-memory automata (FMA) in [13], which are also known as register automata (RA). The study of FMA was continued and extended in [21], in which pebble automata (PA) were also introduced. Each of both models has its own advantages and disadvantages. Languages accepted by FMA are closed under standard language operations: intersection, union, concatenation, and Kleene star. In addition, from the computational point of view, FMA are a much easier model to handle. Their emptiness problem is decidable, whereas the same problem for PA is not. However, the PA languages possess a very nice logical property: closure under all boolean operations,\(^1\) whereas FMA languages are not closed under complementation.

Later in [5] first-order logic for data languages was considered, and, in particular, the so-called data automata was introduced. It was shown that data automata define the fragment of existential monadic second order logic for data languages in which the first order part is restricted to two variables only. An important feature of data automata is that their emptiness problem is decidable, even for the infinite words, but is at least as hard as reachability for Petri nets. The automata themselves always work nondeterministically and seemingly cannot be determinized, see [4]. It was also shown that the satisfiability problem for the three-variable first order logic is undecidable.

\(^1\)Though it is unknown whether they are closed under Kleene star.
Another logical approach is via the so called linear temporal logic with \( n \) register freeze quantifier over the labels \( \Sigma \), denoted LTL\(^↓\)\(_n\)(\(\Sigma, X, U\)), see [8]. It was shown that one way alternating \( n \) register finite-memory automata accept all LTL\(^↓\)\(_1\)(\(\Sigma, X, U\)) languages and the emptiness problem for one way alternating one register finite-memory automata is decidable. Hence, the satisfiability problem for LTL\(^↓\)\(_1\)(\(\Sigma, X, U\)) is decidable as well. Adding one more register or past time operators to LTL\(^↓\)\(_1\)(\(\Sigma, X, U\)) makes the satisfiability problem undecidable.

Looking at the existing of work in this direction, one can immediately observe that: (1) the classes of data languages are very heterogeneous, i.e., they are often incomparable, and (2) one quickly obtains undecidable mechanisms. Thus, the questions remains whether there is an appropriate notion of regularity for data languages which has the five desirable properties mentioned above.

In this thesis we continue the study of pebble automata (PA) for data languages. The first time PA is introduced is in the context of regular languages over finite alphabets [11]. It was shown that PA accepts precisely regular languages. In the context of data languages, PA are introduced in [21] Intuitively, PA are finite state automata with a finite number of pebbles. The pebbles are placed on/lifted from the input word in the stack discipline – first in last out – and are intended to mark positions in the input word. One pebble can only mark one position and the most recently placed pebble serves as the head of the automaton. The automaton moves from one state to another depending on the current label and the equality tests among data values in the positions currently marked by the pebbles, as well as, the equality tests among the positions of the pebbles.

Furthermore, as defined in [21], there are two types of PA, according to the position of the new pebble placed. In the first type, the ordinary PA, also called strong PA, the new pebbles are placed at the beginning of the string. In the second type, called weak PA, the new pebbles are placed at the position of the most recent pebble. Obviously, two-way weak PA is just as expressive as two-way ordinary PA. However, it is known that one-way nondeterministic weak PA are weaker than one-way ordinary PA, see [21, Theorem 4.5].

The following are the main results in this thesis.

- The robustness of strong/weak PA languages: the alternating, nonde-
terministic and deterministic (strong/weak) PA have the same recognition power.

- The strict hierarchy of strong/weak PA languages: more pebbles means stronger computation power.

- Monadic second order logic for data languages is strictly stronger than pebble automata.

- The introduction of a subclass of PA languages, the so called top view weak PA languages, for which the emptiness problem is decidable. This subclass seems to be a tight boundary between the decidable and undecidable fragments of PA languages.

Some of the results settle questions left open in [21, 25] such as Theorems 3, 4, 20, 24, and 39.

Roughly speaking, top view weak PA are weak PA where the equality test is performed only between the data values seen by the two most recently placed pebbles. Obviously, for the case of two pebbles there is no difference between top-view weak PA and weak PA. Like the weak and strong PA case, top view weak PA are quite robust: alternating, nondeterministic and deterministic top view weak PA have the same recognition power.

It is also shown that top view weak PA can be simulated by one-way alternating one-register FMA. Therefore, their emptiness problem is decidable. For practical purposes, the most interesting feature of top view weak PA is, perhaps, their containment of all \( \text{LTL}_1^1(\Sigma, x, \mathcal{U}) \) languages. In fact, the number of pebbles of top view weak PA needed to simulate an \( \text{LTL}_1^1(\Sigma, x, \mathcal{U}) \) sentence linearly depends on its \textit{freeze quantifier rank}, introduced in Section 3.2.

Thus, at this point it seems that top view weak PA are the only known model for finite data strings with the five desirable properties mentioned above. How far this model can be applied in the practical settings remains to be seen, but looking at its potential, we hope that something useful might come out of it.

This thesis is organized as follows. In Chapters 2 and 3 we review the models of computation and the logics for data languages. We establish the robustness of pebble automata in Chapter 4. Then, in Chapter 5 we establish the strict hierarchy of pebble automata languages. In Chapter 6 we establish
the undecidability of the emptiness problem for pebble automata. Finally, in Chapter 7 we introduce the top view automata.
Chapter 2

Models of computation

In this chapter we will review two models of computation for data languages: finite-memory automata (FMA), which are also known as register automata (RA), and pebble automata (PA).

We will use the following notation. We always denote by $\Sigma$ a finite alphabet of labels and by $\mathcal{D}$ an infinite set of data values. A $\Sigma$-data word $w = (\sigma_1, a_1) \cdots (\sigma_n, a_n)$ is a finite sequence over $\Sigma \times \mathcal{D}$, where $\sigma_i \in \Sigma$ and $a_i \in \mathcal{D}$. A $\Sigma$-data language is a set of $\Sigma$-data words. The idea is that the alphabet $\Sigma$ is accessed directly, while the data values can only be tested for equality.

We assume that neither of $\Sigma$ and $\mathcal{D}$ contain the left-end marker $\langle$ or the right-end marker $\rangle$. The input word to the automaton is of the form $\langle w \rangle$, where $\langle$ and $\rangle$ mark the left-end and the right-end of the input word.

We will also use the following notations. For $w = (\sigma_1, a_1) \cdots (\sigma_n, a_n)$,

\[
\begin{align*}
\text{Proj}_\Sigma(w) &= \sigma_1 \cdots \sigma_n \\
\text{Proj}_\mathcal{D}(w) &= a_1 \cdots a_n \\
\text{Cont}_\Sigma(w) &= \{\sigma_1, \ldots, \sigma_n\} \\
\text{Cont}_\mathcal{D}(w) &= \{a_1, \ldots, a_n\}
\end{align*}
\]

Finally, we will use the Greek symbols $\alpha, \beta, \sigma, \ldots$, possibly indexed, to denote the labels in $\Sigma$ and the symbols $a, b, c, d, \ldots$, possibly indexed, to denote the data values in $\mathcal{D}$.
2.1 Finite-memory automata (FMA)

In this section we are going to sketch roughly the definition of finite-memory automata (FMA). Readers interested in its more formal treatment can consult [8, 13]. In essence, a register FMA, or, shortly $k$-FMA, is a finite state automaton equipped with a head to scan the input and $k$ registers, numbered from 1 to $k$. Each register can store exactly one data value from $\mathcal{D}$. The automaton is two-way if the head can move to the left or to the right. It is alternating if it is allowed to branch into a finite number of parallel computations.

More formally, a two-way alternating $k$-FMA over the label $\Sigma$ is a tuple $A = \langle \Sigma, Q, q_0, u_0, \mu, F \rangle$ where

- $Q_0$, $q_0 \in Q$ and $F \subseteq Q$ are the finite state of states, the initial state and the set of final states, respectively.
- $u_0 = a_1 \cdots a_k$ is the initial content of the registers.
- $\mu$ is a set of transitions of the following form.
  - $i$) $(q, \sigma, V) \rightarrow q'$ where $\sigma \in \Sigma \cup \{\leftarrow, \rightarrow\}, V \subseteq \{1, \ldots, k\}$ and $q, q' \in Q$. That is, if the automaton $A$ is in state $q$ and the head is currently reading a position labeled with $\sigma$ and $V$ is the set of all registers containing the current data value, then the automaton can enter the state $q'$.
  - $ii$) $q \rightarrow (q', I)$ where $I \subseteq \{1, \ldots, k\}$ and $q, q' \in Q$. That is, if the automaton $A$ is in state $q$, then the automaton can enter the state $q'$ and store the current data value into the registers whose indices belong to $I$.
  - $iii$) $q \rightarrow (q_1 \land \cdots \land q_i)$ and $q \rightarrow (q_1 \lor \cdots \lor q_i)$ where $i \geq 1$ and $q, q' \in Q$. That is, if the automaton $A$ is in state $q$, then it can decide to perform conjunctive or disjunctive branching into the states $q_1, \ldots, q_i$.
  - $iv$) $q \rightarrow (q', \text{act})$ where $\text{act} \in \{\text{left}, \text{right}\}$ and $q, q' \in Q$. That is, if the automaton $A$ is in state $q$, then it can enter the state $q'$ and move to the next or the previous word position.
A finite-memory automata is called non deterministic if the branchings of state (in item (iv)) are all disjunctive. It is called one-way if the head is not allowed to move to the previous word position.

A configuration $\gamma = [j, q, b_1 \cdots b_k]$ of the automaton $A$ consists of the current position of the head in the input word $j$, the state of the automaton $q$ and the content of the registers $b_1 \cdots b_k$. The initial configuration is $\gamma_0 = [0, q_0, u_0]$. The configuration $\gamma$ is called accepting if the state is a final state in $F$.

From each configuration $\gamma$, the automaton performs legitimate computation according to the transition relation and enters another configuration $\gamma'$. If the transition is branching, then it can split into several configurations $\gamma'_1, \ldots, \gamma'_m$.

We now define the notion of leads to acceptance for a configuration $\gamma$ as follows.

- Every accepting configuration leads to acceptance.
- If $\gamma'$ is the configuration obtained from $\gamma$ by applying a non-branching transition, then $\gamma$ leads to acceptance if and only if $\gamma'$ leads to acceptance.
- If $\gamma'_1, \ldots, \gamma'_m$ are the configurations obtained from $\gamma$ by applying a disjunctive branching transition, then $\gamma$ leads to acceptance if and only if at least one of $\gamma'_1, \ldots, \gamma'_m$ leads to acceptance.
- If $\gamma'_1, \ldots, \gamma'_m$ are the configurations obtained from $\gamma$ by applying a conjunctive branching transition, then $\gamma$ leads to acceptance if and only if all of $\gamma'_1, \ldots, \gamma'_m$ lead to acceptance.

An input word $w$ is accepted by $A$ if the initial configuration leads to acceptance. As usual, $L(A)$ denotes the language accepted by $A$.

**Theorem 1** (See [8, Theorems 4.2 and 5.2]) The emptiness problem for one-way alternating 1-FMA is decidable, but not primitive recursive.

### 2.2 Pebble automata (PA)

In this section we present the definition of pebble automata.
Definition 2 (See [21, Definition 2.3]) A two-way alternating $k$-pebble automaton or, in short, $k$-PA, over $\Sigma$ is a system $A = \langle Q, q_0, F, \mu, U, N, D \rangle$ whose components are defined as follows.

- $Q$, $q_0$ and $F$ are the set of states, the initial state and the set of final states, respectively;
- $Q$ is partitioned into $U \cup N \cup D$, where
  - $U \subseteq Q - F$ is the set of universal states;
  - $N \subseteq Q - F$ is the set of nondeterministic states; and
  - $D$ is the set of deterministic states;
- $\mu \subseteq C \times D$ is the transition relation, where
  - $C$ is a set whose elements are of the form $(i, \sigma, P, V, q)$ where $1 \leq i \leq k$, $\sigma \in \Sigma$, $P, V \subseteq \{i + 1, \ldots, k\}$ and $q \in Q$; and
  - $D$ is a set whose elements are of the form $(q, \text{act})$, where $q \in Q$ and $\text{act} \in \{\text{stay, left, right, place-pebble, lift-pebble}\}$.

Elements of $\mu$ will be written as $(i, \sigma, P, V, q) \rightarrow (p, \text{act})$.

Given a word $w = (\sigma_1^a) \cdots (\sigma_n^a) \in (\Sigma \times D)^*$, a configuration of $A$ on $\prec w \succ$ is a triple $[i, q, \theta]$, where $i \in \{1, \ldots, k\}$, $q \in Q$, and $\theta : \{i, i + 1, \ldots, k\} \rightarrow \{0, 1, \ldots, n, n + 1\}$, where $0$ and $n + 1$ are the positions of the end markers $<$ and $>$, respectively. The function $\theta$ defines the position of the pebbles and is called the pebble assignment.

The initial configuration is $[k, q_0, \theta_0]$ where $\theta_0(k) = 0$. That is, in the start of the computation pebble $k$ is positioned in the left-end marker $<$.

A transition $(i, \sigma, P, V, p) \rightarrow \beta$ applies to a configuration $[j, q, \theta]$, if

1. $i = j$ and $p = q$,
2. $P = \{l > i : \theta(l) = \theta(i)\}$,
3. $V = \{l > i : a_{\theta(l)} = a_{\theta(i)}\}$, and
4. $\sigma_{\theta(i)} = \sigma$. 

12
Note that in a configuration \([i, q, \theta]\), pebble \(i\) is in control, serving as the head pebble.

Next, we define the transition relation \(\vdash\) as follows: \([i, q, \theta] \vdash A [i', q', \theta']\), if there is a transition \(\alpha \rightarrow (p, \text{act}) \in \mu\) that applies to \([i, q, \theta]\) such that 
\[
q' = p, \quad \theta'(j) = \theta(j), \quad \text{for all } j > i, \quad \text{and}
\]
- if \(\text{act} = \text{stay}\), then \(i' = i\) and \(\theta'(i) = \theta(i)\);
- if \(\text{act} = \text{left}\), then \(i' = i\) and \(\theta'(i) = \theta(i) - 1\);
- if \(\text{act} = \text{right}\), then \(i' = i\) and \(\theta'(i) = \theta(i) + 1\);
- if \(\text{act} = \text{lift-pebble}\), then \(i' = i + 1\);
- if \(\text{act} = \text{place-pebble}\), then \(i' = i - 1\), \(\theta'(i - 1) = 0\), and \(\theta'(i) = \theta(i)\).

As usual, we denote the reflexive transitive closure of \(\vdash\) by \(\vdash^*\). When the automaton \(A\) is clear from the context, we will omit the subscript \(A\). For a subset \(\mu' \subseteq \mu\), we will also denote by \(\gamma_1 \vdash_{\mu'} \gamma_2\), when the relation \(\gamma_1 \vdash \gamma_2\) is obtained by a transition in \(\mu'\). For a configuration \([i, q, \theta]\), where \(q \in D\), there exists exactly one transition that applies to it.

If \(\text{act} \in \{\text{lift-pebble, place-pebble, right, stay}\}\), then the automaton is called one-way. It is called nondeterministic if \(U = \emptyset\). If \(U = N = \emptyset\), then it is called deterministic.

The acceptance criteria is based on the notion of leads to acceptance below. For every configuration \(\gamma = [i, q, \theta]\),

- if \(q \in F\), then \(\gamma\) leads to acceptance;
- if \(q \in U\), then \(\gamma\) leads to acceptance if and only if for all configurations \(\gamma'\) such that \(\gamma \vdash \gamma'\), \(\gamma'\) leads to acceptance;
- if \(q \notin F \cup U\), then \(\gamma\) leads to acceptance if and only if there is at least one configuration \(\gamma'\) such that \(\gamma \vdash \gamma'\), and \(\gamma'\) leads to acceptance.

A \(\Sigma\)-data word \(w \in (\Sigma \times D)^*\) is accepted by \(A\), if the initial configuration \(\gamma_0\) leads to acceptance. The language \(L(A)\) consists of all data words accepted by \(A\).

As usual, the computation of \(A\) on \(w\) can be viewed as a computation tree, where
• if a node \( \pi \) is labeled with a configuration \([i, q, \theta]\), where \( q \in D \cup N \),
then \( \pi \) has only one child labeled with a configuration \( \gamma' \), where \( \gamma \vdash \gamma' \);

• if a node \( \pi \) is labeled with a configuration \([i, q, \theta]\), where \( q \in U \), then
for all configuration \( \gamma' \) such that \( \gamma \vdash \gamma' \), there exists a child of \( \pi \) labeled
with \( \gamma' \).

The following theorem shows that PA languages are robust and the proof is presented in Section 4.2 of Chapter 4.

**Theorem 3** For each \( k \geq 1 \), alternating two-way \( k \)-PA and deterministic one-way \( k \)-PA have the same recognition power.

In view of Theorem 3, we will always assume that pebble automata under consideration are deterministic and one way.

Next, we define the hierarchy of languages accepted by PA. For \( k \geq 1 \),
\( PA_k \) is the set of all languages accepted by \( k \)-PA, and \( PA \) is the set of all
PA languages.\(^1\) That is,

\[
PA = \bigcup_{k \geq 1} PA_k.
\]

Furthermore, as mentioned in the Introduction chapter there is another weaker version of pebble automata. In the model defined above, the new pebble is placed in the beginning of the input word. An alternative would be to place the new pebble at the position of the most recent one. The model defined this way is usually referred as weak PA. (Whereas, the model defined above is referred as strong PA.)

Formally, it is defined by setting \( \theta'(i-1) = \theta(i) \) (and keeping \( \theta'(i) = \theta(i) \))
in the case of \( \text{act} = \text{place-pebble} \) of the definition of the transition relation \( \vdash_A \). Obviously, two-way weak PA are just as expressive as two-way strong PA. However, one-way deterministic weak \( k \)-PA are weaker than strong \( k \)-PA. Thus, by weak PA we always mean one-way weak PA.

The following theorem shows that weak \( k \)-PA languages are also robust and the proof is presented in Section 4.3 of Chapter 4.

**Theorem 4** For each \( k \geq 1 \), alternating weak \( k \)-PA and deterministic weak \( k \)-PA have the same recognition power.

\(^1\)Note that we use the same notation, whose meaning, however, will be always clear from the context.
In view of Theorem 4, we will always assume that weak pebble automata under consideration are deterministic and one way.

Next, we define the hierarchy of languages accepted by weak PA. For $k \geq 1$, $wPA_k$ is the set of all languages accepted by weak $k$-PA, and $wPA$ is the set of all weak PA languages. That is,

$$wPA = \bigcup_{k \geq 1} wPA_k.$$  

**Remark 5** Theorem 3 immediately implies that PA languages are closed under all boolean operations. Similarly, Theorem 4 also immediately implies that weak PA languages are closed under all boolean operations.

In Chapter 5 we will see that the hierarchies of PA and weak PA languages are both strict. That is, $PA_1 \subset PA_2 \subset PA_3 \subset \cdots$ and $wPA_1 \subset wPA_2 \subset wPA_3 \subset \cdots$. 

15
Chapter 3

Logic for data languages

In this chapter we will review two logics for data languages. The first is the standard first-order and monadic second-order logic. The second is the linear temporal logic equipped with freeze quantifiers.

3.1 First-order and monadic second-order logic

Formally, a $\Sigma$-data word $w = (\sigma_1) \cdot (\sigma_n)$ is represented by the logical structure with domain $\{1, \ldots, n\}$ and the following predicates:

- the natural ordering $<$ on the domain with its induced successor $+1$;
- the predicate $\sigma(x)$ is satisfied if and only if $\sigma_x = \sigma$;
- the equivalence relation $\sim$, where $x \sim y$ if and only if $a_x = a_y$.

The atomic formulas in this logic are of the form $x < y$, $y = x + 1$, $x \sim y$, or $\sigma(x)$ for $\sigma \in \Sigma$.

The first-order logic $\text{FO}[\Sigma, +1, <, \sim]$ is obtained by closing the atomic formulas under the propositional connectives and first-order quantification over $\{1, \ldots, n\}$. The monadic second-order formulas $\text{MSO}[\Sigma, +1, <, \sim]$ are obtained by adding quantification over unary predicates on $\{1, \ldots, n\}$. A sentence $\varphi$ defines the set of words

$$L(\varphi) = \{w : w \models \varphi\}.$$

If $L = L(\varphi)$ for some sentence $\varphi$, then we say that the sentence $\varphi$ expresses the language $L$. 

16
We use the same notation $\text{FO}[\Sigma, +1, <, \sim]$ and $\text{MSO}[\Sigma, +1, <, \sim]$ to denote the classes of languages expressible by sentences in $\text{FO}[\Sigma, +1, <, \sim]$ and $\text{MSO}[\Sigma, +1, <, \sim]$, respectively. That is,

$$\text{FO}[\Sigma, +1, <, \sim] = \{ L(\varphi) : \varphi \text{ is an } \text{FO}[\Sigma, +1, <, \sim] \text{ sentence} \}$$

and

$$\text{MSO}[\Sigma, +1, <, \sim] = \{ L(\varphi) : \varphi \text{ is an } \text{MSO}[\Sigma, +1, <, \sim] \text{ sentence} \}.$$ 

The following proposition states that every language in $\text{FO}[\Sigma, +1, <, \sim]$ is accepted by a $k$-PA, where $k$ is the quantifier rank of the first-order sentence.

**Proposition 6** (See [21, Theorem 4.1]) Let $\varphi \in \text{FO}[\Sigma, +1, <, \sim]$ with quantifier rank $k$. Then, $L(\varphi) \in \text{PA}_k$.

**Proof.** First, we note that $\text{PA}_k$ is closed under boolean operations.

Let $\varphi(x_1, \ldots, x_m)$ be a formula with the free variables $x_1, \ldots, x_m$ and $k$ be the quantifier rank of $\varphi(x_1, \ldots, x_m)$. We will prove that for all $l_1, \ldots, l_m \geq 1$, the language $L(\varphi(l_1, \ldots, l_m))$ is accepted by an $(m + k)$-PA.

**Basis:** $k = 0$. The automaton places pebbles 1 to $m$ on the positions $l_1, \ldots, l_m$ of the input, respectively. When the pebbles are placed, the automaton remembers (in its internal states) the relative order of the positions $l_1, \ldots, l_m$ as well as the satisfaction of the unary predicates $\sigma(l_1), \ldots, \sigma(l_m)$ for each $\sigma \in \Sigma$, and the equalities among the data values on the positions $l_1, \ldots, l_m$. Thus, it can determine whether the input word satisfies $\varphi(l_1, \ldots, l_m)$.

**Induction step:** Let $\varphi(x_1, \ldots, x_m) = Qy\psi(y, x_1, \ldots, x_m)$ where $Q \in \{\forall, \exists\}$ and $\psi(y, x_1, \ldots, x_m)$ is of quantifier rank $k - 1$.

Let $l_1, \ldots, l_m \geq 1$. An $(m + k)$-PA $A$ that accepts $L(\varphi(l_1, \ldots, l_m))$ works as follows. First, it places pebbles $(k + m)$ to $(k + 1)$ on the positions $l_1, \ldots, l_m$. Then, it iterates pebble $(m + 1)$ through all possible positions in the input word $w$. On each iteration, the automaton $A$ recursively calls a $(k - 1)$-PA $A'$ and treats the position of pebbles $(k + m)$ to $(k + 1)$ as the positions $l_1, \ldots, l_m$ and the position of pebble $k$ as the assignment value for $y$.
- If $Q = \forall$, then $\mathcal{A}$ accepts $w$ if and only if $\mathcal{A}'$ accepts on all iterations.
- If $Q = \exists$, then $\mathcal{A}$ accepts $w$ if and only if $\mathcal{A}'$ accepts on at least one iteration.

We end this section with Theorem 7 below which states that every language accepted by pebble automaton can be expressed by an MSO[{$\Sigma$, $+, 1$, $<$, $\sim$}] sentence.

**Theorem 7** ([21, Theorem 4.2]) $PA \subseteq MSO[\Sigma, +, 1, <, \sim]$.

We will see in Chapter 5 that this inclusion in Theorem 7 is actually strict.

### 3.2 Linear temporal logic with freeze quantifier

In this section we recall the definition of Linear Temporal Logic (LTL) augmented with one register freeze quantifier [8]. We consider only one-way temporal operators “next” $\mathcal{X}$ and “until” $\mathcal{U}$, and do not consider their past time counterparts.

Roughly, the logic $LTL_1^\downarrow(\Sigma, \mathcal{X}, \mathcal{U})$ is standard LTL augmented with a register to store a data value. Formally, the formulas are defined as follows.

- Both True and False belong to $LTL_1^\downarrow(\Sigma, \mathcal{X}, \mathcal{U})$.
- The empty formula $\epsilon$ belongs to $LTL_1^\downarrow(\Sigma, \mathcal{X}, \mathcal{U})$.
- For each $\sigma \in \Sigma$, $\sigma$ is in $LTL_1^\downarrow(\Sigma, \mathcal{X}, \mathcal{U})$.
- If $\varphi, \psi$ are in $LTL_1^\downarrow(\Sigma, \mathcal{X}, \mathcal{U})$, then so are $\neg \varphi$, $\varphi \lor \psi$ and $\varphi \land \psi$.
- $\uparrow$ is in $LTL_1^\downarrow(\Sigma, \mathcal{X}, \mathcal{U})$.
- If $\varphi$ is in $LTL_1^\downarrow(\Sigma, \mathcal{X}, \mathcal{U})$, then so is $\mathcal{X}\varphi$.
- If $\varphi$ is in $LTL_1^\downarrow(\Sigma, \mathcal{X}, \mathcal{U})$, then so is $\downarrow \varphi$.
- If $\varphi, \psi$ are in $LTL_1^\downarrow(\Sigma, \mathcal{X}, \mathcal{U})$, then so is $\varphi \mathcal{U} \psi$. 

18
Intuitively, the predicate ↑ is intended to mean that the current data value is the same as the data value in the register, while ↓ φ is intended to mean that the formula φ holds when the register contains the current data value. This will be made precise in the definition of the semantics of $LTL^1_1(\Sigma, X, U)$ below.

An occurrence of ↑ within the scope of some freeze quantification ↓ is bound by it; otherwise, it is free. A sentence is a formula with no free occurrence of ↑.

Next, we define the freeze quantifier rank of a sentence ϕ, denoted by $fqr(\varphi)$.

- For each $\sigma \in \Sigma$, $fqr(\sigma) = 0$.
- $fqr(\text{True}) = fqr(\text{False}) = fqr(\epsilon) = fqr(\bot) = 0$.
- $fqr(X\varphi) = fqr(\neg \varphi) = fqr(\varphi)$, for every $\varphi$ in $LTL^1_1(\Sigma, X, U)$.
- $fqr(\varphi \lor \psi) = fqr(\varphi \land \psi) = fqr(\varphi U\psi) = \max(fqr(\varphi), fqr(\psi))$, for every $\varphi$ and $\psi$ in $LTL^1_1(\Sigma, X, U)$.
- $fqr(\downarrow \varphi) = fqr(\varphi) + 1$, for every $\varphi$ in $LTL^1_1(\Sigma, X, U)$.

Finally, we define the semantics of $LTL^1_1(\Sigma, X, U)$. Let $w = (\sigma_1^a \cdots \sigma_n^a)$ be a $\Sigma$-data word. For a position $i = 1, \ldots, n$, a data value $a$ and a formula $\varphi$ in $LTL^1_1(\Sigma, X, U)$, $w, i \models_a \varphi$ means that $\varphi$ is satisfied by $w$ at position $i$ when the content of the register is $a$. As usual, $w, i \not\models_a \varphi$ means the opposite. The satisfaction relation is defined inductively as follows.

- $w, i \models_a \epsilon$ for all $i = 1, 2, \ldots, n$ and $a \in \mathcal{D}$.
- $w, i \models_a \text{True}$ and $w, i \not\models_a \text{False}$, for all $i = 1, 2, 3, \ldots$ and $a \in \mathcal{D}$.
- $w, i \models_a \sigma$ if and only if $\sigma_i = \sigma$.
- $w, i \models_a \varphi \lor \psi$ if and only if $w, i \models_a \varphi$ or $w, i \models_a \psi$.
- $w, i \models_a \varphi \land \psi$ if and only if $w, i \models_a \varphi$ and $w, i \models_a \psi$.
- $w, i \models_a \neg \varphi$ if and only if $w, i \not\models_a \varphi$.
- $w, i \models_a X\varphi$ if and only if $1 \leq i < n$ and $w, i + 1 \models_a \varphi$. 

19
\[ w, i \models_a \varphi \psi \text{ if and only if there exists } j \geq i \text{ such that } \\
- w, j \models_a \psi \text{ and } \\
- w, j' \models_a \varphi, \text{ for all } j' = i, \ldots, j - 1. \]

\[ w, i \models_a \downarrow \varphi \text{ if and only if } w, i \models_a \varphi \]

\[ w, i \models_a \uparrow \text{ if and only if } a = a_i. \]

For a sentence \( \varphi \) in \( \text{LTL}_1(\Sigma, X, U) \), we define the \( \Sigma \)-data language \( L(\varphi) \) by

\[
L(\varphi) = \{ w \mid w, 1 \models_a \varphi \text{ for some } a \in \mathcal{D} \}.
\]

Note that since \( \varphi \) is a sentence, all occurrences of \( \uparrow \) in \( \varphi \) are bound. Thus, it makes no difference which data value \( a \) is used in the statement \( w, 1 \models_a \varphi \) in the definition of \( L(\varphi) \).

**Theorem 8** (See [8, Theorem 4.1]) For every \( \varphi \in \text{LTL}_1(\Sigma, X, U) \), there exists a one-way alternating 1-FMA \( A_\varphi \) such that \( L(\varphi) = L(A_\varphi) \). Moreover, the construction of \( A_\varphi \) is effective.
Chapter 4

Robustness of PA languages

In this chapter we prove that for each \( k \geq 1 \), two-way alternating and one-way deterministic \( k \)-PA have the same recognition power. The determinization itself is done inductively from pebble 1 to pebble \( k \). The basis is determination of the behavior of pebble 1. In the induction step, assuming that pebbles 1, \ldots, \( i \) behave deterministically, we show how to determinize the behavior of pebble \((i + 1)\). The proof is a straightforward adaptation of the proof of the equivalence between two-way alternating and one-way deterministic finite state automaton in [16].

This chapter is organized as follows. We start by reviewing the proof from [16]. Then, in Section 4.2 we present the equivalence proof of alternating and deterministic strong PA. In Section 4.3 we present the equivalence proof of alternating and deterministic weak PA.

4.1 Two-way alternating finite state automata

A two-way alternating finite state automaton over the finite alphabet \( \Sigma \) is a system \( \mathcal{M} = (Q, q_0, F, \Delta, D, N, U) \), where

- \( Q \), \( q_0 \) and \( F \subseteq Q \) are the set of states, initial state and the set of final states, respectively;
- \( Q \) is partitioned into \( D \cup N \cup U \), where \( N \cap F = U \cap F = \emptyset \);
- \( \Delta \) is a set of transitions of the form \((p, \sigma) \rightarrow (q, \text{act})\), where \( p, q \in Q \), \( \sigma \in \Sigma \) and \( \text{act} \in \{\text{left}, \text{right}, \text{stay}\} \).
The states in \( D, N \) and \( U \) are called the deterministic, nondeterministic and universal states, respectively. The states in \( N \) and \( U \) are the states in which the automaton can perform the disjunctive and conjunctive branching, respectively.

We assume that the automaton \( M \) behaves as follows.

- The input to \( M \) is of the form \( ◁w⊿ \), where \( w \in \Sigma^* \) and \( ◁, ⊿ \notin \Sigma \) are the left-end and the right-end markers of the input.
- The automaton \( M \) starts the computation with the head reading the right-end marker \( ⊿ \).
- The automaton \( M \) can only enter a final state when the head of the automaton reads the right-end marker \( ⊿ \).
- When the automaton \( M \) performs disjunctive and conjunctive branching the head of the automaton is stationery.

That is, if \( (p, \sigma) \rightarrow (q, \text{act}) \) and \( p \in N \cup U \), then \( \text{act} = \text{stay} \).

Given a word \( w = \sigma_1 \cdots \sigma_n \in \Sigma^* \), a configuration of \( M \) on \( ◁w⊿ \) is a triple \( [q, ◁w⊿, l] \), where \( l \in \{0, \ldots, n+2\} \) and \( q \in Q \). The positions 0 and \( n+1 \) are positions of the end markers \( ◁ \) and \( ⊿ \), respectively. The initial configuration is \( \gamma_0 = [q_0, ◁w⊿, n+1] \). When \( l = n + 2 \), it means that the head of the automaton “falls off” the right side of the input word and the automaton finishes the computation.

The set of transitions \( \Delta \) induces the relation \( \vdash \) among the configurations as follows. \( [q, ◁w⊿, l] \vdash [q', ◁w⊿, l'] \) if there exists a transition \( (q, \sigma_l) \rightarrow \) \( (q', \text{act}) \in \Delta \) and

- if \( l' = l \), then \( \text{act} = \text{stay} \);
- if \( l' = l - 1 \), then \( \text{act} = \text{left} \); and
- if \( l' = l + 1 \), then \( \text{act} = \text{right} \).

The acceptance criteria is based on the notion of leads to acceptance below. For every configuration \( \gamma = [q, ◁w⊿, l] \),

- if \( q \in F \), then \( \gamma \) leads to acceptance;
- if \( q \in U \), then \( \gamma \) leads to acceptance if and only if for all configurations \( \gamma' \) such that \( \gamma \vdash \gamma' \), \( \gamma' \) leads to acceptance;
• if \( q \notin F \cup U \), then \( \gamma \) leads to acceptance if and only if there is at least one configuration \( \gamma' \) such that \( \gamma \vdash \gamma' \), and \( \gamma' \) leads to acceptance.

The word \( w \) is accepted by \( M \) if the initial configuration \( \gamma_0 \) leads to acceptance.

As usual, a computation of \( M \) on the input \( \langle w \rangle \) can be viewed as a computation tree where each node is labeled with a configuration and

• if a node \( \pi \) is labeled with a configuration \([q, \langle w \rangle, l]\), where \( q \in D \cup N \), then \( \pi \) has only one child labeled with a configuration \( \gamma' \), where \( \gamma \vdash \gamma' \);

• if a node \( \pi \) is labeled with a configuration \([q, \langle w \rangle, l]\), where \( q \in U \), then for all configuration \( \gamma' \) such that \( \gamma \vdash \gamma' \), there exists a child of \( \pi \) labeled with \( \gamma' \).

It is shown in [16] that every two-way alternating finite state automaton can be simulated by one-way deterministic finite state automaton. One important notion introduced in [16] is the notion of closed terms, which we will describe below.

For each state \( q \in Q \), we define a new symbol \( \bar{q} \) and let \( \bar{Q} = \{ \bar{q} : q \in Q \} \). If \( S \subseteq Q \), then \( \bar{S} = \{ \bar{p} : p \in S \} \). We define a term to be an object \( q \rightarrow S \), where \( q \in Q \) and \( S \subseteq Q \cup \bar{Q} \). A term \( q \rightarrow S \) is closed, if \( S \subseteq \bar{Q} \). A partial response is a set of terms, and a response is a set of closed terms. Note that since \( Q \) is finite, there are only finitely many closed terms and responses.

A configuration \( \gamma = [q, \langle w \rangle, l] \) induces a closed term \( q \rightarrow \bar{S} \), for some \( S \subseteq Q \), if there exists a computation tree of \( M \) on \( \langle w \rangle \) such that

• the root is labeled with the configuration \( \gamma \);

• all the leaf nodes are labeled with a configuration \([p, \langle w \rangle, l + 1]\), for some \( p \in S \);

• for every \( p \in S \), there exists a leaf node labeled with a configuration \([p, \langle w \rangle, l + 1]\);

• every interior node is labeled with a configuration \([s, \langle w \rangle, j]\), for some \( 0 \leq j \leq l \) and \( s \in Q \).

We define a response \( R(\langle w \rangle, l) \) as the set of closed terms induced by the configurations \([q, \langle w \rangle, l]\). In other words, a closed term \( p \rightarrow \bar{S} \in R(\langle w \rangle, l) \),

23
where \( S \subseteq Q \), if and only if there exists a configuration \([p, \langle w\triangleright, l \rangle]\) that induces \( p \rightarrow \hat{S} \).

Now the main point in the proof in [16] is that given a response \( R(\langle w\triangleright, l \rangle) \), we can construct the response \( R(\langle w\triangleright, l + 1 \rangle) \) without simulating the automaton \( \mathcal{M} \) on \( \langle w\triangleright \rangle \). This is done by defining a proof system \( \mathcal{S}(\langle w\triangleright, l \rangle, \sigma_l) \), where the closed terms in \( \mathcal{R}(\langle w\triangleright, l \rangle) \) and the transitions \((p, \sigma_l) \rightarrow (q, \text{act}) \in \Delta\) are the axioms. Now, the response \( \mathcal{R}(\langle w\triangleright, l + 1 \rangle) \) is precisely the set of closed terms provable in \( \mathcal{S}(\mathcal{R}(\langle w\triangleright, l \rangle), \sigma_l) \) [16, Claim in pp. 149]. In [16] such set of closed terms is denoted by \( \text{CTH}(\mathcal{R}(\langle w\triangleright, l \rangle), \sigma_l) \).

The construction of one-way deterministic automaton \( \mathcal{M}' \) that accepts the same language as \( \mathcal{M} \) is as follows. The states of \( \mathcal{M}' \) are exactly the responses. The transitions of \( \mathcal{M}' \) are of the form

\[(R, \sigma_l) \rightarrow (\text{CTH}(R, \sigma_l), \text{right}),\]

where \( R \) is a response. This is the essence of the proof in [16] that we are going to use in this chapter.

### 4.2 Converting an alternating strong PA into a deterministic one

In this section we prove that, for all \( k \geq 1 \), two-way alternating \( k \)-PA and one-way deterministic \( k \)-PA have the same recognition power. As we have mentioned earlier, the proof is a direct generalization of the corresponding proof from [16].

For a two-way alternating \( k \)-PA \( \mathcal{A} = \langle \Sigma, Q, q_0, F, U, N, D \rangle \), we will show how to simulate it by a one-way deterministic \( k \)-PA \( \mathcal{A}' \). We start by normalizing the behavior of \( \mathcal{A} \) as follows.

1. On the input word \( \langle w\triangleright \rangle \), \( \mathcal{A} \) starts the computation with pebble \( k \) above the right-end marker \( \triangleright \).

2. The state set \( Q \) is partitioned into \( Q_1 \cup \cdots \cup Q_k \), where \( Q_i \) is the set of states when pebble \( i \) is the head pebble.

Similarly, we denote by \( U_i \), \( N_i \) and \( D_i \) the set of universal, nondeterministic and deterministic states, respectively, and we denote by \( \mu_i \) the set of transitions when pebble \( i \) is the head pebble.
3. Each $Q_i$ is further partitioned into $Q_{i, \text{stay}} \cup Q_{i, \text{right}} \cup Q_{i, \text{left}} \cup Q_{i, \text{place}} \cup Q_{i, \text{lift}}$, where

- if $(i, \sigma, P, V, q) \rightarrow (p, \text{stay})$, then $q \in Q_{i, \text{stay}}$;
- if $(i, \sigma, P, V, q) \rightarrow (p, \text{right})$, then $q \in Q_{i, \text{right}}$;
- if $(i, \sigma, P, V, q) \rightarrow (p, \text{left})$, then $q \in Q_{i, \text{left}}$;
- if $(i, \sigma, P, V, q) \rightarrow (p, \text{place-pebble})$, then $q \in Q_{i, \text{place}}$; and
- if $(i, \sigma, P, V, q) \rightarrow (p, \text{lift-pebble})$, then $q \in Q_{i, \text{lift}}$.

4. The automaton can only do universal and existential branching while the head pebble is stationery. That is, if $(i, \sigma, P, V, q) \rightarrow (p, \text{act})$ and $q \in U \cup N$, then $\text{act} = \text{stay}$.

5. The automaton places the new pebble above the right-end marker $\triangleright$.

6. The automaton lifts the pebble only when it is above the right-end marker $\triangleright$.

7. When the head pebble is reading the left-end or the right-end markers $\triangleleft$ and $\triangleright$, the automaton cannot place a new pebble.

8. Only pebble $k$ can enter a final state and it does so only after reading the right-end marker $\triangleright$.

Such normalization can be achieved by adding some extra states to $A$.

We will also need the following notions. A pebble-$i$ assignment $\theta$ is a pebble assignment when pebble $i$ is the most recently placed pebble. That is, the domain of $\theta$ is $\{i, i+1, \ldots, k\}$.

Let $\theta$ be a pebble-$i$ assignment on an input word $w = (\sigma_1^{a_1}) \cdots (\sigma_n^{a_n})$. We define the assignment $\text{Succ}(\theta) = \theta'$ as follows.

- If $\theta(i) \leq n$, then $\theta'$ is the pebble-$i$ assignment such that for each $j = i, i+1, \ldots, k$,

  $$\theta'(j) = \begin{cases} 
  \theta(j), & \text{if } j = i+1, \ldots, k, \\
  \theta(j) + 1, & \text{if } j = i.
  \end{cases}$$

- If $\theta(i) = n+1$, then $\theta'$ is the pebble-$(i+1)$ assignment such that for each $j = i+1, \ldots, k$, $\theta'(j) = \theta(j)$.
Similarly, for a pebble-\(i\) assignment \(\theta\), we define the assignment \(\text{Pred}(\theta) = \theta'\) as follows.

- If \(1 \leq \theta(i)\), then \(\theta'\) is the pebble-\(i\) assignment such that for each 
\(j = i, i+1, \ldots, k\),
\[
\theta'(j) = \begin{cases}  
\theta(j), & \text{if } j = i+1, \ldots, k, \\
\theta(j) - 1, & \text{if } j = i.
\end{cases}
\]

- If \(\theta(i) = 0\), then \(\theta'\) is the pebble-\((i+1)\) assignment such that for each 
\(j = i+1, \ldots, k\), \(\theta'(j) = \theta(j)\).

In the following sections we present the determinization of \(\mathcal{A}\), starting from pebble 1 and finishing with pebble \(k\). We will denote by \(\mathcal{A}^{(i)}\) the equivalent of \(\mathcal{A}\), where the behavior of pebbles \(1, \ldots, i\) is one-way and deterministic. In this notation, \(\mathcal{A}^{(k)}\) is the one-way deterministic equivalent of \(\mathcal{A}\).

### 4.2.1 Determinizing pebble 1

The determinization of \(\mathcal{A}\) follows closely the one described in [16, Section 4]. For completeness, we present it here. The end result of the determinization is such that pebble 1 is placed above the left-end marker \(<\) and lifted when it reaches the right-end marker \(>\).

We will need a bit more notation, some of which is just a repetition of that introduced in Section 4.1. For each \(q \in Q\), we introduce a new symbol \(\bar{q}\). We denote by \(\bar{Q} = \{\bar{q} : q \in Q\}\). If \(S \subseteq Q\), then \(\bar{S} = \{\bar{p} : p \in S\}\). We define a term to be an object of the form \(q \rightarrow S\) where \(q \in Q\) and \(S \subseteq Q \cup \bar{Q}\). A term \(q \rightarrow S\) is closed, if \(\bar{S} \subseteq \bar{Q}\). A partial response is a set of terms, while a response is a set of closed terms.

Let \(w = (a_1^{(s_1)}) \cdots (a_n^{(s_n)})\) be a data word and let \(\theta\) be a pebble-1 assignment. The determinization of pebble 1 is based on the following three concepts: response \(\mathcal{R}(w, \theta)\), partial response \(\mathcal{PR}(w, \theta)\), and the proof system \(\mathcal{S}(\mathcal{R}, \sigma, P, V)\). We will define these concepts one by one starting with the concept of response \(\mathcal{R}(w, \theta)\).

The response \(\mathcal{R}(w, \theta)\) is defined as follows. For a set of states \(S \subseteq Q\), a closed term \(q \rightarrow \bar{S}\) belongs to \(\mathcal{R}(w, \theta)\) if there is a computation tree \(T\) of \(\mathcal{A}\) on \(w\) whose root is labeled \([1, q, \theta]\) such that
(1) if $\theta(1) \leq n$, then each leaf is labeled $[1, p, \text{Succ}(\theta)]$ for some $\bar{p} \in S$;

(2) if $\theta(1) = n + 1$, then each leaf is labeled $[2, p, \text{Succ}(\theta)]$ for some $\bar{p} \in S$;

(3) each internal node in the computation tree $T$ is labeled $[1, q', \theta']$, where $0 \leq \theta'(1) \leq \theta(1)$; and

(4) for each $\bar{p} \in S$, there is a leaf labeled $[1, p, \text{Succ}(\theta)]$.

Remark 9 Let $w_1, w_2$ be data words and let $\theta_1$ and $\theta_2$ be pebble-1 assignments on $<w_1>$ and $<w_2>$, respectively, such that $\theta_1(1) = \theta_2(1) = 0$. That is, on both assignments pebble 1 is reading the left-end marker $\prec$. Then, by clause (3) of the above definition, $\mathcal{R}(w_1, \theta_1) = \mathcal{R}(w_2, \theta_2)$.

Now we define the concept of a partial response $\mathcal{PR}(w, \theta)$. For a set of states $S \subseteq Q \cup \bar{Q}$, a term $q \rightarrow S$ is in $\mathcal{PR}(w, \theta)$, if there is a computation tree $T$ of $\mathcal{A}$ on $w$ whose root is labeled $[1, q, \theta]$ such that

- if $\theta(1) \leq n$, then each leaf is labeled either $[1, p, \text{Succ}(\theta)]$ for some $\bar{p} \in S$ or $[1, p, \theta]$ for some $p \in S$;
- if $\theta(1) = n + 1$, then each leaf is labeled either $[2, p, \text{Succ}(\theta)]$ for some $\bar{p} \in S$ or $[1, p, \theta]$ for some $p \in S$;
- each internal node in the computation tree $T$ is labeled $[1, q', \theta']$ such that $q' \in Q_1$ and $0 \leq \theta'(1) \leq \theta(1)$;
- if $\theta(1) \leq n$, then for each $\bar{p} \in S$, there is a leaf labeled $[1, p, \text{Succ}(\theta)]$;
- if $\theta(1) = n + 1$, then for each $\bar{p} \in S$, there is a leaf labeled $[2, p, \text{Succ}(\theta)]$;

and

- for each $p \in S$, there is a leaf labeled $[1, p, \theta]$.

We call the tree $T$ a witness for $q \rightarrow S \in \mathcal{PR}(w, \theta)$.

Finally, for a response $\mathcal{R}$, $\sigma \in \Sigma$, and $P, V \subseteq \{2, \ldots, k\}$ we define the proof system $\mathcal{S}(\mathcal{R}, \sigma, P, V)$.

1. $q \rightarrow \{q\}$

2. $q \rightarrow B \cup \{p\}, p \rightarrow C$ 
   $\frac{q \rightarrow B \cup C}{q \rightarrow B \cup C}$
3. \((1, \sigma, P, V, q) \rightarrow (p_i, \text{stay}) \in \mu_1, \text{ for each } i = 1, \ldots, m \) and \(q \in U \)
\[ q \rightarrow \{p_1, \ldots, p_m\} \]

4. \((1, \sigma, P, V, q) \rightarrow (p, \text{stay}) \in \mu_1 \) and \(p \notin U \)
\[ q \rightarrow \{p\} \]

5. \((1, \sigma, P, V, q) \rightarrow (p, \text{right}) \in \mu_1 \)
\[ q \rightarrow \{\bar{p}\} \]

6. \((1, \sigma, P, V, q) \rightarrow (p, \text{left}) \in \mu_1 \), \(p \rightarrow S \in \mathcal{R}, \) and \(S \subseteq Q_1 \)
\[ q \rightarrow S \]

7. \((1, \sigma, P, V, q) \rightarrow (p, \text{lift-pebble}), \text{ if } \sigma = \triangleright \text{ and } P, V = \emptyset \)
\[ q \rightarrow \{\bar{p}\} \]

We denote by \(\text{TH}(\mathcal{R}, \sigma, P, V)\) be the set of terms “provable” using the proof system \(S(\mathcal{R}, \sigma, P, V)\).

The following proposition is the pebble 1 counterpart of the similar result in [16, pp. 149].

**Proposition 10** For each word \(w = (\sigma_1 a_1) \cdots (\sigma_n a_n)\) and pebble-1 assignment \(\theta\) on \(<w>\),
\[ \mathcal{P}R(w, \theta) = \text{TH}(\mathcal{R}(w, \text{Pred}(\theta)), \sigma, P, V), \]
where

- \(1 \leq \theta(1) \leq n + 1;\)
- \(P = \{l : \theta(l) = \theta(1)\};\)
- \(V = \{l : a_{\theta(l)} = a_{\theta(1)}\}; \text{ and}\)
- \(\sigma = \sigma_{\theta(1)}.\)

**Proof.** The proof follows closely that in [16]. We will prove first that \(\mathcal{P}R(w, \theta) \subseteq \text{TH}(\mathcal{R}(w, \text{Pred}(\theta)), \sigma, P, V)\). The prove is by induction on the size of witnesses for terms in \(\mathcal{P}R(w, \theta)\). Let \(q \rightarrow S \in \mathcal{P}R(w, \theta)\).

For the basis, the witness for \(q \rightarrow S \in \mathcal{P}R(w, \theta)\) consists of a single node with the label \([1, q, \theta]\). Then, \(S = \{q\}\) and \(q \rightarrow \{q\}\) is provable by rule 1.

For the induction step, let \(q \rightarrow S \in \mathcal{P}R(w, \theta)\) be witnessed by a tree \(T\) with more than one node. We have five cases to consider.
1. The state $q$ is a universal state, i.e., $q \in U_1$. Let 
\[
(1, \sigma, P, V, q) \rightarrow (p_1, \text{stay}) \in \mu_1; \\
\vdots \\
(1, \sigma, P, V, q) \rightarrow (p_m, \text{stay}) \in \mu_1;
\]
In this case, the root of $T$ is labeled $[1, q, \theta]$ and its immediate children $\pi_1, \ldots, \pi_m$ are labeled $[1, p_1, \theta], \ldots, [1, p_m, \theta]$, respectively. The complete subtree rooted at $\pi_i$, $i = 1, \ldots, m$, witnesses $p_i \rightarrow S_i \in \mathcal{P}R(w, \theta)$, where $S_i$ is the set of states in the labels of the leaves in the subtree. Furthermore, $S_1 \cup \cdots \cup S_m = S$. By the induction hypothesis, $p_i \rightarrow S_i \in \text{TH}(\mathcal{R}(w, \text{Pred}(\theta)), \sigma, P, V)$. Combining rules 2 and 3, we obtain $q \rightarrow S \in \text{TH}(\mathcal{R}(w, \text{Pred}(\theta)), \sigma, P, V)$.

2. The case of a deterministic (respectively, nondeterministic) state $q \in D_1$ (respectively, $q \in N_1$) is just like case 1 above, except that we use rule 4 instead of rule 3.

3. $(1, \sigma, P, V, q) \rightarrow (p, \text{right}) \in \mu_1$. Then $S = \{\bar{p}\}$ and, by rule 5, we obtain $q \rightarrow \{\bar{p}\} \in \text{TH}(\mathcal{R}(w, \text{Pred}(\theta)), \sigma, P, V)$.

4. $(1, \sigma, P, V, q) \rightarrow (p, \text{lift-pebble}) \in \mu_1$, where $\sigma = \triangledown$ and $P, V = \emptyset$. Then $S = \{\bar{p}\}$, and by rule 7, we obtain $q \rightarrow \{\bar{p}\} \in \text{TH}(\mathcal{R}(w, \text{Pred}(\theta)), \sigma, P, V)$.

5. $(1, \sigma, P, V, q) \rightarrow (p, \text{left}) \in \mu_1$. Then the (only) child $\pi$ of the root of $T$ is labeled $[1, p, \text{Pred}(\theta)]$. Every path from $\pi$ to a leaf of $T$ must pass through a node with label of the form $[1, r, \theta]$. That is, pebble 1 must visit the position $\theta(1)$ again.

Let $\Lambda = \{\rho_1, \ldots, \rho_l\}$ be the set of the descendants of $\pi$ such that

(a) each $\rho_i$ is labeled $[1, r_i, \theta]$,
(b) no node between $\pi$ and $\rho_i$ has a label whose third component is $\theta$, and
(c) each path from $\pi$ to a leaf passes through a node in $\Lambda$.

Let $T'$ be the unique subtree of $T$ whose root is $\pi$ and whose set of leaves is $\Lambda$. Then, $T'$ is a witness of $p \rightarrow \{\bar{r}_1, \ldots, \bar{r}_l\} \in \mathcal{P}R(w, \text{Pred}(\theta))$. Since this is a closed term, $p \rightarrow \{\bar{r}_1, \ldots, \bar{r}_l\} \in \mathcal{R}(w, \text{Pred}(\theta))$, and, by
rule 6, \( q \rightarrow \{ r_1, \ldots, r_l \} \in \text{TH}(\mathcal{R}(w, \text{Pred}(\theta)), \sigma, P, V) \). The complete subtree of \( T \) rooted at \( \rho \) witnesses \( r_i \rightarrow S_i \in \mathcal{PR}(w, \theta) \), where \( S_i \) is the set of states in the labels of the leaves in the subtree. By the induction hypothesis, \( r_i \rightarrow S_i \in \text{TH}(\mathcal{R}(w, \text{Pred}(\theta)), \sigma, P, V) \), and, by rule 2, we obtain \( q \rightarrow \bigcup_{1 \leq i \leq l} S_i \in \text{TH}(\mathcal{R}(w, \text{Pred}(\theta)), \sigma, P, V) \). Since \( \bigcup_{1 \leq i \leq l} S_i = S \), this case follows.

Now we will prove that \( \text{TH}(\mathcal{R}(w, \text{Pred}(\theta)), \sigma, P, V) \subseteq \mathcal{PR}(w, \theta) \). The proof is by induction on the \( S(\mathcal{R}, \sigma, P, V) \) proof length. Let \( q \rightarrow S \in \text{TH}(\mathcal{R}(w, \text{Pred}(\theta)), \sigma, P, V) \).

- If the last step in the derivation of \( q \rightarrow S \) is by rule 1, 3, 4, 5, or 7, then we immediately have a computation tree witnessing \( q \rightarrow S \in \mathcal{PR}(w, \theta) \).

- If the last step in the derivation is by rule 2, let \( q \rightarrow B \cup \{ p \} \) and \( p \rightarrow C \) be the antecedents from which \( q \rightarrow B \cup C \) is derived (i.e., \( S = B \cup C \)). By the induction hypothesis, there are computation trees \( T \) and \( T' \) which witness \( q \rightarrow B \cup \{ p \} \) and \( p \rightarrow C \), respectively. Then, replacing each leaf of \( T \) labeled \( [1, p, \theta] \) with the tree \( T' \) (whose root is labeled \( [1, p, \theta] \)), we obtain the tree that witnesses \( q \rightarrow B \cup C \in \mathcal{PR}(w, \theta) \).

- If the last step in the derivation is by rule 6, then let \( q \rightarrow B \) be derived from
  \[
  (1, \sigma, P, V, q) \rightarrow (p, \text{left}), \quad p \rightarrow \bar{S} \in \mathcal{R}(w, \text{Pred}(\theta)), \quad S \subseteq Q_1.
  \]
  Since \( p \rightarrow \bar{S} \in \mathcal{R}(w, \text{Pred}(\theta)) \), there is a computation tree \( T' \) such that
  - the root of \( T' \) is labeled \( [1, p, \text{Pred}(\theta)] \);
  - each leaf of \( T' \) is labeled \( [1, r, \theta] \) for some \( r \in S \); and
  - for each \( r \in S \), there is a leaf of \( T' \) labeled \( [1, r, \theta] \).

Now the tree \( T \) such that
  - the root of \( T \) is labeled \( [1, q, \theta] \),
  - the root has only one immediate child \( \pi \) labeled \( [1, p, \text{Pred}(\theta)] \),
  - the subtree rooted at \( \pi \) is the tree \( T' \),
is a witness of the term $q \rightarrow S \in \mathcal{P}(w, \theta)$.

This completes the proof of the proposition. \hfill \Box

We denote by $\text{CTH}(\mathcal{R}, \sigma, P, V)$ the set of all closed terms in $\text{TH}(\mathcal{R}, \sigma, P, V)$. Since, by Proposition 10, $\text{TH}(\mathcal{R}(w, \text{Pred}(\theta)), \sigma, P, V) = \mathcal{P}(w, \theta)$, we have

$$\text{CTH}(\mathcal{R}(w, \text{Pred}(\theta)), \sigma, P, V) = \mathcal{R}(w, \theta)$$

and the determinization of $\mu_1$ is based on this equality. To some extent, the set of “states” of the deterministic version of $\mu_1$ is the set of responses $\mathcal{R}(w, \theta)$ that is finite. On the “input” $\sigma, P, V$, from the “state” $\mathcal{R}(w, \text{Pred}(\theta))$, pebble 1 (deterministically) moves right and enters the state $(\mathcal{R}, \theta)$. We describe this idea more precisely below, but before we need to modify a bit the behavior of pebble 2.

The modifications $\tilde{Q}_2$, $\tilde{U}_2$, $\tilde{N}_2$, $\tilde{D}_2$, and $\tilde{\mu}_2$, of $Q_2$, $U_2$, $N_2$, $D_2$, and $\mu_2$, respectively, are defined as follows.

- $\tilde{Q}_2 = Q_2 \cup 2^{Q_2} \cup 2^{2^{Q_2}}$;
- $\tilde{U}_2 = U_2 \cup (2^{Q_2} - \{\emptyset\})$;
- $\tilde{N}_2 = N_2 \cup 2^{2^{Q_2}}$;
- $\tilde{D}_2 = D_2$; and
- $\tilde{\mu}_2$ consists of the following transitions.

- Every transition in $\mu_2$, in which the action is not “place-pebble” is in $\tilde{\mu}_2$.
- For each $\sigma \in \Sigma$, $P, V \subseteq \{3, \ldots, k\}$, and $S_1, \ldots, S_m \subseteq Q_2$, $\tilde{\mu}_2$ contains the transition

$$\left(2, \sigma, P, V, \{S_1, \ldots, S_m\}\right) \rightarrow (S_i, \text{stay})$$

for each $i = 1, 2, \ldots, m$. By definition, from the state $\{S_1, \ldots, S_m\} \in \tilde{Q}_2$ pebble 2 performs existential branching.

---

1 As usual, $2^B$ is the power set of $B$. 
For each $\sigma \in \Sigma$, $P, V \subseteq \{3, \ldots, k\}$, and $S \subseteq Q_2$, $\tilde{\mu}_2$ contains the transition

$$(2, \sigma, P, V, S) \rightarrow (q, \text{stay})$$

for each $q \in S$. By definition, from the state $S \subseteq Q_2$ pebble 2 performs universal branching.

For each transition $$(2, \sigma, P, V, q) \rightarrow (p, \text{place-pebble}) \in \mu_2$$, $\tilde{\mu}_2$ contains the transition

$$(2, \sigma, P, V, q) \rightarrow ((p, \emptyset), \text{place-pebble})$$

That is, the transition $$(2, \sigma, P, V, q) \rightarrow (p, \text{place-pebble})$$ of $\mu_2$ is replaced with the transition $$(2, \sigma, P, V, q) \rightarrow ((p, \emptyset), \text{place-pebble})$$.

Next we define the sets of states $Q'_1$ and the set of transitions $\mu'_1$ for the "deterministic" pebble 1.

- $Q'_1$ consists of elements of the form $(q, R)$, where $q \in Q_1$ and $R$ is a response; and
- for each $q \in Q_1$, $\mu'_1$ contains the following transitions.
  - $$(1, q, \emptyset, \emptyset, (q, \emptyset)) \rightarrow ((q, R), \text{right})$$, where $R = R(w, \theta)$, for some $w$ and $\theta$ such that $\theta(1) = 0$. By Remark 9, $R(w, \theta)$ does not depend on $w$ and $\theta$, i.e., it is well defined.
  - $$(1, \sigma, P, V, (q, R)) \rightarrow ((q, \text{CTH}(R, \sigma, P, V)), \text{right})$$, for each response $R$, each label $\sigma \in \Sigma$, and each $P, V \subseteq \{2, \ldots, k\}$.
  - $$(1, \triangleright, \emptyset, \emptyset, (q, R)) \rightarrow ((S_1, \ldots, S_m), \text{lift-pebble})$$, where for each $j = 1, \ldots, m$,
    * $q \rightarrow \bar{S}_j \in \text{CTH}(R, \triangleright, \emptyset, \emptyset)$, and
    * $\bar{S}_j \subseteq Q_2$.

The intuition behind the "lift-pebble" transitions of $\mu'_1$ is as follows. Let $R = R(w, \theta)$, where $\theta$ is a pebble-1 assignment such that $\theta(1) = n + 1$, $n$ being the length of $w$, and let $\theta'$ be a pebble-2 assignment such that for $i = 2, \ldots, k$, $\theta'(i) = \theta(i)$. Then "$q \rightarrow \bar{S}_j \in \text{CTH}(R, \triangleright, \emptyset, \emptyset)$" means that there is a computation tree $T$ such that

\footnote{We use the “prime” sign to indicate that the behavior of pebble 1, described by $\mu'_1$, is deterministic, and we use the “tilde” sign to indicate that the behavior of pebble 2, described by $\tilde{\mu}_2$, is still alternating.}
• the root of $\mathcal{T}$ is labeled the configuration $[1, q, \theta]$;
• all non leaf nodes are labeled 1-configurations, i.e., configurations where the head pebble is pebble 1;
• all leaves are labeled configurations of the form $[2, p, \theta']$, for some $p \in S_j$; and
• for each $p \in S_j$, there is a leaf labeled the configuration $[2, p, \theta']$.

Since $\text{CTH}(\mathcal{R}, \mathcal{B}, \emptyset, \emptyset)$ consists of the closed terms $q \rightarrow \tilde{S}_1, \ldots, q \rightarrow \tilde{S}_m$, there are exactly $m$ possible “choices” $S_1, \ldots, S_m$ of sets of states once pebble 1 is lifted, see Figure 4.1.

So, once we have deterministically simulated pebble 1, we have to indicate that there are $m$ possible “choices” of sets of states for pebble 2, which is done by the state $\{S_1, \ldots, S_m\} \in 2^Q$. From this state the automaton nondeterministically chooses set of states $S_j$ which pebble 2 enters, and from $S_j$ the automaton branches conjunctively into each state in $S_j$, see Figure 4.2.

We proceed to show that the sets of transitions $\mu_1 \cup \mu_2$ and $\mu'_1 \cup \tilde{\mu}_2$ are “equivalent.” Recall that for a subset $\mu'$ of $\mu$, $\gamma \vdash_{\mu'} \gamma'$ denotes that $\gamma \vdash \gamma'$ is results in a transition from $\mu'$.

Let $w = (a_1^1) \cdots (a_n^m)$ be a data word and let $\theta$ be a pebble-2 assignment on $\triangleleft w \triangleright$. For each $i = 0, \ldots, n + 1$, we denote by $\theta_i$ the pebble-1 assignment
such that

\[ \theta_i(j) = \begin{cases} 
\theta(j), & \text{if } j = 2, \ldots, k, \\
i, & \text{if } j = 1.
\end{cases} \]

We will prove first that the transitions in \( \mu_1 \cup \mu_2 \) can be simulated by the transitions in \( \mu'_1 \cup \tilde{\mu}_2 \). Assume

\[ [2, p_1, \theta] \vdash_{\mu_2} [1, p_2, \theta_{n+1}] \vdash_{\mu_1} [1, p_3, \theta_{n+1}] \vdash_{\mu_1} [2, p_4, \theta]. \]

Then there is a closed term \( p_2 \to S \in R(w, \theta_{n+1}) \) such that \( S \subseteq Q_2 \) and \( p_4 \in S \).

We will show that there is a “deterministic” run from the configuration \([2, p_1, \theta]\) to the configuration \([2, p_4, \theta]\) by the means of the transitions in \( \mu'_1 \cup \tilde{\mu}_2 \).

By the definition of \( \tilde{\mu}_2 \), we have

\[ [2, p_1, \theta] \vdash_{\tilde{\mu}_2} [1, (p_2, \emptyset), \theta_0], \quad (4.1) \]
and, by the definition of \( \mu'_1 \),

\[
[1, (p_2, \emptyset), \theta_0] \vdash_{\mu'_1} [1, (p_2, R(\prec w\succ, \theta_0)), \theta_1].
\] (4.2)

Then, applying Proposition 10 repeatedly, we obtain

\[
[1, (p_2, R(\prec w\succ, \theta_0)), \theta_1] \vdash_{\mu'_1} [1, (p_2, R(\prec w\succ, \theta_1)), \theta_2]
\]

\[
[1, (p_2, R(\prec w\succ, \theta_1)), \theta_2] \vdash_{\mu'_1} [1, (p_2, R(\prec w\succ, \theta_2)), \theta_3]
\]

\[
\vdots
\]

\[
[1, (p_2, R(\prec w\succ, \theta_{n-1})), \theta_n] \vdash_{\mu'_1} [1, (p_2, R(\prec w\succ, \theta_n)), \theta_{n+1}],
\]

implying

\[
[1, (p_2, R(\prec w\succ, \theta_0)), \theta_1] \vdash^{*}_{\mu'_1} [1, (p_2, R(\prec w\succ, \theta_n)), \theta_{n+1}] \quad (4.3)
\]

Again, by the definition of \( \mu'_1 \),

\[
[1, (p_2, R(\prec w\succ, \theta_0)), \theta_1] \vdash_{\mu'_1} [1, (p_2, R(\prec w\succ, \theta_n)), \theta_{n+1}]
\] (4.4)

where \( p_2 \rightarrow S_j \in \text{CTH}(\prec w\succ, \theta_{n+1}), \) \( j = 1, \ldots, m. \)

Let \( S = S_j \). Then, by the definition of \( \tilde{\mu}_2 \),

\[
[2, \{S_1, \ldots, S_m\}, \theta] \vdash_{\tilde{\mu}_2} [2, S_j, \theta], \quad (4.5)
\]

and, since \( p_4 \in S \),

\[
[2, S_j, \theta] \vdash_{\tilde{\mu}_2} [2, p_4, \theta]. \quad (4.6)
\]

Finally, combining (1)–(6), we obtain the desired run

\[
[2, p_1, \theta] \vdash_{\tilde{\mu}_2} [1, (p_2, \emptyset), \theta_0]
\]

\[
\vdash^{*}_{\mu'_1} [1, (p_2, R(\prec w\succ, \theta_n)), \theta_{n+1}]
\]

\[
\vdash_{\mu'_1} [2, \{S_1, \ldots, S_m\}, \theta]
\]

\[
\vdash_{\tilde{\mu}_2} [2, S_j, \theta]
\]

\[
\vdash_{\tilde{\mu}_2} [2, p_4, \theta].
\]

Next we will show how the transitions in \( \mu'_1 \cup \tilde{\mu}_2 \) are simulated by the transitions in \( \mu_1 \cup \mu_2 \). Assume that we have the following sequence of computation steps.
(1) \([2, q, \theta] \vdash \bar{\mu}_2 [1, (p, \emptyset), \theta_0]\)

(2) \([1, (p, \emptyset), \theta_0] \vdash \mu'_1 [1, (p, R(<w>, \theta_0)), \theta_1]\)

(3) \([1, (p, R(<w>, \theta_0)), \theta_1] \vdash \mu'_1 \cdots \vdash \mu'_1 [1, (p, R(<w>, \theta_n)), \theta_{n+1}]\)

(4) \([1, (p, R(<w>, \theta_n)), \theta_{n+1}] \vdash \mu'_1 [2, \{S_1, \ldots, S_m\}, \theta]\)

(5) \([2, \{S_1, \ldots, S_m\}, \theta] \vdash \bar{\mu}_2 [2, S_i, \theta]\)

(6) \([2, S_i, \theta] \vdash \bar{\mu}_2 [2, s, \theta]\), for each \(s \in S_i\).

By the definition of \(\bar{\mu}_2\), (1) implies
\[\[2, q, \theta] \vdash \mu_2 [1, p, \theta_{n+1}]\],
and, by the definition of \(\mu'_1\) and Proposition 10, (2)–(4) imply
\[p \rightarrow S_i \in R(<w>, \theta_{n+1}), \text{ where } S_i \subseteq Q_2.\]

That is, for each \(s \in S_i\),
\[\[1, p, \theta_{n+1}] \vdash^* \mu_1 [2, s, \theta]\],
completing the proof of “equivalence” of \(\mu_1 \cup \mu_2\) and \(\mu'_1 \cup \bar{\mu}_2\).

4.2.2 Determinizing pebble \(i\)

Now, assuming that the behavior of pebbles \(1, \ldots, i-1\) is one-way and deterministic, we will determinize pebble \(i\). The determinization will result in placing pebble \(i\) above the left-end marker \(<\) and lifting the pebble when it reaches the right-end marker \(>\).

The idea is very similar to that in Subsection 4.2.1, with the exception that now during the computation the automaton can place pebble \((i-1)\). The effect of such placement is the state of pebble \(i\) changes, i.e., the state the automaton enters after pebble \((i-1)\) is lifted is not necessarily the same in which it has been placed. Figure 4.3 is an example of a sequence of moves of pebble 2 of a two pebble automaton \(A\) on the input data word \((\sigma_1^a)(\sigma_2^a)(\sigma_3^a)(\sigma_4^a)\). Recall that, by our normalization of \(A\) in Section 4.2, the computation starts with pebble 2 above the right-end marker \(>\). We assume
that the behavior of pebble 1 is already determinized as explained in the previous section.

For example, the pair of states \((q_2, q'_2)\) in the run of pebble 1 indicates that pebble 2 first arrives at the symbol \(\sigma_3\) when \(A\) is in the state \(q_2\), upon which pebble 1 is placed. When pebble 1 has finally finished its computation, i.e., when it is lifted after reading the right-end marker \(\triangledown\), \(A\) enters the state \(q'_2\) from which pebble 2 continues the computation. This pair \((q_2, q'_2)\) can be viewed as the term \(q_2 \rightarrow \{q'_2\}\) and has to be included as an “axiom” in the proof system \(\text{TH}(R, \sigma, \emptyset, \emptyset)\). The corresponding modifications of the proof system and the automaton are elaborated below.

Let \(Q_1, \ldots, Q_{i-1}\) be the set of states of pebbles 1, \ldots, \((i-1)\), respectively, and let \(\mu_1, \ldots, \mu_{i-1}\) be the set of transitions of pebbles 1, \ldots, \((i-1)\), respectively. We assume that the behavior of pebbles 1, \ldots, \((i-1)\), according to \(\mu_1, \ldots, \mu_{i-1}\), is deterministic.

Let \(w = (\sigma^1_{a_1}) \cdots (\sigma^n_{a_n})\) and \(\theta\) be a pebble-\(i\) assignment on \(w\). We define a set of terms \(s(\mu_i, w, \theta)\) as follows. For \(p, q \in Q_i\), the term \(p \rightarrow \{q\}\) is in \(s(\mu_i, w, \theta)\) if and only if there are \(s_1, s_2 \in Q_{i-1}\) such that

1. \((i, \sigma_{\theta(i)}, P, V, p) \rightarrow (s_1, \text{place-pebble}) \in \mu_i\), where
   - \(P = \{l > i : \theta(l) = \theta(i)\}\) and
   - \(V = \{l > i : a_{\theta(l)} = a_{\theta(i)}\}\);
2. \([i-1, s_1, \theta_0] \vdash [i-1, s_2, \theta_{n+1}],\) where
   - \(\theta_m(i-1) = m, m = 0, \ldots, n + 1,\) and
   - \(\theta_m(j) = \theta(j), m = 0, \ldots, n + 1, j = i, \ldots, k,\)

and

Figure 4.3: The sequence of moves of \(A\) on \((\sigma^1_{a_1})(\sigma^2_{a_2})(\sigma^3_{a_3})(\sigma^4_{a_4})\).
3. \((i, \nu, \emptyset, \emptyset, s_2) \to (q, \text{lift-pebble}) \in \mu_{i-1}\).

Since pebbles \(1, \ldots, (i-1)\) all behave deterministically, for each \(p \in Q, \text{place}\), there is exactly one state \(q \in Q_i\) such that the term \(p \to \{q\}\) is in \(s(\mu_i, w, \theta)\).

For a pebble-\(i\) assignment \(\theta\), we define the response \(R(w, \theta)\) as follows. For a set \(S \subseteq Q\), a closed term \(q \to \bar{S}\) belongs to \(R(w, \theta)\) if there is a computation tree \(T\) of \(A\) on \(w\) whose root is labeled \([i, q, \theta]\) such that

- if \(\theta(i) \leq n\), then each leaf is labeled \([i, p, \text{Succ}(\theta)]\) for some \(p \in S\);
- if \(\theta(i) = n + 1\), then each leaf is labeled \([i + 1, p, \text{Succ}(\theta)]\) for some \(p \in S\);
- each internal node in \(T\) is labeled \([j, q', \theta']\), where
  - \(j \leq i\) and
  - if \(j = i\), then \(0 \leq \theta'(i) \leq \theta(i)\);
- for each \(p \in S\), there is a leaf labeled \([1, p, \text{Succ}(\theta)]\).

The partial response \(PR(w, \theta)\) is defined in a similar manner. For a set \(S \subseteq Q \cup \bar{Q}\), a term \(q \to S\) belongs to \(PR(w, \theta)\) if there is a computation tree \(T\) of \(A\) on \(w\) whose root is labeled \([i, q, \theta]\) such that

- if \(\theta(i) \leq n\), then each leaf is labeled either \([i, p, \text{Succ}(\theta)]\) for some \(\bar{p} \in S\) or \([i, p, \theta]\) for some \(p \in S\);
- if \(\theta(i) = n + 1\), each leaf is labeled either \([i + 1, p, \text{Succ}(\theta)]\) for some \(\bar{p} \in S\) or \([i, p, \theta]\) for some \(p \in S\);
- each internal node in \(T\) is labeled \([j, q', \theta']\), where
  - \(j \leq i\) and
  - if \(j = i\), then \(0 \leq \theta'(i) \leq \theta(i)\);
- if \(\theta(i) \leq n\), for each \(\bar{p} \in S\), there is a leaf labeled \([i, p, \text{Succ}(\theta)]\);
- if \(\theta(i) = n + 1\), for each \(\bar{p} \in S\), there is a leaf labeled \([i + 1, p, \text{Succ}(\theta)]\); and
- for each \(p \in S\), there is a leaf labeled \([i, p, \theta]\).
Finally, the pebble $i$ proof system is exactly $S(R, \sigma, P, V)$ of pebble 1.

Proposition 11 below is a generalization of Proposition 10 and the proof of the former is similar to that of the latter, and, thus omitted.

**Proposition 11**  For each data word $w = (\sigma_1) \cdots (\sigma_n)$ and pebble-$i$ assignment $\theta$ on $\langle w \rangle$,

$$\mathcal{P}R(w, \theta) = TH(P, \sigma, P, V),$$

where

- $\mathcal{P} = R(w, Pred(\theta)) \cup s(\mu_i, w, \theta)$;
- $1 \leq \theta(i) \leq n + 1$;
- $P = \{l > i : \theta(l) = \theta(i)\}$;
- $V = \{l > i : a_{\theta(l)} = a_{\theta(i)}\}$; and
- $\sigma = \sigma_{\theta(i)}$.

We start with an intuitive description of the deterministic simulation of pebble $i$. The “main” states of pebble $i$ are still of the form $(q, R)$, where $q \in Q_i$ and $R$ is a response.

Let $w = (\sigma_1) \cdots (\sigma_n)$ be an input word, $\theta$ be a pebble-$i$ assignment such that $1 \leq \theta(i) \leq n$, and let $R$ be a response. From the configuration $[i, (q, R), \theta]$ the automaton acts as follows.

1. Places pebble $(i - 1)$ and simulates it starting from each possible state, in order to compute the set of terms $s(\mu_i, w, \theta)$.

2. Then pebble $i$ moves right and enters the state $(q, CTH(P, \sigma, P, V))$, where

   - $\mathcal{P} = R \cup s(\mu_i, w, \theta)$,
   - $\sigma = \sigma_{\theta(i)}$,
   - $P = \{l > i : \theta(l) = \theta(i)\}$, and
   - $V = \{l > i : a_{\theta(l)} = a_{\theta(i)}\}$.
The formal definitions are as follows. Let \( Q_1, \ldots, Q_i \) be the sets of states of pebbles \( 1, \ldots, i \), respectively, and let \( \mu_1, \ldots, \mu_i \) be the sets of transitions of pebbles \( 1, \ldots, i \), respectively. Recall that the behavior of the pebbles \( 1, \ldots, (i - 1) \) according to \( \mu_1, \ldots, \mu_{i-1} \), is deterministic.

Like in the case of pebble 1, we need to modify a bit the behavior of pebble \( (i + 1) \). The modifications \( \tilde{Q}_{i+1}, \tilde{U}_{i+1}, \tilde{N}_{i+1}, \tilde{D}_{i+1}, \) and \( \tilde{\mu}_{i+1} \), of \( Q_{i+1}, U_{i+1}, N_{i+1}, D_{i+1}, \) and \( \mu_{i+1} \), respectively, are as follows.

- \( \tilde{Q}_{i+1} = Q_{i+1} \cup 2^{Q_{i+1}} \cup 2^{2^{Q_{i+1}}} \);
- \( \tilde{U}_{i+1} = U_{i+1} \cup 2^{Q_{i+1}} - \{ \emptyset \} \);
- \( \tilde{N}_{i+1} = N_{i+1} \cup 2^{2^{Q_{i+1}}} \);
- \( \tilde{D}_{i+1} = D_{i+1} \), and
- \( \tilde{\mu}_{i+1} \) consists of the following transitions.

  - Every transition in \( \mu_{i+1} \), in which the action is not “place-pebble” is in \( \tilde{\mu}_{i+1} \).

  - For each \( \sigma \in \Sigma, P, V \subseteq \{i + 2, \ldots, k\} \), and \( S_1, \ldots, S_m \subseteq Q_{i+1} \), \( \tilde{\mu}_{i+1} \) contains the transition
    \[
    (i + 1, \sigma, P, V, \{S_1, \ldots, S_m\}) \rightarrow (S_j, \text{stay})
    \]
    for each \( j = 1, 2, \ldots, m \). By definition, from the state \( \{S_1, \ldots, S_m\} \in \tilde{Q}_{i+1} \) pebble \( (i + 1) \) performs existential branching.

  - For each \( \sigma \in \Sigma, P, V \subseteq \{i + 2, \ldots, k\} \), and \( S \subseteq Q_{i+1} \), \( \tilde{\mu}_{i+1} \) contains the transition
    \[
    (i + 1, \sigma, P, V, S) \rightarrow (q, \text{stay})
    \]
    for each \( q \in S \). By definition, from the state \( S \subseteq Q_{i+1} \) pebble \( (i + 1) \) performs universal branching.

  - For each transition \( (i + 1, \sigma, P, V, q) \rightarrow (p, \text{place-pebble}) \in \mu_{i+1} \), \( \tilde{\mu}_{i+1} \) contains the transition
    \[
    (i + 1, \sigma, P, V, q) \rightarrow ((p, \emptyset), \text{place-pebble}).
    \]

That is, like in the case of pebble 1, the transition \( (i + 1, \sigma, P, V, q) \rightarrow (p, \text{place-pebble}) \) of \( \mu_{i+1} \) is replaced with the transition \( (i + 1, \sigma, P, V, q) \rightarrow ((p, \emptyset), \text{place-pebble}) \).
Next, we define the sets of states $Q'_1, \ldots, Q'_i$ and the sets of transitions $\mu'_1, \ldots, \mu'_i$ such that the behavior of pebbles $1, \ldots, i$ according to $\mu'_1, \ldots, \mu'_i$ is deterministic. We start with the definition of the sets of states $Q'_1, \ldots, Q'_i$.

- $Q'_i$ consists of elements of the forms
  - $(q, PR)$, where $q \in Q_i$ and $PR$ is a partial response and
  - $(q, X, PR)$ where $q \in Q_i$, $X \subseteq Q_{i,\text{place}}$ and $PR$ is a partial response.

The intuitive meaning of the state $(q, PR)$ is like in the previous section, and the state $(q, X, PR)$ is used to simulate pebble $(i - 1)$ in order to compute the set $s(\mu_i, w, \theta)$. The set $X$ is supposed to contain the states of pebble $i$ from which the automaton has yet to simulate pebble $(i - 1)$.

- For each $j = 1, \ldots, i - 1$, the states in $Q'_j$ are of the form
  $$((q, X, PR, s), p)$$
  where $q \in Q_i$, $X \subseteq Q_{i,\text{place}}$, $PR$ is a partial response, $s \in Q_{i,\text{place}}$, and $p \in Q_j$. The intuitive meaning of these states is as follows.

  - The triple $(q, X, PR)$ is used to remember the state of pebble $i$ while simulating pebble $(i - 1)$;
  - the component $s \in Q_{i,\text{place}}$ is used to remember the starting state of the simulation of pebble $(i - 1)$; and
  - the last component $p \in Q_j$ is the current state of the simulation.

The sets of transitions $\mu'_1, \ldots, \mu'_{i-1}$ are defined as follows.

- For each $j = 1, \ldots, i - 2$, and each transition $(j, \sigma, P, V, p) \rightarrow (t, \text{act})$ in $\mu_j$, $\mu'_j$ contains the transition
  $$((j, \sigma, P, V, ((q, X, PR, s), p)) \rightarrow ((q, X, PR, s), t), \text{act}).$$

- For each transition $(i - 1, \sigma, P, V, p) \rightarrow (t, \text{act})$ in $\mu_{i-1}$ such that $\text{act} \neq \text{lift-pebble}$, $\mu'_{i-1}$ contains the transition
  $$(i - 1, \sigma, P, V, ((q, X, PR, s), p)) \rightarrow ((q, X, PR, s), t), \text{act}).$$
• For each transition \((i - 1, \triangleright, \emptyset, \emptyset, p) \rightarrow (t, \text{lift-pebble})\) in \(\mu_{i-1}\), \(\mu'_{i-1}\) contains the transition
\[
(i - 1, \triangleright, \emptyset, ((q, X, \mathcal{PR}, s), p)) \rightarrow
((q, X, \mathcal{PR} \cup \{s \rightarrow \{t\}\}), \text{lift-pebble}) \in \mu'_{i-1}.
\]

Finally, \(\mu'_{i}\) consists of the following transitions.

• For each \(q \in Q_i\), \(\mu'_{i}\) contains the transition
\[
(i, \triangleleft, \emptyset, ((q, \emptyset))) \rightarrow ((q, \mathcal{R}), \text{right}),
\]
where \(\mathcal{R} = \mathcal{R}(w, \theta)\), for some \(w\) and \(\theta\) such that \(\theta(i) = 0\). By Remark 9, this \(\mathcal{R}(w, \theta)\) is well defined.

• For each state \(q \in Q_i\), each response \(\mathcal{R}\), each label \(\sigma \in \Sigma\), and each \(P, V \subseteq \{i + 1, \ldots, k\}\), \(\mu'_{i}\) contains the transition
\[
(i, \sigma, P, V, (q, \mathcal{R})) \rightarrow ((q, Q_{i, \text{place}}, \mathcal{R}), \text{stay}).
\]
This transition starts the computation of the set of terms \(s(\mu_i, w, \theta)\).

• For each state \(q \in Q_i\), each partial response \(\mathcal{PR}\), each nonempty set \(X \subseteq Q_{i, \text{place}}\), each label \(\sigma \in \Sigma\), and each \(P, V \subseteq \{i + 1, \ldots, k\}\), \(\mu'_{i}\) contains the transition
\[
(i, \sigma, P, V, (q, X, \mathcal{PR})) \rightarrow ((q, X - \{s\}, \mathcal{PR}, s, t), \text{place-pebble}),
\]
where \(X \neq \emptyset\), \(s \in X\), and \((i, \sigma, P, V, s) \rightarrow (t, \text{place-pebble})\). The purpose of these transitions is to simulate pebble \((i - 1)\) from the state \(s\), that is the state of pebble \(i\) before pebble \((i - 1)\) is placed for the simulation. Note that this is a place-pebble transition, i.e., the state \((q, X - \{s\}, \mathcal{PR}, s, t)\) is in \(Q'_{i-1}\).

• For each state \(q \in Q_i\), each partial response \(\mathcal{PR}\), each label \(\sigma \in \Sigma\), and each \(P, V \subseteq \{i + 1, \ldots, k\}\), \(\mu'_{i}\) contains the transition
\[
(i, \sigma, P, V, (q, \emptyset, \mathcal{PR})) \rightarrow ((q, \text{CTH}(\mathcal{PR}, \sigma, P, V)), \text{right}).
\]
The purpose of these transitions is as follows. When the automaton has finished the simulation of pebble \((i - 1)\) from all possible states, which is indicated by \(X = \emptyset\), pebble \(i\) computes \(\text{CTH}(\mathcal{PR}, \sigma, P, V)\), enters the state \((q, \text{CTH}(\mathcal{PR}, \sigma, P, V))\), and moves right.
• Finally, $\mu'_i$ contains the transition

$$(i, \triangleright, \emptyset, \emptyset, (q, R)) \rightarrow (\{S_1, \ldots, S_m\}, \text{lift-pebble}),$$

where $S_j \subseteq Q_{i+1}$ and $q \rightarrow S_j \in \text{CTH}(R, \triangleright, \emptyset, \emptyset)$, for each $j = 1, \ldots, m$.

The purpose of these existential branching transitions is the same as their pebble 1 counterpart.\(^3\)

The proof of the equivalence of $\mu_1 \cup \cdots \cup \mu_i \cup \mu_{i+1}$ and $\mu'_1 \cup \cdots \cup \mu'_i \cup \tilde{\mu}_{i+1}$ is similar to the corresponding proof for the case of pebble 1 and is omitted.

### 4.2.3 Determinizing the automaton $\mathcal{A}$

At last, we define the deterministic $k$-PA $\mathcal{A}' = \langle Q', q'_0, \mu', F' \rangle$ that accepts the language of $\mathcal{A} = \langle Q, q_0, \mu, F \rangle$. By the induction step described in the previous section, we assume that the behavior of pebbles $1, \ldots, k-1$ is deterministic.

• $Q' = Q'_1 \cup \cdots \cup Q'_{k-1} \cup Q'_k \cup \{q_{\text{acc}}, q_{\text{rej}}\}$, where $Q'_1, \ldots, Q'_{k-1}, Q'_k$ are the modification of the sets of states $Q_1, \ldots, Q_{k-1}, Q_k$, respectively, as defined in the previous section;

• $q'_0 = (q_0, \emptyset)$;

• $F' = \{q_{\text{acc}}\}$; and

• $\mu'$ consists of the transitions of $\mu'_1 \cup \cdots \cup \mu'_{k-1} \cup \mu'_k$, where $\mu'_1, \ldots, \mu'_{k-1}, \mu'_k$ are the modification of the set of transitions $\mu_1, \ldots, \mu_{k-1}, \mu_k$, respectively, as defined in the previous section, and the following transitions.

\[
(k, \triangleright, \emptyset, \emptyset, (q_0, R)) \rightarrow (q_{\text{acc}}, \text{right}),
\]

if there is a subset $S$ of $F$ such that $q_0 \rightarrow \overline{S} \in \text{CTH}(R, \triangleright, \emptyset, \emptyset)$; and

\[
(k, \triangleright, \emptyset, \emptyset, (q_0, R)) \rightarrow (q_{\text{rej}}, \text{right}),
\]

if there is no subset $S$ of $F$ such that $q_0 \rightarrow \overline{S} \in \text{CTH}(R, \triangleright, \emptyset, \emptyset)$.

\(^3\)Since no new pebble is placed when the head pebble is above the right-end marker $\triangleright$, there is no need to compute the set of terms $s(\mu_i, w, \theta)$. 

43
The proof of the equivalence $\mathcal{A}$ and $\mathcal{A}'$ is similar to the corresponding proof for the case of pebble 1, and, again, is omitted.

4.3 Converting an alternating weak PA into a deterministic one

In this section we prove that alternating and deterministic weak $k$-PA have the same recognition power. For every one-way alternating weak $k$-PA, we will construct its equivalent one-way deterministic weak $k$-PA. This is done in two steps.

1. First, we show how to transform the one-way nondeterministic weak $k$-PA into its equivalent one-way deterministic weak $k$-PA.

2. Then, we show how to transform the one-way alternating weak $k$-PA into its equivalent one-way nondeterministic weak $k$-PA.

4.3.1 From nondeterministic to deterministic

We start with the simple case. We will show how to determinize nondeterministic weak 2-PA. The idea can be generalized to arbitrary number of pebbles.

Let $\mathcal{A} = (Q, q_0, F, \mu)$ be a nondeterministic weak 2-PA. We start by normalizing the behavior of $\mathcal{A}$ as follows.

N1. For every configuration $\gamma$ of $\mathcal{A}$, there exists a transition in $\mu$ that applies to it.

N2. Only pebble 2 can enter a final state and it does so only after it reads the right-end marker $\triangleright$.

N3. Immediately after pebble 2 moves right, pebble 1 is placed.

N4. Pebble 1 is lifted only when it reaches the right-end marker $\triangleright$.

Such normalization can be done by adding some extra states to $\mathcal{A}$. The normalization N4 is especially important, as it implies that nondeterminism on pebble 1 is now limited only to deciding which state to enter. There is no
nondeterminism in choosing which action to take, i.e. either to lift pebble 1 or to keep on moving right.

Next, we note that immediately after pebble 1 is lifted, there can be two choices of actions for pebble 2:

- to place pebble 1 again; or
- moves pebble 2 to the right.

The following fifth normalization is supposed to handle this situation:

N5. Immediately after pebble 1 is lifted, pebble 2 moves right.

In other words, while pebble 2 is reading a specific position, pebble 1 makes exactly one pass, from the position of pebble 2 to the right end of the input, instead of making several rounds of passes by placing pebble 1 again immediately after it is lifted. Since there are only finitely many states, there can only be finitely many passes. So, the normalization N4 can be achieved by simultaneously simulating all possible passes in one pass.

With the normalization N1–N5, there is no nondeterminism in choosing which action to take for pebble 2. The same as for pebble 1, the nondeterminism for pebble 2 is now limited only in deciding which states to take. This is summed up in the following remark.

**Remark 12** For each $i = 1, 2$, if $(i, P, V, p) \rightarrow (q_1, \text{act}_1)$ and $(i, P, V, p) \rightarrow (q_2, \text{act}_2)$, then $\text{act}_1 = \text{act}_2$.

Now that the nondeterminism is reduced to deciding which state to enter, the determinization of $\mathcal{A}$ becomes straightforward. Similar to the classical proof of the equivalence between nondeterministic and deterministic finite state automata, we can take the power set of the states of $\mathcal{A}$ to deterministically simulate $\mathcal{A}$.

Now the normalization steps N1–N5 can be performed similarly for weak $k$-PA $\mathcal{A}$.

N1’. For every configuration $\gamma$ of $\mathcal{A}$, there exists a transition in $\mu$ that applies to it.

N2’. Only pebble $k$ can enter a final state and it does so only after it reads the right-end marker $\triangleright$. 

45
N3’. For each \( i = 2, \ldots, k \), immediately after pebble \( i \) moves right, pebble \( (i - 1) \) is placed.

N4’. For each \( i = 1, \ldots, k - 1 \), pebble \( i \) is lifted only when it reaches the right-end marker \( \triangleright \).

N5’. For each \( i = 1, \ldots, k - 1 \), immediately after pebble \( i \) is lifted, pebble \( i + 1 \) moves right.

The implication of such normalization is the nondeterminism is reduced to only deciding which state to enter. Then, similar to the case of 2 pebble, we can take the power set of the states of \( A \) to deterministically simulate \( A \).

4.3.2 From alternating to nondeterministic

Let \( A = (\Sigma, Q, q_0, \mu, F, U) \) be one-way alternating weak \( k \)-PA. Adding some extra states, we can normalize \( A \) as follows.

- For every \( p \in U \), if \( (i, \sigma, V, p) \rightarrow (q, \text{act}) \in \mu \), then \( \text{act} = \text{stay} \).
- Every pebble can be lifted only after it reads the right-end marker \( \triangleright \).
- Only pebble \( k \) can enter a final state and it does so only after it reads the right-end marker \( \triangleright \).

We assume that \( Q \) is partitioned into \( Q_1 \cup \cdots \cup Q_k \) where \( Q_i \) is the set of states, where pebble \( i \) is the head pebble, for each \( i = 1, \ldots, k \). We can further partition each \( Q_i \) into four sets of states: \( Q_{i, \text{stay}} \), \( Q_{i, \text{right}} \), \( Q_{i, \text{place}} \), \( Q_{i, \text{lift}} \) such that for every \( i \), \( \sigma \), \( V \), \( q \) and \( p \),

- if \( q \in Q_{i, \text{stay}} \) and \( (i, \sigma, V, q) \rightarrow (p, \text{act}) \in \mu \), then \( \text{act} = \text{stay} \);
- if \( q \in Q_{i, \text{right}} \) and \( (i, \sigma, V, q) \rightarrow (p, \text{act}) \in \mu \), then \( \text{act} = \text{right} \);
- if \( q \in Q_{i, \text{place}} \) and \( (i, \sigma, V, q) \rightarrow (p, \text{act}) \in \mu \), then \( \text{act} = \text{place-peek} \);
- if \( q \in Q_{i, \text{lift}} \) and \( (i, \sigma, V, q) \rightarrow (p, \text{act}) \in \mu \), then \( \text{act} = \text{lift-peek} \).

Now we define a nondeterministic weak \( k \)-PA \( A' = (\Sigma, Q', q'_0, \mu', F') \) where \( Q' = 2^Q \), \( q'_0 = \{ q_0 \} \) and \( F' = 2^F \).

The set \( \mu' \) contains the following transitions. For every \( i = 1, 2, \ldots, k \), for every \( V \subseteq \{ i + 1, \ldots, k \} \), for every \( S \in Q' \), for every \( \sigma \in \Sigma \),
• if $S$ contains a state $q \in U$, then
  \[ (i, \sigma, V, S) \rightarrow (((S - \{q\}) \cup S_q), \text{stay}) \in \mu' \]
  where $S_q = \{p \mid (i, \sigma, V, q) \rightarrow (p, \text{stay}) \in \mu\}$;
• if $S$ contains a state $q \in Q_{i,\text{stay}}$ and $S \cap U = \emptyset$, then
  \[ (i, \sigma, V, S) \rightarrow (((S - \{q\}) \cup \{p\}), \text{stay}) \in \mu' \]
  where $(i, \sigma, V, q) \rightarrow (p, \text{stay}) \in \mu$;
• if $S$ contains a state $q \in Q_{i,\text{place}}$ and $S \cap Q_{i,\text{stay}} = \emptyset$, then
  \[ (i, \sigma, V, S) \rightarrow (((S - \{q\}) \cup \{p\}), \text{place-pebble}) \in \mu' \]
  where $(i, \sigma, V, q) \rightarrow (p, \text{place-pebble}) \in \mu$; and
• if $S \cap Q_{i,\text{place}} = \emptyset$ and $(S \cap Q_i) \subseteq Q_{i,\text{right}}$, then
  \[ (i, \sigma, V, S) \rightarrow ((S - (S \cap Q_i)) \cup S', \text{right}) \in \mu' \]
  where $S' = \{p \mid (i, \sigma, V, q) \rightarrow (p, \text{right}) \in \mu$ and $q \in S \cap Q_i\}$.

The following proposition immediately implies that $L(A) = L(A')$.

**Proposition 13** For every $w \in (\Sigma \times D)^*$ of length $n$, for every $S \subseteq Q$, for every $i = 1, \ldots, k$, for every pebble assignment $\theta$, the following Statements 1 and 2 are equivalent.

1. The initial configuration $[0, q_0, \theta_0]$ leads to acceptance and $S$ is the set of states such that
   • $[0, q_0, \theta_0] \vdash [i, p, \theta]$, for all $p \in S$;
   • $[0, q_0, \theta_0]$ leads to acceptance if and only if $[i, p, \theta]$ leads to acceptance, for all $p \in S$.

2. There exists an accepting run of $A'$ on $w$
   \[ [0, \{q_0\}, \theta_0] \vdash^*_{A'} [i, S, \theta] \vdash^*_{A'} [k, R, \theta_f] \]
   for some $R \in F'$ and pebble assignment $\theta_f$, where $\theta_f(k) = n + 1$.

The proof is by routine induction on the run of $A$ on $w$.  

47
Chapter 5

Hierarchy of PA languages

In this chapter we are going to establish the strict hierarchy of PA languages. More precisely, we are going to show that for every \( k = 1, 2, \ldots, (k + 1) \)-PA are more strictly powerful than \( k \)-PA. We also show a similar result for the case of weak PA. In addition, we also establish the inclusion in Theorem 7 is strict. That is, MSO[\( \Sigma, +1, <, \sim \)] is strictly stronger that PA.

Throughout this chapter, the set \( \Sigma \) consists of only one label \( \sigma \). Therefore, for the convenience of presentation, we will omit the label in presenting the data word. Instead of writing \( (\sigma a) \), we will write only \( a \), where \( a \) is a data value in \( \mathcal{D} \).

The main idea of this chapter is viewing a word of even length \( w = a_0b_0 \cdots a_nb_n \) as a directed graph \( G_w = (V_w, E_w) \) with the symbols that appear in \( a_0b_0 \cdots a_nb_n \) as the vertices in \( V_w \) and \( (a_0, b_0), \ldots, (a_n, b_n) \) as the edges in \( E_w \). We say that \( w \) induces the graph \( G_w \).

With regards to graph reachability, we prove that PA behaves similarly to the first order logic: For any positive integer \( k \), \( k \) pebbles are sufficient for recognizing the existence of a path of length \( 2^k - 1 \) from the vertex \( a_0 \) to the vertex \( b_n \), but are not sufficient for recognizing the existence of a path of length \( 2^{k+1} - 2 \) from the vertex \( a_0 \) to the vertex \( b_n \).

Based on this result, we establish the following relationships among the classes of languages over infinite alphabets which were previously unknown.

1. The strict hierarchy of the PA languages based on the number of pebbles.

2. The separation of monadic second order logic from the PA languages.
3. The separation of the one-way deterministic FMA languages from the PA languages.

**Related work.** There is an analogy between the result in this chapter with the classical first-order quantifier lower bounds for \((s, t)\)-reachability which states as follows. \((s, t)\)-reachability in graphs is expressible in a first order sentence of quantifier rank \(k\) if and only if the distance from \(s\) to \(t\) is less than or equal to \(2^k\). See, for example, [30].

As far as the author can see, this classical result does not imply our result here, as explained in the following paragraphs. In [28], the author introduces a logic for directed graphs, where the domain consists of the edges only, and excludes the vertices. It is established in [28] that every first-order sentence \(\varphi\) in such logic can be appropriately translated into a first-order sentence \(\psi\) in logic over word structure such that \(w \models \psi\) if and only if \(G_w \models \varphi\). Since PA is already stronger than first-order logic (Proposition 6), the result in this paper can be seen as a “stronger” version of the classical result. Indeed, our result in this paper can be used to reestablish the classical quantifier bound for \((s, t)\)-reachability for the logic introduced in [28]. That is, \((s, t)\)-reachability in directed graphs is expressible in a first order sentence of quantifier rank \(k\), where the quantification is over the edges, if and only if the distance from \(s\) to \(t\) is less than or equal to \(2^k\). See [28, Proposition 4, Theorem 5].

Other related results are those established in [2, 10, 24]. To the best of our knowledge, those results has no connection with the result in this paper. In [2] it is established that \((s, t)\)-reachability in directed graph is not in monadic NP, while in [10, 24] it is established that undirected graph connectivity is not in monadic NP. All these results hold even in the presence of built-in relations. However, no lower bound on first-order quantifier rank is established. Furthermore, the logic introduced in [28], which can be seen as a first-order logic for PA as a model of computation for graphs, behaves differently from the standard logic for graphs. In the logic, where the edges are the domain, \((s, t)\)-reachability in directed graph can be expressed in monadic NP. See [28, Proposition 3].
5.1 Words of $\mathcal{D}^*$ as graphs

Let $w = a_0b_0 \cdots a_nb_n \in \mathcal{D}^*$ be a word of even length. The word $w$ induces a directed graph $G_w = (V_w, E_w)$ whose the set of vertices is $V_w = \{a_0, b_0, \ldots, a_n, b_n\}$, that is, the symbols that appear in $w$, and the set of edges is $E_w = \{(a_0, b_0), \ldots, (a_n, b_n)\}$. With such view, we will use the term graph when referring to an even length word. We also write $s_w = a_0$ and $t_w = b_n$ to denote the first and the last symbol in $w$, respectively. For convenience, we consider only the words $w$ in which $s_w$ and $t_w$ occur only once.

We need the following basic graph terminology. Let $a$ and $b$ be vertices in a graph $G$. A path of length $m$ from $a$ to $b$ is a sequence of $m$ edges in $G$: $(a_{i_1}, b_{i_1}), \ldots, (a_{i_m}, b_{i_m})$ such that $a_{i_1} = a$, $b_{i_m} = b$ and for each $j = 1, \ldots, m-1$, $b_{i_j} = a_{i_{j+1}}$. The distance from $a$ to $b$ is the length of a minimal path from $a$ to $b$ in $G$. If there is no path from $a$ to $b$ in $G$, then we set the distance be $\infty$.

We define the following reachability languages. For $m \geq 1$,

$$\mathcal{R}_m = \{w : \text{the distance from } s_w \text{ to } t_w \text{ in } G_w \text{ is } \leq m\}$$

and

$$\mathcal{R} = \bigcup_{m=1,2,\ldots} \mathcal{R}_m$$

**Remark 14** When processing an input word $w$, an automaton $A$ can remember by its state whether its head pebble is currently at an odd-number or even-number position in $w$.\(^1\) Therefore, we will always assume that the input word $w$ is of even length (which also naturally fits the subject of this chapter). In addition, unless indicated otherwise, we always denote an input word $w$ by $a_0b_0 \cdots a_nb_n$. That is, the odd-position symbols are denoted by $a_i$’s and the even-position symbols by $b_i$’s.

Again, we remind the reader that in $k$-PA the pebbles placed on the input word are numbered from $k$ to $i$.

**Proposition 15** For each $k = 1, 2, \ldots$, $\mathcal{R}_{2^k-1} \in \mathcal{PA}_k$.

\(^1\)We count that the leftmost symbol of $w$ is in position 1.
Proof. The proof of this proposition is the standard implementation of Savitch’s algorithm for \((s,t)\)-reachability [22]. It is by induction on \(k\). For the basis, we prove that \(\mathcal{R}_3 \in \text{PA}_2\). On input word \(w = a_0b_0\cdots a_nb_n\), a 2-PA \(\mathcal{A}_2\) acts as follows.

- Pebble 2 iterates through all \(a_i\).

- On each iteration, the automaton uses pebbles 1 to check whether \(b_0 = a_i\).

- If \(b_0 = a_i\), the automaton moves pebble 2 to the right to read \(b_i\), and check whether \(b_i = a_n\).

Now, we prove the induction step. On input word \(w = a_0b_0\cdots a_nb_n\), a \(k\)-PA \(\mathcal{A}_k\) acts as follows.

- Pebble \(k\) iterates through all \(a_i\).

- On each iteration, the automaton recursively uses pebbles \((k-1), \ldots, 1\) to check whether there is a path from \(a_0\) to \(a_i\). That is, \(\mathcal{A}_k\) makes a recursive call of \(\mathcal{A}_{k-1}\).

- If such path exists, the automaton lifts all pebbles \(1, \ldots, (k-1)\); moves pebble \(k\) to the right to read \(b_i\), and recursively uses pebbles \(k-1, \ldots, 1\) to check whether there is a path from \(b_i\) to \(b_n\). That is, \(\mathcal{A}_k\) makes a recursive call of \(\mathcal{A}_{k-1}\) one more time.

\(\mathcal{A}_k\) accepts \(w\) if there is an index \(i\) such that there are a path from \(a_0\) to \(a_i\) and a path from \(b_i\) to \(b_n\). It is straightforward to show that \(\mathcal{R}_{2^k-1} \notin \text{PA}_k\). \(\Box\)

Lemma 16 below is the backbone of most of the results presented in this chapter. Its proof is quite long. Therefore, we only sketch it for the case of \(k = 2\) in Section 5.2. A complete proof of the lemma can be found in Section 5.3.

Let \(n_k = 2^{k+1} - 2\), for \(k = 1, 2, \ldots\).

Lemma 16 Let \(\mathcal{A}\) be a \(k\)-pebble automaton such that \(L(\mathcal{A}) \subseteq \mathcal{R}\). Then \(\mathcal{R}_{n_k} \notin L(\mathcal{A})\).

51
Intuitively, the proof is as follows. We will show that for every $k$-PA $A$, there exists $w \in \mathcal{R}_{n_k} - \mathcal{R}_{n_{k-1}}$ such that $A$ cannot recognize a path from $s_w$ to $t_w$. The proof is by induction on $k$. The basis, $k = 1$, is relatively trivial to demonstrate as 1 pebble is not even capable of comparing data values.

Let $A$ be a $k$-PA and let $A_{k-1}$ be $(k-1)$-PA subautomaton of $A$. By the induction hypothesis, there exists $w \in \mathcal{R}_{n_k} - \mathcal{R}_{n_{k-1}}$ such that $A_{k-1}$ cannot recognize a path from $s_w$ to $t_w$. The induction step is as follows. Let $w'$ be a data word isomorphic to $w$ such that $[w] \cap [w'] = \emptyset$.

We define the data word $u$ such that

$$u = w v_1 a_1 a_2 v_2 a_2 a_3 v_3 w';$$

where

- $v_1, v_2, v_3$ depend on the number of states of $A$;
- $a_1 = t_w$;
- $a_3 = s_w'$;
- $a_2, a_3 \notin [w] \cup [w']$.

It is obvious that $u \in \mathcal{R}_{n_k} - \mathcal{R}_{n_{k-1}}$. We claim that $A$ cannot recognize a path from $s_u$ to $t_u$. The proof is roughly as follows.

- When pebble $k$ is above $wv_1a_1a_2$, by the induction hypothesis, the subautomaton $A_{k-1}$ cannot recognize the path from $s_{w'}$ to $t_{w'}$, thus, the path from $s_u$ to $t_u$.
- When pebble $k$ is above $v_2a_2a_3v_3w'$, by the induction hypothesis, the subautomaton $A_{k-1}$ cannot recognize the path from $s_w$ to $t_w$, thus, the path from $s_u$ to $t_u$.

Therefore, the automaton cannot recognize the path from $s_u$ to $t_u$. We want to notify the reader here that the stack discipline of the pebbles is very crucial in the application of the induction hypothesis. Pebble $k$ is fixed on its current position once we start using the other $(k - 1)$ pebbles, and cannot be moved unless all the other $(k - 1)$ pebbles are lifted.

**Corollary 17** $\mathcal{R}_{n_k} \notin PA_k$. 

52
Proof. The corollary immediately follows from the lemma, because \( R_{n_k} \subseteq R \).

Corollary 18 \( R \notin PA \).

Proof. Assume to the contrary that \( R = L(A) \) for a \( k \)-PA \( A \). Then, \( R_{n_k} \subseteq L(A) \), which contradicts Lemma 16.

The following theorem establishes the proper hierarchy of the PA languages.

Theorem 19 For each \( k = 1, 2, \ldots \), \( PA_k \subsetneq PA_{k+1} \).

Proof. By Proposition 15, \( R_{2k+1-1} \in PA_{k+1} \). Suppose \( R_{2k+1-1} \in PA_k \). Let \( A \) be a \( k \)-PA that accepts \( R_{2k+1-1} \). Then, \( L(A) \subseteq R \) and by Lemma 16, \( R_{2k+1-2} \not\subseteq L(A) \). But this is a contradiction since \( L(A) = R_{2k+1-1} \supseteq R_{2k+1-2} \).

Another consequence of Lemma 16 is that the inclusion of PA in MSO[\( \Sigma, +1, <, \sim \)] provided by Theorem 7 is proper. Combined with Theorem 3, it answers a question left open in [21]: Is monadic second order logic for data languages strictly stronger than alternating PA?

Theorem 20 \( PA \subsetneq MSO[\Sigma, +1, <, \sim] \).

Proof. Without loss of generality, we may assume that MSO[\( \Sigma, +1, <, \sim \)] contains two constant symbols, \( \text{min} \) and \( \text{max} \), which denote minimum and the maximum elements of the domain, respectively.\(^2\)

Then the language \( R \) can be expressed in MSO[\( \Sigma, +1, <, \sim \)] as follows. There exist unary predicates \( S_{\text{odd}} \) and \( P \) such that

- \( S_{\text{odd}} \) is the set of all odd elements in the domain where \( \text{min} \in S_{\text{odd}} \) and \( \text{max} \notin S_{\text{odd}} \).
- The predicate \( P \) satisfies the conjunction of the following FO[\( \Sigma, +1, <, \sim \)] sentences:

\(^2\)For a word \( w = a_1 \cdots a_n \), the minimum and the maximum elements are 1 and \( n \), respectively, whereas not 0 and \( n + 1 \) are reserved for the end-markers \( \langle \) and \( \rangle \).
\(- P \subseteq S_{\text{odd}} \text{ and } \min \in P \text{ and } \max - 1 \in P, \)
\(- \text{for all } x \in P - \{\max - 1\}, \text{ there exists exactly one } y \in P \text{ such that } \text{val}(x + 1) = \text{val}(y), \text{ and} \)
\(- \text{for all } x \in P - \{\min\}, \text{ there exists exactly one } y \in P \text{ such that } \text{val}(y + 1) = \text{val}(x). \)

Now, the theorem follows from Corollary 18. \(\square\)

Next, we define a restricted version of the reachability languages. Recall that \([w]\) denotes the set of symbols occurring in \(w\).

For a positive integer \(m \geq 1\), the language \(R_m^+\) consists of all words of the form

\[c_0c_1 \cdots c_1c_2 \cdots c_2c_3 \cdots \cdots c_m \cdots c_m^{m-3}\cdots \cdots c_m^{m-2}\cdots \cdots c_m^{m-1}\cdots \cdots c_m^{m-1}c_m^{m-1}\cdots \cdots c_m^{m-1}c_m^{m-2}\cdots \cdots c_m^{m-1}c_m^{m-1}\cdots \cdots c_m^{m-1}c_m^{m-2}\cdots \cdots c_m^{m-1}c_m^{m-1}\cdots \cdots c_m^{m-1}c_m^{m-2}\cdots \cdots c_m^{m-1}c_m^{m-1}\cdots \cdots c_m^{m-1}c_m^{m-2}\cdots \cdots c_m^{m-1}c_m^{m-1}\]

where

- \(c_0, \ldots, c_m \in \mathcal{D}\), and \(c_i \neq c_{i+1}\), for all \(i = 0, \ldots, m - 1\);
- \(u_1, \ldots, u_{m-1} \in \mathcal{D}^*\) and each of \(u_1, \ldots, u_{m-1}\) is of even length;
- \(\text{for each } i = 1, \ldots, m - 1, c_i \notin [u_i]\).

The language \(R^+\) is defined as

\[R^+ = \bigcup_{m=1,2,\ldots} R_m^+.\]

In some sense, we can compare the relation between \(\mathcal{R}\) and \(R^+\), like the relation between nondeterministic and deterministic transitive closure logic.

The following lemma is the corresponding result to Proposition 15 and Lemma 16.

**Lemma 21** For each \(k = 1, 2, \ldots\), \(R_k^+ \in wPA_k\), but \(R_{k+1}^+ \notin wPA_k\).

The proof of the Lemma 21 can be found in Section 5.4. It immediately implies the strict hierarchy for weak PA languages.

**Theorem 22** For each \(k = 1, 2, \ldots\), \(wPA_k \subsetneq wPA_{k+1}\).
Remark 23 Actually, in the proof of Lemma 16 we show that for every $k$-PA $A$, there exist graphs $G \in \mathcal{R}^+_{n_k}$ and $G' \not\in \mathcal{R}^+$ such that either $A$ accepts both $G$ and $G'$, or rejects both of them. Therefore, $\mathcal{R}^+ \not\in \text{PA}$. In fact, $\mathcal{R}^+$ is not accepted by alternating one-way PA either.

The following theorem answers a question left open in [21, 25]: Can two-way deterministic FMA be simulated by pebble automata.\(^3\)

**Theorem 24** The language $\mathcal{R}^+$ is accepted by one-way deterministic FMA, but is not accepted by pebble automata.

**Proof.** Note that $\mathcal{R}^+$ is accepted by a one-way deterministic one register FMA. On input word $w = a_0b_0 \cdots a_nb_n$, the automaton stores $b_0$ in the register and then moves right until it finds a vertex $a_{i_1} = b_0$. If it finds one, then it stores $b_{i_1}$ in register 1 and moves right again until it finds another vertex $a_{i_2} = b_{i_1}$. It repeats the process until either of the following holds.

- $a_n$ is the same as the content of the register, or,
- it cannot find a vertex currently stored in the register.

In the former case, the automaton accepts the input graph, and in the latter case does not.

However, by Remark 23, $\mathcal{R}^+$ is not a PA language. \(\square\)

### 5.2 Sketch of the proof of Lemma 16

Recall that for $i = 1, 2, \ldots, n_i = 2^{i+1} - 2$. An equivalent recursive definition is $n_1 = 2$, and $n_{i+1} = 2n_i + 2$, for $i \geq 2$.

Let $m > 1$ be a positive integer. We define $G_{i,m}$ to be the set of graphs $G$ of the following form:

\[
G = c_0c_1 \cdots c_{n_i-2}c_{n_i-1} \cdots d_{n_i-2}d_{n_i-1} \cdots e_{n_i-2}e_{n_i-1} \cdots \]

where the strings $v_1, w_1, \ldots, v_{n_i-1}, w_{n_i-1}$ are defined as follows. For each $j = 1, \ldots, n_i - 1$,

- $v_j = e_{j,1}e_{j+1,1} \cdots e_{j,m-1}e_{j+1,m-1}$.

\(^3\)See [13, Definition 1] for the formal definition of FMA.
- \( w_j = f_j,1f_{j+1,1} \cdots f_{j,m-1}f_{j+1,m-1} \); and
- the symbols

\[ \{c_0, \ldots, c_{n_i}, d_0, \ldots, d_{n_i-1}\} \cup \{e_{i,j}, f_{i,j} : 1 \leq i \leq n_i - 1, 1 \leq j \leq m - 1\} \]

are pairwise different.

**Remark 25** Note that \( G_{i,m} \subseteq \mathcal{R}_{n_i} \). We just put \( u_j = v_jd_{j-1}d_jw_j, j = 1, \ldots, n_i - 1 \).

Figure 5.2 illustrates the graph induced by an element of \( \mathcal{G}_{k,m} \).

Let \( G \) be the graph depicted in 5.1. We define the graph \( \overline{G} \) as

\[
\overline{G} = c_0c_1 \cdots d_0d_1 \cdots c_1c_2 \cdots d_1d_2 \cdots \cdots c_{n_i-2}c_{n_i-1} \cdots d_{n_i-2}d_{n_i-1} \quad (5.2)
\]

That is, the graph \( \overline{G} \) is \( G \) without the suffix \( w_{n_i-1}c_{n_i-1}c_{n_i} \). Obviously, this graph is not in \( \mathcal{R} \). Let

\[ \mathcal{G}_{i,m} = \{\overline{G} : \in \mathcal{G}_{i,m}\}. \]

We claim that for each \( k \)-PA \( \mathcal{A} \), there exists an positive integer \( m \) such that \( \mathcal{A} \) cannot “distinguish” between \( G \) and \( \overline{G} \), where \( G \in \mathcal{G}_{k,m} \). The proof is by induction on \( k \) and in the remainder of this section we will sketch it for the case of \( k = 3 \). \(^4\) **Note that in this case, \( n_3 = 14 \).**

Let \( \mathcal{A} = \langle Q, q_0, \mu, F \rangle \) be a 3-PA. We can assume that \( \mathcal{A} \) is one-way and deterministic, and that \( \mu \) does not contain transitions of the form \((i, a, P, V, q) \to (q, \text{act})\), i.e., \( \Theta_\mathcal{A} = \emptyset \). Recall that, pebbles are placed on the input word in the order from 3 to 1.

We normalize the behavior of each pebble as follows. The automaton \( \mathcal{A} \) can enters into a final state only after pebble 3 reaches the right-end marker \( \triangleright \) and for each \( i = 2, 1 \) the following holds.

- immediately after pebble \((i + 1)\) moves right, pebble \( i \) is placed;
- pebble \( i \) is lifted only when it reaches the right end \( \triangleright \) of the input; and
- immediately after pebble \( i \) is lifted, pebble \((i + 1)\) moves right.

\(^4\) By “cannot distinguish” we mean that the automaton finishes its computations on both \( G \) and \( \overline{G} \) in the same state. This notion will be precisely explained in Section 5.3.
\(^5\) The complete proof is presented in Section 5.3.
We will need one more bit of notation. Let $\beta_i$ be defined by the following recursion:

$$\beta_1 = |Q|$$

and, for $i \geq 2$,

$$\beta_i = |Q|^2\left( (\beta_{i-1})! \right)^{|Q|}.$$

Note that $\beta_j$ divides $\beta_i$, whenever $j \leq i$.

**Figure 5.1:** The graph induced by $G \in \mathcal{G}_{k,m}$. 

The twisted arrow $\cup\cup$ represents the order in which the edges are written in $G \in \mathcal{G}_{k,m}$. 

57
Let \( m = \beta_4 \), \(^6\) and let \( G \) and \( \overline{G} \) be as follows.

\[
G = c_0c_1 \cdots d_0d_1 \cdots c_1c_2 \cdots \cdots c_{12}c_{13} \cdots d_{12}d_{13} \cdots c_{13}c_{14}
\]

\[
\overline{G} = c_0c_1 \cdots d_0d_1 \cdots c_1c_2 \cdots \cdots c_{12}c_{13} \cdots d_{12}d_{13}
\]

where \( G \in \mathcal{G}_{3,m} \). Note that for each \( i = 0, \ldots, 8 \), we can partition \( G \) and \( \overline{G} \) as follows, where “+6” comes from \( n_2 = 6 \).

\[
G = \begin{array}{c}
\begin{array}{l}
\vdots \\
\cdots c_0c_1 \cdots d_0d_1 \cdots c_1c_2 \cdots \cdots d_{i+5} \cdots c_{i+5}c_{i+6} \cdots \cdots d_{12}d_{13} \cdots c_{13}c_{14}
\end{array}
\end{array}
\]

\[
\overline{G} = \begin{array}{c}
\begin{array}{l}
\vdots \\
\cdots c_0c_1 \cdots d_0d_1 \cdots c_1c_2 \cdots \cdots d_{i+5} \cdots c_{i+5}c_{i+6} \cdots \cdots d_{12}d_{13}
\end{array}
\end{array}
\]

where \( G' \in \mathcal{G}_{2,m} \), \( \overline{G}' \in \mathcal{G}_{2,m} \) and \( u' \) is isomorphic to \( u'' \).

Now (for the induction hypothesis) we assume that in the 2-runs of \( A \) on \( G \) and \( \overline{G} \), if

1. pebble 3 is not inside \( G' \) and is not inside \( \overline{G}' \),
2. both \( G' \) and \( \overline{G}' \) do not contain the symbol under pebble 3, and
3. pebble 2 enters the patterns \( G' \) and \( \overline{G}' \) in the same state,\(^7\)

then pebble 2 exits \( G' \) and \( \overline{G}' \) in the same state; and prove (for the induction step) the following claim.

**Claim 1** The automaton \( A \) finishes the computation on both \( G \) and \( \overline{G} \) in the same state.

**Proof.** (Sketch) We divide the graph \( G \) into three parts: \( A \), \( B \) and \( C \), and divide the graph \( \overline{G} \) into two parts: \( D \) and \( E \), as illustrated below.

\[
G = \begin{array}{c}
\begin{array}{l}
\vdots \\
\cdots c_0c_1 \cdots d_0d_1 \cdots c_1c_2 \cdots \cdots d_{i+5} \cdots c_{i+5}c_{i+6} \cdots \cdots d_{12}d_{13} \cdots c_{13}c_{14}
\end{array}
\end{array}
\]

\[
\overline{G} = \begin{array}{c}
\begin{array}{l}
\vdots \\
\cdots c_0c_1 \cdots d_0d_1 \cdots c_1c_2 \cdots \cdots d_{i+5} \cdots c_{i+5}c_{i+6} \cdots \cdots d_{12}d_{13}
\end{array}
\end{array}
\]

The proof of the claim is divided into three stages according to the above division.

\(^6\)Here the number 4 comes from the number of pebbles (3) plus 1, and in the general proof it will be \( \beta_{i+1} \).

\(^7\)Of course, by “entering a pattern” we mean “arriving at its first symbol.”
**Stage 1**: We show that pebble 3 exits section $A$ (in the run of $A$ on $G$) in the same state as it exits section $D$ (in the run of $A$ on $G$). Recall that pebble 3 is the first pebble being placed on the input.

First, note that there exist suffixes $G_1$ of $G$ and $\overline{G}_1$ of $\overline{G}$, such that $G_1 \in G_{2,m}$ and $[A] \cap [G_1] = [D] \cap [\overline{G}_1] = \emptyset$, as depicted in Figure 5.2 below.

![Figure 5.2: When pebble 3 is above $A$ and $D$, we apply the induction hypothesis, according to which pebble 2 exits $G_1$ and $\overline{G}_1$ in the same state.](image)

Since $A$ starts the computation in the initial state $q_0$, pebble 3 reads the symbol $c_0$ in $G$ in the same state (i.e., $q_0$) as it reads the symbol $c_0$ in $\overline{G}$. Therefore, after pebble 2 is placed, it enters $G_1$ in the run of $A$ on $G$ in the same state as it enters $\overline{G}_1$ in the run of $A$ on $\overline{G}$. Furthermore, since $[A] \cap [G_1] = [D] \cap [\overline{G}_1] = \emptyset$, the symbol under pebble 3 (i.e., $c_0$) is not in $G_1$ and $\overline{G}_1$.

By the induction hypothesis, pebble 2 exits $G_1$ and $\overline{G}_1$ in the same state. Thus, when pebble 3 moves right, it reads $c_1$ in the run on $G$ in the same state as it reads $c_1$ in the run on $\overline{G}$. This scenario repeats until pebble 3 exits sections $A$ and $D$. Therefore, pebble 3 exits section $A$ in the same state as it exits section $D$.

**Stage 2**: We show that in the run of $A$ on $G$ pebble 3 enters and exits section $B$ in the same state.

At this stage we show that the behavior of pebble 3 when scanning $v_7d_6d_7w_7c_7c_8$ is periodic. Namely, there exists an integer $1 \leq p \leq \beta_3$ such that on every other $2p$ moves (to the right), pebble 3 enters the same state. We call such number $p$ an *interval* of pebble 3.
The proof of the existence of the interval \( p \) is by induction on the number of pebbles. In fact, we show that for \( i = 1, 2, 3 \), the number \( \beta_i \) is the largest possible interval of pebble \( i \) and \( \beta_{i+1} \) is a multiple of all possible intervals of pebble \( i \). Since the length of the section \( B = d_6d_7w_7 \) is \( 2m \), pebble 3 enters and exists this section in the same state. The complete proof of this stage is presented in Subsection 5.3.1.

**Stage 3**: Pebble 3 exits section \( C \) (in the run of \( A \) on \( G \)) in the same state as it exits section \( E \) (in the run of \( A \) on \( \overline{G} \)).

First, note that there exist prefixes \( G_2 \) of \( G \) and \( \overline{G}_2 \) of \( \overline{G} \) such that \( G_2 \cap \overline{G}_2 \) = \( E \cap \overline{G}_2 \) = \( \emptyset \), see Figure 5.3 below. Let \( G_3 \) be the suffix of \( G \) such that \( G = G_2G_3 \) and \( G_4 \) be the suffix of \( \overline{G} \) such that \( \overline{G} = \overline{G}_2G_4 \). Then the graphs \( G_3 \) and \( G_4 \) are isomorphic.

![Diagram](image)

**Figure 5.3**: When pebble 3 is above \( C \) and \( E \), we apply the induction hypothesis, according to which pebble 2 exits \( G_2 \) and \( \overline{G}_2 \) in the same state.

By Stages 1 and 2, pebble 3 enters section \( C \) (in the run of \( A \) on \( G \)) in the same state as it enters section \( E \) (in the run of \( A \) on \( \overline{G} \)). Thus, pebble 3 reads the symbol \( c_7 \) in section \( C \) in the same state as it reads the symbol \( d_6 \) in section \( E \). Therefore, pebble 2 enters \( G_2 \) and \( \overline{G}_2 \) in the same state. In addition, since \( [C] \cap [G_2] = [E] \cap [\overline{G}_2] = \emptyset \), the symbol under pebble 3 does not occur in either of \( G_2 \) or \( \overline{G}_2 \). Thus, by the induction hypothesis, pebble 2 exits \( G_2 \) and \( \overline{G}_2 \) in the same state.

Next, the graphs \( G_3 \) and \( G_4 \) are isomorphic and the length of the pattern between \( G_2 \) and the first symbol of \( C \) (the position of pebble 3 in the run of
\( \mathcal{A} \) on \( G \) equals to the length of the pattern between \( \overline{G}_2 \) and the first symbol of \( E \) (the position of pebble 3 in the run of \( \mathcal{A} \) on \( \overline{G} \)). Thus, the computation of pebble 2 inside \( G_3 \) (in the run of \( \mathcal{A} \) on \( G \)) is the same as its computation inside \( G_4 \) (in the run of \( \mathcal{A} \) on \( \overline{G} \)).

Therefore, pebble 3 exits \( G_3 \) in the same state as it exits \( G_4 \). That is, pebble 3 moves right and reads \( c_8 \) in \( C \) in the same state as it moves right and reads \( d_7 \) in \( E \). This scenario repeats until pebble 3 finishes scanning sections \( C \) and \( E \). Thus, pebble 3 exits section \( C \) (in the run of \( \mathcal{A} \) on \( G \)) in the same state as it exits section \( E \) (in the run of \( \mathcal{A} \) on \( \overline{G} \)).

This completes the sketch of the proof of our claim that \( \mathcal{A} \) finishes the computation on both \( G \) and \( \overline{G} \) in the same state. \( \square \)

### 5.3 Proof of Lemma 16

This section contains the proof of Lemma 16 that we outlined in Section 5.2. Namely, in Subsection 5.3.1 we present the formalization of Stage 2 of the proof, whereas the proofs of Stages 1 and 3 are presented in Subsection 5.3.2.

Recall that \( \mathcal{A} = \langle Q, q_0, F, \mu \rangle \) is a strong \( k \)-PA that accepts the language \( \mathcal{R}_{nk} \). By Theorem 3, we may assume that \( \mathcal{A} \) is deterministic and one-way. We also assume that \( \Theta_{\mathcal{A}} = \emptyset \). Recall that pebbles are placed on the input word in the order from \( k \) to 1.

By adding some extra states, we can normalize the behavior of each pebble \( i, i = k - 1, \ldots, 2, 1 \), as follows.

- After pebble \( (i + 1) \) moves right, pebble \( i \) is immediately placed (at the left-end marker \( \leftarrow \));

- pebble \( i \) is lifted only when it reaches the right-end marker \( \Rightarrow \) of the input; and

- immediately after pebble \( i \) is lifted, pebble \( (i + 1) \) moves right.

Furthermore, only pebble \( k \) can enter a final state and it may do so only after it reads the right-end marker \( \Rightarrow \) of the input.

Also, instead of writing “the automaton \( \mathcal{A} \) enters state \( q \) with pebble \( i \) as the head pebble,” we will write only “pebble \( i \) enters the state \( q \).” That is, each pebble is actually viewed as an automaton itself.
We start some terminology which we will need for the proof. Recall that $[w]$ denotes the set of symbols that occur in the word $w$.

- Let $\gamma = [i, q, \theta]$ be an $i$-configuration of $A$ on the word $w = c_1 \cdots c_n$. For $j = i, \ldots, k$, we denote by $\text{val}_{\theta,w}(j) = c_{\theta(j)}$, i.e. the symbol under pebble $j$ with respect to the assignment $\theta$ on the word $w$.

- Let $\gamma = [i, q, \theta]$ be an $i$-configuration on a word $uvw$. For $i \leq j \leq k$, we say that pebble $j$ is outside the infix $v$, if it is not located above $v$ and $\text{val}_{\theta,uvw}(j) \notin [v]$. The configuration $\gamma$ is outside the infix $v$, if all pebbles from $i$ to $k$ are outside $v$.

- Let $\rho : \mathcal{D} \to \mathcal{D}$ be a 1-1 mapping. The mapping $\rho$ It induces an automorphism on $\mathcal{D}^*$, also denoted by $\rho$, that is defined as follows. If $w = c_1 \cdots c_n \in \mathcal{D}^*$, then $\rho(w) = \rho(c_1) \cdots \rho(c_n)$.

- Let $w_1, w_2 \in \mathcal{D}^*$. We say that $w_1$ is isomorphic to $w_2$, denoted by $w_1 \cong w_2$, if there exists a 1-1 mapping $\rho : \mathcal{D} \to \mathcal{D}$ such that $w_1 = \rho(w_2)$. Note that if $w_1 \cong w_2$, then $w_1 \in L(A)$ if and only if $w_2 \in L(A)$. (Recall that we assume the automaton $A$ does not have constant symbols in its description, i.e. $\Theta_A = \emptyset$.)

**Definition 26** Let $u_1, u_2 \in \mathcal{D}^*$. We say that $u_1$ and $u_2$ are $A$-equivalent, if $A$ finishes the computation on both inputs $u_1$ and $u_2$ in the same state.

Let $uvw \in \mathcal{D}^*$ and $\gamma = [i, q, \theta]$ be an $i$-configuration of $A$ on $uvw$. We will use the following notation.

- $q(\gamma) = q$ and $\theta(\gamma) = \theta$.

- Let $\gamma$ be outside $v$. The pebble assignment $\theta$ modulo $v$ is a pebble assignment $\theta'$ on $uw$, written as

$$\theta' = \theta \mod v,$$

where for all $j = i, \ldots, k$,

$$\theta'(j) = \begin{cases} 
\theta(j) & \text{if } 1 \leq \theta(j) \leq |u| \\
\theta(j) - |v| & \text{if } |u| + |v| + 1 \leq \theta(j) \leq |u| + |v| + |w|
\end{cases}$$
Recall that for $1 \leq i \leq k$, an $i$-configuration is a configuration $[i, q, \theta]$ in which the head pebble is pebble $i$. An $i$-run is a run from an $i$-configuration to an $i$-configuration in which pebble $(i + 1)$ is never lifted. Let $j < i$. A $j$-run yielded from an $i$-configuration $\gamma$ is a $j$-run that starts with an $j$-configuration $\gamma'$ where $\gamma'$ is the first $j$-configuration such that $\gamma \vdash^* \gamma'$.

Let $w \in \mathcal{D}^*$ and $\gamma$ be an $i$-configuration of $\mathcal{A}$ on $w$. Let $v$ be a prefix of $w$. We define $R_{\gamma,j}(w, v)$ to be the state of pebble $j$ in which it exits $v$ in the $j$-run of $\mathcal{A}$ on $w$ yielded from $\gamma$.

### 5.3.1 Stage 2 of the proof of Lemma 16

Recall that $\mathcal{A} = (Q, q_0, \mu, F)$ is a strong $k$-PA, $\beta_1 = |Q|$, and for $i \geq 2$,

$$\beta_{i+1} = |Q|^2 \left( (|Q|) \right)^{|Q|}$$

Let $m = \beta_{k+1}$ (fixed until the end of this section) and let $G \in \mathcal{G}_{k,m}$. Let $i < k$ and let $G_i \in \mathcal{G}_{i,m}$ be an infix of $G$:

$$G = w_1 G_i w_2$$

(5.3)

for some $w_1, w_2 \in \mathcal{D}^*$.

Let

$$G_i = G_{i-1} \underbrace{\cdots}_{v_n(i-1)} \underbrace{d_{n(i-1)} \cdots d_{n(i-1)}}_{w_n(i-1)} \underbrace{c_{n(i-1)} \cdots c_{n(i-1)+1}}_{w_n(i-1)}$$

$$\cdots \underbrace{\cdots}_{v_n(i-1)+1} \underbrace{d_{n(i-1)} \cdots d_{n(i-1)+1}}_{w_n(i-1)+1} \underbrace{G_i'_{i-1}}_{w_n(i-1)+1}$$

where $G_{i-1}, G_i'_{i-1} \in \mathcal{G}_{i-1,m}$. We call the string

$$v_{n(i-1)+1} d_{n(i-1)} \cdots d_{n(i-1)+1} w_{n(i-1)+1}$$

the middle section of the graph $G_i$. For example, if

$$G_1 = c_0 c_1 \cdots d_0 d_1 \cdots \in \mathcal{G}_{1,m}$$

63
Proposition 27  Let \( G \) outside \( G_i \). We are going to prove that in the \( i \)-run of \( A \) yielded from \( \gamma \) pebble \( i \) enters the symbols \( d_{n(\gamma)+1}c_{n(\gamma)+2} \) in the same state.

For what follows it will be convenient to rename the symbols in the middle section and \( c_{n(\gamma)+1}c_{n(\gamma)+2} \) of \( G_i \) as

\[
\begin{array}{cccc}
\cdots & d_{n(\gamma)} & d_{n(\gamma)+1} & \cdots \\
\varepsilon_{n(\gamma)+1} & \cdots & \varepsilon_{n(\gamma)+1} & \cdots \\
\end{array}
\]

(5.4)

where

- for \( j = 1, \ldots, m-1 \), \( x_j = \varepsilon_{n(\gamma)+1,j} \) and \( y_j = \varepsilon_{n(\gamma)+2,j} \);
- \( x_m = d_{n(\gamma)+1} \) and \( y_m = d_{n(\gamma)+1} \);
- for \( j = m+1, \ldots, 2m-1 \), \( x_j = f_{n(\gamma)+1,j} \) and \( y_j = f_{n(\gamma)+2,j} \);
- \( x_{2m} = c_{n(\gamma)+1} \) and \( y_{2m} = c_{n(\gamma)+2} \);

see the notation in the beginning of Section 5.2.

Recall that the symbols

\[
\{x_1, y_1, \ldots, x_{2m-1}, y_{2m-1}\} = [\varepsilon_{n(\gamma)+1} d_{n(\gamma)+1} w_{n(\gamma)+1}]
\]

are pairwise different and neither of them occurs in either \( G_{i-1} \) or \( G'_{i-1} \).

The following proposition implies that pebble \( i \) enters the patterns \( d_{n(\gamma)}d_{n(\gamma)+1} \) and \( c_{n(\gamma)+1}c_{n(\gamma)+2} \) in the same state, as explained below.

Proposition 27  Let \( G \in G_{k,m}, G_i \in G_{i,m} \), and let \( \gamma \) be an \((i+1)\)-configuration outside \( G_i \). Let \((q_1^j, q_2^j)\), \( j = 1, \ldots, 2m \), be a pair of states, where

- \( q_1^j \) is the state in which pebble \( i \) enters \( x_j \) in the \( i \)-run of \( A \) yielded from \( \gamma \), and
- \( q_2^j \) is the state in which pebble \( i \) enters \( y_j \) in the \( i \)-run of \( A \) yielded from \( \gamma \).
Then, there exist two positive integers \( p, m_0 \leq \beta_i \) such that \( p|\beta_{i+1} \) and for all \( j, j' \) such that \( m_0 < j, j' < 2m \) the following holds. If \( j \equiv j' \pmod{p} \), then \( (q^i_1, q^i_2) = (q^{i+p}_1, q^{i+p}_2) \).

This proposition states that after every \( 2p \) moves to the right pebble \( i \) repeats its states. That is, \( (q^i_1, q^i_2) = (q^{i+p}_1, q^{i+p}_2) \). The repetition starts from an index \( m_0 < \beta_i \) and the repetition period is \( p \) that divides \( \beta_{i+1} = m \).

Since the number of symbols between \( x_m \) and \( x_{2m} \) is a factor of \( m \), it follows that pebble \( i \) enters \( x_m \) and \( x_{2m} \) in the same state.

We precede the proof of Proposition 27 with a brief note on the number of different repetition of states. Since \( \gamma \) is outside \( G_i \), the pebble assignment \( \theta \) becomes irrelevant after pebbles \( i, (i-1), \ldots, 1 \) enter \( G_i \). That is, the values \( p \) and \( m_0 \) depend only on the states \( q_i, q_{i-1}, \ldots, q_1 \) in which pebbles \( i, i-1, \ldots, 1 \) enter \( G_i \). Since the number of states of \( A \) is finite, there are only finitely many possible repetition of states.

Each repetition of states has its own values of \( p \) and \( m_0 \), and we call the pair \( (p, m_0) \) a period of states. We will use the letter \( \phi \), possibly indexed, to denote a period of states. The components of \( \phi = (p, m_0) \) are denoted by \( p(\phi) = p \) and \( m_0(\phi) = m_0 \). The value \( p(\phi) \) is called the interval of \( \phi \).

**Proof. (of Proposition 27)** The proof is by induction on \( i \).

**The base case:** \( i = 1 \). Then, let

\[
G = u_1 G_1 u_2 = u_1 c_0 c_1 x_1 y_1 \cdots x_{m-1} y_{m-1} x_m y_m x_{m+1} y_{m+1} \cdots x_{2m-1} y_{2m-1} c_1 c_2 u_2.
\]

for some \( u_1, u_2 \in D^* \) and \( G_1 \in G_{1,m} \). Let \( \gamma \) be a 2-configuration outside \( G_1 \). By definition, none of the symbols seen by pebbles \( 2, \ldots, k \) occur in \( G_1 \). Therefore, in the 1-run of \( A \), starting at \( x_1 \) pebble 1 only moves (right) by the means of transitions of the form

\[(1, q, \emptyset, \emptyset) \rightarrow (q', \text{right}).\]

Since \( A \) is deterministic and the number of states is finite, starting from some moment \( m_0 \leq |Q| \), the same sequence of transitions repeats itself. The length of such sequence \( p \) is \( \leq |Q| \), and, since \( \beta_2 = |Q|^2((|Q|)!)|Q| \), it follows that have \( p|\beta_2 \). That is, the desired period is \( (p, m_0) \).

65
The repetition of states which pebble 1 eventually enters depends on the state of pebble 1 in which it enters $G_1$. Therefore, there are at most $|Q|$ possible periods of states for pebble 1.

**The induction hypothesis:** Let $G_{i-1} \in G_{i-1,m}$ be an infix of $G$ where

$$G = u_1 G_{i-1} u_2$$

for some $u_1, u_2 \in \mathcal{D}^*$. Then, for every $i$-configuration $\gamma$ outside $G_{i-1}$, in the $(i-1)$-run of $A$ on $G$ yielded from $\gamma$ pebble $(i-1)$ enters into a period of states when scanning the middle section of $G_{i-1}$. Furthermore, there are at most $|Q|$ such periods of states.

**The induction step:** Let $G_i \in G_{i,m}$ be an infix of $G$ where

$$G = u_1 G_i u_2$$

for some $u_1, u_2 \in \mathcal{D}^*$. Let

$$G'_{i-1}, G''_{i-1} \in G_{i-1,m},$$

and let

- $x_1 y_1 \cdots x_{2m-1} y_{2m-1}$ be the middle section of $G_i$,
- $x'_1 y'_1 \cdots x'_{2m-1} y'_{2m-1}$ be the middle section of $G'_{i-1}$, and
- $x''_1 y''_1 \cdots x''_{2m-1} y''_{2m-1}$ be the middle section of $G''_{i-1}$.

By the induction hypothesis, when all pebbles $i, i + 1, \ldots, k$ are outside $G'_{i-1}$ and $G''_{i-1}$, pebble $(i-1)$ enters into a period of states when scanning the middle sections $x'_1 y'_1 \cdots x'_{2m} y'_{2m}$ and $x''_1 y''_1 \cdots x''_{2m} y''_{2m}$ of $G'_{i-1}$ and $G''_{i-1}$, respectively.

Let $\phi_1, \ldots, \phi_l$ be all periods of states of pebble $(i-1)$ in both $x'_1 y'_1 \cdots x'_{2m} y'_{2m}$ and $x''_1 y''_1 \cdots x''_{2m} y''_{2m}$.

We contend that there is a pair of positive integers $(h, h')$ such that

1. $1 \leq h < h' \leq m$,
2. $h \equiv h' \mod p(\phi_1), \ldots, p(\phi_l)$, and
3. $(q^h_1, q^h_2) = (q^{h'}_1, q^{h'}_2)$.
Indeed, let $M$ be the least common multiple of $p(\phi_1), \ldots, p(\phi_l)$. Consider the sequence of pairs of states $(q_1^{1+iM}, q_2^{1+iM}), i = 0, \ldots, |Q|^2$. Since $l \leq |Q|$ and $1 \leq p(\phi_1), \ldots, p(\phi_l) \leq \beta_{l-1}$, $M \leq \left((\beta_{l-1})!\right)^{|Q|}$, implying $|Q|^2 M \leq \beta_i$. Therefore, for some $i, j = 0, \ldots, |Q|^2$, $i < j$, $(q_1^{1+iM}, q_2^{1+iM}) = (q_1^{1+jM}, q_2^{1+jM})$, which proves our contention (with $h = 1 + iM$ and $h' = 1 + jM$).

Let $(h, h')$ be the pair satisfying conditions 1–3 above that is the smallest with respect to the lexicographic order of $\{1, \ldots, 2m\} \times \{1, \ldots, 2m\}$.

We claim that the desired period of states is $(p, m_0)$, where $p = h' - h$ and $m_0 = h$ as stated below.

**Claim** For all $j, j' \geq m'$, if $j \equiv j' \pmod{p}$, then $(q_1^j, q_2^j) = (q_1^{j'}, q_2^{j'})$.

Obviously, it suffices to prove the claim for the case of $j' = j + p$, which we will do by induction on $j$.

The base case $j = h$ and $j' = h'$ is immediate, because $(q_1^h, q_2^h) = (q_1^{h'}, q_2^{h'})$ by our choice of $h$ and $h'$, and for the induction step we assume

$$(q_1^{j-1}, q_2^{j-1}) = (q_1^{j-1+p}, q_2^{j-1+p})$$

and will prove

$$(q_1^{j}, q_2^{j}) = (q_1^{j+p}, q_2^{j+p}).$$

For the proof of $q_1^j = q_1^{j+p}$ we will need the following notation. In the $(i - 1)$-run of $A$ when pebble $i$ is above $y_{j-1}$, let $(s_1, s_2, s_3, s_4, s_5)$ be the 5-tuple of states of pebble $(i - 1)$ that is defined as follows.

- $s_1$ is the state of pebble $(i - 1)$ in which it is placed on the input string (at the left-end marker symbol $\triangleleft$).
- $s_2$ is the state of pebble $(i - 1)$ in which it arrives at the symbol $x'_{j-1}$.
- $s_3$ is the state of pebble $(i - 1)$ in which it arrives at the symbol $x''_{j-1}$.
- $s_4$ is the state of pebble $(i - 1)$ in which it exits symbol $x''_{2m-1}$, i.e., in which it arrives at $x''_{2m}$.
- $s_5$ is the state of pebble $(i - 1)$ in which it arrives at the right-end marker $\triangleright$. 

67
Similarly, in the \((i - 1)\)-run of \(A\) when pebble \(i\) is above \(y_{j+p-1}\), let \((s'_1, s'_2, s'_3, s'_4, s'_5)\) be the 5-tuple of states of pebble \(i - 1\) such that

- \(s'_1\) is the state of pebble \((i - 1)\) in which it is placed on the input string (at the left-end marker symbol \(<\)).
- \(s'_2\) is the state of pebble \((i - 1)\) in which it arrives at the symbol \(x'_{j-1}\).
- \(s'_3\) is the state of pebble \((i - 1)\) in which it arrives at the symbol \(x''_{j-1}\).
- \(s'_4\) is the state of pebble \((i - 1)\) in which it exits symbol \(x''_{2m-1}\), i.e., in which it arrives at \(x''_{2m}\).
- \(s'_5\) is the state of pebble \((i - 1)\) in which it arrives at the right-end marker \(\triangleright\).

\[
\begin{array}{cccccc}
  s_1 & s_2 & s_3 & s_4 & s_5 \\
  \downarrow & \downarrow & \downarrow & \downarrow & \downarrow \\
  < & x_{j-1} & y_{j-1} & x'_{j-1} & x''_{2m-1} & \triangleright \\
\end{array}
\]

Similarly, in the \((i - 1)\)-run of \(A\) when pebble \(i\) is above \(y_{j+p-1}\), let \((s'_1, s'_2, s'_3, s'_4, s'_5)\) be the 5-tuple of states of pebble \(i - 1\) such that

\[
\begin{array}{cccccc}
  s'_1 & s'_2 & s'_3 & s'_4 & s'_5 \\
  \downarrow & \downarrow & \downarrow & \downarrow & \downarrow \\
  x_{j+p-1} & y_{j+p-1} & x'_{j+p-1} & x''_{2m-1} & \triangleright \\
\end{array}
\]

\[
\begin{array}{cccccc}
  p & p \\
\end{array}
\]

Figure 5.4: Positions of \(x'_{j-1}, y_{j-1}, x''_{j-1}\) and \(x'_{j+p-1}, y_{j+p-1}, x''_{j+p-1}\).

Note the words \(x'_{j-1} \ldots y_{j-1} \ldots x''_{j-1}\) and \(x'_{j+p-1} \ldots y_{j+p-1} \ldots x''_{j+p-1}\) are isomorphic. Recall that \(p \equiv 0 \mod p(\phi_1), p(\phi_2)\), where \(\phi_1\) is the period of pebble \((i - 1)\) on \(x_1' y_1' \ldots x_{2m-1}' y_{2m-1}'\), and \(\phi_2\) is the period of pebble \((i - 1)\) on \(x_1'' y_1'' \ldots x_{2m-1}'' y_{2m-1}''\).

For the proof of that \(q_{j+1}^j = q_{j+p}^j\) it suffices to show that \((s_1, s_2, s_3, s_4, s_5) = (s'_1, s'_2, s'_3, s'_4, s'_5)\), which is done as follows.

- Since \(q_{j+1}^j = q_{j+p}^j\), \(s_1 = s'_1\) is immediate.
• By the induction hypothesis, pebble \((i - 1)\) enters a period of states when reading the middle section \(x_1'y_1' \cdots x_{2m-1}''y_{2m-1}'\) of \(G'_{i-1}\). Since \(s_1 = s_1'\), pebble \((i - 1)\) enters the same period of states, say \(\phi\), in both cases: when pebble \(i\) is above \(y_{j-1}\) and when it is above \(y_{j+p-1}\). Since \(p(\phi)\) divides \(p\), \(s_2 = s_2'\) follows.

• The equality \(s_3 = s_3'\), will follow, if we show that for each

\[
\ell = 1, \ldots, |x_{j-1}'\cdots y_{j-1}''\cdot x_{j-1}''\cdot x_{j-1}''| (= |x_{j+p-1}'\cdots y_{j+p-1}''\cdots x_{j+p-1}''|),
\]

pebble \((i - 1)\) arrives at position \(\ell\) in \(x_{j-1}'\cdots y_{j-1}''\cdot x_{j-1}''\cdot x_{j-1}''\) in the same state as it arrives at position \(\ell\) in \(x_{j+p-1}'\cdots y_{j+p-1}''\cdots x_{j+p-1}''\). By the induction hypothesis of Proposition 27, when pebbles \(1, \ldots, (i-2)\) run on the input word they enter periods of states in their respective middle sections in both \(G''_{i-1}\) and \(G''_{i-1}'\).

Note that each position in \(x_{j+p-1}'\cdots y_{j+p-1}''\cdot x_{j+p-1}''\cdot x_{j+p-1}''\) is a shift to the right by \(p\) of the corresponding position in \(x_{j-1}'\cdots y_{j-1}''\cdot x_{j-1}''\cdot x_{j-1}''\). Since \(p\) is a factor of all intervals of pebbles \(1, \ldots, (i-2)\), these pebbles enter the infix \(x_{j+p-1}'\cdots y_{j+p-1}''\cdot x_{j+p-1}''\) (i.e., enter \(x_{j+p-1}'\)) in the same state as they enter the infix \(x_{j-1}'\cdots y_{j-1}''\cdot x_{j-1}''\) (i.e., enter \(x_{j-1}'\)). In addition, since \(x_{j+p-1}'\cdots y_{j+p-1}''\cdot x_{j+p-1}''\cdot x_{j+p-1}''\) and \(x_{j-1}'\cdots y_{j-1}''\cdot x_{j-1}''\cdot x_{j-1}''\) are isomorphic, and \(\gamma\) is outside \(G_i\) (see the statement of Proposition 27), all pebbles \(1, \ldots, (i-2)\) “behave the same” on \(x_{j+p-1}'\cdots y_{j+p-1}''\cdot x_{j+p-1}''\cdot x_{j+p-1}''\) and \(x_{j-1}'\cdots y_{j-1}''\cdot x_{j-1}''\cdot x_{j-1}''\). Thus, they all exit both strings in the same state.

After this, each of the pebbles \(1, \ldots, (i-2)\) continues its computation and enter a period of states in their respective middle section of \(G''_{i-1}\). Again, since \(p\) is a factor of all intervals of pebbles \(1, \ldots, (i-2)\), each of the pebbles exits its respective middle sections in \(G''_{i-1}\) in the same state. Therefore, in both cases, all pebbles \(1, \ldots, (i-2)\) reach the right-end marker \(\triangleright\) in the same state. Consequently, pebble \((i - 1)\) is in the same state in the respective positions in \(x_{j-1}'\cdots y_{j-1}''\cdot x_{j-1}''\cdot x_{j-1}''\) and \(x_{j+p-1}'\cdots y_{j+p-1}''\cdot x_{j+p-1}''\cdot x_{j+p-1}''\), \(s_3 = s_3'\) follows.

• By the induction hypothesis, pebble \((i - 1)\) enters a period of states when scanning the middle section \(x_1'y_1''\cdots x_{2m-1}''y_{2m-1}'\) of \(G''_{i-1}\), and since \(s_3 = s_3'\), it enters the same period of states, say \(\phi\), in both cases.
when pebble \( i \) is above \( y_{j-1} \) and when it is above \( y_{j+p-1} \). Since \( p(\phi) \) divides \( p \), we have \( s_4 = s'_4 \).

- Finally, \( s_5 = s'_5 \) follows from \( s_4 = s'_4 \).

The proof of \( q^j_2 = q^{j+p}_2 \) is similar (and is omitted). This completes the proof of the claim and the whole proposition. \( \square \)

5.3.2 Proof of Lemma 16

In this section we formalize the sketch of the proof of Lemma 16 from Section 5.2. Recall that \( \mathcal{A} \) is a strong \( k \)-PA that accepts \( \mathcal{R}_{n_k} \). For \( m = \beta_{k+1} \) (defined in Subsection 5.3.1) we are going to show that for all \( G \in \mathcal{G}_{k,m} \), the graphs \( G \) and \( \overline{G} \) are \( \mathcal{A} \)-equivalent.

We start with the following remark.

**Remark 28** Let \( G \in \mathcal{G}_{k,m} \), where

\[
G = c_0c_1 \cdots d_0d_1 \cdots c_1c_2 \cdots \cdots c_{n-k-2}c_{n-k-1} \cdots d_{n-k-2}d_{n-k-1} \cdots c_{n-1}c_n
\]

\[
\overline{G} = c_0c_1 \cdots d_0d_1 \cdots c_1c_2 \cdots \cdots c_{n-k-2}c_{n-k-1} \cdots d_{n-k-2}d_{n-k-1}
\]

Then, for any \( i = 1, \ldots, k \), for any infix \( G_i \in \mathcal{G}_{i,m} \) of \( G \), there exists an infix \( \overline{G}_i \) of \( \overline{G} \) such that

\[
G = u \underbrace{G_i}_{u'} \overline{u'}
\]

\[
\overline{G} = u \underbrace{\overline{G}_i}_{\overline{u'}} \overline{u''}
\]

and \( u' \cong u'' \). Also, if \( i < 1 \), then \([u] \cap [u'] = [u] \cap [u''] = \emptyset\).

We observe next that for any \( G_i \in \mathcal{G}_{i,m} \), the following.

a) There exists a prefix \( G_{i-1} \) and \( w_1, w_2 \in \mathcal{D}^* \) such that of \( G_i \)

\[
G_i = G_{i-1}w_1
\]

\[
\overline{G}_i = \overline{G}_{i-1}w_2
\]

\( w_1 \cong w_2 \), and

\([G_{i-1}] \cap [u'] = [\overline{G}_{i-1}] \cap [u''] = \emptyset\).
It is illustrated as follows. For some $0 \leq j \leq n_k - n_i$,

$$G_i = \begin{array}{c}
\underbrace{c_j c_{j+1} \cdots d_j d_{j+1} \cdots d_{j+n_i-1-2d_j+n_{i-1}-1} \cdots c_{j+n_i-1-1} c_{j+n_i-1}}_w \\
\cdots d_{j+n_i-1} d_{j+n_i-1} d_{j+n_i-1-2d_j+n_{i-1}-1} \cdots c_{j+n_i-1-1} c_{j+n_i-1}
\end{array}$$

$$\overline{G}_i = \begin{array}{c}
\underbrace{c_j c_{j+1} \cdots d_j d_{j+1} \cdots d_{j+n_i-1-2d_j+n_{i-1}-1}}_w \\
\cdots c_{j+n_i-1-1} c_{j+n_i-1-1} d_{j+n_i-1-1} d_{j+n_i-1} \cdots d_{j+n_i-1-2d_j+n_{i-1}}
\end{array}$$

b) There exists a suffix $G_{i-1}'$ and $w \in \mathcal{D}^*$ such that of $G_i$

$$G_i = w G_{i-1}'$$
$$\overline{G}_i = w \overline{G}_{i-1}$$

and

$$[G_{i-1}'] \cap [u] = [\overline{G}_{i-1}'] \cap [u] = \emptyset.$$ 

It is illustrated as follows. For some $0 \leq j \leq n_k - n_i$,

$$G_i = \begin{array}{c}
\underbrace{c_j c_{j+1} \cdots d_j d_{j+1} \cdots d_{j+n_i-1-1} d_{j+n_i-1}}_w \\
\cdots c_{j+n_i-1-1} c_{j+n_i-1-1} c_{j+n_i-1-1} c_{j+n_i-1} d_{j+n_i-1-1} \cdots c_{j+n_i-1-1} c_{j+n_i-1}
\end{array}$$

$$\overline{G}_i = \begin{array}{c}
\underbrace{c_j c_{j+1} \cdots d_j d_{j+1} \cdots d_{j+n_i-1-1} d_{j+n_i-1}}_w \\
\cdots c_{j+n_i-1-1} c_{j+n_i-1-1} c_{j+n_i-1-1} d_{j+n_i-1} \cdots d_{j+n_i-1-2d_j+n_{i-1}}
\end{array}$$

The reason why we cut $G_i$ and $\overline{G}_i$ into two, as shown in items (a) and (b), is that we can show, by induction on $i$, that $i$ pebbles cannot distinguish between $G_i$ and $\overline{G}_i$.

Lemma 16 follows from the Lemma 29 below. This lemma essentially states that, for each $i = 1, \ldots, k$, if $G_i \in \mathcal{G}_{i,m}$, then pebble $i$ of $\mathcal{A}$ cannot distinguish between $G_i$ and $\overline{G}_i$.

Recall that for $w \in \mathcal{D}^*$, a prefix $v$ of $w$, and an $(i+1)$-configuration $\gamma$, $R_{\gamma,i}(w,v)$ denotes the state in which pebble $i$ exits $v$ in the $i$-run of $\mathcal{A}$ on $w$ yielded from $\gamma$. 

71
Lemma 29 Let \( 1 \leq i \leq k - 1 \). Let \( G_i \) and \( \overline{G}_i \) be infixes of \( G \) and \( \overline{G} \), respectively, as described in Remark 28. Let \( \gamma_1 \) be an \((i + 1)\)-configuration of \( A \) on \( G \) and \( \gamma_2 \) be an \((i + 1)\)-configuration of \( A \) on \( \overline{G} \) such that

1. \( \gamma_1 \) and \( \gamma_2 \) are outside \( G_i \) and \( \overline{G}_i \), respectively,
2. \( q(\gamma_1) = q(\gamma_2) \),
3. \( \theta(\gamma_1) \mod G_i = \theta(\gamma_2) \mod \overline{G}_i \), and
4. if \( i = 1 \), then for each \( j = 2, \ldots, k \),

- if pebble \( j \) in the configuration \( \gamma_1 \) (respectively, \( \gamma_2 \)) is above \( u \), then it is outside \( u' \) (respectively, \( u'' \)),
- if pebble \( j \) in the configuration \( \gamma_1 \) (respectively, \( \gamma_2 \)) is above \( u' \) (respectively, \( u'' \)), then it is outside \( u \).

Then, \( R_{\gamma_1,i}(G, G) = R_{\gamma_2,i}(\overline{G}, \overline{G}) \).

The reason for which we need the cumbersome prerequisite (4) in the lemma is that when \( i = 1 \), the non-intersection between \([u]\) and \([u'']\) fails. Prerequisite (4) can be seen as a compensation for the non-intersection property \([u] \cap [u''] = \emptyset\). Without this condition, the lemma will not hold for \( i = 1 \).

Proof. (of Lemma 29) The proof is by induction on \( i \).

The base case: \( i = 1 \). That is,

\[
\begin{align*}
G &= uG_1u' = u v_0 c_0 d_0 v_0 c_1 d_1 v_0 c_2 d_2 v_0 c_3 d_3 v_0 \\
\overline{G} &= u\overline{G}_1u'' = u v_0 c_0 d_0 v_0 c_1 d_1 v_0 c_2 d_2 v_0 c_3 d_3 v_0 
\end{align*}
\]

for some \( u, u'u'' \in \mathcal{D}^* \) and \( G_1 = c_0 v_0 d_0 v_0 d_1 v_0 c_1 d_2 v_0 c_2 d_3 v_0 c_3 d_3 v_0 \in \mathcal{G}_{1,m} \). Let \( \gamma_1, \gamma_2 \) be 2-configurations on \( G \) and \( \overline{G} \), respectively, that they satisfy prerequisites 1 – 4 of the lemma. We will prove that \( R_{\gamma_1,1}(G, G) = R_{\gamma_2,1}(\overline{G}, \overline{G}) \).

From prerequisites 2 – 4 of the lemma it follows that

\[
R_{\gamma_1,1}(G, u) = R_{\gamma_2,1}(\overline{G}, u). 
\tag{5.6}
\]

Next, since, by prerequisite 1, both \( \gamma_1 \) and \( \gamma_2 \) are outside \( G_1 \) and \( \overline{G}_1 \), respectively, it follows from (5.6) that

\[
R_{\gamma_1,1}(G, uc_0 d_0 \cdots d_0 d_1) = R_{\gamma_2,1}(\overline{G}, uc_0 d_0 \cdots d_0 d_1). 
\tag{5.7}
\]
Then, by Proposition 27,
\[ R_{\gamma_1,1}(G, uc_0c_1 \cdots d_0d_1) = R_{\gamma_1,1}(G, uc_0c_1 \cdots d_0d_1 \cdots c_0c_1), \]  
(5.8)
and combining (5.7), (5.8), and the definition of \( G_1 \) we obtain
\[ R_{\gamma_1,1}(G, uG_1) = R_{\gamma_2,1}(\overline{G}, u\overline{G}_1). \]  
(5.9)
Finally, by prerequisites 1, 3, 4, and (5.9),
\[ R_{\gamma_1,1}(G, G) = R_{\gamma_2,1}(\overline{G}, \overline{G}). \]

**The induction step:** Assume that the lemma holds for all \( j \leq i \) and prove it for \( i + 1 \).

Let
\[ G = uG_{i+1}u' \]
\[ \overline{G} = u\overline{G}_{i+1}u'' \]
u, \( u' \), \( u'' \) ∈ \( D^* \), where \( G_{i+1} \in \mathcal{G}_{i+1,m} \). Let \( \gamma_1, \gamma_2 \) be \((i + 2)\)-configurations on \( G \) and \( \overline{G} \), respectively, satisfying prerequisites (1) – (3) of the lemma. We will prove that \( R_{\gamma_1,i+1}(G, G) = R_{\gamma_2,i+1}(\overline{G}, \overline{G}) \).

By observation b) of Remark 28, there is a suffix \( G_i \) of \( G_{i+1} \) and \( w \) ∈ \( D^* \) such that
\[ G_{i+1} = wG_i \]
\[ \overline{G}_{i+1} = w\overline{G}_i \]
and
\[ [G_i] \cap [u] = [\overline{G}_i] \cap [u] = \emptyset. \]

Therefore, when pebble \((i + 1)\) is above \( u \) in both runs of \( A \) on \( G \) and \( \overline{G} \), by the induction hypothesis,
\[ R_{\gamma_1,i+1}(G, u) = R_{\gamma_2,i+1}(\overline{G}, u). \]  
(5.10)
In what follows we will show that
\[ R_{\gamma_1,i+1}(G, uG_{i+1}) = R_{\gamma_2,i+1}(\overline{G}, u\overline{G}_{i+1}). \]
We divide the graph $G_{i+1}$ into three parts: $A$, $B$, and $C$, and divide the graph $\overline{G}_{i+1}$ into two parts: $D$ and $E$, as illustrated below.

\[ G_{i+1} = \begin{array}{c}
\begin{array}{c}
\varepsilon_G 1 \ldots \cdot c_n 1 c_{n+1} \ldots \cdot d_n d_{n+1} \ldots \cdot c_{n+i+1} d_{n+i+1} \ldots \cdot c_{n+i+1} - 1 c_{n+i+1} \\
A \end{array}
\end{array}
\]

\[ \overline{G}_{i+1} = \begin{array}{c}
\begin{array}{c}
\varepsilon_{\overline{G}} 1 \ldots \cdot c_n 1 c_{n+1} \ldots \cdot d_n d_{n+1} \ldots \cdot d_{n+i+1} - 2 d_{n+i+1 - 1} \\
D \end{array}
\end{array}
\]

Note that $A = D$ and $C \cong E$.

Next, we break the $(i+1)$-runs of $A$ on $G$ and $\overline{G}$ yielded from $\gamma_1$ and $\gamma_2$, respectively, into the following three stages, according to the above division.

**Stage 1**: Pebble $(i+1)$ is above the section $A$ in the $(i+1)$-run of $A$ on $G$ and pebble $(i+1)$ is above the section $D$ in the $(i+1)$-run of $A$ on $\overline{G}$.

Let $G_i$ be a suffix of $G_{i+1}$ such that

- $G_i \in G_{i,m}$ and
- $[A] \cap [G_i] = [D] \cap [\overline{G}_i] = \emptyset$,

see Figure 5.5 below. (Such a suffix exists by the definition of $G_i$.)

Figure 5.5: When pebble $(i+1)$ is above $A$ and $D$, we apply the induction hypothesis, according to which pebble $i$ exits $G_i$ and $\overline{G}_i$ in the same state.

By non-intersection condition $[A] \cap [G_i] = \emptyset$, This implies that when pebble $i$ is above $A$, it is outside $G_i$. Similarly, since $[D] \cap [\overline{G}_i] = \emptyset$, when pebble $i$ is above $D$, it is outside $\overline{G}_i$. That is, at this stage of computation,
all \(i\)-configurations in the \((i + 1)\)-run of \(A\) on \(G\) and \(\overline{G}\) yielded from \(\gamma_1\) and \(\gamma_2\), respectively, are outside \(G_i\) and \(\overline{G}_i\), respectively.

By (5.10), pebble \((i + 1)\) reads the symbol \(c_0\) in \(G_{i+1}\) in the same state as it reads the symbol \(c_0\) in \(\overline{G}_{i+1}\), and, after pebble \(i\) is placed, we apply the induction hypothesis according to which pebble \(i\) exits \(G\) in the run of \(A\) on \(G\) in the same state as it exits \(\overline{G}\) in the run of \(A\) on \(\overline{G}\). That is, pebble \((i + 1)\) moves right and reads the symbol \(c_1\) in the same state in both runs on \(G\) and on \(\overline{G}\). This scenario repeats until pebble \((i + 1)\) finishes scanning sections \(A\) and \(D\) in the corresponding runs of \(A\) on \(G\) and \(\overline{G}\). Therefore,

\[
R_{\gamma_1,i+1}(G, uA) = R_{\gamma_2,i+1}(\overline{G}, uD).
\] (5.11)

Note that when we apply the induction hypothesis for \(i = 1\), we do not violate prerequisite 4 of the lemma, because \([D] \cap [u'] = \emptyset\).

**Stage 2**: Pebble \((i + 1)\) is above section \(B\) in the \((i + 1)\)-run of \(A\) on \(G\).

By Proposition 27, in the \((i + 1)\)-run of \(A\) on \(G\), pebble \((i + 1)\) enters the patterns \(d_n d_{n+1}\) and \(c_{n+1} c_{n+2}\) in the same state. That is,

\[
R_{\gamma_1,i+1}(G, uAB) = R_{\gamma_1,i+1}(G, uA).
\] (5.12)

**Stage 3**: Pebble \((i + 1)\) is above section \(C\) in the \((i + 1)\)-run of \(A\) on \(G\) and pebble \((i + 1)\) is above section \(E\) in the \((i + 1)\)-run of \(A\) on \(\overline{G}\).

By (5.11) and (5.12),

\[
R_{\gamma_1,i+1}(G, uAB) = R_{\gamma_2,i+1}(\overline{G}, uD).
\] (5.13)

Recall that the sections \(A\), \(B\), \(D\) are as follows.

- \(A = c_0 c_1 v_1 d_0 d_1 \cdots c_{n_i} c_{n_i+1} v_{n_i+1}\) is a prefix of \(G_{i+1}\).
- \(B = d_n d_{n+1} w_{n+1}\) is an infix of \(G_{i+1}\).
- \(D = c_0 c_1 v_1 d_0 d_1 \cdots c_{n_i} c_{n_i+1} v_{n_i+1}\) is a prefix of \(G_{i+1}\).

We will prove that

\[
R_{\gamma_1,i+1}(G, uG_{i+1}) = R_{\gamma_2,i+1}(\overline{G}, u\overline{G}_{i+1}).
\]

The proof is very similar to that of Stage 1.

Let \(G_i\) be a prefix of \(G_{i+1}\) such that
• \( G_i \in \mathcal{G}_{i,m} \) and

• \([C] \cap [G_i] = [E] \cap [\overline{G}_i] = \emptyset\),

see Figure 5.6 below.

\[
G = \begin{pmatrix}
\cdots & d_{n_i-2} & d_{n_i-1} & \cdots & c_{n_i-1} & c_{n_i} & \cdots & d_{n_i-1} & d_{n_i} & \cdots \\
\cdots & d_{n_i+1} & d_{n_i+2} & \cdots & c_{n_i+1} & c_{n_i+2} & \cdots & d_{n_i+1} & d_{n_i+2} & \cdots \\
\end{pmatrix}
\]

\[
C = \begin{pmatrix}
\cdots & c_{n_i-1} & c_{n_i} & \cdots & d_{n_i} & d_{n_i+1} & \cdots & d_{n_i+2} & d_{n_i+1} & \cdots \\
\cdots & c_{n_i+1} & c_{n_i+2} & \cdots & d_{n_i+1} & d_{n_i+2} & \cdots & d_{n_i+1} & d_{n_i+2} & \cdots \\
\end{pmatrix}
\]

By induction hypothesis, pebble \( i \) exits \( G_i \) and \( \overline{G}_i \) in the same state.

Pebble \((i + 1)\) is above \( C \) and \( E \).

\[
\mathcal{G} = \begin{pmatrix}
\cdots & d_{n_i-2} & d_{n_i-1} & \cdots & c_{n_i-1} & c_{n_i} & \cdots & d_{n_i-1} & d_{n_i} & \cdots \\
\cdots & d_{n_i+1} & d_{n_i+2} & \cdots & c_{n_i+1} & c_{n_i+2} & \cdots & d_{n_i+1} & d_{n_i+2} & \cdots \\
\end{pmatrix}
\]

\[
\pi_i = \begin{pmatrix}
\cdots & d_{n_i-2} & d_{n_i-1} & \cdots & c_{n_i-1} & c_{n_i} & \cdots & d_{n_i-1} & d_{n_i} & \cdots \\
\cdots & d_{n_i+1} & d_{n_i+2} & \cdots & c_{n_i+1} & c_{n_i+2} & \cdots & d_{n_i+1} & d_{n_i+2} & \cdots \\
\end{pmatrix}
\]

\[
V_1 = \begin{pmatrix}
\cdots & c_{n_i} & \cdots & d_{n_i} & \cdots & c_{n_i} & \cdots & d_{n_i} & \cdots \\
\cdots & c_{n_i+1} & c_{n_i+2} & \cdots & d_{n_i+1} & d_{n_i+2} & \cdots & d_{n_i+1} & d_{n_i+2} & \cdots \\
\end{pmatrix}
\]

\[
V_2 = \begin{pmatrix}
\cdots & c_{n_i} & \cdots & d_{n_i} & \cdots & c_{n_i} & \cdots & d_{n_i} & \cdots \\
\cdots & c_{n_i+1} & c_{n_i+2} & \cdots & d_{n_i+1} & d_{n_i+2} & \cdots & d_{n_i+1} & d_{n_i+2} & \cdots \\
\end{pmatrix}
\]

Figure 5.6: When pebble \((i + 1)\) is above \( C \) and \( E \), we apply the induction hypothesis according to which pebble \( i \) exits \( G_i \) and \( \overline{G}_i \) in the same state.

Again, it follows from the non-intersection condition \([C] \cap [G_i] = \emptyset\) that when pebble \((i + 1)\) is above \( C \), it is outside \( G_i \). Similarly, since \([E] \cap [\overline{G}_i] = \emptyset\), when pebble \((i + 1)\) is above \( E \), it is outside \( \overline{G}_i \). That is, at this stage of computation, all \( i \)-configurations in the \((i + 1)\)-runs of \( \mathcal{A} \) on \( G \) and \( \overline{G} \) yielded from \( \gamma_1 \) and \( \gamma_2 \), respectively, are outside \( G_i \) and \( \overline{G}_i \), respectively. In addition, \( V_1' u'_i \cong V_2 u''_i \), where the patterns \( V_1 \) of \( G \) and \( V_2 \) of \( \overline{G} \) are as indicated in Figure 5.6.

It follows from (5.13) that pebble \((i + 1)\) reads the symbol \( c_{n_i+1} \) in section \( C \) of \( G_{i+1} \) in the same state as it reads the symbol \( d_{n_i} \) in section \( E \) of \( \overline{G}_{i+1} \). After pebble \( i \) is placed, we apply the induction hypothesis according to which pebble \( i \) exits \( G \) in the run of \( \mathcal{A} \) on \( G \) in the same state as it exits \( \overline{G} \) in the run of \( \mathcal{A} \) on \( \overline{G} \). That is, pebble \((i + 1)\) moves right and reads the symbol \( c_{n_i+2} \) in section \( C \) of \( G_{i+1} \) in the same state as it reads the symbol \( d_{n_i} \) after moving right in section \( E \) of \( \overline{G}_{i+1} \). This scenario repeats until pebble \((i + 1)\) finishes scanning sections \( C \) and \( E \) in both the runs of \( \mathcal{A} \) on \( G \) and \( \overline{G} \), respectively. Therefore,

\[
R_{\gamma_1,i+1}(G, uABC) = R_{\gamma_2,i+1}(\overline{G}, uDE),
\]

76
or, equivalently,

\[ R_{\gamma_1,i+1}(G, uG_{i+1}) = R_{\gamma_2,i+1}(G, uG_{i+1}) \]  

(5.14)

Finally, we will prove that

\[ R_{\gamma_1,i+1}(G, G) = R_{\gamma_2,i+1}(G, G) \]  

(5.15)

By observation a) of Remark 28, there exists a prefix \( G_i \) of \( G_{i+1} \) and \( w_1, w_2 \in \mathcal{D}^* \), such that

\[ G_{i+1} = G_i w_1 \]
\[ \overline{G}_{i+1} = \overline{G}_i w_2 \]

\( w_1 u' \cong w_2 u'' \), and

\[ [G_i] \cap [u'] = [\overline{G}_i] \cap [u''] = \emptyset. \]

Therefore, when pebble \((i+1)\) is above \( u' \) in the run of \( A \) on \( G \) and pebble \((i+1)\) is above \( u'' \) in the run of \( A \) on \( \overline{G} \) we apply the induction hypothesis for \( G_i \) and \( \overline{G}_i \) to obtain (5.15). This completes the proof of Lemma 16.

\[ \square \]

### 5.4 Proof of Lemma 21

The proof of Lemma 21 is divided into two parts. In Subsection 5.4.1 we show that \( R_k^+ \in \text{wPA}_k \) and in Subsection 5.4.2 we show that \( R_{k+1}^+ \notin \text{wPA}_k \).

Recall that the language \( R_k^+ \), \( k = 1, 2, \ldots \), consists of words of the form

\[
c_{0}c_1 \cdots c_{2}c_3 \cdots \cdots c_{m-3}c_{m-2} \cdots \cdots c_{m-2}c_{m-1} \cdots \cdots c_{m-1}c_m
\]

such that

- \( c_0, \ldots, c_m \in \mathcal{D} \);

- \( w_1, \ldots, w_{m-1} \in \mathcal{D}^* \) and each of \( w_1, \ldots, w_{m-1} \) is of even length; and

- for each \( i = 1, \ldots, m-1 \), \( c_i \notin [w_i] \).
5.4.1 The proof of $\mathcal{R}_k^+ \in \text{wPA}_k$

The weak $k$-PA $\mathcal{A}$ that accepts $\mathcal{R}_k^+$ works as follows. It accepts the word $w = a_0b_0 \cdots a_nb_n$, if there are positions

$$0 = 2i_k < 2i_{k-1} < \cdots < 2i_1 = 2n$$

(5.16)

of pebbles $k, \ldots, 1$, respectively, (i.e., the symbol under pebble $j$ is $b_{i_j}$, $j = k, \ldots, 1$) such that $b_{i_j} = a_{i_{j-1}}$, $j = k, \ldots, 1$. By (5.16) such positions (if exists) can be guessed (non-deterministically) in a “weak fashion.”

5.4.2 The proof of $\mathcal{R}_{k+1}^+ \not\in \text{wPA}_k$

The proof is very similar to the proof of Lemma 16 and, for the sake of completeness, we sketch it below.

Assume to the contrary that $\mathcal{R}_{k+1}^+ \in \text{wPA}_k$ and $\mathcal{A} = \langle Q, q_0, \mu, \{F\} \rangle$ be a weak $k$-PA that accepts $\mathcal{R}_{k+1}^+$.

Consider the graphs

$$G = c_0c_1 \cdots d_0d_1 \cdots c_1c_2 \cdots d_1d_2 \cdots c_2c_3 \cdots \cdots d_{i-2}d_{i-1} \cdots c_{i-1}c_i,$$

and

$$\overline{G} = c_0c_1 \cdots d_0d_1 \cdots c_1c_2 \cdots d_1d_2 \cdots c_2c_3 \cdots \cdots d_{i-2}d_{i-1}$$

where

- for each $j = 1, \ldots, 2(i-1)$, $u_i = e_{i,1}e_{i+1,1} \cdots e_{i,m-1}e_{i+1,m-1}$; and
- the symbols

$$\{c_0, \ldots, c_i, d_0, \ldots, d_{i-1}\} \cup \{e_{j,j'} : 1 \leq j \leq i + 1, 1 \leq j \leq m - 1\}$$

are pairwise different.

We denote by $G_{k,m}^+$ the set of all such graphs $G$ and denote by $\overline{G}_{k,m}^+$ the set of all such graphs $\overline{G}$. Obviously, $G \in \mathcal{R}_{k+1}^+$, whereas $\overline{G} \not\in \mathcal{R}_{k+1}^+$.

We shall prove that for $m = \beta_{k+1}$, $\mathcal{A}$ ends the computation on both $G$ and $\overline{G}$ in the same state. The proof is similar to that in Section 5.3 and is based on the following claim.
Claim. Let \( G \in \mathcal{G}^+_{k,m} \) and \( G' \in \mathcal{G}'^+_{k,m} \) and let \( G = wG_i \) and \( G' = wG'_i \), where \( w \in \mathcal{D}^* \), and \( G_i \in \mathcal{G}^+_{i,m} \) and \( G'_i \in \mathcal{G}'^+_{i,m} \):

\[
G_i = c_0 c_1 \cdots u_1 d_0 d_1 \cdots c_1 c_2 \cdots \cdots d_{i-2} d_{i-1} \cdots c_{i-1} c_i
\]

and

\[
G'_i = c_0 c_1 \cdots u_1 d_0 d_1 \cdots c_1 c_2 \cdots \cdots d_{i-2} d_{i-1}
\]

Let \( \gamma_1 \) be an \((i+1)\)-configuration of \( \mathcal{A} \) on \( G \) and \( \gamma_2 \) be an \((i+1)\)-configuration of \( \mathcal{A} \) on \( G' \) such that

- \( \gamma_1 \) and \( \gamma_2 \) are outside \( G_1 \) and \( G'_1 \), respectively;
- \( q(\gamma_1) = q(\gamma_2) \); and
- \( \theta(\gamma_1) \mod G_1 = \theta(\gamma_2) \mod G'_1 \).

Then, \( R_{\gamma_1,i}(G, G) = R_{\gamma_2,i}(G', G) \).

Proof. The proof is by induction on \( i \).

The base case: \( i = 1 \). Since \( m \) is large enough, in the 1-run of \( \mathcal{A} \) on \( G \) yielded from \( \gamma_1 \), pebble 1 enters a repetition of states when scanning reads \( c_0 c_1 u_1 d_0 d_1 u_2 c_1 c_2 \). By our choice of \( m \), the length of the interval of states divides \( m \). Therefore, pebble 1 enters \( d_0 d_1 \) in the same state as it enters \( c_1 c_2 \), i.e.,

\[
R_{\gamma_1,1}(G, wc_0 c_1 u_1 d_0 d_1) = R_{\gamma_1,1}(G, G).
\]

Since \( q(\gamma_1) = q(\gamma_2) \), we have

\[
R_{\gamma_1,1}(G, wc_0 c_1 u_1 d_0 d_1) = R_{\gamma_2,1}(G', G),
\]

and, combining these two equalities, we obtain

\[
R_{\gamma_1,1}(G, G) = R_{\gamma_2,1}(G', G).
\]

The induction step: Assume that the claim holds for \( i - 1 \) and prove it for \( i \).
By the induction hypothesis,
\[ R_{\gamma_1,i}(G, wc_0c_1u_1) = R_{\gamma_2,i} (\overline{G}, wc_0c_1u_1) \quad (5.17) \]

Then, similar to the proof in Subsection 5.3.1, it can be shown that
\[ R_{\gamma_1,i}(G, wc_0c_1u_1) = R_{\gamma_1,i}(G, wc_0c_1u_1d_0d_1u_2) \quad (5.18) \]

Combining (5.17) and (5.18), we obtain
\[ R_{\gamma_1,i}(G, wc_0c_1u_1d_0d_1u_2) = R_{\gamma_2,i}(G, wc_0c_1u_1) \]

Finally, since the suffix
\[ c_1c_2 \cdots d_1d_2 \cdots c_3 \cdots \cdots d_{i-2}d_{i-1} \cdots c_{i-1}c_i \]

of \( G \) is isomorphic to the suffix
\[ d_0d_1 \cdots c_1c_2 \cdots \cdots c_{i-2}c_{i-1} \cdots d_{i-2}d_{i-1}, \]

of graph \( \overline{G} \),
\[ R_{\gamma_1,i}(G, G) = R_{\gamma_2,i}(\overline{G}, G). \]

\[ \square \]
Chapter 6

Undecidability of PA languages

In this chapter we will show that the emptiness problem for PA languages is undecidable. It consists of two sections. In Section 6.1 we show the undecidability of the emptiness problem for weak 3-PA languages, whereas in Section 6.2 we show the undecidability of the same problem for strong 2-PA languages.

6.1 Undecidability of weak 3-PA languages

Theorem 30 The emptiness problem for weak 3-PA is undecidable.

The proof is very similar to the proof of the undecidability of the emptiness problem for weak 5-PA in [21]. We observe that the same proof can be easily adopted to weak 3-PA. The details are provided below. It uses a reduction from the Post Correspondence Problem (PCP), which is well known to be undecidable [12]. An instance of PCP is a sequence of pairs \((x_1, y_1), \ldots, (x_n, y_n)\), where each \(x_1, y_1, \ldots, x_n, y_n \in \{\alpha, \beta\}^*\).

This instance has a solution if there exist indexes \(i_1, \ldots, i_m \in \{1, \ldots, n\}\) such that \(x_{i_1} \cdots x_{i_m} = y_{i_1} \cdots y_{i_m}\). The PCP asks whether a given instance of the problem has a solution.

In the following we show how to encode a solution of an instance of PCP into a data word which possesses properties that can be checked by a weak 3-PA. Let \(\Sigma = \{1, \ldots, n, \alpha, \beta, \$\}\). We denote by \(x_i = \nu_{i,1} \cdots \nu_{i,l_i}\), for each
Each string \(x_i\) is encoded as \(\text{Enc}(x_i) = (\nu_{a_i,1}^{i_1}) \cdots (\nu_{a_i,l_i}^{i_{l_i}})\) where \(a_{i,1}, \ldots, a_{i,l_i}\) are pairwise different.

The string \(x_{i_1}, x_{i_2}, \ldots, x_{i_m}\) can be encoded as

\[
\text{Enc}(x_{i_1}, x_{i_2}, \ldots, x_{i_m}) = \left(\frac{i_1}{b_1}\right)\text{Enc}(x_{i_1}) \left(\frac{i_2}{b_2}\right)\text{Enc}(x_{i_2}) \cdots \left(\frac{i_m}{b_m}\right)\text{Enc}(x_{i_m})
\]

where all the data values that appear in it are pairwise different. Note that even if \(i_j = i_{j'}\) for some \(j, j'\), the data values that appear in \(\text{Enc}(x_{i_j})\) do not appear in \(\text{Enc}(x_{i_{j'}})\) and vice versa. The idea is each data value is used to mark a place in the string.

Similarly, the string \(y_{j_1}, y_{j_2}, \ldots, y_{j_l}\) can be encoded as

\[
\text{Enc}(y_{j_1}, y_{j_2}, \ldots, y_{j_l}) = \left(\frac{j_1}{c_1}\right)\text{Enc}(y_{j_1}) \left(\frac{j_2}{c_2}\right)\text{Enc}(y_{j_2}) \cdots \left(\frac{j_l}{c_l}\right)\text{Enc}(y_{j_l})
\]

where the data values that appear in it are pairwise different.

Now the data word

\[
\left(\frac{i_1}{b_1}\right)\text{Enc}(x_{i_1}) \cdots \left(\frac{i_m}{b_m}\right)\text{Enc}(x_{i_m}) \left(\frac{\delta}{d}\right) \left(\frac{j_1}{c_1}\right)\text{Enc}(y_{j_1}) \cdots \left(\frac{j_l}{c_l}\right)\text{Enc}(y_{j_l})
\]

constitutes a solution to the instance of PCP if and only if

\[
i_1 i_2 \cdots i_m = j_1 j_2 \cdots j_l \quad \text{(6.1)}
\]

\[
\text{Proj}_\Sigma(\text{Enc}(x_{i_1}) \cdots \text{Enc}(x_{i_m})) = \text{Proj}_\Sigma(\text{Enc}(y_{j_1}) \cdots \text{Enc}(y_{j_l})) \quad \text{(6.2)}
\]

Now, in order to be able to check such property with weak 3-PA, we demand the following additional criterion.

1. \(b_1 \cdots b_m = c_1 \cdots c_l\);

2. \(\text{Proj}_D(\text{Enc}(x_{i_1}) \cdots \text{Enc}(x_{i_m})) = \text{Proj}_D(\text{Enc}(y_{j_1}) \cdots \text{Enc}(y_{j_l}))\)

3. For any two positions \(h_1\) and \(h_2\) where \(h_1\) is to the left of the delimiter \(\delta\) and \(h_2\) is to the right of the delimiter \(\delta\), if both of them have the same data value, then both of them are labelled with the same label.

All the criteria (1)–(3) imply Equations 6.1 and 6.2.

Because the data values that appear in \(\text{Proj}_D(\text{Enc}(x_{i_1}), \ldots, \text{Enc}(x_{i_m}))\) are pairwise different, all of them are checkable by three pebbles in the “weak” manner. For example, to check criterion (1), the automaton does the following.
• Check that \( b_1 = c_1 \).

• Check that for each \( i = 1, \ldots, m - 1 \), there exists \( j \) such that \( a_i a_{i+1} = b_j b_{j+1} \).

It can be done by placing pebble 3 to read \( a_i \) and pebble 2 to read \( a_{i+1} \), then using pebble 3 to search on the other side of \$ for the index \( j \).

• Finally, check that \( b_m = c_l \).

criterion (2) can be checked similarly and criterion (3) is straightforward. The reduction is now complete and we prove that the emptiness problem for weak 3-PA is undecidable.

### 6.2 Undecidability of strong 2-PA languages

**Theorem 31** The emptiness problem for 2-PA languages is undecidable.

For the proof of Theorem 31 we reduce Hilbert’s tenth problem (existence of solutions of Diophantine equations) to the emptiness problem for 2-PA languages. Namely, we show that the set of solutions of a Diophantine equation is accepted by a 2-PA. Since the former is undecidable ([17], see also [7] or [18]), the latter is undecidable as well.

We precede the proof of Theorem 31 with a number of examples exhibiting an unexpectedly strong recognition power of 2-PA. These examples are rather simple, but despite their simplicity, they are the backbones of the proof of Theorem 31.

For the rest of the section, in all of the examples, the set \( \Sigma \) consists of only two labels: \( \sigma \) and \$. The label \$ is used only as a delimiter. We will not be concerned with the data value of the label \$. Thus, instead of writing \((\sigma a)\) for some data value \( a \in \mathcal{D} \), we will write only \$. For all other positions, which are labelled with \( \sigma \), we will write only the data value \( a \), instead of \((\sigma a)\).

The first example deals with the language \( L_{\text{diff}} \) consisting of all data words in which every data value occurs at most one time:

\[
L_{\text{diff}} = \{ a_1 \cdots a_n : n \geq 1, a_i \in \mathcal{D}, \text{ for each } i = 1, \ldots, n, \text{ and } a_i \neq a_j, \text{ whenever } i \neq j \}.
\]
The words of $L_{\text{diff}}$ will be used for representation of positive integers in “unary” notation: a word $w \in L_{\text{diff}}$ represents the integer $|w|$.$^1$ Of course in such way a positive integer has infinitely many equivalent representations, but as we will see in the sequel, the integer equality can be tested by 2-PA.

**Example 32** The language $L_{\text{diff}}$ is accepted by a 2-PA that works as follows. Pebble 2 advances through the input from left to right. At each step pebble 2 check that the label is not $\&$ and pebble 1 scans the input to verify that the input symbol under pebble 2 differs from all the others, see also [21, the example in Section 2.4 and Theorem 4.1].

Example 33 below employs the following notation. For two $\Sigma$-data words $u, v \in L_{\text{diff}}$ we write $u \sim v$, if the data values in $u$ is a permutation of those in $v$.

**Example 33** Let

$$L_{\text{perm}} = \{u\& v : u, v \in L_{\text{diff}} \text{ and } u \sim v\}.$$  

This language is accepted by 2-PA that works as follows. Pebble 2 advances through the input from left to right. In each step pebble 2 scans the input and finds the symbol under pebble 1 on the other “half” of the input.$^2$ Verifying that both $u$ and $v$ are in $L_{\text{diff}}$ can be done in two swaps, see Example 32.$^3$

Our next example shows that positive integers represented by elements of $L_{\text{diff}}$ can be tested for equality by 2-PA.

**Example 34** Let $L_{\text{eq}}$ consist of all words of the form

$$u\& a_1 b_1 \cdots a_n b_n\& v,$$

where

- $u, v \in L_{\text{diff}},$

---

$^1$Cf. [6, Section 7], where a similar representation was used for the proof of undecidability of the emptiness problem languages accepted by a kind of a *register* automata called po-2-DFA.$^1$

$^2$Naturally, the first half of the input consists of the symbols occurring before “$\&$” and the second half of the input consists of the symbols occurring after it.

$^3$Recall that we are dealing with two-way automata.
• \(a_1 \cdots a_n \sim u\), and
• \(b_1 \cdots b_n \sim v\).

This language is accepted by 2-PA that, like in Example 32, verifies that both \(u\) and \(v\) are in \(L_{\text{diff}}\), and then, like in Example 33, verifies that \(a_1 \cdots a_n \sim u\) and \(b_1 \cdots b_n \sim v\).

The following example shows that 2-PA can accept non-semi-linear languages.

**Example 35** Let \(L_{sq}\) consist of all words of the form
\[
 u\$a_1v_1\$a_2v_2\$ \cdots \$a_nv_n,
\]
where
• \(u \in L_{\text{diff}}\), and
• \(u \sim a_1 \cdots a_n \sim v_1 \sim \cdots \sim v_n\).

By Example 33, \(L_{sq}\) is accepted by 2-PA, because the membership test involves only verifying permutations of words. Obviously,
\[
\{|w| : w \in L_{sq}\} = \{n^2 + 3n - 1 : n = 1, 2, \ldots\}
\]
is not semi-linear.

The reduction of Hilbert’s tenth problem is based on Examples 36 and 37 below which illustrate the core idea lying behind the proof. Namely, these examples show that 2-PA can test atomic integer equalities.

**Example 36** Let \(L_{\text{add}}\) be the language consisting of all words of the form
\[
 u\$v\$a_1c_1 \cdots a_mc_m\$b_1c_{m+1} \cdots b_nc_{m+n}\$w,
\]
where
A1. \(u, v, w \in L_{\text{diff}}\),
A2. \(a_1 \cdots a_m \sim u\),
A3. \(b_1 \cdots b_n \sim v\), and
Example 37 Let $L_{\text{mul}}$ be the language consisting of all words of the form

$$u \circ v \circ a_{1,1} b_{1,1} \cdots a_{1,n} b_{1,n} \circ \cdots \circ a_{m,1} b_{m,1} \cdots a_{m,n} b_{m,n} \circ w,$$

where

M1. $u$, $v$ and $w$ are in $L_{\text{diff}},$

M2. $a_1 \cdots a_m \sim u$,

M3. $b_{i,1} \cdots b_{i,n} \sim v$, for each $i = 1, \ldots, m$, and

M4. $c_{1,1} \cdots c_{1,n} \cdots c_{m,1} \cdots c_{m,n} \sim w$.

Obviously,

$$|u| \times |v| = |w|.$$

Like in Example 32 it can be shown that two pebbles are sufficient to verify condition M1; and like in Example 33 it can be shown that two pebbles are sufficient to verify conditions M2–M4. Thus, $L_{\text{mul}}$ is accepted by 2-PA.

Example 38 below is a straightforward extension of Example 36.

Example 38 Let $m$ be a positive integer and let $L_{\text{add},m}$ be the language consisting of all words of the form

$$v_1 \circ \cdots \circ v_m \circ a_{1,1} b_{1,1} \cdots a_{1,|v_1|} b_{1,|v_1|} \circ \cdots \circ a_{m,1} b_{m,1} + \cdots + a_{m,|v_m|} b_{m,|v_m|} \circ v,$$

where

A1. $v_1, \ldots, v_m, v \in L_{\text{diff}},$

A2. $a_{i,1} \cdots a_{i,|v_i|} \sim v_i$, $i = 1, \ldots, m$, and
Obviously, 

$$|v_1| + \cdots + |v_m| = |v|.$$ 

The language $L_{\text{add},m}$ is accepted by a 2-PA similar to that described in Example 33.

At last, we have arrived at the proof of Theorem 31. The intuition lying behind the proof is as follows. Examples 34, 36, and 37 show how, by verifying only permutations of words, 2-PA can simulate the equality test and can verify the results of the arithmetic operations: addition and multiplication, respectively. For the proof of Theorem 31 we (quite naturally) extend these examples to testing results of polynomial evaluation, or, more precisely, to languages corresponding to polynomial evaluation. Loosely speaking, the language $L_f$ corresponding to a polynomial $f(x_1, \ldots, x_m)$ with positive integer coefficients consist of all words of the form

$$v_1 \cdots v_m \cdots \text{ “an evaluation of } f(|v_1|, \ldots, |v_m|) \text{” } \cdots w,$$

where $v_1, \ldots, v_m, w \in L_{\text{diff}}$ and $|w| = f(|v_1|, \ldots, |v_m|)$. Since only permutations of words are needed to simulate the evaluation of $f(|v_1|, \ldots, |v_m|)$, $L_f$ is accepted by 2-PA, see the proof of Theorem 31 below.

**Proof. (of Theorem 31)** We start with recalling Hilbert’s tenth problem that can equivalently be restated as follows. *Given two polynomials $f'(x_1, \ldots, x_m)$ and $f''(x_1, \ldots, x_m)$ with positive integer coefficients, do there exist positive integers $n_1, \ldots, n_m$ such that*

$$f'(n_1, \ldots, n_m) = f''(n_1, \ldots, n_m)? \quad (6.3)$$

It was shown in [17] (see also [7] or [18]) that Hilbert’s tenth problem is undecidable.

First, using Example 37, we show that 2-PA can recognize the values of monomials over positive integers. For this, with each sequence of monomials $M_1(x_1, \ldots, x_m), \ldots, M_k(x_1, \ldots, x_m)$ over positive integers we associate the language $L_{M_1,\ldots,M_k}$ defined by the following recursion.

- If $M_1(x_1, \ldots, x_m)$ is a constant $n$, then $L_{M_1}$ consists of all words of the form

$$v_1 \cdots v_m \cdot v,$$
where \( v_1, \ldots, v_m, v \in L_{\text{diff}} \) and \(|v| = n\), i.e., for all \( v_1, \ldots, v_m \in L_{\text{diff}}, \)
\[
M_1([v_1], \ldots, [v_m]) = |v| (= n).
\]

- If \( M_1(x_1, \ldots, x_m) \) is of the form \( x_j M(x_1, \ldots, x_m) \), then \( L_{M_1} \) consists of all words of the form

\[
v_1v_2 \cdots v mv_1^b c_1 b_1, v_1 v_2 \cdots v_m v_2^b c_1 b_1 \cdots c_m b_{m_1} \cdots c_{m_2} b_{m_2} \cdots v_n,
\]

where

\[
- \quad v_1 v_2 \cdots v_m v' \in L_M, \text{ i.e., } |v'| = M([v_1], \ldots, [v_m]);
- \quad a_1 \cdots a_m \sim v_j;
- \quad b_i, b_i n \sim v', \text{ for each } i = 1, \ldots, m; \text{ and }
- \quad c_1, c_1 n \cdots c_{m_2} \cdots c_{m_2} \sim v.
\]

That is,

\[
|v| = |v_j||v'| = |v_j|M([v_1], \ldots, [v_m]) = M_1([v_1], \ldots, [v_m]).
\]

Assume that the language \( L_{M_1, \ldots, M_k} \) has been defined, and let \( M_{k+1} \) be a constant \( n \). Then \( L_{M_1, \ldots, M_{k+1}} \) consists of all words of the form \( w sv \), where \( w \in L_{M_1, \ldots, M_k} \), \( v \in L_{\text{diff}} \), and \(|v| = n\).

If \( M_{k+1}(x_1, \ldots, x_m) \) is of the form \( x_j M(x_1, \ldots, x_m) \), then \( L_{M_1, \ldots, M_{k+1}} \) is defined similarly to the second clause of the definition of \( L_{M_1} \).

Now, let

\[
f'(x_1, \ldots, x_m) = M_1'(x_1, \ldots, x_m) + \cdots + M_{k'}'(x_1, \ldots, x_m)
\]

and let

\[
f''(x_1, \ldots, x_m) = M_1''(x_1, \ldots, x_m) + \cdots + M_{k''}''(x_1, \ldots, x_m),
\]

where \( M_i'(x_1, \ldots, x_m) \), \( i' = 1, \ldots, k' \), and \( M_i''(x_1, \ldots, x_m) \), \( i'' = 1, \ldots, k'' \), are monomials. Then, like in Example 38, we can “extend” \( L_{M_1', \ldots, M_k', M_1'', \ldots, M_k''} \) to the language \( L_{f', f''} \) that for each \( m \)-tuple \( v_1, \ldots, v_m \in L_{\text{diff}} \) contains a word of the form

\[
v_1 v_2 \cdots v_m v' \cdots v'' w \cdots w', w \in L_{\text{diff}}.
\]
where $|w'| = f'(|v_1|, \ldots, |v_m|)$ and $|w''| = f''(|v_1|, \ldots, |v_m|)$.

Finally, let $L_{f'=f''}$ be the languages consisting of all words of the form

$$v_1$\cdots$v_m$\cdots$w'$$w''$$a'_1$$a''_1$$\cdots$$a'_n$$a''_n,$$

where

- $v_1$\cdots$v_m$\cdots$w'$$w'' \in L_{f',f''},$
- $a'_1$$\cdots$$a'_n \sim w'$, and
- $a''_1$$\cdots$$a''_n \sim w''$.

Then, like in Example 34, one can show that $L_{f'=f''}$ is accepted by 2-PA.

Since, obviously, (6.3) has a solution if and only if $L_{f'=f''}$ is non-empty, our reduction (and, therefore, the proof of Theorem 31) is complete. □

---

4Note that delimited patterns of $L_{M'_1,\ldots,M'_k,M''_1,\ldots,M''_k}$ can be detected by using just the finite memory (states) of an appropriate 2-PA.
Chapter 7

Top view weak PA

In this chapter we will introduce a weaker model of weak PA, which we call top view weak PA. Roughly speaking, top view weak PA are weak PA where the equality test is performed only between the data values seen by the two most recently placed pebbles. Obviously, for the case of two pebbles there is no difference between top-view weak PA and weak PA. Like the weak and strong PA case, top view weak PA are quite robust: alternating, nondeterministic and deterministic top view weak PA have the same recognition power.

It is also shown that top view weak PA can be simulated by one-way alternating one-register FMA. Therefore, their emptiness problem is decidable. For practical purposes, the most interesting feature of top view weak PA is, perhaps, their containment of all $LTL^1_1(\Sigma,\mathit{X},\mathit{U})$ languages. In fact, the number of pebbles of top view weak PA needed to simulate an $LTL^1_1(\Sigma,\mathit{X},\mathit{U})$ sentence linearly depends on its freeze quantifier rank, introduced in Section 3.2.

This chapter is organized as follows. We start by showing in Section 7.1 the decidability of the emptiness problem for weak 2-PA languages. Then, in Section 7.2 we study the complexity issues of weak 2-PA languages. In Section 7.3 we introduce top view weak PA and study their connection with $LTL^1_1(\Sigma,\mathit{X},\mathit{U})$ languages.
7.1 Decidability of weak 2-PA

Now we are going to show that the emptiness problem for weak 2-PA is decidable. The proof is by simulating weak 2-PA by one-way alternating 1-FMA. In fact, the simulation can be easily generalized to arbitrary number of pebbles. That is, weak \(k\)-PA can be simulated by one-way alternating \((k - 1)\)-FMA. This result settles a question left open in [21]: Can weak PA be simulated by alternating FMA? We refer the reader to Appendix 7.5 for the details of the proof.

**Theorem 39** For every weak 2-PA \(A\), there exists a one-way alternating 1-FMA \(A'\) such that \(L(A) = L(A')\). Moreover, the construction of \(A'\) from \(A\) is effective.

Now, by Theorem 39, we immediately obtain the decidability of weak 2-PA because the emptiness problem for one-way alternating 1-FMA is decidable [8, Theorem 4.4].

**Corollary 40** The emptiness problem for weak 2-PA is decidable.

We devote the rest of this section to the proof of Theorem 39.

Let \(A = (Q, q_0, \mu, F)\) be a weak 2-PA. We assume that \(A\) is deterministic. Furthermore, we normalize the behavior of \(A\) as follows.

- Pebble 1 is lifted only after it reads the right-end marker symbol \(\triangleright\).
- Only pebble 2 can enter a final state and it does so after it reads the right-end marker \(\triangleright\).
- Immediately after pebble 2 moves right, pebble 1 is placed.
- Immediately after pebble 1 is lifted, pebble 2 moves right.

On input word \(w = (\sigma_1^d_1) \cdots (\sigma_n^d_n)\), the run of \(A\) on \(\langle w \rangle\) can be depicted as a tree shown in Figure 7.1.

The meaning of the tree is as follows.

- \(q_0, q_1, \ldots, q_n, q_{n+1}\) are the states of \(A\) when pebble 2 is the head pebble reading the positions 0, 1, \ldots, \(n, n+1\), respectively, that is, the symbols \(\langle \sigma_i^d_1 \rangle, \ldots, (\sigma_i^d_n), \triangleright\), respectively.
• $q_f$ is the state of $A$ after pebble 2 reads the symbol $\triangleright$.

• For $1 \leq i \leq j \leq n$, $p_{i,j}$ is the state of $A$ when pebble 1 is the head pebble above the position $j$ while pebble 2 is above the position $i$.

• For $1 \leq i \leq n$, the state $p_i$ is the state of $A$ immediately after pebble 1 is lifted and pebble 2 is above the position $i$.

It must be noted that there is a transition $(2, \sigma_i, 0, p_i) \rightarrow (q_{i+1}, \text{right})$ applied by $A$ that is not depicted in the figure.

Now the simulation of $A$ by a one-way alternating 1-FMA $A'$ becomes straightforward by transforming the tree in Figure 7.1 into a tree depicting the computation of $A'$ on the same word $w$. 

Figure 7.1: The tree representation of a run of $A$ on $w = (\sigma_1) \cdots (\sigma_n)$. 

92
Figure 7.2: The corresponding run of $\mathcal{A}'$ to the one in Figure 7.1.

Roughly, the automaton $\mathcal{A}'$ is defined as follows.

- The states of $\mathcal{A}'$ are elements of $Q \cup (Q \times Q)$;
- the initial state is $q_0$; and
- the set of final states is $F \cup \{(p, p) : p \in Q\}$.

For each placement of pebble 1 on position $i$, the automaton performs the following “Guess–Split–Verify” procedure which consists of the following steps.

---

1. Actually $\mathcal{A}'$ needs some other auxiliary states. However, for the intuitive explanation here the set $Q \cup (Q \times Q)$ suffices. We refer the reader to Section 7.5 for the details.

93
1. From the state $q_i$, $A'$ “guesses” the state in which pebble 1 is eventually lifted, i.e. the state $p_i$, and stores it in its internal state. That is, $A'$ enters into the state $(q_i, p_i)$.

2. $A'$ “splits” its computation (conjunctively) into two branches.

   - In one branch, assuming that the guess $p_i$ is correct, $A'$ moves right and enters into the state $q_{i+1}$, simulating the transition $(2, \emptyset, p_i) \rightarrow (q_{i+1}, \text{right})$. After this, it recursively performs the Guess–Split–Verify procedure for the next placement of pebble 1 on position $(i + 1)$.

   - In the other branch $A'$ stores the data value $d_i$ in its register and simulates the run of pebble 1 on $(\sigma_1 d_1) \cdots (\sigma_n d_n)$ to “verify” that the guess $p_i$ is correct. That is, $A'$ accepts only if it ends in the state $(p_i, p_i)$.

Figure 7.1 shows the corresponding run of $A'$ on the same word.

### 7.2 Complexity of weak 2-PA

In this subsection we are going to determine the time complexity of three specific problems related to weak 2-PA.

**Emptiness problem.** The emptiness problem for weak 2-PA. That is, given a weak 2-PA $A$, is $L(A) = \emptyset$?

**Labelling problem.** Given a weak 2-PA $A$ over the labels $\Sigma$ and a sequence of data values $d_1 \cdots d_n \in D^n$, is there a sequence of labels $\sigma_1 \cdots \sigma_n \in \Sigma^n$ such that $(\sigma_1 d_1) \cdots (\sigma_n d_n) \in L(A)$?

**Data value membership problem.** Given a weak 2-PA $A$ over the labels $\Sigma$ and a sequence of finite labels $\sigma_1 \cdots \sigma_n \in \Sigma^n$, is there a sequence of data values $d_1 \cdots d_n \in D^n$ such that $(\sigma_1 d_1) \cdots (\sigma_n d_n) \in L(A)$?

The emptiness problem, as we have seen in the previous section, is decidable. The labeling and data value membership problem are definitely decidable. To solve the labeling problem, one simply iterates all possible sequence $\sigma_1 \cdots \sigma_n \in \Sigma^n$ and runs $A$ to check whether $(\sigma_1 d_1) \cdots (\sigma_n d_n) \in L(A)$. Such straightforward algorithm requires $O(|\Sigma|^n \cdot n^2)$ computational steps.
Similarly, to solve the data value membership problem, one can iterate all possible sequence of data values \(d_1 \cdots d_n\) and run \(A\) to check whether \((\sigma_1) \cdots (\sigma_n) \in L(A)\). Since the word is of length \(n\), one simply needs to consider up to \(n\) different data values. Such algorithm takes \(O(n^n \cdot n^2)\) computational steps.

We are going to show that the emptiness problem is not primitive recursive, while both the labeling and data value membership problems are NP-complete.

We start the proof with a few simple examples of languages accepted by weak 2-PA. Though simple, they are very crucial in determining the complexity of the emptiness problem for weak 2-PA.

**Example 41** Let \(\Sigma = \{\alpha, \beta\}\). We define the \(\Sigma\)-data language \(L_{inc}\) which consists of the data words of the following form:

\[
\frac{\alpha}{a_1} \cdots \frac{\alpha}{a_m} \frac{\beta}{b_1} \cdots \frac{\beta}{b_n},
\]

where

- the data values \(a_1, \ldots, a_m\) are pairwise different;
- the data values \(b_1, \ldots, b_n\) are pairwise different;
- \(\text{Proj}_\Sigma(w_1) = \alpha^m\);
- \(\text{Proj}_\Sigma(w_2) = \beta^n\);
- \(\text{Cont}_D(w_1) \subseteq \text{Cont}_D(w_2)\).

All these conditions can be checked by weak 2-PA. The intention of data words in \(L_{inc}\) is to represent the inequality \(m \leq n\).

**Example 42** Let \(\Sigma = \{\alpha, \beta\}\). For a fixed \(l \geq 0\), we define the language \(L_{inc,+1}\) which consists of the data words of the following form:

\[
\frac{\alpha}{a_1} \cdots \frac{\alpha}{a_m} \frac{\beta}{b_1} \cdots \frac{\beta}{b_n},
\]

where
• the data values $a_1, \ldots, a_m$ are pairwise different;

• the data values $b_1, \ldots, b_n$ are pairwise different;

• $\text{Proj}_\Sigma(w_1) = \alpha^m$;

• $\text{Proj}_\Sigma(w_2) = \beta^n$;

• For each $a_i \in \text{Cont}_\Sigma(w_1)$, $a_i \neq b_1$.

• $\{a_1, \ldots, a_m\} \subseteq \{b_2, \ldots, b_n\}$.

Again, all these conditions can be checked by weak 2-PA. The intention of data words in $L_{\text{inc},+1}$ is to represent the inequality $m + 1 \leq n$.

**Example 43** Let $\Sigma = \{\alpha, \beta\}$. For a fixed $l \geq 0$, we define the language $L_{\text{inc},-1}$ which consists of the data words of the following form:

$$
\begin{array}{c}
\begin{array}{cc}
\alpha & \ldots & \alpha \\
\alpha & \ldots & \alpha \\
\hline
w_1 & w_2 & \\
\end{array}
\end{array}
$$

where

• the data values $a_1, \ldots, a_m$ are pairwise different;

• the data values $b_1, \ldots, b_n$ are pairwise different;

• $\text{Proj}_\Sigma(w_1) = \alpha^m$;

• $\text{Proj}_\Sigma(w_2) = \beta^n$;

• The symbol $a_1 \notin \{b_1, \ldots, b_n\}$;

• For each $i = 2, \ldots, m$, $a_i \in \{b_1, \ldots, b_n\}$.

Again, all these conditions can be checked by weak 2-PA. The intention of data words in $L_{\text{inc},-1}$ is to represent the inequality $m - 1 \leq n$.

**Theorem 44** The emptiness problem for weak 2-PA is not primitive recursive.
Proof. The proof is by simulation of incrementing counter automata. It follows closely the proof of similar lower bound for one-way alternating 1-FMA [8, Theorem 2.9]. It is known that the emptiness problem for incrementing counter automata is decidable [19, Theorem 6], but not primitive recursive [23]. We refer the reader to Section 7.4 for the formal definition of incrementing counter automata.

In short, an incrementing $l$-counter automaton over $\Sigma$ is an automaton with $l$ counters, operates on words over $\Sigma$, and the value in each counter is allowed to erroneously increase, hence, the name incrementing. A configuration is a tuple $(q, \sigma, v)$ where $q$ is a state, $\sigma$ is the current symbol read and $v : \{1, \ldots, l\} \rightarrow \mathbb{N}$, where $v(i)$ denotes the value stored in counter $i$.

Now a configuration $(q, v)$ can be encoded as a $(Q \cup \Sigma \cup \{c_1, \ldots, c_l\})$-data word as follows.

\[
(q) (\sigma) (c_1) \cdots (c_1, a_{1,v(1)}) \cdots (c_l) \cdots (c_l, a_{l,v(l)}),
\]

where the symbols $a_{1,1}, \ldots, a_{l,v(l)}$ are pairwise different. The labels $c_1, \ldots, c_l$ are used as pointers that the current data value is part of the encoding of the counters $v(1), \ldots, v(l)$, respectively.

Since the automaton allows for erroneous increment of values in each counter, we can check the validity of the application of each transition, like in Examples 41, 42 and 43.

Before we proceed to study the labeling and data value membership problems, we present the following example. It will be useful in our proof of the NP-completeness of both problems.

**Example 45** Consider a $\Sigma$-data language $L_\sim$ defined as follows. A $\Sigma$-data word $w = (\sigma_1) \cdots (\sigma_n) \in L_\sim$ if and only if for all $i, j = 1, \ldots, n$, if $a_i = a_j$, then $\sigma_i = \sigma_j$. That is, $w \in L_\sim$ if and only if whenever two positions in $w$ carry the same data value, their labels are the same.

The language $L_\sim$ is accepted by weak 2-PA which works in the following manner. Pebbles 2 iterates through all possible positions in $w$. At each iteration, pebble 1 is placed and scans through all the positions to the right of pebble 2, checking whether there is a position with the same data value of pebble 2. If there is such position, then the labels seen by pebbles 1 and 2 are the same.
Now we are going to show the NP-completeness of the labeling problem. It is by a reduction from graph 3-colorability problem.

Given an undirected graph $G = (V, E)$, let $V = \{1, \ldots, n\}$ and $E = \{(i_1, j_1), \ldots, (i_m, j_m)\}$. We can take $i_1 j_1 \cdots i_m j_m$ as the sequence of data values. Then, we construct a weak 2-PA $\mathcal{A}$ over the alphabet $\Sigma = \{\vartheta_R, \vartheta_G, \vartheta_B\}$ that accepts data words of even length in which the following hold.

- For all odd position $x$, the label on position $x$ is different from the label on position $x + 1$.
- For every two positions $x$ and $y$, if they have the same data value, then they have the same label. (See Example 45)

Thus, the graph $G$ is 3-colorable if and only if there exists $\sigma_1 \cdots \sigma_{2m} \in \{\vartheta_R, \vartheta_G, \vartheta_B\}^*$ such that

$$
\left(\frac{\sigma_1}{i_1}\right)\left(\frac{\sigma_2}{j_1}\right)\cdots\left(\frac{\sigma_{2m-1}}{i_m}\right)\left(\frac{\sigma_{2m}}{j_m}\right) \in L(\mathcal{A}),
$$

and the NP-completeness of the labeling problem follows.

The NP-completeness of data value membership problem can be established in a similar spirit. The reduction is from the following variant of graph 3-colorability, called 3-colorability with constraint. Given a graph $G = (V, E)$ and three integers $n_r$, $n_g$, $n_b$ in unary form, can the graph $G$ be colored with the colors $R$, $G$ and $B$ such that the numbers of vertices colored with $R$, $G$ and $B$ are $n_r$, $n_g$ and $n_b$, respectively?

The polynomial time reduction to data value membership problem is as follows. Let $V = \{1, \ldots, n\}$ and $E = \{(i_1, j_1), \ldots, (i_m, j_m)\}$.

We define $\Sigma = \{\vartheta_R, \vartheta_G, \vartheta_B, \nu_1, \ldots, \nu_n\}$. We take

$$
\nu_1 \nu_1 \cdots \nu_{n_r} \vartheta_R \vartheta_R \cdots \nu_1 \nu_1 \cdots \nu_{n_g} \vartheta_G \vartheta_G \cdots \nu_1 \nu_1 \cdots \nu_{n_b} \vartheta_B \vartheta_B \cdots
$$

as the sequence of finite labels.

Then, we construct a weak 2-PA over $\Sigma$ that accepts data words of the form

$$
\left(\frac{\nu_1}{c_1}\right)\left(\frac{\nu_{j_1}}{d_1}\right)\cdots\left(\frac{\nu_{i_m}}{c_m}\right)\left(\frac{\vartheta_R}{a_1}\right)\cdots\left(\frac{\vartheta_R}{a_{n_r}}\right)\left(\frac{\vartheta_G}{a_1'}\right)\cdots\left(\frac{\vartheta_G}{a_{n_g}'}\right)\left(\frac{\vartheta_B}{a_1''}\right)\cdots\left(\frac{\vartheta_B}{a_{n_b}''}\right)
$$

where
• \( \nu_{i_1}, \nu_{j_1}, \ldots, \nu_{i_m}, \nu_{j_m} \in \{ \nu_1, \ldots, \nu_n \} \);

• in the sub-word \({\nu_{i_1} \choose c_1}{\nu_{j_1} \choose d_1}\cdots{\nu_{i_m} \choose c_m}{\nu_{j_m} \choose d_m}\), every two positions with the same labels have the same data value, see Example 45;

• the data values \( a_1, \ldots, a_n, a'_1, \ldots, a'_{ng}, a''_1, \ldots, a''_{nb} \) are pairwise different;

• For each \( i = 1, \ldots, m \), the data values \( c_i, d_i \) appear among \( a_1, \ldots, a_n, a'_1, \ldots, a'_{ng}, a''_1, \ldots, a''_{nb} \) such that the following holds:

  – if \( c_i \) appears among \( a_1, \ldots, a_n \), then \( d_i \) appears among \( a'_1, \ldots, a'_{ng} \) or \( a''_1, \ldots, a''_{nb} \);
  
  – if \( c_i \) appears among \( a'_1, \ldots, a'_{ng} \), then \( d_i \) appears either among \( a_1, \ldots, a_n \) or \( a''_1, \ldots, a''_{nb} \); and
  
  – if \( c_i \) appears among \( a''_1, \ldots, a''_{nb} \), then \( d_i \) appears among \( a_1, \ldots, a_n \) or \( a'_1, \ldots, a'_{ng} \).

Note that we can store the integers \( r, g, b, m \) in the internal states \( A \), thus, enable \( A \) to “count” up to \( n_r, n_g, n_b \) and \( m \). We have each state for the numbers \( 1, \ldots, n_r, 1, \ldots, n_g, 1, \ldots, n_b \) and \( 1, \ldots, m \). Furthermore, the unary form of \( n_r, n_g \) and \( n_b \) is crucial here to ensure that the number of the states of \( A \) is still polynomial in the length of the input.

Now the graph \( G \) is 3-colorable with constraint if and only if there exits \( c_1 d_1 \cdots c_m d_m a_1 \cdots a_n a'_1 \cdots a'_{ng} a''_1 \cdots a''_{nb} \) such that

\[
\left( \nu_{i_1} \right) \left( \nu_{j_1} \right) \cdots \left( \nu_{i_m} \right) \left( \nu_{j_m} \right) \left( \partial_R \right) \left( \partial_R \right) \cdots \left( \partial_G \right) \left( \partial_G \right) \left( \partial_B \right) \left( \partial_B \right)
\]

is accepted by \( A \), and the NP-completeness of data value membership problem follows.

**7.3 Top view weak \( k \)-PA**

In this section we are going to restrict the definition of weak \( k \)-PA so that its emptiness problem becomes decidable. Roughly speaking, top view weak PA are weak PA where the equality test is performed only between the data values seen by the last and the second last placed pebbles. That is, if
pebble \( i \) is the head pebble, then it can only compare the data value it reads with the data value read by pebble \((i + 1)\). It is not allowed to compare its data value with those read by pebble \( i + 2, \ldots, k \).

Formally, the transitions of top view weak \( k \)-PA \( A = (Q, q_0, \mu, F) \) are of the form

\[
(i, \sigma, V, q) \rightarrow (q', \text{act})
\]

where \( V \) is either \( \emptyset \) or \( \{i + 1\} \).

The definition of top view weak \( k \)-PA is defined by setting

\[
V = \begin{cases} 
\emptyset, & \text{if } a_{\theta(i+1)} \neq a_{\theta(i)} \\
\{i + 1\}, & \text{if } a_{\theta(i+1)} = a_{\theta(i)}
\end{cases}
\]

in the definition of transition relation in Section 2.2. Note that top view weak \( 2 \)-PA are just the same as weak \( 2 \)-PA. We can also define the alternating version of top view weak \( k \)-PA. However, just like in the case of weak \( k \)-PA, alternating, nondeterministic and deterministic top view weak \( k \)-PA have the same recognition power.

**Theorem 46** For every top view weak \( k \)-PA \( A \), there is a one-way alternating 1-FMA \( A' \) such that \( L(A') = L(A) \). Moreover, the construction of \( A' \) is effective.

**Proof.** The proof is a straightforward generalization of the proof of Theorem 39. Each placement of a pebble is simulated by “Guess–Split–Verify” procedure. Since each pebble \( i \) can only compare its data value with the one seen by pebble \( i + 1 \), \( A' \) does not need to store the data values seen by pebble \( i + 2, \ldots, k \). It only need to store the data value seen by pebble \( i + 1 \), thus, one register suffices.

Following Theorem 46, we immediately obtain the decidability of the emptiness problem for top view weak \( k \)-PA.

**Corollary 47** The emptiness problem for top view weak \( k \)-PA is decidable.

**Remark 48** Since the emptiness problem for ordinary \( 2 \)-PA and for weak \( 3 \)-PA is already undecidable (See Theorem 30 and [15, Theorem 4]), it seems that top view weak PA is a tight boundary of a subclass of PA languages for which the emptiness problem is decidable.
Theorem 49  For every sentence $\psi \in \text{LTL}^1(\Sigma, X, U)$, there exists a top view weak $k$-PA $A_\psi$, where $k = \text{fqr}(\psi) + 1$, such that $L(A_\psi) = L(\psi)$.

Proof. Let $\psi$ be an LTL$^1(\Sigma, X, U)$ sentence. We construct an alternating top view weak $k$-PA $A_\psi$, where $k = \text{fqr}(\psi) + 1$ such that given a data word $w$, the automaton $A_\psi$ checks whether $w, 1 \models \psi$. $A_\psi$ accepts if it is so. Otherwise, it rejects.

Intuitively, the computation of $w, 1 \models \psi$ is done recursively as follows. The automaton $A_\psi$ “consists of” the automata $A_\varphi$ for all sub-formula $\varphi$ of $\psi$, including $A_\epsilon$ to represent the empty formula $\epsilon$.

- The automaton $A_\epsilon$ accepts every data words.
- If $\psi = \sigma \varphi$, then check whether the current label is $\sigma$. If it is not, then $A$ rejects immediately. Otherwise, $A_\psi$ proceeds to run $A_\varphi$.
- If $\psi = \varphi \lor \varphi'$, then $A_\psi$ nondeterministically chooses one of $A_\varphi$ or $A_{\varphi'}$ and proceeds to run one of them.
- If $\psi = \varphi \land \varphi'$, then $A_\psi$ splits its computation (by conjunctive branching) into two and proceed to run both of $A_\varphi$ and $A_{\varphi'}$.
- If $\psi = X \varphi$, then $A_\psi$ moves to the right one step. If it reads the right-end marker, then the automaton rejects immediately. Otherwise, it proceeds to run $A_\varphi$.
- If $\psi = \uparrow \varphi$, then $A_\psi$ checks whether the data value seen by its head pebble is the same as the one seen by the second last placed pebble. If it is not the same, then it rejects immediately. Otherwise, it proceeds to run $A_\varphi$.
- If $\psi = \downarrow \varphi$, then $A_\psi$ places a new pebble and proceeds to run $A_\varphi$.
- If $\psi = \varphi U \varphi'$, then $A_\psi$ it runs $A_{\varphi' \lor (\varphi \land X(\varphi U \varphi'))}$.
- If $\psi = \neg \varphi$, then $A_\psi$ runs $A_\varphi$. If $A_\varphi$ accepts, then $A_\psi$ rejects. Otherwise, $A_\psi$ accepts.

Note that since $\text{fqr}(\varphi) = k$, on each computation path then the automaton $A_\psi$ only needs to place the pebble $k$ times, thus, $A_\psi$ requires only $k + 1$. It is a straight forward induction to show that $L(A_\psi) = L(\psi)$. □
Our next results deals with the expressive power of LTL\(^1\)(\(\Sigma, x, u\)) based on the freeze quantifier rank. It is an analog of the classical hierarchy of first order logic based on the ordinary quantifier rank. We start by defining an LTL\(^1\)(\(\Sigma, x, u\)) sentence for the language \(R^+_m\) defined in Section 5.1.

First, we recall the definition of \(R^+_m\). Let \(\Sigma = \{\sigma\}\) be a singleton alphabet. For an integer \(m \geq 1\), the language \(R^+_m\) consists of \(\Sigma\)-data words of the form

\[
\left(\sigma_{a_0}\right) \left(\sigma_{a_1}\right) \cdots \left(\sigma_{a_i}\right) \left(\sigma_{a_{i+1}}\right) \cdots \cdots \cdots \left(\sigma_{a_{m-2}}\right) \left(\sigma_{a_{m-1}}\right) \cdots \left(\sigma_{a_m}\right)
\]

where

- for each \(i = 0, 1, \ldots, m - 1\), \(a_i \neq a_{i+1}\);
- for each \(i = 1, \ldots, m - 1\), \(a_i \notin \text{Cont}_D(w_i)\).

The language \(R^+\) is defined as

\[
R^+ = \bigcup_{m=1,2,\ldots} R^+_m.
\]

**Lemma 50** For each \(k = 1, 2, 3, \ldots\), there exists a sentence \(\psi_k\) in LTL\(^1\)(\(\Sigma, x, u\)) such that \(L(\psi_k) = R^+_k\) and

- \(\text{fqr}(\psi_1) = 1\); and
- \(\text{fqr}(\psi_k) = k - 1\), when \(k \geq 2\).

**Proof.** First, we define a formula \(\varphi_k\) such that \(\text{fqr}(\varphi_k) = k - 1\) and for every data word \(w = (\sigma_{d_1}) \cdots (\sigma_{d_n})\), for every \(i = 1, \ldots, n\),

\[
w, i \models_d \varphi_k \text{ if and only if } \left(\sigma_{d_i}\right) \cdots \left(\sigma_{d_n}\right) \in R^+_k. \quad (7.1)
\]

We construct \(\varphi_k\) inductively as follows.

- \(\varphi_1 := X(\neg \top) \land \neg (X\text{True})\).

- For each \(k = 1, 2, 3, \ldots\),

\[
\varphi_{k+1} := X(\neg \top) \land X\left(\downarrow X(\neg \top u(\top \land \varphi_k))\right)
\]

102
Note that since \( \text{fqr}(\varphi_1) = 0 \), then for each \( k = 1, 2, \ldots \), \( \text{fqr}(\varphi_k) = k - 1 \).

Assuming first that \( \varphi_k \) satisfies the property in Equation 7.1, the desired sentence \( \psi_k \) is defined as follows.

- \( \psi_1 := \downarrow (X(\neg \uparrow) \land \neg (X\text{True})) \).
- For each \( k = 2, 3, \ldots \),

\[
\psi_k := \downarrow (X(\neg \uparrow)) \land X(\downarrow X((\neg \uparrow)U(\uparrow \land \varphi_{k-1})))
\]

Since \( \text{fqr}(\varphi_{k-1}) = k - 2 \), then \( \text{fqr}(\psi_k) = k - 1 \).

Now we want to show that the formula \( \varphi_k \) satisfies Equation 7.1. The proof is by induction on \( k \). The base case, \( k = 1 \), is trivial. Suppose, for the induction hypothesis, the formula \( \varphi_k \) satisfies Equation 7.1.

The induction step is as follows. Let \( w = (\sigma_{d_1}) \cdots (\sigma_{d_n}) \). We have the following chain of application of the semantics of LTL.

\[
\begin{align*}
\wedge, i &= \downarrow (X(\neg \uparrow) \land X(\downarrow X((\neg \uparrow)U(\uparrow \land \varphi_k)))) \\
\end{align*}
\]

For the first part, we have

\( w, i \models \varphi_{k+1} \) if and only if \( d_i \neq d_{i+1} \) \hspace{1cm} (7.2)

Now we evaluate the second part.

\[
\begin{align*}
\wedge, i + 1 &= \downarrow (X((\neg \uparrow)U(\uparrow \land \varphi_k))) \\
\end{align*}
\]

Equation 7.3 holds if and only if there exists \( j \) such that \( i + 2 \leq j \) and
1. $w, j \models_{d_{i+1}} \top \land \varphi_k$,

2. $w, j' \models_{d_{i+1}} \neg \top$, for each $j' = i + 1, \ldots, j - 1$.

By the semantics of LTL and the induction hypothesis, Clause 1 holds if and only if $d_j = d_{i+1}$ and $(\sigma_{d_j}) \cdots (\sigma_{d_n}) \in R^+_k$. The meaning of Clause 2 is $d_j' \neq d_{i+1}$, for each $j' = i + 1, \ldots, j - 1$. Both clauses, together with Equation 7.2, means that $(\sigma_{d_i}) \cdots (\sigma_{d_n}) \in R^+_{k+1}$. This completes the induction hypothesis.

**Lemma 51** For each $k = 1, 2, \ldots$, the language $R^+_{k+1}$ is not expressible by a sentence in $LTL_1^\downarrow(\Sigma, X, U)$ of freeze quantifier rank $k - 1$.

**Proof.** By Lemma 21, $R^+_{k+1}$ is not accepted by weak $k$-PA, thus, it is also not accepted by top-view $k$-PA. Then, by Theorem 49, $R^+_{k+1}$ is not expressible by $LTL_1^\downarrow(\Sigma, X, U)$ sentence of freeze quantifier rank $k - 1$.

Combining both Lemmas 50 and 51, we obtain the following strict hierarchy of $LTL_1^\downarrow(\Sigma, X, U)$ based on its freeze quantifier rank.

**Theorem 52** For each $k = 1, 2, \ldots$, the class of sentences in $LTL_1^\downarrow(\Sigma, X, U)$ of freeze quantifier rank $k + 1$ is strictly more expressive than those of freeze quantifier rank $k$.

### 7.4 Counter Automata

A Minsky $k$-counter automata (CA), with $\epsilon$-transitions and zero testing, is a tuple $A = \langle \Sigma, Q, q_0, \delta, F \rangle$, where

- $\Sigma$ is a finite alphabet;
- $Q$ is a finite set of states;
- $q_0$ is the initial state;
- $\delta \subseteq Q \times (\Sigma \cup \{\epsilon\}) \times L \times Q$ is a transition relation over the instruction set $L = \{\text{inc, dec, ifz}\} \times \{1, \ldots, k\}$;
- $F \subseteq Q$ is the set of accepting set, such that $q' \notin F$ whenever $(q, \epsilon, l, q') \in \delta$. 

104
Given a word \( w = \sigma_1 \cdots \sigma_n \in \Sigma^* \), a configuration of \( A \) is a triple \([i, q, v]\) where \( 0 \leq i \leq n \), \( q \in Q \) and a counter valuation \( v : \{1, \ldots, k\} \to \mathbb{N} \), where \( \mathbb{N} \) is the set of natural number \( \{1, 2, 3, \ldots\} \).

The initial configuration is \([0, q_0, v_0]\) where \( v_0(j) = 0 \) for each \( j = 1, \ldots, k \). The run of \( A \) on \( w \) is a sequence \([0, q_0, v_0], [1, q_1, v_1], \ldots, [n, q_n, v_n]\) where for each \( i = 0, \ldots, n-1 \), there exists a transition \((q_i, \sigma_{i+1}, l, q_{i+1}) \in \delta\) and

- if \( l = (\text{inc}, j) \) for some \( j = 1, \ldots, k \), then \( v_{i+1}(j) = v_i(j) + 1 \) and for all other \( j' \neq j \), \( v_{i+1}(j') = v_i(j') \).
- if \( l = (\text{dec}, j) \) for some \( j = 1, \ldots, k \), then \( v_i(j) > 0 \) and \( v_{i+1}(j) = v_i(j) - 1 \) and for all other \( j' \neq j \), \( v_{i+1}(j') = v_i(j') \).
- if \( l = (\text{ifz}, j) \) for some \( j = 1, \ldots, k \), then \( v_i(j) = 0 \) and \( v_{i+1} = v_i \).

The word \( w \) is accepted by \( A \) if \( q_n \in F \). As usual, we denote by \( L(A) \) the set of all words over \( \Sigma \) accepted by \( A \).

We say that the automaton \( A \) is incrementing if its counters may erroneously increase at any time. More precisely, The run of an incrementing \( A \) on \( w \) is a sequence of configurations \([0, q_0, v_0], [1, q_1, v_1], \ldots, [n, q_n, v_n]\) where for each \( i = 0, \ldots, n-1 \), there exists a transition \((q_i, \sigma_{i+1}, l, q_{i+1}) \in \delta\) and

- if \( l = (\text{inc}, j) \) for some \( j = 1, \ldots, k \), then \( v_{i+1}(j) \geq v_i(j) + 1 \) and for all other \( j' \neq j \), \( v_{i+1}(j') \geq v_i(j') \).
- if \( l = (\text{dec}, j) \) for some \( j = 1, \ldots, k \), then \( v_i(j) \geq 0 \) and \( v_{i+1}(j) \geq v_i(j) - 1 \) and for all other \( j' \neq j \), \( v_{i+1}(j') \geq v_i(j') \).
- if \( l = (\text{ifz}, j) \) for some \( j = 1, \ldots, k \), then \( v_i(j) = 0 \) and for all \( j' = 1, \ldots, k \), \( v_{i+1}(j') \geq v_i(j') \).

**Theorem 53** [8, Theorem 2.9] (See also [19, Theorem 6] and [23]) The non-emptiness problem for incrementing counter automata is decidable, but not primitive recursive.

### 7.5 Generalization of Theorem 39

Let \( A = \langle Q, q_0, \mu, F \rangle \) be a weak \( k \)-PA. We will show how to construct one-way alternating \((k - 1)\)-RA \( A' \). For our convenience, we assume that \( A \) is deterministic. We also assume that \( A \) behaves as follows.
• For every configuration $\gamma$ of $A$, there exists a transition in $\mu$ that applies to it.

• Only pebble $k$ can enter a final state and it does so only after it reads the right-end marker $\triangleright$.

• For every $i = 2, \ldots, k$, immediately after pebble $i$ moves right, pebble $i - 1$ is placed.

• For every $i = 1, \ldots, k - 1$, pebble $i$ is lifted only when it reaches the right-end marker $\triangleright$.

• For every $i = 1, \ldots, k - 1$, immediately after pebble $i$ is lifted, pebble $(i + 1)$ moves right.

See Subsection 4.3.1 on how this normalization can be done.

We also assume that the set of states $Q$ is partitioned into $Q_1 \cup \cdots \cup Q_k$ where $Q_i \cap Q_j$ whenever $i \neq j$ and $Q_i$ is the set of states when pebble $i$ is in control.

The automaton $A' = \langle Q', q_0', u_0', \mu', F' \rangle$ is defined as follows.

• The set of states is $Q' = Q \cup Q^2 \cup Q^3 \cup Q \times Q$, where $\tilde{Q} = \{ \tilde{q} \mid q \in Q \}$ and $Q \times Q = \{ (q, p) \mid p, q \in Q \}$.

• The initial state is $q_0' = q_0 \in Q_k$.

• The initial assignment is $\#^{k-1}$.

• The set of final states is $F' = F \cup \{ (q, q) : q \in Q \}$.

For our convenience, we number the registers of $A'$ from 2 to $k$, not from 1 to $(k - 1)$. The set of transitions $\mu'$ consists of the following.

• For $i = k - 1, \ldots, 1$, we have the following transitions.

  1. For each transition $(i, \sigma, V, q) \rightarrow (q', \text{right}) \in \mu$, there are transitions $((q, p), \sigma, V) \rightarrow (q', p) \in \mu'$ for all $p \in Q_{i+1}$.

  2. For each transition $(i, P, V, q) \rightarrow (q', \text{place-pebble}) \in \mu$, there are the following transitions in $\mu'$. For every $p \in Q_{i+1}$,

     $$(q, p) \rightarrow (\tilde{q}, p), \{i\}$$
Let $\rho \equiv \bigwedge_{p_j \in Q_i} \left( (q, p_j, p) \right)$

\[
(q, p_j, p) \rightarrow (p_j, p) \land (q', p_j) \quad \text{for every } p_j \in Q_i
\]

- For $i = k$, there are the following transitions in $\mu'$.

  1. For each transition $(k, \sigma, \emptyset, q) \rightarrow (q', \text{right}) \in \mu$, there is a transition $(q, \sigma, \emptyset) \rightarrow (q') \in \mu'$.

  2. For each transition $(k, \sigma, \emptyset, q) \rightarrow (q', \text{place-pebble}) \in \mu$, there are the following transitions in $\mu'$.

\[
q \rightarrow \tilde{q}, \{ k \}
\]

\[
\tilde{q} \rightarrow \bigwedge_{p_j \in Q_k} (q, p_j)
\]

\[
(q, p_j) \rightarrow p_j \land (q', p_j) \quad \text{for every } p_j \in Q_k
\]

We can show the following proposition by straightforward induction on $i$.

**Proposition 54** Let $w = (\sigma_1) \cdots (\sigma_n)$ be a $\Sigma$-data word. For $i = 1, \ldots, k - 1$, there exists an $i$-run $[i, q_1, \theta_1] \vdash^* [i, q_2, \theta_2]$ of $A$ on $w$ and $[i, q_2, \theta_2] \vdash [i + 1, q_3, \theta_3]$ if and only if the configuration $[\theta(i), (q_1, q_3), u_2 \cdots u_k]$ of $A'$ on $w$ leads to acceptance, where $u_j = a_{\theta(j)}$, for $j = i + 1, \ldots, k$.

Then, by the definition of $\mu'$, we can easily deduce the following. For each $\ell = 1, \ldots, n$,

\[
[k, q_1, \theta_1] \vdash_A [k - 1, q_2, \theta_2] \vdash^*_A [k - 1, q_3, \theta_3] \vdash_A [k, q_4, \theta_4] \vdash_A [k, q_5, \theta_5]
\]

is a $k$-run of $A$ on $w$, where $\theta_1(k) = \theta_2(k) = \theta_3(k) = \theta_4(k) = \ell$ and $\theta_5(k) = \ell + 1$ and $\theta_3(k - 1) = n + 1$ if and only if

\[
[k, q_1, \#^{k-2}a_{\ell-1}] \vdash [k, q_1, \#^{k-2}a_{\ell}]
\]

\[
[k, q_1, \#^{k-2}a_{\ell}] \vdash [k, q_1, q_4, \#^{k-2}a_{\ell}]
\]

\[
[k, (q_1, q_4), \#^{k-2}a_{\ell}] \vdash [k, q_4, \#^{k-2}a_{\ell}]
\]

\[
[k, (q_1, q_4), \#^{k-2}a_{\ell}] \vdash [k, q_2, q_4, \#^{k-2}a_{\ell}]
\]

\[
[k, q_4, \#^{k-2}a_{\ell}] \vdash [\ell + 1, q_3, \#^{k-2}a_{\ell}]
\]

and the configuration $[\ell, (q_2, q_4), \#^{k-2}a_{\ell}]$ leads to acceptance.

Now, the equivalence between $L(A)$ and $L(A')$ follows immediately.
Chapter 8

Conclusion

In this thesis we studied pebble automata for data languages. In particular we have shown that in either strong or weak PA more pebbles provide more expressive power. Furthermore, we also establish a fragment of PA languages for which the emptiness problem is decidable, the so called top view weak PA.

As shown in our thesis, top view weak PA inherit nice properties mentioned in Chapter 1.

1. Expressiveness: Top view weak PA strictly contain the languages expressible by $\text{LTL}_1^1(\Sigma, x, u)$.

2. Decidability: The emptiness problem is decidable.

3. Efficiency: The model checking problem, that is, testing whether a given string of length $n$ is accepted by a specific deterministic top view weak $k$-PA can be solved in $O(n^k)$ time.

4. Closure properties: Top view weak $k$-PA languages are closed under all boolean operations.

5. Robustness: Alternation and nondeterminism do not add expressive power to top view weak $k$-PA languages.

Moreover, various techniques used in the thesis show the unexpectedly rich structures inherent even in the finite data strings. One example of such technique is that in Chapter 5, where a sequence of data values can be
viewed as a directed graph. Another example is in the proof of undecidability of strong 2-PA, where it is shown that strong 2-PA can simulate arithmetic operations. We believe that our study can contribute to the reasoning aspect of study of data languages. Some of our results are published in [29, 27, 15, 26].

There are still lots of work to be done. In order to be applicable in program verification and XML, the settings should be infinite strings and unranked trees, respectively. Thus, the question remains whether it is possible to extend top-view weak PA to the settings of infinite strings and unranked trees, while still preserving the five properties mentioned above. If it is not possible, then are there models for data languages in those settings possessing such properties?
Bibliography


RA are considered for the purpose of the modelling of the past and future. The evolution of the concepts is also part of the analysis. The thesis is divided into seven parts: the first three chapters deal with the past and future evolution, the fourth chapter deals with the evolution of the concepts, and the last three chapters deal with the future evolution of the concepts.
The last operation before a new operation starts is the functional operation. This means that we set it to zero before starting the operation.

Next, we note that: the language of the three automata PA, the language of the three automata PA, and the language of the three automata PA is the language of the three automata PA.

In summary, as we have seen,[15], the language of the three automata PA, the language of the three automata PA, and the language of the three automata PA is the language of the three automata PA.
In the previous sections, we defined the logic of the first-order theory of the finite memory automata (FMA) that are register automata (RA) - Register Automata. [5] A variant of the first-order logic, called RA, can be seen as a first-order logic with a register that always contains the current state of the automaton. However, the first-order logic with a register is not enough to express certain properties of the automaton, such as the existence of certain states or transitions.

The extended first-order logic with a register is called existential monadic second-order logic (EMSO). In this logic, we can express properties of the automaton that cannot be expressed in first-order logic. For example, we can express that there exists a state in the automaton that is reachable from a given state.

In this section, we will study the logic of the finite memory automata with registers. We will show that this logic is expressive enough to express certain properties of the automaton, such as the existence of certain states or transitions.

The logic of the finite memory automata with registers is called LTL$^1$ (Σ,X,U) (n register freeze quantifier) one-way alternating n-register automata (RA). We will study this logic in detail and show that it is expressive enough to express certain properties of the automaton.
רשימת האיורים

29 ................................................................. 4.1.
30 ...................................................................... 4.2.
33 ................................................................. 4.3.
53 ...................................................................... 5.1.
55 ...................................................................... 5.2.
64 ................................................................. 5.3.
70 ...................................................................... 5.4.
88 ................................................................. 7.1.
89 ...................................................................... 7.2.

G ∈ ℋ, n ≤ |V(G)| ≤ m.

G_1 G_2, דוגמה למקרה של בניית אוטומטים בין שני אוטומטים G_1, G_2.

G_1 G_2, דוגמה למקרה של בניית אוטומטים בין שני אוטומטים G_1, G_2.

w = (σ_1, d_1) ... (σ_n, d_n), A כדי לה髻י מודל למותם של A.
7.3 חלשים המבטיים מלמטה ע"פ חלוקה

7.4 אוטומטים עם מונים

7.5 ההכלה של משפח 39

8 סיכום

בibliוגרף
תקנון עניינים

1. .................................................................לוגיקה
2. ..........................................................................
3. ..........................................................................
4..............................................................................
5..............................................................................
6..............................................................................
7..............................................................................
8..............................................................................
9..............................................................................
10. .........................................................................
11. ...........................................................................
12. ...........................................................................
13. ...........................................................................
14. ...........................................................................
15. ...........................................................................
16. ...........................................................................
17. ...........................................................................
18. ...........................................................................
19. ...........................................................................
20. ...........................................................................
21. ...........................................................................
22. ...........................................................................
23. ...........................................................................
24. ...........................................................................
25. ...........................................................................
26. ...........................................................................
27. ...........................................................................
28. ...........................................................................
29. ...........................................................................
30. ...........................................................................
31. ...........................................................................
32. ...........................................................................
33. ...........................................................................
34. ...........................................................................
35. ...........................................................................
36. ...........................................................................
37. ...........................................................................
38. ...........................................................................
39. ...........................................................................
40. ...........................................................................
41. ...........................................................................
42. ...........................................................................
43. ...........................................................................
44. ...........................................................................
45. ...........................................................................
46. ...........................................................................
47. ...........................................................................
48. ...........................................................................
49. ...........................................................................
50. ...........................................................................
51. ...........................................................................
52. ...........................................................................
53. ...........................................................................
54. ...........................................................................
55. ...........................................................................
56. ...........................................................................
57. ...........................................................................
58. ...........................................................................
59. ...........................................................................
60. ...........................................................................
61. ...........................................................................
62. ...........................................................................
63. ...........................................................................
64. ...........................................................................
65. ...........................................................................
66. ...........................................................................
67. ...........................................................................
68. ...........................................................................
69. ...........................................................................
70. ...........................................................................
71. ...........................................................................
72. ...........................................................................
73. ...........................................................................
74. ...........................................................................
75. ...........................................................................
76. ...........................................................................
77. ...........................................................................
78. ...........................................................................
79. ...........................................................................
80. ...........................................................................
81. ...........................................................................
82. ...........................................................................
83. ...........................................................................
84. ...........................................................................
85. ...........................................................................
86. ...........................................................................
87. ...........................................................................
88. ...........................................................................
89. ...........................................................................
90. ...........................................................................

Technion - Computer Science Department - Ph.D. Thesis PHD-2009-07 - 2009
המחקר נועש בנהיה פורפ"ח, מייאל קמינסקי, בפקולטה למדעי המחשב

אני מודה לטכניון ולרפאלו מיראמ מישן על התמיכת כספית הנדיבת

בהשכלה מקדימה

Technion - Computer Science Department - Ph.D. Thesis PHD-2009-07 - 2009
אוטומטים עם הולקים عبر שפות נתונים: הפרדה,سرعة ואיציוות

hiyor leshem milui hakik shel hetirushot lekiput hetavor doktor bamediyim moshav

טומי טא

השב לסמוט הטכניון – המרכז הטכניولي לישראל

נוף חשמונאי אפריל 2009
אוטמותים עם חלוקים עבוריしましょう
הפרדה, כריעות וא-כריעות

טומי טאך
Technion - Computer Science Department - Ph.D. Thesis PHD-2009-07 - 2009
החלקים המחלפים בולין, גורר את נתונתقه. מכס שבעית הספיקות של ILTL$^2(\Sigma,X,U)$ אי

לא ניתן להכרעה. המספר גורר נסף שואף להפרת העבר הפונט את הוליקה

לבטל ניצח להכרעה.

בתחדזה זו אנושיכים את חくり-ה. ח iletiות LTL$^2$กรณום בכרר חקוד($.LIFO$) -

החלקה משמחת לסיום

המודיך בולין הקול. ל חくなりました כולם מחו בית וחלקה ש/browser את אחרי משמש
cארש האוטומט. האוטומט עביר ממצק שלצבר בהתחוםلب窕ית יהי ששונין בז' התוחיות

שנמצאות חתח התחולים.

בנוסח, כフィersion -ככ[15], קיימים שני מודלים של Prosecutor: ח الذى, התחולים הדוהים מדッツיב ביתהתחלקה הקול.

המודל המואר הקורא קיימים, החולים הדוהים מדッツיב ביתהתחלקה הקול.

המודול המואר קורא קיימים, החולים הדוהים מדッツיב ביתהתחלקה הקול.

המודל המואר קורא קיימים, החולים הדוהים מדッツיב ביתהתחלקה הקול.

המודל המואר קורא קיימים, החולים הדוהים מדッツיב ביתהתחלקה הקול.

המודל המואר קורא קיימים, החולים הדוהים מדッツיב ביתהתחלקה הקול.

המודל המואר קורא קיימים, החולים הדוהים מדッツיב ביתהתחלקה הקול.

המודל המואר קורא קיימים, החולים הדוהים מדッツיב ביתהתחלקה הקול.

המודל המואר קורא קיימים, החולים הדוהים מדッツיב ביתהתחלקה הקול.

המודל המואר קורא קיימים, החולים הדוהים מדッツיב ביתהתחלקה הקול.

המודל המואר קורא קיימים, החולים הדוהים מדッツיב ביתהתחלקה הקול.

המודל המואר קורא קיימים, החולים הדוהים מדッツיב ביתהתחלקה הקול.

המודל המואר קורא קיימים, החולים הדוהים מדッツיב ביתהתחלקה הקול.

המודל המואר קורא קיימים, החולים הדוהים מדッツיב ביתהתחלקה הקול.

המודל המואר קורא קיימים, החולים הדוהים מדッツיב ביתהתחלקה הקול.

המודל המואר קורא קיימים, החולים הדוהים מדッツיב ביתהתחלקה הקול.

המודל המואר קורא קיימים, החולים הדוהים מדッツיב ביתהתחלקה הקול.

המודל המואר קורא קיימים, החולים הדוהים מדッツיב ביתהתחלקה הקול.

המודל המואר קורא קיימים, החולים הדוהים מדッツיב ביתהתחלקה הקול.

המודל המואר קורא קיימים, החולים הדוהים מדッツיב ביתהתחלקה הקול.

המודל המואר קורא קיימים, החולים הדוהים מדッツיב ביתהתחלקה הקול.

המודל המואר קורא קיימים, החולים הדוהים或多Faces מדッツיב ביתהתחלקה הקול.

המודל המואר קורא קיימים, החולים הדוהים מדッツיב ביתהתחלקה הקול.

המודל המואר קורא קיימים, החולים הדוהים מדッツיב ביתהתחלקה הקול.

המודל המואר קורא קיימים, החולים הדוהיםomedical.health מדッツיב ביתהתחלקה הקול.

המודל המואר קורא קיימים, החולים הדוהיםomedical.health מדッツיב ביתהתחלקה הקול.

המודל המואר קורא קיימים, החולים הדוהיםomedical.health מדッツיב ביתהתחלקה הקול.

המודל המואר קורא קיימים, החולים הדוהיםomedical.health מדッツיב ביתהתחלקה הקול.

המודל המואר קורא קיימים, החולים הדוהיםomedical.health מדッツיב ביתהתחלקה הקול.

המודל המואר קורא קיימים, החולים הדוהיםomedical.health מדッツיב ביתהתחלקה הקול.

המודל המואר קורא קיימים, החולים הדוהיםomedical.health מדッツיב ביתהתחלקה הקול.

המודל המואר קורא קיימים, החולים הדוהיםomedical.health מדッツיב ביתהתחלקה הקול.

המודל המואר קורא קיימים, החולים הדוהיםomedical.health מדッツיב ביתהתחלקה הקול.

המודל המואר קורא קיימים, החולים הדוהיםomedical.health מדッツיב ביתהתחלקה הקול.

המודל המואר קורא קיימים, החולים הדוהיםomedical.health מדッツיב ביתהתחלקה הקול.

המודל המואר קורא קיימים, החולים הדוהיםomedical.health מדッツיב ביתהתחלקה הקול.

המודל המואר קורא קיימים, החולים הדוהיםomedical.health מדッツיב ביתהתחלקה הקול.

המודל המואר קורא קיימים, החולים הדוהיםomedical.health מדッツיב ביתהתחלקה הקול.

המודל המואר קורא קיימים, החולים הדוהיםomedical.health מדッツיב ביתהתחלקה הקול.

המודל המואר קורא קיימים, החולים הדוהיםomedical.health מדッツיב ביתהתחלקה הקול.

המודל המואר קורא קיימים, החולים הדוהיםomedical.health מדッツיב ביתהתחלקה הקול.

המודל המואר קורא קיימים, החולים הדוהיםomedical.health מדッツיב ביתהתחלקה הקול.

המודל המואר קורא קיימים, החולים הדוהיםomedical.health מדッツיב ביתהתחלקה הקול.

המודל המואר קורא קיימים, החולים הדוהיםomedical.health מדッツיב ביתהתחלקה הקול.

המודל המואר קורא קיימים, החולים הדוהיםomedical.health מדッツיב ביתהתחלקה הקול.

המודל המואר קורא קיימים, החולים הדוהיםomedical.health מדッツיב ביתהתחלקה הקול.

המודל המואר קורא קיימים, החולים הדוהיםomedical.health מדッツיב ביתהתחלקה הקול.

המודל המואר קורא קיימים, החולים הדוהיםomedical.health מדッツיב ביתהתחלקה הקול.

המודל המואר קורא קיימים, החולים הדוהיםomedical.health מדッツיב ביתהתחלקה הקול.

המודל המואר קורא קיימים, החולים הדוהיםomedical.health מדッツיבペットחית את

שנתו של תחית שבתאש שמסת תשליפת החולים האזרחים. מנור Mỹ שברקה של שטי

חולים, יז הביא של תחית ב chiranelקשלים בה Chattanooga.בודה–יתתחית

בchers, המודל שלchersתשליפת הינו ייצב: לאוטומטיות מתחלפים, א–.
نظرית

לגcreateView אוטומטים למילוי מעלי אלפ-בית סופי מובנים תở חסירת. לאחרונה יש פיעול
מחקרייה חותם לcreateView אווטומטים ל밀וי עיצום מעלי אלפ-בית אינסופי, שבע דווי
תחטכתיים למעלים בצלון.] פיעולו וז מוצעת מצד אחוד של ידי היאור באמות
פרומריל ו(solution) של מעורות שביעות מתפרנסי על אינסופי של מכתב, ומוצר יש עלידי חפוש
אתור משוט אווטומטיstit לתח

ניתן להפרcdn עלייה שלישית של שית יحرك פסיקות מתפרנסי. לقرأ
נסקור בכרדה שית י獗יה של איה פספורtı המפרנס-ב[10]. חקר פסיט הנחיתות שע
המפרנסיה של אוטומטים מספר יזכר פסיפי - FMA (Finite Memory Automata) [5].
( RA ) Register Automata ( אוטומטים אלה הוא בוטים עם אוטומטים ביבלי אוטמייר – PB Pebble [8], שבם ובגדוד אוטומטים עם חלקים – disputedAUTS ( PA ) Automata
ידי פסיקת תחט פטאלן סטסדרטיה על השפה, בעית, יחות, יחור, נזרור וטיסירה.
בנסופ לג', במקודד המבר החשוש, הן.Window חליפות פסיקת מברק: פּוּטֶי הירוקת
عبارة בניה פיתורב, בועדת ישנה הגרדה פיתור שער
בעבר בניה פיתורב, בועדה ישנה פיתור שער
בעבר בניה פיתורב, בועדה ישנה פיתור שער
ליגוי בוטנ ירכות,כנן, פסיקת חתונ לכל פסיקת הבולאינוית, בשטח היפוסית
מתכנתות על-ידי RA – פסיקת חותות פשולים.
בשלצ בוח האזור בו [3]-ל דיג 설명 מדריך את פסיקת הנחיתות למ덩 הומצא
אותם הנחיתות מדריך פרגמנט של לדיגカンטרוד הנומרים הקויית מתכותר
– פסיקת הנחיתות. פרגמנט nak נדיגカンטרוד והן improש הקויית שא
ל photoshop אוונאיה. אלו שמידות על.backgroundColor של בעי גי הים ולעשתו
 PSA מברק - השפות שמתאימות לדיגカンטרוד及び תמונת פועלום באמו.
- וברק שמידות והחברות שמתאימות לדיגカンטרוד
פּוּטֶי הקויית
בעבר לדיגカンטרוד ביבלי פטעלת השפות שמתאימות לדיגカンטרוד
ליגוי בוטנ ירכות,כנן, פסיקת חתונ לכל פסיקת הבולאינוית, בשטח היפוסית

בשלצ בוח האזור בו [3]-ל דיג 설명 מדריך את פסיקת הנחיתות למ덩 הומצא
אותם הנחיתות מדריך פרגמנט של צילוםカンטרוד הנומרים הקויית מתכותר
– פסיקת הנחיתות. פרגמנט nak נדיגカンטרוד והן improש הקויית שא
ל photoshop אוונאיה. אלו שמידות על.backgroundColor של בעי גי הים ולעשתו
 PSA מברק - השפות שמתאימות לדיגカンטרוד及び תמונת פועלום באמו.
- וברק שמידות והחברות שמתאימות לדיגカンטרוד
פּוּטֶי הקויית
בעבר לדיגカンטרוד ביבלי פטעלת השפות שמתאימות לדיגカンטרוד
ליגוי בוטנ ירכות,כנן, פסיקת חתונ לכל פסיקת הבולאינוית, בשטח היפוסית

רשימת האイורım

29 ........................................................................ 4.1 הקבוצה מצבי שנותワーク קונצוצע המ sonra תולק-1
30 ........................................................................ 4.2 החסינתה הקושרת של תולק-2
33 ................................................... (\(\sigma_1,a_1\)) (\(\sigma_2,a_2\)) (\(\sigma_3,a_3\)) (\(\sigma_4,a_4\)) 4.3 סדרת התנועות של \(\mathcal{A}\) על \(T\)
53 ................................................................. \(G \in G_{k,m}\) 5.1 הגרף המושרה על \(T\)
54 ........................................................................ 5.2 כשקו \(G_1\) בתולק-3 נצמד בקטעים \(D\) וא \(A\) \(T\) האינדוקציה, תולק-2 \(T\) איות מ-5
55 ........................................................................ 5.3 כשקו \(G_2\) בתולק-3 נצמד בקטעים \(E\) וא \(C\) \(T\) האינדוקציה, תולק-2 \(T\) איות מ-5
64 ........................................................................ 5.4 המימו של \(T\)
65 ........................................................................ 5.5 כשקו \(G_1\) (\(i+1\)) \(T\) האינדוקציה, \(T\) איות מ-7
66 ........................................................................ 5.6 כשקו \(G_1\) \(T\) האינדוקציה, \(T\) איות מ-8
70 ........................................................................ 5.7 \(w = (\sigma_1,d_1)\ldots(\sigma_n,d_n)\) 7.1 לע הירצה של \(\mathcal{A}\) על \(T\)
71 ........................................................................ 7.2 שמתארמה לירצה ביאו-5
7.3 חלשים המbrtcיםملמuida עמ k חלוקים

7.4 אוטומטים עמ מנין

7.5 הכללה של משפט 39

8. סיכום

בבלוגרפיה
המחקר נעשה בהנחיית פרופ’ מיכאל קמינסקי, بكلולטת لمורשה המԽשב

אני מודה לטרינון ורפסל ומיראס Miša על התמיכות כספית הנדיבת

בוחתלמודתי
אוטומטים עם חל続き עזר שפורטים

הפרדה,_CLIועות וא_CLIועות

היבר לשום שלף חלليك, ולש הדירישות לקצבת
הונאות וניקור במדעי המחשב

טו_ טון

הושב לסטה הטכניון – המרכז הטכניologiי _ישראלי

נופ חסמ"ט

אפריל 2009

Technion - Computer Science Department - Ph.D. Thesis PHD-2009-07 - 2009
אוטומטים על חלוקים עברו שפועה והפרדה, צירוף או-צירוף

טומי טאם