POLITECNICO DI MILANO
Facoltà di Ingegneria
Corso di Laurea Specialistica in Ingegneria Informatica

Floyd languages for infinite words

Advisor:
Prof. Matteo Pradella

Thesis by:
Federica Panella,
matricola 749952

Anno Accademico 2010-2011

POLITECNICO DI MILANO
Facoltà di Ingegneria
Corso di Laurea Specialistica in Ingegneria Informatica

Linguaggi di Floyd per parole infinite

Relatore:
Prof. Matteo Pradella

Tesi di Laurea di:
Federica Panella,
matricola 749952

Anno Accademico 2010-2011

Abstract

Nowadays many software systems are designed to work without interruption: for instance operating systems are required to provide continuously their services to the users, and web applications and verification of critical and concurrent systems represent a typical scenario where endless computations are involved. The behavior of systems operating forever can be formally specified by means of ω-languages, a class of languages defined as sets of infinite words of symbols.

Theory on ω-languages dates back to the sixties, with the pioneering works of Büchi and Muller on finite-state automata able to recognize infinite strings([5], [19]). More recently, ω-languages have been studied in [2] as an extension of Visibly Pushdown Languages (VPL), a class of deterministic context-free languages which model strings as nested words of symbols. This work of thesis follows this line of research with the aim of investigating Floyd languages (FL), a traditional formalism introduced in [12] and rediscovered in the recent work [8], generalizing them to languages of infinite words.

FLs are one of the early classes of Deterministic Context-Free (DCF) languages and are defined by operator precedence grammars, renamed Floyd Grammars (FG) in [8]. Research on FLs began decades ago (from the first work of Floyd, [12], and [9]) and has been then interrupted after the introduction of advanced parsing techniques for more expressive languages, as LR(k). Renewal of interest in the study of FLs, which motivates the investigation carried on in this thesis, derived from the discovery of recent surprising results: along the path of research on parentheses-like languages resumed with the definition of VPLs and their characterization in terms of a class of automata (named Visibly Pushdown Automata), recent studies revealed the potential of FLs, proving that FLs are a proper superclass of VPLs and represent the largest known DCF family closed under all classical operations enjoyed by regular languages. Furthermore, the recent paper [17] characterized FLs by means of a complementary class of stack-based automata (Floyd Automata) and this result leads quite naturally to a further interesting extension of FLs to ω-languages. The main issue of this thesis is, thus, the investigation of this novel class of ω-languages and the focus of this study consists in the characterization of suitable abstract machines to define these languages. More to the point, ω-languages have been traditionally characterized in terms of finite-state automata, augmented with different kinds of properties and acceptance conditions which completely determine their expressive power. Classical conditions on how ω-automata recognize strings refer to the sequence of states of the automata occurring in the computation for the acceptance of a word: typical requirements, named after their authors, as Büchi, Muller and Rabin [21] conditions, constrain in particular ways the set of states belonging to these sequences and that are visited infinitely often. The distinguishing feature of FLs w.r.t. classical regular ω-languages is that they define context-free languages and their corresponding automata are endowed with a non-finite memory unit.

Thus, the acceptance condition cannot be restricted to the analysis of visited states, but naturally has to be related to the evolution of the content of the stack. Another interesting issue, that will be investigated in this work, deals with how the introduction of non-determinism in the moves of the automata may influence the recognizability power of these formalisms.

Actually, as FLs are an open field of study, both classical and more recent results of theory on ω-languages do not have correspondence for this family of languages, and this work of thesis aims at starting research along this direction.

Sommario

Molti dei sistemi software che pervadono il mondo che ci circonda sono progettati per funzionare per sempre: si pensi alle applicazioni web o embedded o ai sistemi operativi, che hanno come tipico requisito di garantire ai loro utenti la continua disponibilità di determinati servizi, ed ogni loro (anche temporanea) interruzione viene percepita come un vero e proprio fallimento. Le proprietà di sistemi che compiono ininterrottamente le loro elaborazioni senza fine possono essere naturalmente specificate e verificate ricorrendo ai concetti e ai formalismi propri della teoria dei linguaggi omega, una classe di linguaggi formali composti da parole di lunghezza infinita.

Gli studi sui linguaggi omega risalgono agli anni Sessanta, quando Büchi e Muller presentarono i loro lavori pionieristici su automi in grado di riconoscere sequenze infinite di simboli ([5], [19]). Più recentemente, gli omega linguaggi sono stati studiati in [2] come un'estensione dei Visibly Pushdown Languages (VPL), una classe di linguaggi deterministici context-free che rappresentano le stringhe finite di un linguaggio come parole dotate di una struttura più ricca di quella che una sequenza lineare di simboli può conferire (nested word). Proseguendo questa linea di ricerca nell'ambito dei linguaggi formali, questo lavoro di tesi mira ad investigare le proprietà dei Linguaggi di Floyd (FL), un formalismo classico introdotto in [12] e riscoperto nel recente lavoro [8], estendendo questa famiglia di linguaggi al mondo dei linguaggi infiniti.

Gli FL rappresentano una delle prime classi di linguaggi deterministici context-free (DCF) e sono definiti da grammatiche a precedenza di operatori (operator precedence grammars), chiamate Floyd Grammars in [8] in memoria di Robert Floyd. Fu proprio Floyd nel 1963 a dare avvio alla ricerca sui FL, ma dopo poco tempo lo studio su questa classe di linguaggi venne interrotto con l'emergere di formalismi più potenti ed espressivi per il parsing dei linguaggi di programmazione, come LR(k). L'interesse sui FL riprese in seguito a nuove ricerche sui linguaggi caratterizzati da una struttura a parentesi delle loro parole, quali i linguaggi di mark-up (XML, HTML), culminate con la definizione dei VPL e la loro caratterizzazione attraverso un classe equivalente di automi (VPA). Recentemente, in [8] si è mostrato come proprio i VPL siano una sottoclasse stretta dei FL, e come quest ultimi rappresentino la più ampia famiglia nota di linguaggi DCF chiusi rispetto a tutte le operazioni tipiche dei linguaggi regolari. Inoltre, ancora più di recente, il lavoro [17] ha caratterizzato anche i FL in termini di una classe equivalente di automi, ed è proprio questo fondamentale risultato a consentire l'introduzione dei Linguaggi di Floyd nel campo della teoria delle parole infinite.

Il tema centrale di questa tesi è, pertanto, lo studio di questa nuova classe di linguaggi omega, con l'obiettivo di darne una definizione appropriata attraverso adeguati formalismi operazionali, in linea con la teoria omega classica. Tradizionalmente, infatti, i linguaggi omega sono stati caratterizzati in termini di macchine astratte, che si contraddistinguono per le modalità di accettazione dei linguaggi da essi riconosciuti. I diversi modi di riconoscimento si riferiscono alle sequenze di stati che vengono attra-

versati un numero finito o infinito di volte dagli automi durante la lettura di una parola: essendo gli FL una classe di linguaggi non contestuali, equivalenti ad una famiglia di pushdown automata, il riconoscimento delle stringhe deve far anche riferimento all'evoluzione della pila durante una computazione e al particolare modo in cui la pila viene aggiornata negli FA, e ciò influenza il modo in cui le condizioni di accettazione devono essere definite per gli ω-FL. Un altro aspetto interessante, che verrà analizzato in questo lavoro di tesi, fa riferimento alle conseguenze che l'introduzione del nondeterminismo ha nel potere espressivo di questa classe di linguaggi.

Di fatto, i Linguaggi di Floyd rappresentano un campo ancora aperto per la ricerca, e i risultati (più antichi o recenti) della teoria dei linguaggi omega non hanno un'analoga corrispondenza per questa famiglia di linguaggi. Ed è proprio questa direzione di ricerca che questo lavoro di tesi si pone l'obiettivo di iniziare.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

"MEFISTOFELE
...Si tenga, insomma, alle parole!
Questa è la porta più sicura per entrare
nel Tempio di Certezza.
STUDENTE
Ma ci sarà un concetto, insieme alla parola.
MEFISTOFELE
Va bene! Ma che questo non ci tormenti troppo!
Dove i concetti mancano
ecco che al punto giusto compare una parola.
Con le parole puoi discutere benissimo,
con le parole edifichi un sistema,
alle parole puoi credere benissimo,
neppure un iota puoi portare via
a una parola."
(Goethe, Faust, l.1990-2000)

Words are the fundamental paradigm formal language theory is centered around, and it is quite astonishing to see how it is so versatile to adapt to describe possibly every aspect of computer science and of so many other even not scientific fields.
Words of formal languages can be used to model real phenomena, for the design and building of complex systems, to define their behavior and analyze which properties hold. Safety and reliability of control systems are believable, up to a degree of certainty, resorting to traditional techniques of specification and verification, typically by means of devices, as abstract machines or automata, which recognize languages of finite words. Not to mention that formal languages, paired with their corresponding classes of automata on finite words, pervade so distant areas as parsing, text and image processing, database querying, genomics, etc. But in all these fields formal language and automata theory deals with languages that are indeed expressive, yet composed only of words that are finite sequences of symbols.

One of the most interesting extension of classical theory is the study of ω-languages, whose words are no longer defined as sequences of symbols of finite length, but infinite sequences of letters are allowed. Research of formal language theory along this direction opened up new perspectives and laid the basis for more and more accurate

modeling of relevant phenomena.

Languages of infinite words may represent systems involved in nonterminating computations, and investigation in this field is motivated by several applications arising in computer science. Many of today's software systems are designed to work forever: operating systems and several web and embedded applications are supposed to provide without interruption their services to the users, and any temporary unavailability is considered as a critical failure. Specification and verification of concurrent and distributed systems may be properly formalized with the results of ω-language theory: indeed, while safety properties may be studied by means of automata on finite-length words, analysis of liveness and fairness requirements implies that endless computations of automata, on words of infinite length, are considered.

The study on infinite words and automata able to recognize them, however, originates in a quite different scenario. Research on ω-languages began in the sixties, when R. Büchi investigated certain monadic second order theories: his work [5] led to the definition of the first class of automata able to accept sets of infinite words (named after its author, Büchi automata) and he proved that they provide a powerful formalism to define properties expressed in these logical theories. Some years later, while studying asynchronous oscillating circuits, D. Muller further developed this theory, establishing new fundamental results.
Following several other results after the pioneering works of Büchi and Muller on finite automata able to recognize infinite strings, ω-languages have been more recently studied in [2] as an extension of Visibly Pushdown Languages (VPL), a class of deterministic context-free languages which model strings as nested words of symbols.

Along this line of research, Floyd languages (FL), a traditional formalism introduced in [12] and rediscovered in the recent work [8], have been recently characterized by means of a perfectly matching class of stack-based automata (Floyd Automata, FA) and this result leads quite naturally to a further interesting extension of FLs to languages of infinite words.

1.1 "Ma ci sarà un concetto, insieme alla parola"... i.e. why ω-VPL and ω-FL are important

Besides traditional ω-languages, investigation on ω-VPLs and in particular ω-FLs is justified by several reasons. First of all, classical finite regular languages assign to words a linear structure, but in many contexts the semantics of words logically imply both a linear and a hierarchical structure. As an example, structured web documents and bonds in chemical compounds can be modeled by words with a dual nature. The distinguishing feature of VPLs and FLs w.r.t. regular languages is that they introduce a model of nested words to represent data with dual linear-hierarchical structure in a natural way. More precisely, VPLs and FLs define context-free languages whose parsing is input-driven, as the symbols themselves determine the structure (i.e. the syntax tree and, thus, mainly even the underlying concept) of the word they belong to. FLs grant words an even more adequate syntactic structure than VPLs, and it is often the case that they reflect deeper semantics.

The ω-automata corresponding to these classes of languages are also endowed with a non-finite memory unit, that allows for acceptance of more complex languages than ω-regular ones. Furthermore, a significant difference of these classes of languages w.r.t other classical pushdown ω-automata is that their finite counterparts both enjoy the

same closure properties as regular languages of finite words. As regards VPLs, these properties extend to the infinite field too, and since FLs provide a more expressive formalism than VPLs, these features motivates research on ω-FLs as well. Indeed, the numerous properties FLs enjoy open up promising perspectives in several relevant application contexts.
A first important field that naturally fits the peculiarities of FLs deals with the processing of structured or semistructured documents in streaming applications. One of the most common modes of representation of XML documents is defined by SAX (Simple API for XML), a standard interface proposed by Sun to process XML documents following an event-based approach founded on the Callback technique. SAX representation of XML documents mirrors the dual-nature structure that both VPLs and FLs assign to words, since documents are viewed as streaming sequences of symbols augmented with a hierarchical structure determined by the correspondence among tags. Parsers based on FLs may benefit of the advantages they provide w.r.t VPLs, as FLs allow to define a more adequate and richer structure for opening and closing contexts and may support the analysis of documents lacking a well-formed standardized layout.

A further fundamental application context of ω-Floyd languages is infinite-state model-checking.
Model checking is a classical powerful technique aimed at the automatic or semiautomatic verification of systems. It is based on a model of the system to be verified, expressed in an operational formalism, and a specification of requirements for the system that should be guaranteed to hold. In order to check the assessment of the wished properties, it compares the available model with the desired specification, given by temporal logic formulas, in an algorithmic way. Indeed, even if it is not possible to guarantee automatically absence of errors in system design, model checking has proved to be a valuable effective approach for verification [7]. Traditional model checking techniques referred only to finite-state systems for decidability reasons, and were mainly aimed at improving reliability of hardware devices, which are naturally described by finite state models with a limited number of states. Recently, however, research began to deal with the study of verification approaches also for systems with an infinite number of states, extending model checking scope of applicability to new domains. In fact, many software systems are more naturally defined resorting to infinite-state models, especially when unbounded data structures or parameters from infinite domains are involved ([10]). A typical scenario, where models able to represent only a finite number of configurations lack the necessary expressive power, is given by domains where the interaction between the components of a system is specified by means of distributed protocols that may support an infinite number of instances of processes, or by communication protocols for unbounded channels.
Past literature has addressed the subject of infinite-state model checking looking for suitable models for the verification of classes of infinite-state systems which admit a decidable model checking problem, and several formalism have been proposed in past years, such as timed automata [3] counter machines [15] and logical formalisms as fragments of LTL with Presburger constraints [6].

Among the traditional formal methods adopted for verification of systems, Floyd languages appear to be a new attractive field of research, distinguishing from the classical approaches by some peculiarities that lead to see this class of languages as a promising formalism for infinite-state model checking.

For instance, FLs naturally support the introduction of parallel algorithms for parsing, and this distinctive feature can be exploited to design an efficient model checker based on this class of languages. Moreover, as regards expressiveness adequacy, the

relevant properties FLs enjoy (they are closed under Boolean operations, product, reversal, suffix/prefix extraction and Kleene star [8]) are shared also by VPLs, that model infinite-state pushdown systems too, but FAs are strictly more expressive in terms of recognition power, allowing to model more complex and practical phenomena.

Within this context, the investigation of the theoretical properties of ω-Floyd languages acquires a decisive role: indeed, the characterization of ω-FLs in terms of a suitable class of abstract machines and the analysis of their distinguishing features represent the first necessary step towards further research on applications of FLs to model-checking, extending the classical results of ω-language and automata theory in this domain.

1.2 Thesis objectives and structure

This thesis stems from this field of research on ω-language theory, with the aim of investigating Floyd languages and generalizing them to languages of infinite words. Indeed, as FLs are an open field of study, properties of Floyd ω-languages have not been analyzed yet. Thus, the main issue of this thesis is the characterization of a suitable operational formalism to define these languages, along with the investigation of the peculiar aspects this class of languages is endowed with.

These themes are presented as follows.
Chapter 2 reminds the fundamental results on Floyd languages of finite words, defining them in terms of grammars and automata.
Chapter 3 outlines Floyd ω-languages and the various acceptance modes for automata on infinite strings.
Chapter 4 shows how the introduction of determinsim or nondeterminism in the moves of the automata may influence the recognizability power of these formalisms.
Chapter 5 presents closure properties Floyd ω-languages enjoy.
Finally, Chapter 6 draws some conclusions.

Chapter 2

Floyd languages of finite words

2.1 State of the art: Floyd languages across the years

Floyd languages (FL) made their first appearance on the scene of formal language science in 1963, when Robert Floyd published a paper on operator precedence grammars, a restricted class of deterministic context-free grammars aimed at deterministic (and thus efficient) parsing of programming languages [12]. The introduction of this family of languages followed the traditional attempt of formal language theory to devise adequate models for parsing, that conjugate practical and valuable properties and expressive power. However, though R.Floyd claimed the benefits such grammars may provide and he outlined several examples of use of this formalism, as a grammar for a programming language ALGOL-like, the path of research of classical theory was traced along different directions and Floyd grammars (FG) were neglected for decades.

The stated objective of investigation of formal language science was the search for languages, mainly context-free ones, enjoying the closure and decidability properties of regular languages but exhibiting more expressiveness and generality for numerous applications. Some time after Floyd's results, a first candidate was found in McNaughton's parenthesis grammars [18]. The peculiar feature of this class of grammars is that the right-hand side of the rules is enclosed within a pair of parentheses; the alphabet is the disjoint union of internal characters and the pair. By considering instead of strings the stencil (or skeletal) trees encoded by parenthesized strings, some typical properties of regular languages, that context-free languages do not enjoy, continue to hold, such as uniqueness of the minimal grammar and Boolean closure within the class of languages having the same rule stencils.

Some years later, in [9], Floyd's operator precedence grammars were investigated again, and their closure properties were analyzed. Indeed, FG suggest an elegant alternative to the parenthesis structure of the rules in McNaughton's grammars. Instead of delimiting each right-hand side with begin and end markers, parenthesis generators are introduced. Generators implicitly stand for the set of necessary parentheses, and they embody precedence relations between the terminal symbols or operators that appear in the rules of a FG.

A straightforward description of the parenthesis structure induced by the rules of a FG is presented in [13] referring to arithmetic expressions, a classical construct to which the notion of precedence or hierarchy between operators perfectly fits. Parsing of arithmetic expressions, as for instance #2 + 5 × 6 × 7#, is trivial whenever the expression

is fully parenthesized, that is when each segment composed of an operator with its operands is surrounded by parentheses. Thus, the previous example will be rewritten as #(2 + ((5 × 6) × 7))#; a parser will scan the word until the first closing parenthesis, reducing the segment enclosed by this last parenthesis and its previous matching open one, and will repeat the parsing in the same way for the subsequent part of the string. The same procedure is applied for parentheses languages generated by McNaugthon's grammars.
Still, common grammars do not have per se markers which delimit the right hand side (r.h.s.) of the rules. A possible solution would insert a proper parenthesis between each pair of operators, or more precisely the proper set of parentheses between them, as in #(n, +(($n\times$, ×)n × and n))# for the example above, where n is any number. This is obtained with parenthesis generator written as $\lessdot, \gtrdot, \doteq$ for open, closing and matching parentheses respectively. These generators express the precedence relations between operators: a relation as $+ \lessdot \times$ formalizes the fact that + has a lower precedence than ×. More generally, the rules of an operator grammar implicitly determine three binary precedence relations ($\lessdot, \gtrdot, \doteq$) over the alphabet, and these relations can be presented in a matrix. For the previous example the precedence matrix is the following one:

	+	×	#
+	$\gtrdot$	$\lessdot$	$\gtrdot$
×	$\gtrdot$	$\gtrdot$	$\gtrdot$
#	$\lessdot$	$\lessdot$	$\doteq$

where numbers n have been neglected.
The precedence matrix, even in the absence of the rules, fully determines the topology of the syntax tree, for any word that is produced by any FG with the same precedence matrix.
However, even this time, research on the interesting properties of FG was suspended again, and this class of grammars was overshadowed by the introduction of more general parsers as LR and LALR parsers.

Decades later, novel interest for parentheses-like languages arose from research on mark-up languages such as XML and HTML, and produced the family of balanced grammars and languages (BALAN) [4]. They generalize the parenthesis grammars along two directions: several pairs of parentheses are allowed, and the right-hand side of the grammar rules permit a regular expression over nonterminal and internal symbols to occur between matching parentheses. Uniqueness of the minimal grammar and closure w.r.t. Boolean operations hold for this family too. Furthermore, balanced as well as parenthesis languages are closed under reversal. Model checking and static program analysis provide a different motivation for such families of languages, that extend the typical finite-state properties to infinite-state pushdown systems.

A fundamental step in research on classes of languages that can model hierarchical data structures or semistructured documents has been made by [1] and [2]: these seminal papers introduced a novel subclass of deterministic context-free languages, Visibly pushdown languages (VPLs), that extend BALAN in important ways for software verification, program analysis and data querying. VPLs provide a natural model for symbolic program execution, as they may properly represent the control flow of sequential computations in typical programming languages with nested, and even recursive, invocations of program modules, allowing algorithmic verification for properties of structured programs that could not be specified in traditional temporal logics if they had been expressed in general context-free languages of words. Moreover, linguistic research and document processing are other motivating areas of this class of languages,

since they rely on a representation of words with linear and hierarchical structure.
VPLs support these fields of research thanks to the particular dual nature they assign to words: VPLs are defined as languages of nested words, which consist of sequences of letters labeling paths of linearly ordered positions, as in classical regular languages, but equipped with additional matching relations between pairs of symbols (nested edges). Intuitively, nested edges model occurrences of matching opening and closing scopes, as for calls and returns of procedures and methods.

VPLs have been characterized both in terms of a subclass of context-free grammars and a particular class of automata, named visibly pushdown automata (VPAs), which resemble classical pushdown automata but differ from them as regards the fact that operations on the stack are input-driven. For every VPL, the input alphabet is partitioned into call, return and internal symbols, and the letters of the input word determine the corresponding transitions of the automaton, i.e. respectively push, pop and neutral moves (which leave the stack unchanged).
Furthermore, the class of languages accepted by VPAs enjoy all the properties exhibited by regular languages: for each partitioned alphabet, the corresponding language family is closed under Boolean operations, concatenation and its transitive closure, and also under reversal (by switching calls and returns alphabets).

Along the field of research marked by VPLs, investigation of FGs started again, in the recent work [8], and rather surprisingly, it turned out that FGs show an unexpected connection with BALAN and VPLs. Actually, VPLs is a proper subclass of FLs, determined by a fixed partitioning of the precedence matrix induced by the alphabetic partition into calls, returns and internals. A FG can be algorithmically derived from a VPA and conversely a VPA can be obtained by a FG whose OPM satisfies the restriction on the precedence relations of the partitioned alphabet.

Moreover, as regards expressiveness and structural adequacy, FG extend VPLs in several ways. First, FLs allow for more general precedence relations among terminal symbols than VPLs, and this property has an impact on the processing of markup languages as XML and HTML. In VPLs the call, return and internal alphabets are disjoint, whereas in FLs the same letter can play a different role in different contexts, as a call or as a return, leading to a more flexible treatment of tags. Second, FL, but not VPL, can handle hierarchical structures such as multi-level lists, which is a common paradigm occurring in several contexts, from mark-up languages to arithmetic expressions over precedence ordered operators. FLs assign to words a more adequate syntactic structure than VPLs, revealing their deep semantics, even when it is not explicitly emphasized with tags (calls or returns). Furthermore, as VPLs, FLs enjoy closure under all traditional language operations(Boolean, concatenation, Kleen star, reversal, suffix/prefix), and hence turned out to be the largest known class of deterministic context-free languages that exhibit all properties of practical interest formal language theory has always been looking for.

Finally, a recent work [17] characterized FLs also in terms of abstract machines, by means of a class of stack-based automata, named Floyd Automata (FAs) and this result naturally leads to a further interesting investigation of FLs in the field of omega-languages, and this is exactly the subject of this work of thesis.

2.2 Floyd grammars and languages

This section recalls important definitions and the basic properties of FG and FL in the finite field, that will be the basis for characterization of Floyd languages as sets of

infinite words. All the following definitions are presented in [8].

Let Σ be an alphabet. The empty string is denoted ε. A *context-free* (CF) grammar is a 4-tuple $G = (N, \Sigma, P, S)$, where N is the nonterminal alphabet, P the rule (or production) set, and S the axiom. The *total* alphabet is $V = N \cup \Sigma$. An *empty rule* has ε as the right hand side (r.h.s.). A *renaming rule* has one nonterminal as r.h.s. A grammar is *reduced* if every rule can be used to generate some string in Σ^*. It is *invertible* if no two rules have identical r.h.s.

The following naming convention will be adopted, unless otherwise specified: lowercase Latin letters $a, b, \ldots$ denote terminal characters; uppercase Latin letters $A, B, \ldots$ denote nonterminal characters; letters $u, v, \ldots$ denote terminal strings; and Greek letters $\alpha, \ldots, \omega$ denote strings over $\Sigma \cup N$. The strings may be empty, unless stated otherwise. The *stencil* of a rule $A \to u_0 A_1 u_1 \ldots u_{k-1} A_k u_k$, $k \geq 0$, is $R \to u_0 R u_1 \ldots u_{k-1} R u_k$, where R is not in N.

A rule is in *operator form* if its r.h.s has no adjacent nonterminals; an *operator grammar* (OG) contains just such rules. Any CF grammar admits an equivalent OG, which can be also assumed to be invertible [14, 22].

For a CF grammar G over Σ, the associated *parenthesis grammar* [18] $\widetilde{G}$ has the rules obtained by enclosing each right part of a rule of G within the parentheses '[' and ']' that are assumed not to be in Σ.

Two grammars G, G' are *equivalent* if they generate the same language, i.e, $L(G) = L(G')$. They are *structurally equivalent* if in addition the corresponding parenthesis grammars are equivalent, i.e, $L(\widetilde{G}) = L(\widetilde{G'})$.

For an OG G and a nonterminal A, the *left and right terminal sets* are

$$\mathcal{L}_G(A) = \{a \in \Sigma \mid A \overset{*}{\Rightarrow} Ba\alpha\} \qquad \mathcal{R}_G(A) = \{a \in \Sigma \mid A \overset{*}{\Rightarrow} \alpha a B\}$$

where $B \in N \cup \{\varepsilon\}$. The grammar name G will be omitted unless necessary.

As mentioned above, R. Floyd formalized the notion of precedence relations between operators as it occurs in a wide-spread construct as arithmetic expressions, and introduced a new class of languages, with the peculiar property that the shape of the derivation tree is solely determined by a binary relation between terminals that are consecutive, or become consecutive after a bottom-up reduction step. The precedence relations can be formally defined as follows.

For an OG G, let α, β range over $(N \cup \Sigma)^*$ and $a, b \in \Sigma$. Three binary operator precedence (OP) relations are defined:

$$\begin{aligned}
\text{equal in precedence:} \quad & a \doteq b \iff \exists A \to \alpha a B b \beta, B \in N \cup \{\varepsilon\} \\
\text{takes precedence:} \quad & a \gtrdot b \iff \exists A \to \alpha D b \beta, D \in N \text{ and } a \in \mathcal{R}_G(D) \\
\text{yields precedence:} \quad & a \lessdot b \iff \exists A \to \alpha a D \beta, D \in N \text{ and } b \in \mathcal{L}_G(D)
\end{aligned} \tag{2.1}$$

For an OG G, the *operator precedence matrix* (OPM) $M = OPM(G)$ is a $|\Sigma| \times |\Sigma|$ array that with each ordered pair (a, b) associates the set M_{ab} of OP relations holding between a and b. Between two OPMs M_1 and M_2, define set inclusion and operations.

$$\begin{aligned}
M_1 \subseteq M_2 \text{ if } \forall a, b : M_{1,ab} \subseteq M_{2,ab}, \\
M = M_1 \cup M_2 \text{ if } \forall a, b : M_{ab} = M_{1,ab} \cup M_{2,ab} \\
M = M_1 \cap M_2 \text{ if } \forall a, b : M_{ab} = M_{1,ab} \cap M_{2,ab}
\end{aligned}$$

Definition 2.1 *G is an* operator precedence *or* Floyd grammar *(FG) if, and only if, $M = OPM(G)$ is a conflict-free matrix, i.e., $\forall a, b$, $|M_{ab}| \leq 1$. Two matrices are compatible if their union is conflict-free. A matrix is total if it contains no empty case.*

Example 2.1 *Arithmetic expressions with prioritized operators, introduced in Section 2.1, are here presented in a variant that includes variables or constants (denoted n) as terminal symbols, and as before is devoid of parentheses. A FG for this construct, together with its OPM, is listed below.*

$$S \to E,$$
$$E \to E + T \mid T \times n \mid n,$$
$$T \to T \times n \mid n$$

	n	$+$	$\times$
n		$\gtrdot$	$\gtrdot$
$+$	$\lessdot$	$\gtrdot$	$\lessdot$
$\times$	$\doteq$		

Notice that the precedence relations defined in the OPM alone completely determine the skeleton of the syntax tree of a word and directly drive its parsing.
Indeed, if a word belongs to the language generated by a FG with a given precedence matrix, then its derivation tree must be compatible with the precedence relations included in the OPM.
Consider, as an example, the word $n \times n + n$, whose syntax tree is depicted in the figure below (Figure 2.1), which clearly emphasizes how the precedence relations between the symbols affect the shape of the tree.

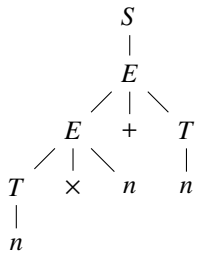


Figure 2.1: Derivation tree for the string $n \times n + n$.

Operator-precedence parsing of a word is fully driven by the relations defined in the OPM, and this peculiarity of operator-precedence grammars and languages is evident considering how a skeleton syntax tree is built while reading the same word as above (Figure 2.2).

Stack		Rest of input
$\#$	,	$n \times n + n \,\#$
$\# \underbrace{\lessdot n \gtrdot}$	,	$\times n + n \,\#$
$\# \underbrace{\lessdot n \gtrdot \times \doteq n \gtrdot}$	,	$+ n \,\#$
$\# \underbrace{\lessdot n \gtrdot \times \doteq n \gtrdot} + \underbrace{\lessdot n \gtrdot}$	,	$\#$
$\# \underbrace{\underbrace{\lessdot n \gtrdot \times \doteq n \gtrdot} + \underbrace{\lessdot n \gtrdot}} \#$		

Figure 2.2: Operator-precedence parsing for the string $n \times n + n$.

An important parameter of a FG is the length of the right hand sides of the rules, which is connected to the equal in precedence relations of a FG alphabet. A rule $A \to A_1 a_1 \dots A_t a_t A_{t+1}$, where each A_i is a possibly missing nonterminal, is associated with relations $a_1 \doteq a_2 \doteq \dots \doteq a_t$. If the $\doteq$ relation is cyclic, there is no finite bound on the length of the r.h.s of a production. Otherwise the length is bounded by $2 \cdot c + 1$, where $c \geq 1$ is the length of the longest $\doteq$-chain.

Next, a normal form and some important families of FGs (precedence-compatible grammars and grammars with bounded right-hand sides) will be defined.

Definition 2.2 *Normal form of FGs*
A FG is in Fischer normal form [11] if it is invertible, the axiom S does not occur in the right hand side (r.h.s.) of any rule, no empty rule exists except possibly $S \to \varepsilon$, the other rules having S as left hand side (l.h.s.) are renaming, and no other renaming rules exist.

Definition 2.3 *Precedence-compatible grammars*
For a precedence matrix M, the class [9] C_M of precedence-compatible grammars FG is

$$C_M = \{G \in FG \mid OPM(G) \subseteq M\}$$

Definition 2.4 *Right-bounded grammars*
The class $C_{M,k}$ of FGs with right bound $k \geq 1$is defined as

$$C_{M,k} = \{G \mid G \in C_M \wedge (\forall \text{ rule } A \longrightarrow \alpha \text{ of } G, |\alpha| \leq k)\}$$

The class of $\doteq$-acyclic FGs is defined as

$$C_{M,\doteq} = \{G \mid G \in C_M : \text{matrix } M \text{ is } \doteq\text{-acyclic}\}$$

A class of FGs is *right bounded* if it is k-right-bounded for some k. The class of $\doteq$-acyclic FGs is obviously right bounded.

Closure properties FLs enjoy the same closure properties as VPLs. The following statements are introduced and proved in [9] and [8].

Theorem 2.1 *(Corol. 5.7 and Theor. 5.8), [9]*
For every precedence matrix M, the class of FLs

$$\{L(G) \mid G \in C_{M,k}\}$$

is a Boolean algebra.

Hence, the proposition holds for languages generated by right-bounded FGs having precedence matrices that are included in, or equal to some matrix M. The top element of the Boolean lattice is the language of the FG that generates all possible syntax trees compatible with the precedence matrix; in particular, if M is total, the top element is Σ^*.

Theorem 2.2 *Let G_1, G_2 be FGs such that $OPM(G_1)$ is compatible with $OPM(G_2)$. Then a FG grammar G can be effectively constructed such that*

$$L(G) = L(G_1)L(G_2) \text{ and } OPM(G) \supseteq OPM(G_1) \cup OPM(G_2).$$

Closure w.r.t Kleene star holds, likewise, for $\doteq$-acyclic FGs.

2.3 Floyd automata

As it is usual in the tradition of formal language science, FLs have been also defined by an ad-hoc family of automata, named in memory of Robert Floyd, and presented in the recent paper [17]. FAs are pushdown automata based on the classical deterministic parsing algorithms for FLs (actually, this section introduces nondeterministic FAs, but Chapter 4 will prove the equivalence between deterministic and nondeterministic FAs). FAs blend some typical features of traditional parsers as LR with characteristic aspects of VPA: indeed operations on the stack reflect the shift and reduce moves of classical bottom-up parsers, and at the same time they are input-driven, as it happens for VPA, where the symbols of the partitioned alphabet determine the type of move to perform (push, pop or neutral move).

The class of languages recognized by FAs corresponds exactly to the family generated by FGs, and the equivalence between these two formalisms is effective: any language produced by a FG can be accepted by a FA such that parsing of words by the automaton is completely determined by the precedence relations between operators defined in the OPM of the grammar, and conversely. More formally, given the strict relation between automata and grammars, to each automaton an *operator precedence alphabet* is assigned, as the following definition makes more precise.
Incidentally, notice that a special symbol #, which does not belong to the alphabet Σ, is used to mark the beginning and the end of any string. This is consistent with the typical operator parsing technique that requires the lookback and lookahead of one character to determine the precedence relation [13]. The precedence relation in the OPM are extended to include #.

Definition 2.5 *An* operator precedence alphabet *is a pair* (Σ, M) *where* Σ *is an alphabet and* M *is a conflict-free operator precedence matrix, i.e. a* $|\Sigma \cup \{\#\}|^2$ *array that with each ordered pair* (a, b) *associates at most one of the operator precedence relations:* $\doteq$, $<$ *or* $>$.

This section reminds basic definitions of Floyd automata which recognize languages of finite words as they are defined in [17].

Definition 2.6 *A nondeterministic* precedence automaton *(or Floyd automaton) is given by a tuple:* $\mathcal{A} = \langle \Sigma, M, Q, I, F, \delta \rangle$ *where:*

- (Σ, M) *is a precedence alphabet,*
- Q *is a set of states (disjoint from* Σ*),*
- $I \subseteq Q$ *is a set of initial states,*
- $F \subseteq Q$ *is a set of final states,*
- $\delta : Q \times (\Sigma \cup Q) \to 2^Q$ *is the transition function.*

The transition function can be seen as the union of two disjoint functions:

$$\delta_{\text{push}} : Q \times \Sigma \to 2^Q \qquad \delta_{\text{flush}} : Q \times Q \to 2^Q$$

A nondeterministic precedence automaton can be represented by a graph with Q as the set of vertices and $\Sigma \cup Q$ as the set of edges: there is an edge from state q to state p labeled by $a \in \Sigma$ if and only if $p \in \delta_{push}(q, a)$ and there is an edge from state q to state

p labeled by $r \in Q$ if and only if $p \in \delta_{flush}(q, r)$. Flush transitions are denoted by a double arrow, whereas push transitions by a simple arrow.

The semantics of the automaton is defined according to the following notations. Let letters $p, q, p_i, q_i, \ldots$ denote states in Q and set $\Sigma' = \{a' \mid a \in \Sigma\}$; symbols in Σ' are called *marked* symbols. Let $\Gamma = (\Sigma \cup \Sigma' \cup \{\#\}) \times Q$; where symbols in Γ are denoted as $[a\ q]$, $[a'\ q]$, or $[\#\ q]$, respectively. Set $symbol([a\ q]) = symbol([a'\ q]) = a$, $symbol([\#\ q]) = \#$, and $state([a\ q]) = state([a'\ q]) = state([\#\ q]) = q$. Given a string $\beta = B_1 B_2 \ldots B_n$ with $q \in Q$ and $B_i \in \Gamma$, set $state(\beta) = state(B_n)$ and $symbol(\beta) = symbol(B_n)$.

A *configuration* is a pair $C = \langle \beta\ ,\ w \rangle$, where $\beta = B_1 B_2 \ldots B_n \in \Gamma^*$, $symbol(B_1) = \#$, and $w = a_1 a_2 \ldots a_m \in \Sigma^* \#$. A configuration represents both the contents β of the stack and the part of input w still to process. Set also $top(C) = symbol(B_n)$ and $input(C) = a_1$.

A computation of the automaton is a finite sequence of moves $C \vdash C_1$; there are three kinds of moves, depending on the precedence relation between $top(C)$ and $input(C)$:

push move: if $top(C) \doteq input(C)$ then $\langle \beta\ ,\ aw \rangle \vdash \langle \beta[a\ \delta_{push}(state(\beta), a)]\ ,\ w \rangle$

mark move: if $top(C) \lessdot input(C)$ then $\langle \beta\ ,\ aw \rangle \vdash \langle \beta[a'\ \delta_{push}(state(\beta), a)]\ ,\ w \rangle$

flush move: if $top(C) \gtrdot input(C)$ then let $\beta = B_1 B_2 \ldots B_n$ with $B_j = [x_j\ q_j]$, $x_j \in \Sigma \cup \Sigma'$ and let i the greatest index such that B_i belongs to $\Sigma' \times Q$. Then $\langle \beta\ ,\ aw \rangle \vdash \langle B_1 B_2 \ldots B_{i-2}[x_{i-1}\ \delta_{flush}(q_n, q_{i-1})]\ ,\ aw \rangle$.

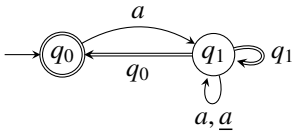
Push and mark moves both push the input symbol on the top of the stack, together with the new state computed by δ_{push}; such moves differ only in the marking of the symbol on top of the stack. The flush move is more complex: the symbols on the top of the stack are removed until the first marked symbol (*included*), and the state of the next symbol below them in the stack is updated by δ_{flush} according to the pair of states that delimit the portion of the stack to be removed; notice that in this move the input symbol is not relevant and it remains available for the following move.

Finally, a configuration $[\#\ q_I]$ is *starting* if $q_I \in I$ and a configuration $[\#\ q_F]$ is *accepting* if $q_F \in F$. The language accepted by the automaton is defined as:

$$L(\mathcal{A}) = \left\{ x \mid \langle [\#\ q_I]\ ,\ x\# \rangle \overset{*}{\vdash} \langle [\#\ q_F]\ ,\ \# \rangle, q_I \in I, q_F \in F \right\}.$$

Example 2.2 *As an example of FA, the automaton depicted in the following figure accepts the Dyck language $\mathcal{L}_{Dyck}$ with the pairs $a, \underline{a}$.*

	a	$\underline{a}$	$\#$
a	$\lessdot$	$\doteq$	
$\underline{a}$	$\lessdot$	$\gtrdot$	$\gtrdot$
$\#$	$\lessdot$		$\doteq$



An example of computation on input $a\underline{a}aa\underline{a}a$ is shown below (Figure 2.3).

	$\langle[\# \, q_0]$	,	$a\underline{a}aa\underline{aa}\#\rangle$
mark	$\langle[\# \, q_0][a' \, q_1]$	,	$\underline{a}aa\underline{aa}\#\rangle$
push	$\langle[\# \, q_0][a' \, q_1][\underline{a} \, q_1]$	,	$aa\underline{aa}\#\rangle$
mark	$\langle[\# \, q_0][a' \, q_1][\underline{a} \, q_1][a' \, q_1]$	,	$a\underline{aa}\#\rangle$
mark	$\langle[\# \, q_0][a' \, q_1][\underline{a} \, q_1][a' \, q_1][a' \, q_1]$	,	$\underline{aa}\#\rangle$
push	$\langle[\# \, q_0][a' \, q_1][\underline{a} \, q_1][a' \, q_1][a' \, q_1][\underline{a} \, q_1]$	,	$\underline{a}\#\rangle$
flush	$\langle[\# \, q_0][a' \, q_1][\underline{a} \, q_1][a' \, q_1]$	,	$\underline{a}\#\rangle$
push	$\langle[\# \, q_0][a' \, q_1][\underline{a} \, q_1][a' \, q_1][\underline{a} \, q_1]$	,	$\#\rangle$
flush	$\langle[\# \, q_0][a' \, q_1][\underline{a} \, q_1]$	,	$\#\rangle$
flush	$\langle[\# \, q_0]$	,	$\#\rangle$

Figure 2.3: Example of computation for language L_{Dyck} of Example 2.2.

Given that the class of FLs has been formalized by means of a complementary family of automata, the natural continuation of the path of research on Floyd formalisms is the generalization of finite models to the infinite scenario, where endless words replace sequence of symbols of finite length, and this will be the subject matter of the next chapters.

Chapter 3

Floyd ω-languages

Floyd automata can be generalized to ω-automata, which recognize Floyd languages comprising words of infinite length.
Traditionally, ω-automata have been classified on the basis of the acceptance condition of infinite words they are equipped with, and this classification suits Floyd ω-automata too. All acceptance conditions refer to the occurrences of states which are visited in a computation of the automaton, and they generally impose constraints on those states that are encountered infinitely (or also finitely) often during a run. The earliest definition for automata recognizing infinite-length words dates back to the work of Büchi [5], while several other acceptance conditions (Muller [19] , Rabin [21], Streett [23]) have been subsequently introduced in order to express more and more powerful recognition modes. All these notions of acceptance can be naturally adapted to ω-automata for Floyd languages and can be characterized according to a peculiar acceptance component of the automaton on ω-words. More formally,

Definition 3.1 *A nondeterministic ω-Floyd automaton (ω-FA) is given by a tuple: $\mathcal{A} = \langle \Sigma, M, Q, I, \mathcal{F}, \delta \rangle$ where:*

- *(Σ, M) is a pair which defines a ω-precedence alphabet, where Σ is an alphabet and M is a conflict-free operator precedence matrix, i.e. a $|\Sigma \cup \{\#\}| \times |\Sigma|$ array that with each ordered pair (a, b) associates at most one of the operator precedence relations: $\doteq$, $\lessdot$ or $\gtrdot$.*
- *Q is a set of states (disjoint from Σ),*
- *$I \subseteq Q$ is a set of initial states,*
- *$\mathcal{F}$ is an acceptance component, distinctive of the class (Büchi, Muller,...) the automaton belongs to,*
- *$\delta : Q \times (\Sigma \cup Q) \to 2^Q$ is the transition function.*

The transition function, as in finite FAs, can be seen as the union of two disjoint functions:

$$\delta_{\text{push}} : Q \times \Sigma \to 2^Q \qquad \delta_{\text{flush}} : Q \times Q \to 2^Q$$

The semantics of the automaton can be expressed adopting notations as for finite FAs. Let $\Sigma' = \{a' \mid a \in \Sigma\}$ be the set of *marked* symbols. The set Γ of stack symbols is defined as for FAs on finite words.

A *configuration* is a pair $C = \langle \beta\ ,\ w \rangle$, where $\beta = B_1 B_2 \ldots B_n \in \Gamma^*$, $q \in Q$, $symbol(B_1) =$ # and $w = a_1 a_2 \ldots \in \Sigma^\omega$. A configuration represents both the contents β of the stack and the (infinite) part of input w still to process. Set also $top(C) = symbol(B_n)$ and $input(C) = a_1$.

An *infinite computation* (run) of the automaton on an infinite word $x \in \Sigma^\omega$ is an ω-sequence of configurations $\mathcal{S} = \langle \beta_0\ ,\ x_0 \rangle \langle \beta_1\ ,\ x_1 \rangle \ldots$, such that $\langle \beta_0\ ,\ x_0 \rangle = \langle [\# q_I]\ ,\ x \rangle$, $q_I \in I$ and $\langle \beta_i\ ,\ x_i \rangle \vdash \langle \beta_{i+1}\ ,\ x_{i+1} \rangle$ for $i \geq 0$ is a move which is defined as for finite FAs and belongs to one of three types (*push move, mark move, flush move*).

A run is said to be *successful* if it satisfies the acceptance condition $\mathcal{F}$, which is peculiar for each recognizing mode.
$\mathcal{A}$ *accepts* $x \in \Sigma^\omega$ iff there is a successful run of $\mathcal{A}$ on x. Furthermore, let

$$L(\mathcal{A}) = \{x \in \Sigma^\omega \mid \mathcal{A} \text{ accepts x}\}$$

be the ω-language *recognized* by $\mathcal{A}$.

This general definition of a ω-FA may be further refined, according to the classical notions of acceptance for infinite words. The first main mode is named after Büchi.

3.1 Büchi condition

Definition 3.2 (Büchi acceptance condition) *A nondeterministic Büchi automaton which recognizes Floyd languages (BFA) is an ω-FA $\langle \Sigma, M, Q, I, \mathcal{F}, \delta \rangle$ whose acceptance component is a set of final states: $\mathcal{F} = F \subseteq Q$.*

There are several possibilities to define acceptance of an ω-word, according to classical Büchi condition. The main notions of acceptance are the following ones:

Definition 3.3 (acceptance by empty stack and final state (E)) *A BFA automaton $\mathcal{A}_E = \langle \Sigma, M, Q, I, F, \delta \rangle$ which recognizes infinite words by empty stack and final state is said to have a* successful *run $\mathcal{S}$ on an infinite word $x \in \Sigma^\omega$ iff there exists a set $\{q_F \mid q_F \in F\}$ such that configurations with stack $[\# q_F]$ occur infinitely often in $\mathcal{S}$.*

Definition 3.4 (acceptance by final state (F)) *A BFA automaton $\mathcal{A}_F = \langle \Sigma, M, Q, I, F, \delta \rangle$ which recognizes infinite words by final state is said to have a* successful *run $\mathcal{S}$ on an infinite word $x \in \Sigma^\omega$ iff configurations $\langle \beta_i\ ,\ x_i \rangle$ with $state(\beta_i) \in F$ occur infinitely often in $\mathcal{S}$.*

These acceptance conditions are closely related, as the following statement proves.

Theorem 3.1 *For any nondeterministic BFA automaton $\mathcal{A}$, the acceptance condition by empty stack and final state is strictly less expressive than acceptance by final state only.*

The following lemma shows the first part of the result:

Lemma 3.2 *Any ω-language recognizable by a nondeterministic BFA $\mathcal{A}$ with E acceptance condition can be recognized by a nondeterministic BFA $\tilde{\mathcal{A}}$ with F acceptance condition.*

Proof Notice that the classical proof of equivalence for nondeterministic CF automata with E and F acceptance conditions does not perfectly fit for FA, since it is no longer possible to check directly whether the stack is empty or not by looking for the bottom-stack symbol. It is necessary to keep in the states information on the evolution of the stack: in the following proof, tagged (untagged) states are those ones reached with empty (non-empty) stack respectively.
Consider a nondeterministic BFA $\mathcal{A} = \langle \Sigma, M, Q, I, F, \delta \rangle$ with E acceptance condition, which recognizes the language $L(\mathcal{A})$. Consider then a nondeterministic BFA $\tilde{\mathcal{A}} = \langle \Sigma, M, \tilde{Q}, \tilde{I}, \tilde{F}, \tilde{\delta} \rangle$ with F acceptance condition and whose precedence matrix M is the same as the one associated with $\mathcal{A}$. Let

- $\tilde{Q} = Q \cup \tilde{I} \cup R$, where $\tilde{I} = \{q'_i \mid q_i \in I\}$ is the set of initial states of $\tilde{\mathcal{A}}$ and its elements are defined so as no push/mark transition ends in them. $R = \{q' \mid q \in Q \text{ and } \exists p_1, p_2 \in Q : \delta_{\text{flush}}(p_1, p_2) \ni q\}$ is the set of tagged states q' whose untagged counterpart q belongs to Q and is reached by a flush transition in $\mathcal{A}$. Let also $c : R \to Q$ be a function that assigns to each tagged state its untagged counterpart (it can also be naturally extended to be defined on a set of states in R),
- $\tilde{F} = \{q' \in R \mid c(q') \in F\}$,
- $\tilde{\delta} : \tilde{Q} \times (\Sigma \cup \tilde{Q}) \to 2^{\tilde{Q}}$ is the transition function defined as follows. The push transition $\tilde{\delta}_{\text{push}} : \tilde{Q} \times \Sigma \to 2^{\tilde{Q}}$ is defined by:
  - $\tilde{\delta}_{\text{push}}(q, a) = \delta_{\text{push}}(q, a) \quad \forall q \in Q, a \in \Sigma$
  - $\forall q_i \in I, a \in \Sigma, s \in Q$ such that $\delta_{\text{push}}(q_i, a) \ni s$, then $\tilde{\delta}_{\text{push}}(q'_i, a) \ni s$,
  - $\forall q \in c(R), a \in \Sigma, s \in Q$ such that $\delta_{\text{push}}(q, a) \ni s$, then $\tilde{\delta}_{\text{push}}(q', a) \ni s$
  - and no other transition is defined.

  The flush transition $\tilde{\delta}_{\text{flush}} : \tilde{Q} \times \tilde{Q} \to 2^{\tilde{Q}}$ is defined by:
  - $\tilde{\delta}_{\text{flush}}(q, r) = \delta_{\text{flush}}(q, r) \quad \forall q, r \in Q$
  - $\forall q, r \in Q, q_i \in I$ such that $\delta_{\text{flush}}(q, q_i) \ni r$, then $\tilde{\delta}_{\text{flush}}(q, q'_i) \ni r'$ where $r = c(r')$
  - $\forall q, s \in Q, r \in c(R)$ such that $\delta_{\text{flush}}(q, r) \ni s$, then $\tilde{\delta}_{\text{flush}}(q, r') \ni s'$ where $s = c(s')$
  - and no other transition is defined.

The automata $\mathcal{A}$ and $\tilde{\mathcal{A}}$ recognize the same language, $L(\mathcal{A}) = L(\tilde{\mathcal{A}})$.
Next, before proving the equivalence holding between the two automata, an example of the construction is given.

Example 3.1 *Consider the nondeterministic BFA E $\mathcal{A}$ which recognizes the language* $L(\mathcal{A}) = c(a(br)^+)^\omega \cup c(ar)^\omega$*, with OPM*

	a	b	c	r
a		$\lessdot$		$\doteq$
b				$\doteq$
c	$\gtrdot$			
r	$\gtrdot$	$\gtrdot$		
#	$\lessdot$		$\lessdot$	

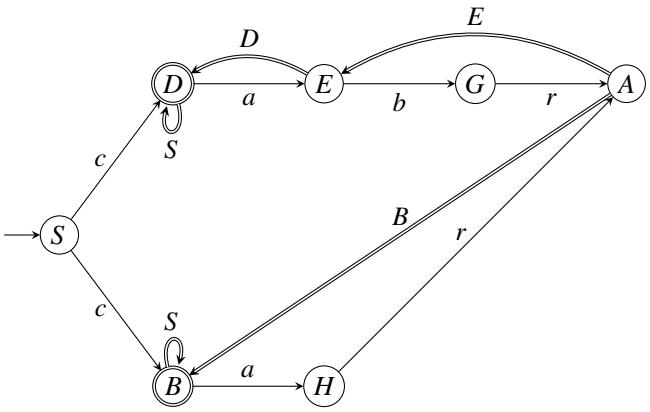


Figure 3.1: ω-automaton $\mathcal{A}$ of Example 3.1.

The BFA E $\mathcal{A} = \langle \Sigma, M, Q, I, F, \delta \rangle$*, with* $\Sigma = \{a, b, c, r\}, I = \{S\}, F = \{B, D\}$*, is depicted in Figure 3.1.*

The BFA F $\tilde{\mathcal{A}} = \langle \Sigma, M, \tilde{Q}, \tilde{I}, \tilde{F}, \tilde{\delta} \rangle$*, with* $\tilde{I} = \{S'\}, \tilde{F} = \{B', D'\}$*, is depicted in Figure 3.2.*

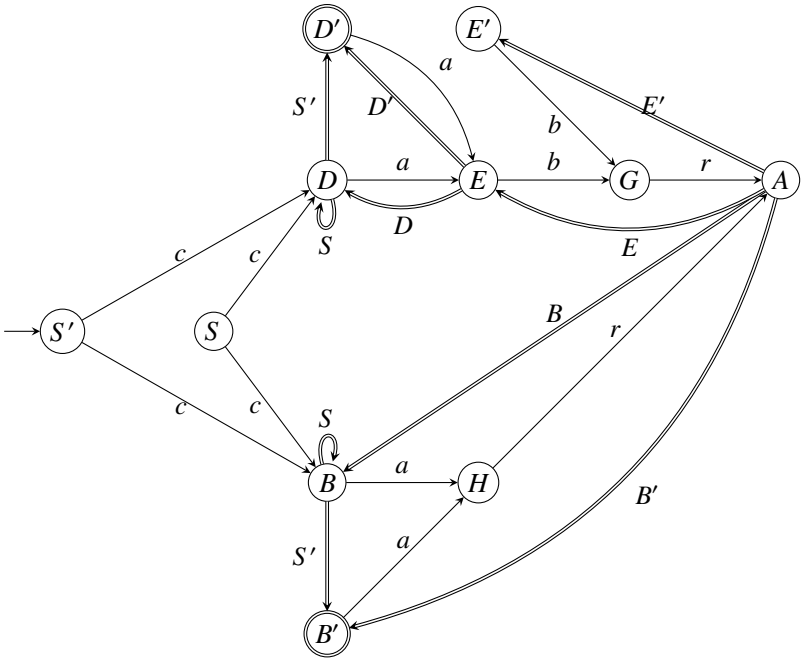


Figure 3.2: ω-automaton $\tilde{\mathcal{A}}$. Notice that $\tilde{\mathcal{A}}$ is not reduced (as regards states S and E').

In order to prove that $L(\mathcal{A}) = L(\tilde{\mathcal{A}})$, consider first of all the direction from left to right of the statement above.
Let $x \in \Sigma^\omega$ be an infinite word recognized by $\mathcal{A}$. Thus $x \in L(\mathcal{A})$ and it can be factorized as $x = w_1 w_2 w_3 \ldots$ where each w_i is the minimal word on Σ^+ that leads the automaton from a configuration with empty stack towards a configuration with empty stack. There exists a successful run in $\mathcal{A}$, $\mathcal{S} = \langle \beta_0 = [\# \, q_0] \, , \, w_1 w_2 w_3 \ldots \rangle \overset{+}{\vdash} \langle \beta_1 = [\# \, q_1] \, , \, w_2 w_3 \ldots \rangle \overset{+}{\vdash} \ldots \overset{+}{\vdash} \langle \beta_i = [\# \, q_i] \, , \, w_{i+1} \ldots \rangle \overset{+}{\vdash} \ldots$ with $q_0 \in I$ and there exist infinitely many $q_i \in F$. Notice that not all q_i reached at the end of a path labeled w_i are necessarily in F.
Furthermore, given $w_1 = c_1 y_1$ (with $c_1 \in \Sigma, y_1 \in \Sigma^*$), if q_0 is the first state encountered at the bottom of the stack and q_1 is the first subsequent state reached with empty stack, then there exists a path

$$q_0 \xrightarrow{c_1} r_1 \overset{y_1}{\rightsquigarrow} r_2 \overset{q_0}{\Longrightarrow} q_1 \tag{3.1}$$

for some $r_1, r_2 \in Q$, and for each $i \geq 1$, given $w_{i+1} = c_{i+1} y_{i+1}$ (with $c_{i+1} \in \Sigma, y_{i+1} \in \Sigma^*$), it holds that

$$q_i \xrightarrow{c_{i+1}} r \overset{y_{i+1}}{\rightsquigarrow} s \overset{q_i}{\Longrightarrow} q_{i+1} \tag{3.2}$$

for some $r, s \in Q$.
Since $\tilde{\mathcal{A}}$ has been built so as there exists a path

$$q_0' \xrightarrow{c_1} r_1 \overset{y_1}{\rightsquigarrow} r_2 \overset{q_0'}{\Longrightarrow} q_1' \tag{3.3}$$

where y_1 labels the same path in $\mathcal{A}$ on (untagged) states in Q, and for each $i \geq 1$

$$q_i' \xrightarrow{c_{i+1}} r \overset{y_{i+1}}{\rightsquigarrow} s \overset{q_i'}{\Longrightarrow} q_{i+1}' \tag{3.4}$$

where y_{i+1} labels the same path in $\mathcal{A}$ on (untagged) states in Q, then $\tilde{\mathcal{A}}$ has a run $\langle \alpha_0 = [\# \, q_0'] \, , \, w_1 w_2 w_3 \ldots \rangle \overset{+}{\vdash} \langle \alpha_1 = [\# \, q_1'] \, , \, w_2 w_3 \ldots \rangle \overset{+}{\vdash} \ldots \overset{+}{\vdash} \langle \alpha_i = [\# \, q_i'] \, , \, w_{i+1} \ldots \rangle \overset{+}{\vdash} \ldots$, where each configuration has empty stack α_i .
Now, considering that there are infinitely many $q_i \in F$ and q_i are reached by flush transitions in $\mathcal{A}$, then there are infinitely many $q_i' \in \tilde{F}$ and $x \in L(\tilde{\mathcal{A}})$.

Conversely, as regards the direction of the proof from right to left, let $x \in \Sigma^\omega$ be an infinite word recognized by $\tilde{\mathcal{A}}$. x can be factorized as $x = w_1 w_2 \ldots$ where each w_i is the minimal word on Σ^+ that leads the automaton $\tilde{\mathcal{A}}$ from a configuration with a tagged state on the top of the stack towards a configuration with a tagged state on the top of the stack. Notice that tagged states must occur infinitely often in a successful run $\tilde{\mathcal{S}}$ of the automaton on x since $\tilde{F}$ consists only of tagged states.
One may argue by induction on the number of configurations with tagged states at the top of the stack occurring in $\tilde{\mathcal{S}}$ that

- $\tilde{\mathcal{S}} = \langle \alpha_0 \, , \, w_1 w_2 w_3 \ldots \rangle \overset{+}{\vdash} \langle \alpha_1 \, , \, w_2 w_3 \ldots \rangle \overset{+}{\vdash} \ldots \overset{+}{\vdash} \langle \alpha_i \, , \, w_{i+1} \ldots \rangle \overset{+}{\vdash} \ldots$ where $\alpha_0 = [\# \, q_0'], q_0' \in \tilde{I}$ and $\forall i \geq 1 \langle \alpha_i \, , \, w_{i+1} \ldots \rangle$ is a configuration with empty stack $\alpha_i = [\# \, q_i']$,

- and there exists a run of $\mathcal{A}$ on x such that $\langle \beta_0 = [\# \, q_0] \, , \, w_1 w_2 w_3 \ldots \rangle \overset{+}{\vdash} \langle \beta_1 = [\# \, q_1] \, , \, w_2 w_3 \ldots \rangle \overset{+}{\vdash} \ldots \overset{+}{\vdash} \langle \beta_i = [\# \, q_i] \, , \, w_{i+1} \ldots \rangle \overset{+}{\vdash} \ldots$ with $q_0 \in I$ and $\forall i \geq 0, q_i = c(q_i')$ is the counterpart of q_i, and each $\langle \beta_i \, , \, w_{i+1} \ldots \rangle$ is with empty stack, too.

Consider the first word $w_1 = c_1 y_1$ (with $c_1 \in \Sigma, y_1 \in \Sigma^*$) of the factorization; then $\langle [\# \, q_0'] \, , \, w_1 w_2 \ldots \rangle \overset{+}{\vdash} \langle \gamma [b \, q_1'] \, , \, w_2 \ldots \rangle$ with $q_0' \in \tilde{I}, \gamma \in \Gamma^*, b \in (\Sigma \cup \{\#\})$ and q_1' is a

tagged state. Each tagged state in $\tilde{Q}$ has no push/mark transitions ending in it, but it can be reached only with flush transitions. Flush transitions to tagged states are labeled only by tagged states and, since by hypothesis the state q'_0 is the only tagged state encountered until q'_1, then it follows that the automaton $\tilde{\mathcal{A}}$ visited a path:

$$q'_0 \xrightarrow{c_1} r_1 \overset{y_1}{\rightsquigarrow} r_2 \overset{q'_0}{\Longrightarrow} q'_1 \tag{3.5}$$

for some $r_1, r_2 \in Q$ which are the starting and ending states of a path on untagged states only; moreover the configuration before the flush has stack $[\#\ q'_0]\psi[x_1\ r_2]$ and $x_1 \succcurlyeq w_2$ and $\tilde{\delta}_{\text{flush}}(r_2, q'_0) \ni q'_1$. Therefore $\langle \alpha_0 = [\#\ q'_0]\ ,\ w_1 w_2 \ldots\rangle \overset{+}{\vdash} \langle \alpha_1 = [\#\ q'_1]\ ,\ w_2 \ldots\rangle$.
Since $\forall q'_i \in \tilde{I}, a \in \Sigma, \tilde{\delta}_{\text{push}}(q'_i, a) = \delta_{\text{push}}(c(q'_i), a)$, then there exists in $\mathcal{A}$

$$q_0 \xrightarrow{c_1} r_1 \overset{y_1}{\rightsquigarrow} r_2 \tag{3.6}$$

ending with stack $[\#\ q_0]\psi[x_1\ r_2]$. Since $\delta_{\text{flush}}(r_2, q_0) = q_1$, then $\mathcal{A}$ follows a run $\langle \beta_0 = [\#\ q_0]\ ,\ w_1 w_2 \ldots\rangle \overset{+}{\vdash} \langle \beta_1 = [\#\ q_1]\ ,\ w_2 \ldots\rangle$ and β_1 has empty stack, too.
Then, suppose that the statement holds for a sequence of less than k transitions labeled, respectively, $w_1 w_2 \ldots w_{k-1}$ and consider the statement for the k^{th} word $w_k = c_k y_k, c_k \in \Sigma, y_k \in \Sigma^*$.
By inductive hypothesis there is a run of $\tilde{\mathcal{A}}$ on the first $k-1$ words ending with a configuration with empty stack and tagged state: $\langle \alpha_0 = [\#\ q'_0]\ ,\ w_1 w_2 \ldots\rangle \overset{+}{\vdash} \langle \alpha_{k-1} = [\#\ q'_{k-1}]\ ,\ w_k \ldots\rangle$ and $\langle \alpha_{k-1} = [\#\ q'_{k-1}]\ ,\ w_k \ldots\rangle \overset{+}{\vdash} \langle \gamma[b\ q'_k]\ ,\ w_{k+1} \ldots\rangle$ for some $\gamma \in \Gamma^*, b \in (\Sigma \cup \{\#\})$, and there is a run of $\mathcal{A}$ $\langle \beta_0 = [\#\ q_0]\ ,\ w_1 w_2 \ldots\rangle \overset{+}{\vdash} \langle \beta_{k-1} = [\#\ q_{k-1}]\ ,\ w_k \ldots\rangle$. As for the base case, q'_k is a tagged state and can be reached in $\tilde{\mathcal{A}}$ only with a flush transition (labeled with a tagged state) and configuration $C_{k-1} = \langle \alpha_{k-1}\ ,\ w_k \ldots\rangle$ has empty stack with a tagged state on the top. Then, since the only tagged state visited from q'_{k-1} until q'_k is q'_{k-1} itself, there must exist a path

$$q'_{k-1} \xrightarrow{c_k} r \overset{y_k}{\rightsquigarrow} s \overset{q'_{k-1}}{\Longrightarrow} q'_k \tag{3.7}$$

for some $r, s \in Q$, and y_k is the label of a path of (untagged) states in Q. Furthermore, before the flush $\tilde{\mathcal{A}}$ ends with a stack $[\#\ q'_{k-1}]\psi_k[x_k\ s]$ and $x_k \succcurlyeq w_{k+1}$ and $\tilde{\delta}_{\text{flush}}(s, q'_{k-1}) \ni q'_k$. Therefore $\langle \alpha_{k-1}\ ,\ w_k \ldots\rangle \overset{+}{\vdash} \langle \alpha_k = [\#\ q'_k]\ ,\ w_{k+1} \ldots\rangle$.
By construction this implies that there exists a path in $\mathcal{A}$

$$q_{k-1} \xrightarrow{c_k} r \overset{y_k}{\rightsquigarrow} s \tag{3.8}$$

ending with stack $[\#\ q_{k-1}]\psi_k[x_k\ s]$ and $s \overset{q_{k-1}}{\Longrightarrow} q_k$. Then $\mathcal{A}$ follows a run on $w_1 \ldots w_k$ defined as $\langle \beta_0\ ,\ w_1 w_2 \ldots\rangle \overset{+}{\vdash} \langle \beta_{k-1}\ ,\ w_k \ldots\rangle \overset{+}{\vdash} \langle \beta_k = [\#\ q_k]\ ,\ w_{k+1} \ldots\rangle$ where $q_k = c(q'_k)$ and β_k has empty stack.
Finally, since $\tilde{\mathcal{S}}$ is a successful run for $\tilde{\mathcal{A}}$, then there exist in $\tilde{\mathcal{S}}$ infinitely many configurations $\langle \alpha_i\ ,\ x_i\rangle$ with $state(\alpha_i) \in \tilde{F}$. Notice also that $state(\alpha_i)$ is tagged since all states in $\tilde{F}$ are tagged states. As it has been shown before, each tagged state is reached by $\tilde{\mathcal{A}}$ iff the automaton $\tilde{\mathcal{A}}$ empties its stack and $\mathcal{A}$ reaches its counterpart q with empty stack, too. Then $\mathcal{A}$ has a run which visits infinitely often states $c(q')$ with empty stack, and considering that tagged states q' belong to $\tilde{F}$ only if $c(q') \in F$, then this run is successful for $\mathcal{A}$. □

The next lemma proves the subsequent part of Theorem 3.1.

Lemma 3.3 *There exist ω-languages that can be recognized by BFA automata (F), but no BFA automaton (E) can recognize them.*

The lemma follows from two observations.

First of all, there are ω-languages which can be accepted by BFA automata with acceptance condition by final state (F) such that there exist BFA automata (E) which can recognize only languages that are weakly equivalent to them, i.e. the syntactic tree assigned to words of the original language may differ from the syntactic tree associated to the words accepted by the BFA (E) automaton.
Consider the two following examples:

Example 3.2 *The simple ω-language $\mathcal{L}_1$ generated by the Floyd grammar with rules and OPM:*

$$\begin{array}{l} S \to A, \\ A \to aA \end{array} \qquad \begin{array}{c|c} & a \\ \hline a & \lessdot \\ \# & \lessdot \end{array}$$

comprises only one word $\mathcal{L}_1 = \{aaa\ldots\} = \{a^\omega\}$, which is structured so as the stack grows indefinitely. It is recognized by the BFA (F) automaton in Figure 3.3, whereas no BFA (E) can accept it.

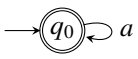


Figure 3.3: ω-automaton recognizing language $\mathcal{L}_1$ of Example 3.2.

Consider now the ω-language $\tilde{\mathcal{L}}_1$ generated by the Floyd grammar with rules and OPM:

$$\begin{array}{l} S \to A, \\ A \to Aa \end{array} \qquad \begin{array}{c|c} & a \\ \hline a & \gtrdot \\ \# & \lessdot \end{array}$$

which is obtained by the former one turning the right recursion in the rules into left recursion. Language $\tilde{\mathcal{L}}_1$ is recognized by the automaton in Figure 3.4 with both (E) and (F) acceptance conditions.

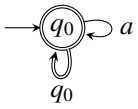


Figure 3.4: ω-automaton recognizing language $\tilde{\mathcal{L}}_1$ of Example 3.2.

Example 3.3 *Let $\mathcal{L}_2 = (\mathcal{L}_{Dyck}(a, \underline{a}, b, \underline{b}))^\omega$ be the language of ω-words composed by an infinite sequence of finite-length words belonging to the Dyck language with the pairs $a, \underline{a}$ and $b, \underline{b}$ (which as usual can be interpreted as a language of calls to two procedures a and b) and where, in general, for a set of finite words $L \subseteq A^*$, one defines*

$L^\omega = \{\alpha \in A^\omega \mid \alpha = w_0 w_1 \ldots \text{ with } w_i \in L \text{ for } i \geq 0\}$.
$\mathcal{L}_2$ can be seen as generated by the FG with rules and related OPM in the following figure. Notice that this FG is the classical grammar that produces words of finite length of the Dyck language over a, $\underline{a}$, b, $\underline{b}$. Infinite-length words are generated whenever only nonterminating derivations, obtained using rules of the FG infinitely many times, are admitted.

$$S \rightarrow aS\underline{a}S \mid bS\underline{b}S \mid \varepsilon$$

	a	$\underline{a}$	b	$\underline{b}$
a	$\lessdot$	$\doteq$	$\lessdot$	
$\underline{a}$	$\lessdot$	$\gtrdot$	$\lessdot$	$\gtrdot$
b	$\lessdot$		$\lessdot$	$\doteq$
$\underline{b}$	$\lessdot$	$\gtrdot$	$\lessdot$	$\gtrdot$
#	$\lessdot$		$\lessdot$	

No BFA (E) can recognize this language. Clearly, the right recursion in the productions of the grammar indefinitely postpones the reduction of the r.h.s. of the rules. As a consequence of this, a Floyd automaton piles up an infinite sequence of words on the stack, as no flush transition can be performed at the end of the parsing of a finite-length word of the Dyck language derived from the r.h.s. of the rules. There is, however, a BFA (F) automaton accepting $\mathcal{L}_2$.

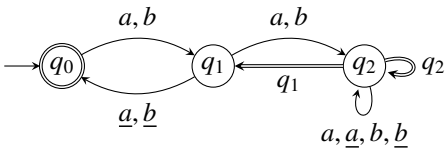


Figure 3.5: BFA automaton recognizing language $\mathcal{L}_2$ with F acceptance condition.

If, instead, $\mathcal{L}_2$ is seen as generated by the Floyd grammar with rules and OPM:

$$S \rightarrow SaS\underline{a} \mid SbS\underline{b} \mid \varepsilon$$

	a	$\underline{a}$	b	$\underline{b}$
a	$\lessdot$	$\doteq$	$\lessdot$	
$\underline{a}$	$\gtrdot$	$\gtrdot$	$\gtrdot$	$\gtrdot$
b	$\lessdot$		$\lessdot$	$\doteq$
$\underline{b}$	$\gtrdot$	$\gtrdot$	$\gtrdot$	$\gtrdot$
#	$\lessdot$		$\lessdot$	

then there exists an automaton with (E) acceptance condition recognizing it (Figure 3.6 - see also Figure 1 in [17]).

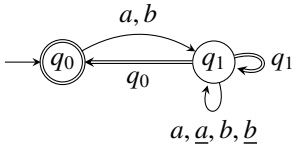


Figure 3.6: BFA automaton recognizing language $\mathcal{L}_2$ with E acceptance condition.

In particular, as regards the two examples above, the automata recognizing the languages generated by the left or right-recursive rules are structurally equivalent in a broad sense: turning recursion from right to left, the syntactic subtrees associated to the words of the language by the former grammar are specular to those assigned by the second grammar.
The second observation, which essentially proves the lemma, is that there are languages that cannot be recognized by empty stack and final state even changing recursion in the rules. Consider the language consisting of words of the Dyck language on the alphabet $\{a, \underline{a}\}$ with a finite but not null number of initial pending calls (generated with infinite derivations of the FG in the following figure, by using rules involving the axiom S only a finite number of times and rules having nonterminal A as l.h.s. an infinite number of times).

$$\mathcal{L} = a^+(\mathcal{L}_{\text{Dyck}}(a, \underline{a}))^\omega \qquad \begin{array}{c} S \to aS \mid aA, \\ A \to AaA\underline{a} \mid \varepsilon \end{array}$$

The precedence relation $a \lessdot a$ is necessary to define Dyck words, but it hinders any automaton from emptying the stack after the parsing of the first sequence of pending calls a^+. Moreover, introducing also a relation $a \gtrdot a$ between the characters of the alphabet would create a conflict in the OPM of the language, whereas the operator precedence matrix of a Floyd grammar has to be conflict-free. A BFA automaton with acceptance condition by final state can, instead, accept this language (Figure 3.7).

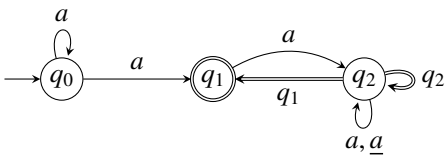


Figure 3.7: BFA automaton (F) recognizing $a^+(\mathcal{L}_{\text{Dyck}}(a, \underline{a}))^\omega$

As a further proof of the inadequacy of the acceptance condition (E), consider the language $\mathcal{L}_{\text{repbdd}}$ (studied in [2]) consisting of infinite words on the alphabet $\{a, \underline{a}\}$ with a finite number of pending calls (incidentally, this language has been proposed as an adequate definition of winning conditions for pushdown games). A nondeterministic BFA automaton with final state acceptance condition can nondeterministically guess which is the prefix of the word containing the last pending call, and then recognizes the language $(\mathcal{L}_{\text{Dyck}}(a, \underline{a}))^\omega$ of correctly nested words. A BFA automaton (E) cannot flush the pending calls among the correctly nested letters a, otherwise it would either introduce conflicts in the OPM or it would not be able to count they are in finite number. Notice, however, that even though the acceptance condition (E) is inadequate to model all ω-languages recognized by BFA, nevertheless it generally allows a more compact definition of automata. As a simple example, consider again the language $\mathcal{L}_2$ of ω-words composed by an infinite sequence of finite-length words belonging to the Dyck language with the pairs $a, \underline{a}$ and $b, \underline{b}$, which can be recognized by final state and empty stack by automaton in Figure 3.6: this automaton can be reduced to have a single state, as in the following Figure 3.8. On the contrary, every BFA automaton (F) accepting the same language must have more than one state.

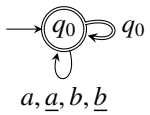


Figure 3.8: Reduced automaton (E) recognizing $\mathcal{L}_2$

Remark Notice that the difference between (E) and (F) recognition modes is immaterial for FAs which recognize words of finite length. In fact, every finite word is ended by the marker #, and $\forall a \in \Sigma \setminus \{\#\}$ the precedence relation $a \gtrdot \#$ holds, hence the stack is always emptied after the parsing of the input word. This is not the case for ω-languages, where obviously words are infinite and there's no ending delimiter.

3.2 Muller condition

Traditionally, Muller automata have been introduced in order to provide an adequate acceptance mode for deterministic automata on ω-words. As regards classical automata on infinite words, in fact, deterministic Büchi automata cannot recognize all ω-regular languages, whereas deterministic Muller automata match the expressive power of nondeterministic Büchi acceptance condition. The notion of Muller acceptance can be defined for Floyd automata as follows.

Definition 3.5 (Muller acceptance condition) *A nondeterministic Muller automaton which recognizes Floyd languages (MFA) is an ω-FA $\langle \Sigma, M, Q, I, \mathcal{F}, \delta \rangle$ whose acceptance component is a collection of subsets of Q, $\mathcal{F} = \mathcal{T} \subseteq 2^Q$, called the* table *of the automaton.*

Let "$\exists^\omega i$" be a shorthand for "there exist infinitely many i" and let $\mathcal{S}$ be a run of the automaton on a given word $x \in \Sigma^\omega$. Then define:

$$In(\mathcal{S}) = \{q \in Q \mid \exists^\omega i \langle \beta_i\ ,\ x_i \rangle \in \mathcal{S} \text{ with } state(\beta_i) = q\}$$

as the set of states that occur infinitely often at the top of the stack of configurations in $\mathcal{S}$.

A MFA automaton $\mathcal{A} = \langle \Sigma, M, Q, I, \mathcal{T}, \delta \rangle$ is said to have a *successful* run $\mathcal{S}$ on an infinite word $x \in \Sigma^\omega$ iff $In(\mathcal{S}) \in \mathcal{T}$, i.e. the set of states occurring infinitely often on the stack is a set in the table $\mathcal{T}$.

3.3 Other acceptance conditions

The classical notions of acceptance for ω-automata, as Rabin and Streett, can be defined for ω-FLs in a similar way.

Other possible acceptance conditions may refer to the evolution of the stack, for instance requiring the height of the stack to be always bounded by a given constant k, or infinitely often bounded, and the like.

3.4 Completeness

Finally, as it is usual for classical automata, there are some reductions that can be operated on FAs or ω-FAs to obtain normalized automata. The following statements introduce two modes to complete a Floyd automaton, that will be considered again in Chapter 5 to prove some closure properties enjoyed by the families of ω-Floyd languages.

3.4.1 Completion of the transition function δ

In general, a FA (or ω-FA) may have a partial transition function, so that not all the possible words which are compatible with the precedence matrix of the automaton can be read along runs on its state-graph. A first definition of completeness of a Floyd automaton concerns the property of its transition function of being total, with respect to a given precedence matrix, as the following statement makes precise.

Definition 3.6 *Complete FA and ω-FA*
Let M be a conflict-free precedence matrix defined on an alphabet Σ.
Denote by $L_{FAM} \subseteq \Sigma^*$ the language comprising all finite words $x \in \Sigma^*$ with syntax trees compatible with M. Denote also by $L_M \subseteq \Sigma^\omega$ the ω-language comprising all infinite words $x \in \Sigma^\omega$ with syntax trees compatible with M.
Then, a FA $\mathcal{A} = \langle \Sigma, M, Q, I, F, \delta \rangle$ (or a ω-FA $\mathcal{A} = \langle \Sigma, M, Q, I, \mathcal{F}, \delta \rangle$) is said to be complete *with respect to matrix M if every word $x \in L_{FAM}$ (or ω-word $x \in L_M$) is the label of some path starting from an initial state of $\mathcal{A}$.*

3.4.2 Completion of the OPM

A further normalization of Floyd automata regards the completion of their precedence matrix. The precedence matrix of a FA on finite or infinite words completely determines the syntax trees of the words recognized by the automaton, and different matrices can assign to the strings of the language it accepts significantly different structure and semantics. It is however possible to complete (even partially) the OPM of an automaton without affecting the language it recognizes. Indeed, the empty entries in a precedence matrix may be filled with any type of precedence relation ($\lessdot, \doteq, \gtrdot$), and the resulting matrix does not have to be necessarily total, but the conflict-freeness property has to be preserved.

The completion of the precedence matrix is correct if the existence of successful runs for the input strings in the automaton with completed OPM holds also in the original automaton and does not depend on the new precedence relations added in the matrix; thus, in order to guarantee the equivalence between an automaton and its normal form, it is necessary to impose that, along a valid run of the completed automaton, a move (mark/push/flush move) can be performed only if even in the initial matrix (and not only in the completed OPM) there was a corresponding precedence relation between the last symbol read to reach the current state and the next input symbol to be considered, guaranteeing the validity of the move even in the non completed FA.
This constraint may be imposed by formalizing a normalized automaton whose states keep additional information on both the portion of the input string read so far and the word still to process. More precisely, in order to avoid inconsistent push/mark moves along a run of the completed automaton, it is necessary to augment the states with a first component, a lookback symbol, which represents the character of the input string

read to reach the current state. This symbol allows to check whether the precedence relation that determines the move of the normalized FA along the run (i.e. a $\doteq, \lessdot$ relation between the last processed letter, which is stored as lookback character, and the next character of the word) is defined also in the original matrix, and, then, the transition will be defined only if it is allowed in the initial automaton. The lookback symbol alone, however, does not suffice to manage the introduction of arbitrary $\gtrdot$ precedence relations in the matrix: indeed, to check the validity of a flush transition even in the original FA, it is necessary to take into account if this type of precedence relation holds also in the original OPM, between the last character read and the next letter still to be processed. While push and mark moves are defined on the basis of the next input symbol, the flush transition function of the automaton does not depend directly on it. Therefore, this letter is introduced as a second component in the state (the lookahead symbol). In this way, the presence of additional entries in the precedence matrix assures that no spurious move can be performed, leading to the acceptance of words which do not belong to the original language.

This normalization of the precedence matrix of an automaton is valid for both FAs, which recognize languages of finite strings, and for ω-FAs: the only difference between these two classes of automata deals with the constraints on the acceptance condition that are required to guarantee the equivalence between the language recognized by the original automaton and its normalized counterpart.

This section introduces first the normal form of automata with completed OPM for FAs, proving the correctness of the formalization, and then the construction for a normalized automaton is naturally extended to the field of ω-languages.

Theorem 3.4 *Normal form of a FA, with completed OPM.*
Let $\mathcal{A} = \langle \Sigma, M, Q, I, F, \delta \rangle$ be a FA with OPM M.
For each conflict-free OPM $M' \supseteq M$ which includes M, it is possible to define a FA with OPM M' which recognizes the same language as $\mathcal{A}$.

Proof Let $\mathcal{A} = \langle \Sigma, M, Q, I, F, \delta \rangle$ be a FA and $M' \supseteq M$ be a conflict-free precedence matrix which includes M.
Consider the FA $\mathcal{A}' = \langle \Sigma, M', Q', I', F', \delta' \rangle$ that has M' as OPM and is defined by:

- $Q' = \hat{\Sigma} \times Q \times \hat{\Sigma}$, where $\hat{\Sigma} = (\Sigma \cup \{\#\})$, i.e. the first component of a state is the lookback symbol, the second component of the triple is a state of $\mathcal{A}$ and the third component of the state is the lookahead symbol,
- $I' = \{\#\} \times I \times \{a \in \hat{\Sigma} \mid M_{\#a} \neq \emptyset\}$ is the set of initial states of $\mathcal{A}'$,
- $F' = \{\#\} \times F \times \{\#\}$
- $\delta' : Q' \times (\Sigma \cup Q') \to 2^{Q'}$ is the transition function defined as follows.
  The push transition $\delta'_{\text{push}} : Q' \times \Sigma \to 2^{Q'}$ is defined by:

  $$\delta'_{\text{push}}(\langle a, q, b \rangle, b) = \{\langle b, p, c \rangle \mid p \in \delta_{\text{push}}(q, b) \wedge M_{ab} \in \{\lessdot, \doteq\} \wedge M_{bc} \neq \emptyset\},$$

  $\forall a \in \hat{\Sigma}, b \in \Sigma, q \in Q$.
  The flush transition $\delta'_{\text{flush}} : Q' \times Q' \to 2^{Q'}$ is defined by:
  $\delta'_{\text{flush}}(\langle a_1, q_1, a_2 \rangle, \langle b_1, q_2, b_2 \rangle) = \{\langle b_1, q_3, a_2 \rangle \mid q_3 \in \delta_{\text{flush}}(q_1, q_2) \ \wedge \ M_{a_1 a_2} = \gtrdot \ \wedge$
  $\wedge \ M_{b_1 a_2} \neq \emptyset\}$,
  $\forall a_1, b_2 \in \Sigma, \forall a_2, b_1 \in \hat{\Sigma}, \forall q_1, q_2 \in Q$.

The FA $\mathcal{A}'$ with OPM M' accepts the same language as $\mathcal{A}$.
Before proving the equivalence between the languages recognized by the two automata, consider the following example, which shows how a normalized automaton may be obtained from a given FA, preserving the language it accepts.

Example 3.4 *Consider the nondeterministic FA $\mathcal{A} = \langle \Sigma, M, Q, I, F, \delta \rangle$ defined on the alphabet $\Sigma = \{a, b\}$ which recognizes the language*
$L(\mathcal{A}) = ab^* \cup a^+$,
with OPM M and state-graph depicted in Figure 3.9.

	a	b	#
a	$\lessdot$	$\lessdot$	$\gtrdot$
b		$\gtrdot$	$\gtrdot$
#	$\lessdot$		$\doteq$

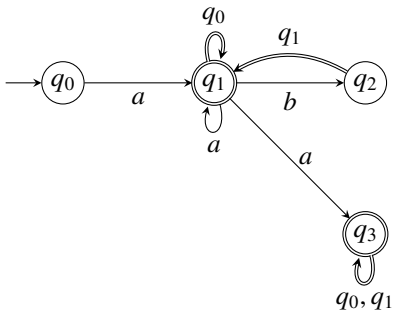


Figure 3.9: FA automaton of Example 3.4, with its OPM.

Notice that the words ab^+a^+ cannot be recognized by $\mathcal{A}$; indeed, symbol b has no precedence relation with letter a and the sequence of transitions $q_1 \xrightarrow{b} q_2 \overset{q_1}{\Longrightarrow} q_1 \xrightarrow{a} q_3$ is not allowed.
Consider, now, a matrix $M' \supseteq M$ which differs from M only as regards the relation M'_{ba}, which is defined as $M'_{ba} = \gtrdot$.

	a	b	#
a	$\lessdot$	$\lessdot$	$\gtrdot$
b	$\gtrdot$	$\gtrdot$	$\gtrdot$
#	$\lessdot$		$\doteq$

If the OPM is completed in this way, keeping the state-graph unchanged, the automaton would recognize even the words ab^+a^+, which do not belong to $L(\mathcal{A})$. In order to guarantee the equivalence between the original automaton $\mathcal{A}$ and an automaton $\mathcal{A}'$ with OPM $M' \supseteq M$ arbitrarily filled, it is necessary to check that each move (push/mark/flush) that can be performed by $\mathcal{A}'$ is also allowed by the precedence relations defined in the original matrix M, so that the introduction of new precedence relations does not influence the language accepted by the automaton.

A normalized automaton $\mathcal{A}' = \langle \Sigma, M', Q', I', F', \delta' \rangle$ with OPM M', defined according to the aforementioned construction, recognizes exactly $L(\mathcal{A})$, though its precedence matrix has been extended to include relations which do not hold in M. The state graph of the automaton is shown in Figure 3.10.

Notice that the words ab^+a^+ are not accepted by $\mathcal{A}'$, though $M'_{ba} = \gtrdot$, and this is consistent with the behavior of the original automaton $\mathcal{A}$, which rejects such strings.

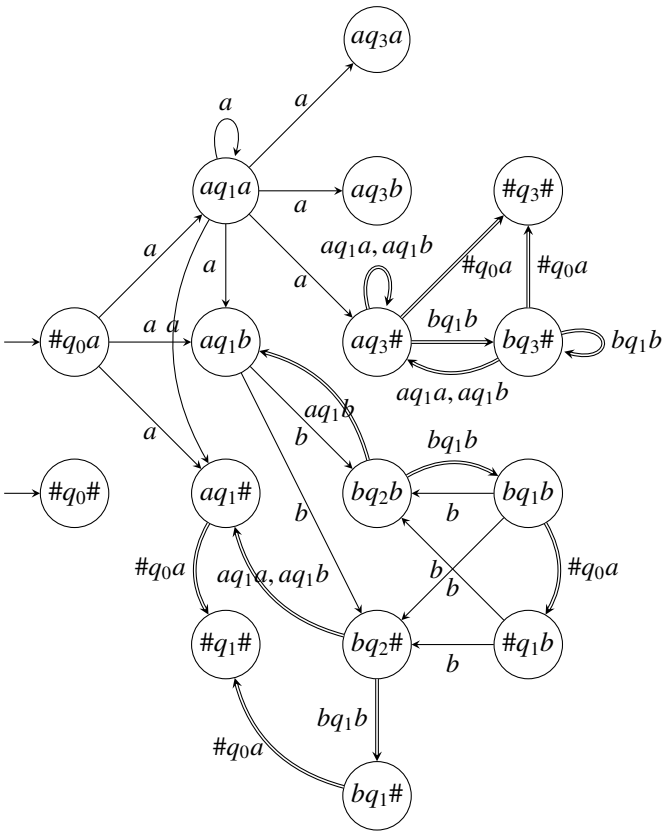


Figure 3.10: FA automaton with OPM M' of Example 3.4.

As usual, if the transition function is not defined, the automaton reaches an error state, not shown in the previous picture.

Now, in order to prove that the two automata $\mathcal{A}$ and $\mathcal{A}'$ accept the same language, $L(\mathcal{A}) = L(\mathcal{A}')$, one may first of all precisely characterize the valid runs of $\mathcal{A}'$ on input strings, i.e. the runs which parse the entire word in input and do not stop in error states. Let x be an input word. A valid run of the automaton $\mathcal{A}'$ on x shows the following structure:
$\mathcal{S}' = C_1 \vdash C_2 \vdash \ldots \vdash C_i \vdash \ldots \vdash C_n$ (for a given n)
where each configuration C_i, $i \leq n$ is
$C_i = \langle \beta_i = B_{i1}B_{i2}\ldots B_{ik}\ ,\ x_i\rangle = \langle [a_1\ \langle a_1\ ,\ q_1,\ a_2\rangle][a_2\ \langle a_2\ ,\ q_2,\ a_3\rangle]\ldots$
$\ldots [a_k\ \langle a_k\ ,\ q_k,\ first(x_i)\rangle]\ ,\ x_i\rangle$, where $first(x_i)$ is the first character of word x_i.
Each state $\langle a_j\ ,\ q_j,\ a_{j+1}\rangle$ in Q' piled up on the stack has, as first component, the symbol a_j read when the element was pushed on the stack and the third component a_{j+1} is the guessed lookahead symbol that will be read at the subsequent move of the automaton and it equals the symbol of the next element on the stack.
This structure necessarily follows from the choice of the transition function: as regards push/mark moves, by definition of the push function, the state placed on the stack after the move has, as first component of the triple, the symbol currently read by the automaton and, moreover, the function δ_{push} is defined from a given state only if the lookahead symbol of the state exactly matches the next input symbol to be read, which

corresponds to the requirement that the letter chosen as lookahead (the third component of the state) has to be guessed correctly (otherwise an error state is reached). On the other side, flush moves remove the elements on the top of the stack until the first marked symbol included, and update the underlying element on the stack restoring exactly its lookback symbol, since the parsing of the input string will be resumed from this character: thus in the updated element, say $[a_{j-1}\ \langle a_{j-1}\ ,\ \delta_{\text{flush}}(q_k, q_{j-1}),\ a_k\rangle]$, the first component of the new state is equal to the previous symbol a_{j-1}, while the third component of the state still equals the next guessed character.

Given this structure of the valid runs in $\mathcal{A}'$, the proof is rather technical but straightforward: one may prove the equivalence between the sets of words accepted by the two automata considering, first, the direction from right to left of the statement above, i.e $L(\mathcal{A}') \subseteq L(\mathcal{A})$.
Let x be a word belonging to $L(\mathcal{A}')$ and let $\mathcal{S}'$ be a successful run of $\mathcal{A}'$ on it.
$\mathcal{S}' = C'_1 \vdash C'_2 \vdash \ldots \vdash C'_i \vdash \ldots \vdash C'_n$
Define a corresponding run of $\mathcal{A}$ on x, which passes through the sequence of states visited by the run $\mathcal{S}'$, without any information on the lookback and lookahead symbols, and expressed as follows:
$\mathcal{S} = C_1 \vdash C_2 \vdash \ldots \vdash C_i \vdash \ldots \vdash C_n$
where the first configurations are $C'_1 = \langle [\#\ \langle \#\ ,\ q_0,\ a_1\rangle]\ ,\ x\rangle$ and $a_1 = \text{first}(x)$, $q_0 \in I$ and $C_1 = \langle [\#\ q_0]\ ,\ x\rangle$. Furthermore, for each following move $C'_i \vdash C'_{i+1}$

- if it is a mark move in $\mathcal{A}'$, then
  $C'_i = \langle \gamma'[a_i\ \langle a_i\ ,\ q_i,\ a_{i+1}\rangle]\ ,\ x_i\rangle \vdash C'_{i+1} = \langle \gamma'[a_i\ \langle a_i\ ,\ q_i,\ a_{i+1}\rangle][a'_{i+1}\ \langle a_{i+1}\ ,\ q_{i+1},\ a_{i+2}\rangle]\ ,\ x_{i+1}\rangle$
  where a_i can be possibly marked, and define in $\mathcal{A}$
  $C_i = \langle \gamma[a_i\ q_i]\ ,\ x_i\rangle \vdash C_{i+1} = \langle \gamma[a_i\ q_i][a'_{i+1}\ q_{i+1}]\ ,\ x_{i+1}\rangle$.
  Notice that this move is valid in $\mathcal{A}$ since, by definition of δ'_{push}, it holds that $q_{i+1} = \delta_{\text{push}}(q_i, a_{i+1})$ and $a_i \lessdot a_{i+1}$ must be a precedence relation also in M.
  Indeed, each state of $\mathcal{A}'$, in general, is defined as $\langle a_i\ ,\ q_i,\ a_{i+1}\rangle$ where there is a relation in M between a_i and a_{i+1}; thus, being the move a mark transition and $M'_{a_i a_{i+1}} = \lessdot$, the same precedence relation must be included also in M.

- if it is a push move, the transition is defined as for the mark move: in $\mathcal{A}'$ it holds that
  $C'_i = \langle \gamma'[a_i\ \langle a_i\ ,\ q_i,\ a_{i+1}\rangle]\ ,\ x_i\rangle \vdash C'_{i+1} = \langle \gamma'[a_i\ \langle a_i\ ,\ q_i,\ a_{i+1}\rangle][a_{i+1}\ \langle a_{i+1}\ ,\ q_{i+1},\ a_{i+2}\rangle]\ ,\ x_{i+1}\rangle$
  and define in $\mathcal{A}$
  $C_i = \langle \gamma[a_i\ q_i]\ ,\ x_i\rangle \vdash C_{i+1} = \langle \gamma[a_i\ q_i][a_{i+1}\ q_{i+1}]\ ,\ x_{i+1}\rangle$.
  where a_{i+1} is not marked and $a_i \doteq a_{i+1}$ in M' and in M.

- if it is a flush move
  $C'_i = \langle \gamma'[a_i\ \langle a_i\ ,\ q_i,\ a_{i+1}\rangle][a'_{i+1}\ \langle a_{i+1}\ ,\ q_{i+1},\ a_{i+2}\rangle] \ldots [a_k\ \langle a_k\ ,\ q_k,\ a_{k+1}\rangle]\ ,\ x_k\rangle \vdash$
  $C'_{i+1} = \langle \gamma'[a_i\ \langle a_i\ ,\ \delta_{\text{flush}}(q_k, q_i),\ a_{k+1}\rangle]\ ,\ x_k\rangle$
  where $a_i \lessdot a_{i+1} \doteq \cdots \doteq a_k \gtrdot first(x_k)$, then consider in $\mathcal{A}$
  $C_i = \langle \gamma[a_i\ q_i][a'_{i+1}\ q_{i+1}] \ldots [a_k\ q_k]\ ,\ x_k\rangle \vdash C_{i+1} = \langle \gamma[a_i\ \delta_{\text{flush}}(q_k, q_i)]\ ,\ x_k\rangle$.
  which is a valid move for the original automaton $\mathcal{A}$ since the lookahead character a_{k+1} equals first(x_k), and even in M it holds that $M_{a_k a_{k+1}} = \gtrdot$, by definition of δ'_{flush}.

Then $\mathcal{S}$ is a valid run for $\mathcal{A}$ on x. Furthermore, since $\mathcal{S}'$ is successful and ends with configuration $C'_n = \langle [\#\ \langle \#\ ,\ q_f,\ \#\rangle]\ ,\ \#\rangle$, where $q_f \in F$, then the run $\mathcal{S}$ built so far ends with an accepting configuration too, $C_n = \langle [\#\ q_f]\ ,\ \#\rangle$.

Conversely, consider the direction from left to right of the statement above, i.e. $L(\mathcal{A}) \subseteq L(\mathcal{A}')$.
Let $x \in L(\mathcal{A})$ and let $\mathcal{S}$ be a successful run of $\mathcal{A}$ on x,
$\mathcal{S} = C_1 \vdash C_2 \vdash \ldots \vdash C_i \vdash \ldots \vdash C_n$.
Then, define the sequence of transitions $\mathcal{S}' = C'_1 \vdash C'_2 \vdash \ldots \vdash C'_i \vdash \ldots \vdash C'_n$, where for each configuration
$C_i = \langle \beta_i = B_{i1}B_{i2}\ldots B_{ik}\ ,\ x_i\rangle$, let
$C'_i = \langle [symbol(B_{i1})\ \langle symbol(B_{i1})\ ,\ state(B_{i1}),\ symbol(B_{i2})\rangle] \ldots$
$\ldots [symbol(B_{in})\ \langle symbol(B_{in})\ ,\ state(B_{in}),\ first(x_i)\rangle]\ ,\ x_i\rangle$
Now prove, by induction on the number of moves in $\mathcal{S}$, that the sequence of moves defined above represents indeed a valid run $\mathcal{S}'$ on x for the normalized automaton $\mathcal{A}'$, and it is successful.
Notice that the first move of the run must be a mark move (unless x is empty), i.e. $C_1 = \langle [\# \ q_0]\ ,\ x\rangle \vdash C_2 = \langle [\# \ q_0][a'_1\ q_1]\ ,\ x_1\rangle$ and $x = a_1x_1$ with $\# \lessdot a_1$ in M. Then the transition defined so far:
$C'_1 = \langle [\# \ \langle \# \ ,\ q_0,\ a_1\rangle]\ ,\ x\rangle \vdash C'_2 = \langle [\# \ \langle \# \ ,\ q_0,\ a_1\rangle][a'_1\ \langle a_1\ ,\ q_1,\ first(x_1)\rangle]\ ,\ x_1\rangle$ is a valid move in $\mathcal{A}'$, since it satisfies the definition of δ'_{push}.
Assuming that $\mathcal{S}'$ is valid up to the $(k-1)$-th move, consider the k-th move: $C_k \vdash C_{k+1}$.
For each type of move (push/mark/flush) one may prove that $C'_k \vdash C'_{k+1}$ defined above is a valid move too: indeed

- push (mark) move: If $C_k = \langle \gamma[a_k\ q_k]\ ,\ x_k\rangle \vdash C_{k+1} = \langle \gamma[a_k\ q_k][a_{k+1}\ q_{k+1}]\ ,\ x_{k+1}\rangle$, then $C'_k = \langle \gamma'[a_k\ \langle a_k\ ,\ q_k,\ a_{k+1}\rangle]\ ,\ x_k\rangle \vdash$
  $\vdash C'_{k+1} = \langle \gamma'[a_k\ \langle a_k\ ,\ q_k,\ a_{k+1}\rangle][a_{k+1}\ \langle a_{k+1}\ ,\ q_{k+1},\ first(x_{k+1})\rangle]\ ,\ x_{k+1}\rangle$ is a valid push (mark) move for $\mathcal{A}'$, according to the definition of δ'_{push}, since $M' \supseteq M$ and $M_{a_ka_{k+1}} \doteq$ ($\lessdot$ respectively), and there is a relation in M between a_{k+1} and the next symbol first(x_{k+1}) since the run $\mathcal{S}$ is valid;

- flush move:
  If $C_k = \langle \gamma[a_i\ q_i][a'_{i+1}\ q_{i+1}]\ldots[a_k\ q_k]\ ,\ x_k\rangle \vdash C_{k+1} = \langle \gamma[a_i\ \delta_{\text{flush}}(q_k, q_i)]\ ,\ x_k\rangle$ with $a_i \lessdot a_{i+1} \doteq \cdots \doteq a_k \gtrdot first(x_k)$, then the move
  $C'_k = \langle \gamma'[a_i\ \langle a_i\ ,\ q_i,\ a_{i+1}\rangle][a'_{i+1}\ \langle a_{i+1}\ ,\ q_{i+1},\ a_{i+2}\rangle]\ldots[a_k\ \langle a_k\ ,\ q_k,\ first(x_k)\rangle]\ ,\ x_k\rangle \vdash$
  $C'_{k+1} = \langle \gamma'[a_i\ \langle a_i\ ,\ \delta_{\text{flush}}(q_k, q_i),\ first(x_k)\rangle]\ ,\ x_k\rangle$
  is valid in $\mathcal{A}'$ since, as before, $M' \supseteq M$ and $M_{a_k first(x_k)} = \gtrdot$ and there is a relation in M between a_i and the next symbol first(x_k) being the run $\mathcal{S}$ valid.

Finally, since $\mathcal{S}$ is successful and ends with an accepting configuration $C_n = \langle [\# \ q_f]\ ,\ \#\rangle$, with $q_f \in F$, then the run $\mathcal{S}'$ is accepting too, ending with a final configuration $C'_n = \langle [\# \ \langle \# \ ,\ q_f,\ \#\rangle]\ ,\ \#\rangle$. □

Remark Notice that the normalization has been defined for a nondeterministic FA, but it holds likewise directly for deterministic Floyd automata. Indeed, it is possible to arbitrarily fill the OPM of a deterministic FA with the following normalization, which derives from the previous one and preserves the property of the original automaton of being deterministic.
Let $\mathcal{A} = \langle \Sigma, M, Q, q_0, F, \delta\rangle$ be a deterministic FA and let $M' \supseteq M$ be a conflict-free precedence matrix which includes M.
Then consider the FA $\mathcal{A}' = \langle \Sigma, M', Q', q'_0, F', \delta'\rangle$ where:

- $Q' = \hat{\Sigma} \times Q \times 2^{\hat{\Sigma}}$,
- $q'_0 = \langle \#, q_0, \{a \in \Sigma \mid M_{\#a} \neq \emptyset\}\rangle$,

- $F' = \{\langle \#, q, B\rangle \mid q \in F, B \subseteq Q, \# \in B\}$,
- $\delta' : Q' \times (\Sigma \cup Q') \to Q'$ is the transition function defined as follows. The push transition $\delta'_{\text{push}} : Q' \times \Sigma \to Q'$ is defined by:

$$\delta'_{\text{push}}(\langle a, q, B\rangle, b) = \langle b, \delta_{\text{push}}(q, b), \{c \in \hat{\Sigma} \mid M_{bc} \neq \emptyset\}\rangle$$

$\forall a \in \hat{\Sigma}, b \in B : M_{ab} \in \{\lessdot, \doteq\}, q \in Q, B \subseteq \hat{\Sigma}$.
The flush transition $\delta'_{\text{flush}} : Q' \times Q' \to Q'$ is defined by:

$$\delta'_{\text{flush}}(\langle a, q_1, A\rangle, \langle b, q_2, B\rangle) = \langle b, \delta_{\text{flush}}(q_1, q_2), \{c \in A \mid M_{ac} = \gtrdot \wedge M_{bc} \neq \emptyset\}\rangle$$

$\forall a \in \Sigma, b \in \hat{\Sigma}, \forall q_1, q_2 \in Q, A, B \subseteq \hat{\Sigma}, B \neq \{\#\}$.

Notice that each state in Q' can be seen as a set of triples which share the first two components (i.e. the lookback symbol and the state): this is consistent with the definition of a deterministic automaton, for which the state reached with a move, reading a given input character, is unique, and this property holds for both push and flush transition functions. Finally, as usual, the previous construction is formalized so as if a move is not defined for a given input and state, or the state computed by the transition function has the lookahead component empty, an error state is reached.

The automaton $\mathcal{A}'$ previously described is deterministic by construction and, as it holds for nondeterministic FAs, the original deterministic automaton and its normal form are equivalent, i.e. $L(\mathcal{A}) = L(\mathcal{A}')$.

The interesting result concerning this normal form is that it is not restricted to FAs, which recognize languages of finite words, but it may be naturally extended also to the field of ω-languages. Indeed, a ω-FA may be normalized following an analogous construction as for finite FAs: the characterization of a run of the normalized automaton $\mathcal{A}'$ is the same as for the finite case and, in particular, for each valid run $\mathcal{S}$ in $\mathcal{A}$ there is a corresponding valid run $\mathcal{S}'$ in $\mathcal{A}'$, and conversely, such that the sequence of states $q_1, q_2, \ldots, q_i \ldots$ visited by $\mathcal{A}$ along this run equals exactly the sequence $q_1, q_2, \ldots, q_i \ldots$ of the second components in the triples visited by $\mathcal{A}'$ along $\mathcal{S}'$. The lookback and the lookahead symbols are managed as for FAs on finite words, as well.

The only difference among the normal forms of FAs and ω-FAs consists in the acceptance component defined for the automata, which constrains the set of states visited along a successful run on a word in input.
More precisely, the acceptance condition of the normal forms for BFA and MFA is expressed as:

BFA $F' = \hat{\Sigma} \times F \times \Sigma$, i.e. a run of the normal form is accepting iff it visits infinitely often final states of $\mathcal{A}$ (in the set F), independently of the lookback and the lookahead symbols considered for these states (indeed, infinite words are not ended by a delimiter #, and final states for $\mathcal{A}'$ are no longer defined as for finite FAs, $F' = \{\#\} \times F \times \{\#\}$). Notice that for BFA (E) this acceptance component may be further refined as $F' = \{\#\} \times F \times \Sigma$.

MFA $\mathcal{T}' = \{P \subseteq Q' \mid \pi_2(P) \in \mathcal{T}\}$
where $\mathcal{T} \subseteq 2^Q$ is the table of $\mathcal{A}$ and π_2 is the projection from $\hat{\Sigma} \times Q \times \hat{\Sigma}$ on Q.

As mentioned above, since the sequence of states in Q visited by $\mathcal{A}$ and $\mathcal{A}'$ along corresponding runs are the same, then these acceptance conditions guarantee exactly the equivalence of the languages recognized by the original automaton and its normal form.

Finally, as for FAs, the normalization may be directly applied for deterministic ω-FAs, and for a given deterministic ω-FA it may be defined an equivalent deterministic ω-FA with OPM which is super-matrix of the OPM of the initial automaton.

Thus, the normal form of automata with arbitrarily completed OPM may be generally defined for FAs on either finite or infinite words, and the following theorem extends the previously stated Theorem 3.4.

Theorem 3.5 *Normal form for FAs and ω-FAs with arbitrarily completed OPM.*
Let $\mathcal{A}$ be a FA (or ω-FA) with OPM M.
For each conflict-free precedence matrix $M' \supseteq M$, it is possible to define a FA (respectively ω-FA) with OPM M' which recognizes the same language as $\mathcal{A}$, and it is deterministic if $\mathcal{A}$ is deterministic.

Chapter 4

Determinism and Nondeterminism

This chapter outlines the relationships holding among the various models of FAs and ω-FAs, characterizing them on the basis of the fact that nondeterminism is allowed or not in the transitions of the automata.

Before introducing the hierarchy of FAs and ω-FAs, the following definition formalizes the notion of determinism for Floyd automata; the statement is valid for automata recognizing either finite or infinite languages. Nondeterministic FAs (on finite or infinite languages) have already been presented in the previous chapters (Chapter 2 and Chapter 3).

Definition 4.1 ***Deterministic FA on finite or infinite words***
A FA $\mathcal{A} = \langle \Sigma, M, Q, I, F, \delta \rangle$ *(or* ω*-FA* $\mathcal{A} = \langle \Sigma, M, Q, I, \mathcal{F}, \delta \rangle$*) is called* deterministic *if*

- $\delta_{push}(q, a)$ *and* $\delta_{flush}(q, p)$ *have at most one element, for every state* $q, p \in Q$ *and letter* $a \in \Sigma$*, and*
- I *is a singleton, i.e.* $\mathcal{A}$ *has one initial state.*

Hence, every word in Σ^* (respectively Σ^ω) is the label of at most one path in the state-graph, starting from an initial state. Every word $x \in L(\mathcal{A})$ recognized by the automaton, instead, labels exactly one run, which is successful for $\mathcal{A}$.

The chapter is organized as follows: the first section analyzes deterministic and nondeterministic FAs which accept languages of finite words, and shows how these two classes of automata have the same recognition power.
The subsequent section deals with ω-FAs and analyzes the expressive power of BFA with (F) acceptance condition and of MFA.
Last, the final section summarizes the hierarchy of automata.

4.1 FAs on finite words

FLs define a subclass of Deterministic Context-Free (DCF) languages; thus a model of automata recognizing exactly this family of languages must be determinizable.
Actually, FAs perfectly match the generative power of Floyd grammars, as it has been

mentioned before (in Section 2.3), but the classical construction for deterministic regular automata does not fit for FAs and it is not trivial to define a deterministic model equivalent to a given nondeterministic FA.
Thus, this section will consider the classical determinization algorithms, showing how they are not suitable for FAs, and ends with the formalization of a deterministic version for FAs.

Let $\mathcal{A} = \langle \Sigma, M, Q, I, F, \delta \rangle$ be a nondeterministic FA.
Following the usual power-set construction, one obtains a deterministic automaton $\tilde{\mathcal{A}} = \langle \Sigma, M, \tilde{Q}, \tilde{I}, \tilde{F}, \tilde{\delta} \rangle$ where:

- $\tilde{Q} = 2^Q$ is the set of subsets of Q,
- $\tilde{I} = I \in \tilde{Q}$ is the set of initial states of $\mathcal{A}$,
- $\tilde{F} = \{J \subseteq Q \mid J \cap F \neq \emptyset\} \subseteq \tilde{Q}$, i.e. $\tilde{F}$ is the set of subsets of Q containing at least one final state of $\mathcal{A}$,
- $\tilde{\delta} : \tilde{Q} \times (\Sigma \cup \tilde{Q}) \to \tilde{Q}$ is the transition function defined as follows. The push transition $\tilde{\delta}_{\text{push}} : \tilde{Q} \times \Sigma \to \tilde{Q}$ is defined in the natural way by

$$\tilde{\delta}_{\text{push}}(J, a) = \bigcup_{p \in J} \delta(p, a);$$

  The flush transition $\tilde{\delta}_{\text{flush}} : \tilde{Q} \times \tilde{Q} \to \tilde{Q}$ is defined likewise by

$$\tilde{\delta}_{\text{flush}}(J, K) = \bigcup_{p \in J, q \in K} \delta_{\text{flush}}(p, q)$$

This model, however, does not match the expressive power of nondeterministic FAs: in fact, the construction may imply the existence of accepting runs in the deterministic automaton that do not correspond to any successful run in the nondeterministic one, or conversely that accepting runs in $\mathcal{A}$ do not have any counterpart in $\tilde{\mathcal{A}}$. The non equivalence between the two automata, in general, is due to the fact that some paths, which are kept distinct in $\mathcal{A}$, can be blithely merged in $\tilde{\mathcal{A}}$, thus leading to inconsistent computations.
The inadequacy of the previous construction is, for instance, proved by the following example.

Example 4.1 *Consider the nondeterministic FA $\mathcal{A} = \langle \Sigma, M, Q, I, F, \delta \rangle$ defined on the alphabet $\Sigma = \{a, \underline{a}, b, c\}$ which recognizes the language:*

$$L(\mathcal{A}) = \{c(a\underline{a})^n \mid n \geq 0\} \cup \{cb^m \mid m \geq 0\}$$

The nondeterministic automaton, together with its OPM, is depicted in Figure 4.1. The final states of the automaton are B and D.

	a	$\underline{a}$	b	c	#
a		$\doteq$			
$\underline{a}$	$\gtrdot$		$\gtrdot$		$\gtrdot$
b			$\doteq$		$\gtrdot$
c	$\lessdot$		$\lessdot$		$\gtrdot$
#				$\lessdot$	$\doteq$

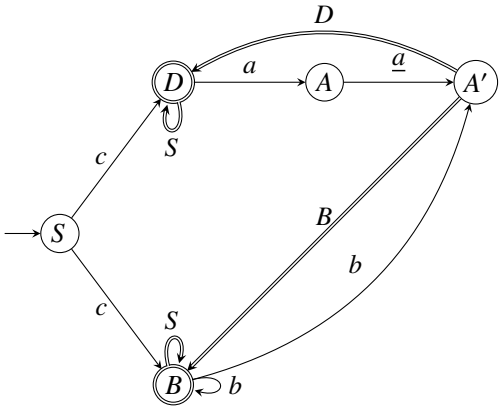


Figure 4.1: Nondeterministic automaton $\mathcal{A}$ of Example 4.1.

Notice also that the fact that the precedence matrix is not acyclic is not relevant: the example would be still valid even if the relation $b \doteq b$ were not defined.
As an example of the behaviour of the automaton, the words $x_0 = ca\underline{a}$ and $x_1 = cbbb$ are accepted, whereas $x_2 = ca\underline{a}b$ does not belong to $L(\mathcal{A})$. The computations of the automaton on these strings are listed in Figure 4.2.

$x_0 = ca\underline{a} \in L(\mathcal{A})$

	$\langle[\# S]$	,	$ca\underline{a}\#\rangle$
mark	$\langle[\# S][c' D]$	,	$a\underline{a}\#\rangle$
mark	$\langle[\# S][c' D][a' A]$	,	$\underline{a}\#\rangle$
push	$\langle[\# S][c' D][a' A][\underline{a} A']$	,	$\#\rangle$
flush	$\langle[\# S][c' D]$	,	$\#\rangle$
flush	$\langle[\# D]$	,	$\#\rangle$

$x_1 = cbbb \in L(\mathcal{A})$

	$\langle[\# S]$	,	$cbbb\#\rangle$
mark	$\langle[\# S][c' B]$	,	$bbb\#\rangle$
mark	$\langle[\# S][c' B][b' B]$	,	$bb\#\rangle$
push	$\langle[\# S][c' B][b' B][b B]$	,	$b\#\rangle$
push	$\langle[\# S][c' B][b' B][b B][b A']$	,	$\#\rangle$
flush	$\langle[\# S][c' B]$	,	$\#\rangle$
flush	$\langle[\# B]$	,	$\#\rangle$

$x_2 = ca\underline{a}b$ does not belong to L($\mathcal{A}$)

	$\langle[\# S]$	,	$ca\underline{a}b\#\rangle$
mark	$\langle[\# S][c' D]$	,	$a\underline{a}b\#\rangle$
mark	$\langle[\# S][c' D][a' A]$	,	$\underline{a}b\#\rangle$
push	$\langle[\# S][c' D][a' A][\underline{a} A']$	,	$b\#\rangle$
flush	$\langle[\# S][c' D]$	,	$b\#\rangle$

stop, without accepting.
The automaton cannot recognize word x_2 since transition $\delta_{\text{push}}(D, b)$ is undefined.

Figure 4.2: Computations for strings x_0, x_1 and x_2.

Now, the automaton $\tilde{\mathcal{A}}$ obtained with the classical determinization algorithm is depicted in Figure 4.3. However, $\tilde{\mathcal{A}}$ is not equivalent to $\mathcal{A}$: in fact $\tilde{\mathcal{A}}$ accepts word x_2, differently from $\mathcal{A}$. The accepting computation of the deterministic automaton on x_2 is reported below (Figure 4.4).

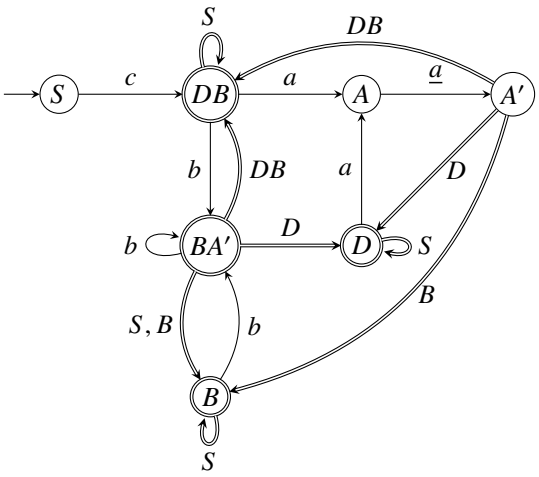


Figure 4.3: Deterministic automaton $\tilde{\mathcal{A}}$ of Example 4.1.

$x_2 = ca\underline{a}b$ does not belong to $\mathrm{L}(\mathcal{A})$, but is recognized by $\tilde{\mathcal{A}}$.

	$\langle[\# S]$	,	$ca\underline{a}b\#\rangle$
mark	$\langle[\# S][c' DB]$	,	$a\underline{a}b\#\rangle$
mark	$\langle[\# S][c' DB][a' A]$	,	$\underline{a}b\#\rangle$
push	$\langle[\# S][c' DB][a' A][\underline{a} A']$	,	$b\#\rangle$
flush	$\langle[\# S][c' DB]$	,	$b\#\rangle$
mark	$\langle[\# S][c' DB][b' BA']$	,	$\#\rangle$
flush	$\langle[\# S][c' DB]$	,	$\#\rangle$
flush	$\langle[\# DB]$	,	$\#\rangle$

The string is erroneously accepted since DB is a final state for the automaton.

Figure 4.4: Computation of $\tilde{\mathcal{A}}$ for string x_2.

Since the natural definition of the transition function in the classical determinization algorithm is not adequate, it is therefore necessary to restrict the push and the flush functions so as to avoid erroneous computations of the deterministic automaton.
A possible alternative algorithm to the classical one previously described may, for instance, try to guarantee that the deterministic automaton $\tilde{\mathcal{A}}$ mirrors the evolution of the stack of the nondeterministic one, allowing only flush transitions towards those states that do not generate bifurcations in the runs (i.e. defining the flush transition function only on pairs $(J, K) \in \tilde{Q} \times \tilde{Q}$ such that the states in J have a unique corresponding image for all the states in K). But, in general, even this approach fails to guarantee equivalence between deterministic and nondeterministic automata.
Likewise, other possible constructions, that constrain the flush function of the deterministic automaton $\tilde{\mathcal{A}}$ on the basis of reachability information on the state graph of $\mathcal{A}$,

either leads to contradictions as regards acceptance of the strings or may undermine determinism.

The key point to define a suitable deterministic FA, equivalent to a given nondeterministic one, is that it is necessary to keep in the states all the information necessary to properly reduce a portion of the input word after a flush, so as to correctly identify the state to be reached by the flush transition and from which the parsing of the string will be resumed. This is consistent with the classical approach of LR parsers, that follow several simultaneous computations on the word in input, keeping information useful to disambiguate, when a reduction has to be performed, the different directions followed while the string has been read. Indeed, the parsing of a FA on an given word mirrors the typical operations of shift-reduce of LR parsers: each mark or push move corresponds to a shift, at the top of the stack, of the current symbol and state, while flush transitions reduce a recognized portion of the input string. It is therefore quite natural to consider a construction that takes inspiration from this modus operandi.

In general, a nondeterministic FA may take different runs on a given word. In the classical power-set construction of a deterministic automaton $\tilde{\mathcal{A}}$ for a given nondeterministic FA $\mathcal{A}$, each mark/push move along the run of $\tilde{\mathcal{A}}$ (for a given input symbol) computes a state which is the collection of the states reached by the corresponding nondeterministic automaton $\mathcal{A}$ reading the same letter. However, considering only a set of states and neglecting the evolution of the stack along the run cannot work: it is fundamental that the deterministic automaton may find, among the states of the nondeterministic automaton pushed on the stack, the starting point of the path that led to the recognition of the portion of input string that has to be reduced (that is identified by the r.h.s of a rule). If the deterministic automaton cannot identify which states belong to the same run, then a flush transition may erroneously blend different paths of $\mathcal{A}$, allowing acceptance of words that do not belong to $L(\mathcal{A})$.
Now, the recognition of the r.h.s of a rule has its beginning with a mark move followed by a sequence (possibly empty) of push transitions. Notice also that there can be nested flush, interspersed among the push moves, and hence, if one considers the parsing from the point of view of the derivations of a FG, the r.h.s to be reduced may be a sentential form of terminals among nonterminal symbols.
In order to keep track of the starting point from which a mark has been performed, a possible solution implies that each state of the nondeterministic automaton is augmented with summary information on the run followed until that state. More precisely, the states of $\tilde{\mathcal{A}}$ may be defined by pairs of states instead of single states of $\mathcal{A}$: $\tilde{\mathcal{A}}$ may simulate $\mathcal{A}$ along the first component of the pair, whereas the second component stores the state that gave origin to a mark transition and it is propagated during push moves. Notice, though, that a state of $\tilde{\mathcal{A}}$ is a set of pairs, since $\mathcal{A}$ is nondeterministic.
The adequacy of the construction is guaranteed, even in presence of nested reductions, if each flush transition in $\tilde{\mathcal{A}}$ restores together with the state, say q, originating a mark move for the current reduction, even the starting point of a mark transition leading to q itself. By forcing the chaining of pairs of states which give rise to mark transitions, it is possible to keep track, from the current state in a run up to the initial state, of the points in the path followed by the nondeterministic automaton which are critical to perform reductions correctly, guaranteeing a proper correspondence among runs in the nondeterministic and the deterministic automata.

Following this approach, a nondeterministic FA can be turned into an equivalent deterministic one as follows.

Given a nondeterministic automaton $\mathcal{A} = \langle \Sigma, M, Q, I, F, \delta \rangle$, consider the deterministic automaton $\tilde{\mathcal{A}} = \langle \Sigma, M, \tilde{Q}, \tilde{I}, \tilde{F}, \tilde{\delta} \rangle$ where:

- $\tilde{Q} = 2^{Q \times (Q \cup \{\perp\}) \times \hat{\Sigma}}$, where $\hat{\Sigma} = (\Sigma \cup \{\#\})$ and $Q \cap \{\perp\} = \emptyset$ and $\perp$ is a symbol that stands for the baseline of the computations (i.e. the pseudo-state before the initial states),
- $\tilde{I} = I \times \{\perp\} \times \{\#\}$ is the set of initial states of $\tilde{\mathcal{A}}$,
- $\tilde{F} = \{J \in \tilde{Q} \mid \exists \langle q, \perp, \# \rangle \in J : q \in F\} \subseteq \tilde{Q}$,
- $\tilde{\delta} : \tilde{Q} \times (\Sigma \cup \tilde{Q}) \to \tilde{Q}$ is the transition function defined as follows. The push transition $\tilde{\delta}_{\text{push}} : \tilde{Q} \times \Sigma \to \tilde{Q}$ is defined by

$$\tilde{\delta}_{\text{push}}(J, a) = \bigcup_{\langle q,p,b \rangle \in J} \{\langle h, t, a \rangle \mid h \in \delta_{\text{push}}(q, a)\},$$

  where $t = \begin{cases} q & \text{if } b \lessdot a \quad \text{(this corresponds to a mark move)} \\ p & \text{if } b \doteq a \quad \text{(this corresponds to a push move)} \end{cases}$

  The flush transition $\tilde{\delta}_{\text{flush}} : \tilde{Q} \times \tilde{Q} \to \tilde{Q}$ is defined as follows:

$$\tilde{\delta}_{\text{flush}}(J, K) = \bigcup_{\langle q,p,b \rangle \in J, \langle p,r,c \rangle \in K} \{\langle h, r, c \rangle \mid h \in \delta_{\text{flush}}(q, p)\}.$$

Notice that, if $n = |Q|$ is the number of states of the nondeterministic automaton $\mathcal{A}$, the deterministic automaton $\tilde{\mathcal{A}}$ that is obtained in this way has a set of states whose size is exponential in n^2, i.e. $|\tilde{Q}| = 2^{|Q||Q \cup \{\perp\}||\Sigma \cup \{\#\}|}$. This construction can be considerably optimized noting that, for each state $\{\langle q, p, a \rangle\} \in \tilde{Q}$, all the possible pairs (q, p) share the symbol $a \in \Sigma$. This is due to the fact that a state in $\tilde{Q}$ can be reached only with a mark/push/flush transition: as regards mark/push moves, given a state $J \in \tilde{Q}$ and a letter $a \in \Sigma$, it holds that $\tilde{\delta}_{\text{push}}(J, a) = \bigcup_{\langle q,p,b \rangle \in J} \{\langle h, t, a \rangle \mid h \in \delta_{\text{push}}(q, a)\}$, thus all the triples $\langle h, t, a \rangle$ in the new computed state $\tilde{\delta}_{\text{push}}(J, a)$ share the same character a. Moreover, as regards flush transitions, assuming that, when the flush has to be performed, all the states on the stack of the automaton have triples sharing the same third component in Σ, then the construction computes a state $\tilde{\delta}_{\text{flush}}(J, K) = \bigcup_{\langle q,p,b \rangle \in J, \langle p,r,c \rangle \in K} \{\langle h, r, c \rangle \mid h \in \delta_{\text{flush}}(q, p)\}$ composed of triples $\langle h, r, c \rangle$ with $h \in \delta_{\text{flush}}(q, p)$ whose letter c is the same for all.
A refined version of the construction can assign to $\tilde{\mathcal{A}}$ a set of states $\tilde{Q} = \hat{\Sigma} \times 2^{Q \times (Q \cup \{\perp\})}$, bounding the size of the set to $|\tilde{Q}| = |\Sigma \cup \{\#\}| \cdot 2^{\Theta(n^2)}$.

This improved construction for the deterministic automaton, along with the proof of correctness, is presented in [16].
Finally, the above constructions allow to state the following theorem [17]:

Theorem 4.1 *Deterministic Floyd automata are equivalent to nondeterministic ones.*

4.2 ω-FAs

This section compares the expressive power of nondeterministic BFA (BFA), deterministic BFA (BFAD), nondeterministic MFA (MFA) and deterministic MFA (MFAD), where, as regards BFA and BFAD, the acceptance condition (F) is taken into account,

given that it is strictly more powerful than the (E) recognition mode. The relationships among the classes of languages recognized by these families of automata (that are denoted as $L_{BFA}, L_{BFAD}, L_{MFA}, L_{MFAD}$ respectively) differ from the relationships holding among the languages recognized by the corresponding classical families of regular ω-automata and, instead, show several similarities with the languages accepted by ω-VPAs.
Indeed, it is a fundamental result of traditional theory on ω-finite-state automata that nondeterministic Büchi automata, nondeterministic and deterministic Muller automata recognize the same class of ω-languages (called *ω-regular or ω-rational* in literature, see e.g. [24], [20]), whereas deterministic Büchi automata are a proper subclass of nondeterministic Büchi automata.

For VPAs on infinite words, instead, the influential paper [2] showed that the classical determinization algorithm of Büchi automata into deterministic Muller automata is no longer valid, and deterministic Muller automata are strictly less powerful than the former class of abstract machines. A similar relationship holds also for ω-FAs, as it will be proved later on, whereas the comparison among the other families of devices mirrors the classical theory on ω-finite state machines.

4.2.1 Comparison between BFA and MFA

Theorem 4.2 *BFA and MFA recognize the same class of languages, i.e. $L_{BFA} = L_{MFA}$.*

This theorem follows from the two subsequent lemmas.

Lemma 4.3 $L_{BFA} \subseteq L_{MFA}$
Each BFA $\mathcal{B} = \langle \Sigma, M, Q, I, F, \delta \rangle$ is equivalent to a MFA $\mathcal{A} = \langle \Sigma, M, Q, I, \mathcal{T}, \delta \rangle$ whose acceptance component $\mathcal{T}$ consists of all subsets of Q including some F-state of B, namely

$$\mathcal{T} = \{P \subseteq Q \mid P \cap F \neq \emptyset\}.$$

Clearly the two automata $\mathcal{A}$ and $\mathcal{B}$ have a successful run on the same words $x \in \Sigma^\omega$, as it happens for classical ω-finite-state machines.

Lemma 4.4 $L_{BFA} \supseteq L_{MFA}$
Any ω-language recognized by a MFA $\mathcal{A}$ can be recognized by a BFA $\mathcal{B}$.

Proof Consider a MFA $\mathcal{A} = \langle \Sigma, M, Q, I, \mathcal{T}, \delta \rangle$ which recognizes the language $L(\mathcal{A})$. It can be assumed, without loss of generality, that $\mathcal{T}$ is a singleton. In fact it is easy to see that $L(\mathcal{A})$ can be expressed as

$$L(\mathcal{A}) = \bigcup_{T \in \mathcal{T}} L(\mathcal{A}_T)$$

where $\mathcal{A}_T = \langle \Sigma, M, Q, I, T, \delta \rangle$ is a MFA whose table corresponds to the singleton set $T \in \mathcal{T}$.
Since the class of languages recognized by BFA, L_{BFA}, is closed under union (a property that will be proved later, see Chapter 5), then, if each language $L(\mathcal{A}_T)$ might be accepted by a BFA, then $L(\mathcal{A})$ would be accepted by a BFA as well.
Now, assume that $\mathcal{T} = T \subseteq Q$ is a singleton.
Then, there exists a BFA $\mathcal{B} = \langle \Sigma, M, \tilde{Q}, I, F, \tilde{\delta} \rangle$ which accepts the same language as $\mathcal{A}$.
The set of states of $\mathcal{B}$ includes elements of two types: states $q \in Q$ of $\mathcal{A}$ and states

(q, R) where $q \in Q$ is a state of $\mathcal{A}$ and R is a set of states (called "box") that will be used to test whether the run of $\mathcal{A}$ is successful.
Intuitively, the automaton $\mathcal{B}$ simulates $\mathcal{A}$, during the parsing of the input string, along a sequence of states q, and then guesses nondeterministically the point after which a successful run $\mathcal{S}$ of $\mathcal{A}$ on x stops visiting the states that occur only finitely often in the run, and $\mathcal{S}$ begins to visit all and only the states in the set T infinitely often.
After this point $\mathcal{B}$ switches to the states of the form (q, R) and collects in the component R of the states (the "box") the states visited by $\mathcal{A}$ during the run, emptying the box as soon as it contains exactly the set T. Every time it empties the box, $\mathcal{B}$ resumes collecting the states that $\mathcal{A}$ will visit from that point onwards. If the final states of $\mathcal{B}$ are defined as those ones when it collects exactly the set T, then $\mathcal{B}$ will visit infinitely often these final states iff $\mathcal{A}$ visits all and only the states in T infinitely often.

More formally, $\mathcal{B}$ is defined by:

- $\tilde{Q} = Q \cup (Q \times 2^Q)$,
- $F = \{(q, T) \mid q \in T\}$,
- $\tilde{\delta} : \tilde{Q} \times (\Sigma \cup \tilde{Q}) \to 2^{\tilde{Q}}$ is the transition function defined as follows. The push transition $\tilde{\delta}_{\text{push}} : \tilde{Q} \times \Sigma \to 2^{\tilde{Q}}$ is defined by:
  - $\tilde{\delta}_{\text{push}}(q, a) = \delta_{\text{push}}(q, a) \cup \{\langle p, \{p\}\rangle \mid p \in \delta_{\text{push}}(q, a)\} \quad \forall q \in Q, a \in \Sigma$
    where a state in the first set $\delta_{\text{push}}(q, a)$ is chosen when $\mathcal{B}$ simulates $\mathcal{A}$ until the point mentioned above, whereas the automaton chooses non deterministically to reach a state of the form $\langle p, \{p\}\rangle$ if it guesses that the subsequent states will be all and only states of T.
  - $\tilde{\delta}_{\text{push}}(\langle q, R\rangle, a) = \begin{cases} \left\{\langle p, R \cup \{p\}\rangle \mid p \in \delta_{\text{push}}(q, a)\right\} & \text{if } R \neq T \\ \left\{\langle p, \{p\}\rangle \mid p \in \delta_{\text{push}}(q, a)\right\} & \text{if } R = T \end{cases}$
    $\forall q \in Q, R \subseteq Q, a \in \Sigma$
    i.e. the box is emptied only when it equals T.

  The flush transition $\tilde{\delta}_{\text{flush}} : \tilde{Q} \times \tilde{Q} \to 2^{\tilde{Q}}$ is defined by:
  - $\tilde{\delta}_{\text{flush}}(q_1, q_2) = \delta_{\text{flush}}(q_1, q_2) \cup \{\langle p, \{p\}\rangle \mid p \in \delta_{\text{flush}}(q_1, q_2)\}$,
    $\forall q_1, q_2 \in Q$
  - $\tilde{\delta}_{\text{flush}}(\langle q_1, R\rangle, q_2) = \begin{cases} \{\langle p, R \cup \{p\}\rangle \mid p \in \delta_{\text{flush}}(q_1, q_2)\} & \text{if } R \neq T \\ \{\langle p, \{p\}\rangle \mid p \in \delta_{\text{flush}}(q_1, q_2)\} & \text{if } R = T \end{cases}$.
  - $\tilde{\delta}_{\text{flush}}(\langle q_1, R_1\rangle, \langle q_2, R_2\rangle) = \begin{cases} \{\langle p, R_1 \cup \{p\}\rangle \mid p \in \delta_{\text{flush}}(q_1, q_2)\} & \text{if } R_1 \neq T \\ \{\langle p, \{p\}\rangle \mid p \in \delta_{\text{flush}}(q_1, q_2)\} & \text{if } R_1 = T \end{cases}$.
    $\forall q_1, q_2 \in Q, R, R_1, R_2 \subseteq Q$.
    where the explanation for the choice among the various cases is similar to the previous comment for $\tilde{\delta}_{\text{push}}$.

The two automata accept the same language.
First, let's show that $L(\mathcal{A}) \subseteq L(\mathcal{B})$.
Let $x \in \Sigma^\omega$ be an infinite word belonging to $L(\mathcal{A})$; then $\mathcal{A}$ has a successful run $\mathcal{S}$ on x.
There exists a finite prefix $v \in \Sigma^+$ of $x = vu_1u_2\ldots$ such that the infinite path followed

by $\mathcal{A}$ after reading v (i.e. on the infinite word $u_1u_2\ldots$) visits all and only states in T infinitely often. Thus, the run $\mathcal{S}$ can be written as: $\mathcal{S} = \langle \alpha_0 = [\#\, q_0]\ ,\ x = vu_1u_2\ldots\rangle \overset{+}{\vdash} \langle \alpha_{|v|}\ ,\ u_1u_2\ldots\rangle \overset{+}{\vdash} \ldots \overset{+}{\vdash} \langle \alpha_i\ ,\ u_i\ldots\rangle \overset{+}{\vdash} \ldots$ where $\{state(\alpha_i) \mid i \geq |v|\} = T$ and $q_0 \in I$.
Then, there is a successful run $\tilde{\mathcal{S}}$ of $\mathcal{B}$ on the same word, which follows singleton states of $\mathcal{A}$ while it reads v,
$\tilde{\mathcal{S}} = \langle \beta_0 = \alpha_0 = [\#\, q_0]\ ,\ x = vu_1u_2\ldots\rangle \overset{+}{\vdash} \langle \beta_{|v|} = \alpha_{|v|}\ ,\ u_1u_2\ldots\rangle$
and then switches to states augmented with a box:
$\langle \beta_{|v|} = \alpha_{|v|}\ ,\ u_1u_2\ldots\rangle \vdash \langle \beta_{|v|+1} = \gamma[a\ \langle p, \{p\}\rangle]\ ,\ \tilde{u_1}u_2\ldots\rangle$,
where $\langle \alpha_{|v|}\ ,\ u_1u_2\ldots\rangle \vdash \langle \alpha_{|v|+1} = \gamma[a\ p]\ ,\ \tilde{u_1}u_2\ldots\rangle$. Since after this point $\mathcal{A}$ visits each state in T and only these states infinitely often, $\mathcal{B}$ will reach infinitely often final states $(q, T) \in F$, emptying infinitely often its box after them and resuming the collection of states with the subsequent state in the run.

Conversely, it holds that $L(\mathcal{B}) \subseteq L(\mathcal{A})$.
Let $x \in \Sigma^\omega$ be an infinite word in $L(\mathcal{B})$.
Define the projection $\pi_1 : Q \cup (Q \times 2^Q) \to Q$ as $\pi_1(q) = q$ and $\pi_1(\langle q, R\rangle) = q, \forall q \in Q, R \subseteq Q$. Given a run $\tilde{\mathcal{S}}$ of the automaton $\mathcal{B}$, let $\pi_1(\tilde{\mathcal{S}})$ be the natural extension of π_1, i.e. if in general
$\tilde{\mathcal{S}} = \langle \beta_0 = [\#\, r_0][a_1\ r_1]\ldots[a_{n0}\ r_{n0}]\ ,\ x_1x_2\ldots\rangle \overset{+}{\vdash} \langle \beta_i = [b_1\ p_1]\ldots[b_{n1}\ p_{n1}]\ ,\ x_2\ldots\rangle \overset{+}{\vdash} \ldots$,
with $r_i, p_i \in Q \cup (Q \times 2^Q), a_i, b_i \in (\Sigma \cup \Sigma')$, then
$\pi_1(\tilde{\mathcal{S}}) = \langle [\#\, \pi_1(r_0)][a_1\ \pi_1(r_1)]\ldots[a_{n0}\ \pi_1(r_{n0})]\ ,\ x_1x_2\ldots\rangle \overset{+}{\vdash} \ldots$.
Clearly, by construction, if $\tilde{\mathcal{S}}$ is a run of $\mathcal{B}$ on an ω-word, then $\pi_1(\tilde{\mathcal{S}}) = \mathcal{S}$ is a run for $\mathcal{A}$ on the same word.
Now, let $\tilde{\mathcal{S}}$ be a successful run for $\mathcal{B}$ on x; $\mathcal{S} = \pi_1(\tilde{\mathcal{S}})$ is a run for $\mathcal{A}$ on x. Furthermore, since only the states augmented with a box are final states, then after a sequence (possibly empty) of singleton states initially traversed by $\mathcal{B}$, the automaton will definitively visit only states of the form (q, R). (Notice, in fact, that from such kind of states no singleton state is reachable again).
By induction on the number of final states reached by $\mathcal{B}$ along its run, it can be proved that, for each pair of final states consecutively reached by $\mathcal{B}$, say (q_{F_i}, R_i) and $(q_{F_{i+1}}, R_{i+1})$, the portion of the run visited between them, let's say $\tilde{\mathcal{S}}_i$, is such that $\pi_1(\tilde{\mathcal{S}}_i) = T$.
Finally, since final states in $\tilde{\mathcal{S}}$ are visited infinitely often, the run $\pi_1(\tilde{\mathcal{S}})$ is successful for $\mathcal{A}$. □

4.2.2 Comparison between BFAD and MFAD

Theorem 4.5 *BFAD are strictly less expressive than MFAD.*

Lemma 4.3 holds likewise for BFAD and MFAD:

Lemma 4.6 $L_{BFAD} \subseteq L_{MFAD}$
Each BFAD $\mathcal{B} = \langle \Sigma, M, Q, q_0, F, \delta\rangle$ is equivalent to a MFAD $\mathcal{A} = \langle \Sigma, M, Q, q_0, \mathcal{T}, \delta\rangle$ whose acceptance component $\mathcal{T}$ consists of all subsets of Q including some F-state of $\mathcal{B}$, namely

$$\mathcal{T} = \{P \subseteq Q \mid P \cap F \neq \emptyset\}.$$

Furthermore, in order to prove that the inclusion is strict, it suffices to consider that, given the alphabet $\Sigma = \{a, b\}$, the language

$$L = \{\alpha \in \Sigma^\omega : \alpha \text{ contains finitely many letters a }\} \tag{4.1}$$

is recognized by a MFAD, but no BFAD can accept it.

First of all, a MFAD that recognizes L is the deterministic automaton $\mathcal{A} = \langle \Sigma, M, Q, q_0, \mathcal{T}, \delta \rangle$ on $\Sigma = \{a, b\}$, with $\mathcal{T} = \{\{q_0\}\}$ and OPM M and state-graph depicted in the following Figure (4.5).

	a	b
a	$\lessdot$	$\lessdot$
b	$\lessdot$	$\lessdot$
#	$\lessdot$	$\lessdot$

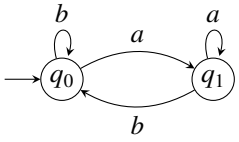


Figure 4.5: MFAD $\mathcal{A}$, with its OPM, of Theorem 4.5.

An example of run of the automaton on a string in L, for instance $x = aababaa(b)^\omega$ is:

	$\langle [\# \; q_0]$	,	$aababaabbbb...\rangle$
mark	$\langle [\# \; q_0][a' \; q_1]$	,	$ababaabbbb...\rangle$
mark	$\langle [\# \; q_0][a' \; q_1][a' \; q_1]$	,	$babaabbbb...\rangle$
...			
mark	$\langle [\# \; q_0][a' \; q_1][a' \; q_1][b' \; q_0][a' \; q_1][b' \; q_0][a' \; q_1][a' \; q_1]([b' \; q_0])$	,	$b^\omega...\rangle$
mark	$\langle [\# \; q_0][a' \; q_1][a' \; q_1][b' \; q_0][a' \; q_1][b' \; q_0][a' \; q_1][a' \; q_1]([b' \; q_0])^+$	,	$b^\omega...\rangle$

The proof that no BFAD can recognize L resembles the classical proof (see [24]) that deterministic Büchi automata are strictly weaker than nondeterministic Büchi ones. Assume that there exists a BFAD $\mathcal{B}$ which recognizes L.

Notice that, in general, according to the definition of push/mark/flush moves of a Floyd automaton (finite or ω), given any configuration $C = \langle \beta \; , \; w \rangle$, the state piled up at the top of the stack with a transition $\langle \beta \; , \; w \rangle \vdash \langle \beta' \; , \; w' \rangle$, namely $state(\beta')$, is exactly the state reached by the automaton on its state-graph. Thus, during a run on a word $x \in \Sigma^\omega$, configurations with stack β_i with $state(\beta_i) \in F$ occur infinitely often iff the automaton visits infinitely often states in F in its graph.

Now, the infinite word $x = b^\omega$ belongs to L, since it contains no (and then a finite number of) letters a. Then, there exists a unique run of $\mathcal{B}$ on this string which visits infinitely often final states. Let b^{n_1} be the prefix read by $\mathcal{B}$ until the first visited final state.

But also $b^{n_1}ab^\omega$ belongs to L, hence there exists a final state reached reading the prefix $b^{n_1}ab^{n_2}$, for some $n_2 \in N$.

In general, one can find a sequence of finite words $b^{n_1}ab^{n_2} \dots ab^{n_k}, (k \geq 1)$ such that the automaton has a unique run on them, and for each such runs it reaches a final state (placing it at the top of the stack) after reading every prefix $b^{n_1}ab^{n_2} \dots ab^{n_i}, \forall \; i \leq k$. Therefore, there exists a (unique) run of $\mathcal{A}$ on the ω-word $w = b^{n_1}ab^{n_2} \dots$ such that $\mathcal{A}$ visits infinitely often final states, and thus reaches infinitely often configurations $C = \langle \beta_i \; , \; w_i \rangle$ with $state(\beta_i) \in F$.

However, w cannot be accepted by $\mathcal{B}$ since it contains infinitely many letters a, and this is a contradiction.

4.2.3 Comparison between BFA and BFAD

The comparison between BFAD and BFA easily follows from the fact that there exists a BFA recognizing language (4.1), i.e. the automaton in the following figure.

Thus, the inclusion is strict.

	a	b
a	$\lessdot$	$\lessdot$
b	$\lessdot$	$\lessdot$
$\#$	$\lessdot$	$\lessdot$

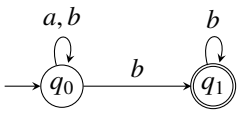


Figure 4.6: BFA for language L (Equation 4.1), with its OPM.

4.2.4 Comparison between BFA and MFAD

The following theorem shows that MFAD are strictly less expressive than BFA, as it also happens for ω-VPA.
Notice that it has already been proved that MFAD are a subclass of BFA, as MFA equal BFA.

The proof of the theorem presented below resembles the analogous proof for ω-VPA in [2]; indeed, that proof is essentially based on topological properties of the state-graph of the automata and it is general enough to adapt to both ω-VPA and ω-FA. Hence, the proof is reported emphasizing those aspect that characterize ω-FA.

Theorem 4.7 *Let L_{repbdd} be the language consisting of ω-words n on the alphabet $\Sigma = \{c, r\}$ such that n has only finitely many pending calls (i.e. letters c not matched by any subsequent corresponding letter r). The language L_{repbdd} is accepted by a BFA, but cannot be accepted by any MFAD.*

Proof A BFA which accepts this language is the nondeterministic automaton $\mathcal{B} = \langle \Sigma, M, Q_B, I_B, F, \delta_B \rangle$ whose precedence matrix M and state graph are:

	c	r
c	$\lessdot$	$\doteq$
r	$\gtrdot$	$\gtrdot$
$\#$	$\lessdot$	

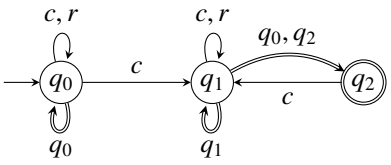


Figure 4.7: BFA automaton recognizing L_{repbdd}

The automaton parses the input string in state q_0 and nondeterministically finds the point in the infinite word from which no pending calls will be found, and this choice corresponds to the transition to state q_1. Then, $\mathcal{B}$ reaches infinitely often a final state iff it reads infinitely often words in $L_{Dyck}(c, r)$.

In order to prove that L_{repbdd} is not recognizable by any MFAD, assume that, instead, there is a MFAD $\mathcal{A} = \langle \Sigma, M, Q, q_0, \mathcal{T}, \delta \rangle$ that accepts this language, where the OPM M is the same as the precedence matrix of $\mathcal{B}$.

Let G be the classical state-graph of the automaton: a word on Σ is said to be *well-matched* (wmw) iff it belongs to the Dyck language on the pairs c, r; a state $q' \in Q$ is said to be reachable from $q \in Q$ in G by a wmw n if there is a sequence of transitions that leads $\mathcal{A}$ from q to q' reading n and ending with a flush.
Then, define two other graphs, which summarizes reachability information on G.
Let $G_1 = (Q, \longrightarrow)$ be a directed graph for $\mathcal{A}$ where there exists an edge $q \longrightarrow q'$ iff q' is reachable from q on G by a wmw n. Edges in G_1 are called summary edges.

Notice that, if q is a state reachable in G by the automaton reading some finite word $x \in \Sigma^*$, then there must be an outgoing edge from q in G_1. In fact, if n is a wmw, then $x(n)^\omega$ contains a finite number of pending calls (those that can be, possibly, in x) and this ω-word must belong to $L(\mathcal{A})$. Therefore, there must be a path in G recognizing it (incidentally, this path is unique since $\mathcal{A}$ is deterministic) and there must be an edge in G_1 departing from q reading a wmw n^+. Note also that G_1 is transitively closed since concatenation of wmws is well-matched. Now, consider the strongly connected components (SCC) of G_1, which are the sets of states mutually reachable by reading wmws. Define a *sink* SCC of G_1 as a strongly connected component S such that there's no state reachable by states in S, reading a wmw, that does not belong to S, i.e. for every $q \in S$ and edge $(q, q') \in G_1$, then $q' \in S$.

Notice that a sink SCC must always exist. Assuming in fact the contrary, if no SCC can be a sink, then for every SCC S_i there must be a wmw n_i that leads from a state in S_i to a state $\tilde{q}_i \notin S_i$, and S_i must not be reachable by $\tilde{q}_i$ with any wmw either, by transitive closure. Consider then a SCC S_1 and take the edge in G_1, corresponding n_1, to reach $\tilde{q}_1$. If $\mathcal{A}$ reads from now on only wmws $(n)^\omega$ it should reach a SCC (according to Muller acceptance condition); then there are two cases: either it reaches S_1 again, and this is a contradiction, or it reaches another SCC S_2. Repeating this reasoning again for S_2, and noting that SCCs are in finite number, a contradiction follows as well.

As mentioned above, another graph is considered for $\mathcal{A}$, $G_2 = (Q, \Rightarrow)$ which is a supergraph of G_1 augmented with so-called *call-edges*: there is a call-edge (q, q') if there can be a mark move from q to q' in G, reading a symbol c.
Now, the proof is based on a lemma (the same as [2]), which shows how the initial hypothesis, that a MFAD for L_{repbdd} can exist, is not valid.
The lemma can be stated as follows

Lemma 4.8 *(in [2])*
There is a sink SCC S of G_1 and a state $q \in S$ reachable from the initial state q_0 in G_2 such that there is a cycle involving q in G_2 that includes a call-edge.

The lemma derives from the same reasoning considered to claim the existence of a sink SCC in G_1.
More precisely, consider that, from any state, $\mathcal{A}$ can read a sequence of wmws to reach a SCC, and in particular $\mathcal{A}$ can always reach a sink SCC (just reads the wmw that brings away from the SCCs, until a sink is reached).
Assume that from the initial state $q_0 \in Q$ $\mathcal{A}$ follows edges corresponding to wmws to reach a state q_1 belonging to a sink SCC S_1 of G_1. Then, it reads a letter c from q_1, and follows a call-edge from this state towards q_1'. If $q_1' \in S_1$, then there is a cycle from q_1 to q_1 which includes a call-edge, and the lemma then holds. Otherwise, let $\mathcal{A}$ read wmws from q_1' to reach a sink SCC S_2. If $S_2 = S_1$ the lemma has been proved, as before, otherwise the automaton may take again a call-edge from q_2 to a state q_2' and it goes on until a previously visited sink SCC is reached (this must happen as SCCs are in finite number): the lemma follows.

Finally, the lemma allows to definitively prove the main theorem. Consider a finite word $x \in \Sigma^*$ such that, reading it, $\mathcal{A}$ reaches q from q_0, using summary edges and call-edges in G_2, and then after the parsing of x, it follows all summary edges in S infinitely often, reading therefore only an infinite sequence of wmw in Σ^+. The word read in this way has only finitely many pending calls, and then must be accepted. Let Q_S be the set

of all states in Q that are traversed while summary edges in S are followed. Clearly, each summary edge in G_1 corresponds to a sequence of states and edges belonging to the original state-graph G of the automaton, namely to the sequence of mark/flush moves actually performed while reading the wmw.
Since the word is accepted and all and only the states in S are visited infinitely often, then, according to Muller acceptance condition, $Q_S \in \mathcal{T}$. Now consider another ω-word that takes $\mathcal{A}$ from q_0 to q, but then follows the cycle cited in the lemma. Along the cycle, $\mathcal{A}$ can read wmws corresponding to summary edges and calls, but these calls cannot be matched by any return: in fact edges in G_2 model only transitions of calls or paths that can be followed with well-matched words. Let Q' be the set of states visited going from q to q along the cycle, then it can be proved that $Q' \subseteq Q_S$. In fact suppose that $\mathcal{A}$ comes back to q after visiting the cycle, and then begins to visit states in S. It is possible to read from now on a finite word with enough symbol r to match the pending calls, reaching some state, say $\tilde{q}$: then the path followed by $\mathcal{A}$ from q, along the cycle and these further sequence of moves till the state $\tilde{q}$, corresponds to the parsing of a wmw, and there must be a summary edge from q to $\tilde{q}$. But q belongs to a sink SCC and therefore $\tilde{q}$ belongs to the sink too, and then $Q' \subseteq Q_S$ since the states in the cycle are traversed while $\mathcal{A}$ follows a summary edge in S.

Finally, consider the infinite word that first goes from q_0 to q, and then alternately takes the cycle in G_2 to reach q and follows all the summary edges in S to come back to q again. This word has infinitely many unmatched calls and the automaton $\mathcal{A}$ visits all the states in $Q_S \in \mathcal{T}$ infinitely often while reading it, thus it should accept the word, which is a contradiction. □

Remark Notice that this theorem reproves the claim in Section 4.2.3, regarding the relationship between BFAD and BFA, since BFAD ⊂ MFAD ⊂ BFA.

4.3 Hierarchy of ω-FAs

The containment relations among the classes of ω-FAs described in the previous sections of this chapter are summarized in Figure 4.8.
Remind that BFA and BFAD are considered with (F) acceptance condition.

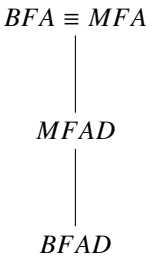


Figure 4.8: Hierarchy for ω-FL. All inclusions are strict.

Finally, concerning some relationships among ω-FAs and other classes of ω-automata, [17] shows that Büchi ω-VPLs are a proper subset of the corresponding ω-FLs. Furthermore, as regards classical ω-languages, ω-finite-state automata on an alphabet Σ can be simulated by ω-FAs which essentially do not resort to the stack for the recognition, i.e. they have the same state-graph of the finite-state devices and their precedence

matrix is such that for each pair of symbols $a, b \in \Sigma$ the precedence relation $a \lessdot b$ holds. Then the ω-FA may pile up on the stack each symbol of the input word, together with the currently reached state, and accepts a word only on the basis of the path followed on the graph, as the ω-finite state automaton does.

Chapter 5

Closure properties

This chapter analyzes the closure properties with respect to classical operations on ω-languages enjoyed by the main classes of ω-FLs, i.e. L_{BFA}, L_{BFAD} and L_{MFAD}.

5.1 Closure properties of BFA

The class of languages accepted by BFA with (F) acceptance condition is closed under intersection and union, and an analogous result holds also for ω-VPA and ω-regular languages.

Theorem 5.1 *Let L_1 and L_2 be ω-languages that can be recognized by two BFA (F) defined over the same alphabet Σ and with compatible precedence matrices M_1 and M_2 respectively. Then $L = L_1 \cap L_2$ is recognizable by a BFA (F) with OPM $M = M_1 \cap M_2$.*

Proof Let $\mathcal{A}_1 = \langle \Sigma, M_1, Q_1, I_1, F_1, \delta_1 \rangle$ and $\mathcal{A}_2 = \langle \Sigma, M_2, Q_2, I_2, F_2, \delta_2 \rangle$ be two BFA (F) with $L(\mathcal{A}_1) = L_1$ and $L(\mathcal{A}_2) = L_2$ and with compatible precedence matrices M_1 and M_2. Suppose, without loss of generality, that Q_1 and Q_2 are disjoint.
Notice that if two OPM are compatible, then if a precedence relation between two letters of the alphabet is defined in both matrices, it is the same. Therefore, during the parsing of the ω-word in input, there can be two cases: if the precedence relation between the last character read and the current symbol is defined in both M_1 and M_2 (and it must be the same, being the matrices compatible) then the automata perform the same type of move (push/mark/flush), otherwise at least one of the two automata stops without accepting the word in input since its transition function is not defined for the current letter.

Now, a BFA (F) which recognizes $L_1 \cap L_2$ is defined in a similar way as for classical finite-state Büchi automata, $\mathcal{A} = \langle \Sigma, M = M_1 \cap M_2, Q, I, F, \delta \rangle$ where:

- $Q = Q_1 \times Q_2 \times \{0, 1, 2\}$,
- $I = I_1 \times I_2 \times \{0\}$,
- $F = Q_1 \times Q_2 \times \{2\}$
- and the transition function $\delta : Q \times (\Sigma \cup Q) \to 2^Q$ is defined so as, $\forall p_1, q_1, p_2, q_2 \in Q, a \in \Sigma$:

$\delta_{\text{push}}(\langle p_1, p_2, x\rangle, a) = \{\langle r_1, r_2, y\rangle \mid r_1 \in \delta_{1\text{push}}(p_1, a),\ r_2 \in \delta_{2\text{push}}(p_2, a)\}$

$\delta_{\text{flush}}(\langle p_1, p_2, x\rangle, \langle q_1, q_2, z\rangle) = \{\langle r_1, r_2, y\rangle \mid r_1 \in \delta_{1\text{flush}}(p_1, q_1), r_2 \in \delta_{2\text{flush}}(p_2, q_2)\}$

and the third component of the states is computed as follows:

- if $x = 0$ and $r_1 \in F_1$ then $y = 1$
- if $x = 1$ and $r_2 \in F_2$ then $y = 2$
- if $x = 2$ then $y = 0$
- $y = x$ otherwise.

Intuitively, during the parsing of an input string, the automaton $\mathcal{A}$ simulates $\mathcal{A}_1$ and $\mathcal{A}_2$ respectively on the first two components of the states, whereas the third component keeps track of the succession of visits of the two automata to their final states: in particular its value is 0 at the beginning, then switches from 0 to 1, from 1 to 2 and then back to 0, whenever the first automaton reaches a final state and the other one visits a final state afterwards. This cycle is repeated infinitely often whenever both the automata reach infinite final states along their run.

More precisely, as for ω-regular languages, the claim follows from the fact that, if an ω-word x belongs to $L_1 \cap L_2$, then $\mathcal{A}_1$ and $\mathcal{A}_2$ have a successful run on it, visiting infinitely often final states (in F_1 and F_2 respectively), and along these runs the two automata perform the same type of move for each input symbol. Then, the ω-word is also recognized by $\mathcal{A}$ along a successful run which passes through the pairs of states of the two automata, switching infinitely often the third component to 2, and thus reaching a final state in F, whenever any state in F_1 and any subsequent state in F_2 occur in the first two components.

Conversely, if the input string x does not belong to $L_1 \cap L_2$, then at least one of the runs of $\mathcal{A}_1$ and $\mathcal{A}_2$ either stops because the transition function of the automaton is undefined for the given input or it does not visit infinitely often final states. Hence, $\mathcal{A}$ cannot have a successful run on x and the word is rejected by $\mathcal{A}$ too. □

A similar result holds also for union of ω-languages recognized by BFA with (F) acceptance condition, as the following statement shows.

Theorem 5.2 *Let L_1 and L_2 be ω-languages that can be recognized by two BFA (F) defined over the same alphabet Σ and with compatible precedence matrices M_1 and M_2 respectively. Then $L = L_1 \cup L_2$ is recognizable by a BFA (F) with OPM $M = M_1 \cup M_2$.*

Proof Let $\tilde{\mathcal{A}_1} = \langle \Sigma, M_1, \tilde{Q}_1, \tilde{I}_1, \tilde{F}_1, \tilde{\delta}_1\rangle$ and $\tilde{\mathcal{A}_2} = \langle \Sigma, M_2, \tilde{Q}_2, \tilde{I}_2, \tilde{F}_2, \tilde{\delta}_2\rangle$ be two BFA (F) accepting the languages $L(\tilde{\mathcal{A}_1}) = L_1$ and $L(\tilde{\mathcal{A}_2}) = L_2$. Assume, without loss of generality, that $\tilde{Q}_1$ and $\tilde{Q}_2$ are disjoint.
Since M_1 and M_2 are compatible, then $M = M_1 \cup M_2$ is conflict-free; therefore, applying the normalization defined in Section 3.4.2, the matrices of $\tilde{\mathcal{A}_1}$ and $\tilde{\mathcal{A}_2}$ may be completed so as to become equal to $M = M_1 \cup M_2$, obtaining two BFA (F) automata $\mathcal{A}_1 = \langle \Sigma, M, Q_1, I_1, F_1, \delta_1\rangle$ and $\mathcal{A}_2 = \langle \Sigma, M, Q_2, I_2, F_2, \delta_2\rangle$ which still recognize languages $L(\mathcal{A}_1) = L_1$ and $L(\mathcal{A}_2) = L_2$ respectively. Q_1 and Q_2 are still disjoint.

Then the ω-language $L = L_1 \cup L_2$ is recognized by the BFA (F) $\mathcal{A} = \langle \Sigma, M, Q = Q_1 \cup Q_2, I = I_1 \cup I_2, F = F_1 \cup F_2, \delta\rangle$ whose transition function $\delta : Q \times (\Sigma \cup Q) \to 2^Q$ is

defined so as its restriction to Q_1 and Q_2 equals respectively $\delta_1 : Q_1 \times (\Sigma \cup Q_1) \to 2^{Q_1}$ and $\delta_2 : Q_2 \times (\Sigma \cup Q_2) \to 2^{Q_2}$, i.e:

$$\delta_{\text{push}}(q,a) = \begin{cases} \delta_{1\text{push}}(q,a) & \text{if } q \in Q_1 \\ \delta_{2\text{push}}(q,a) & \text{if } q \in Q_2 \end{cases}$$
$$\delta_{\text{flush}}(p,q) = \begin{cases} \delta_{1\text{flush}}(p,q) & \text{if } p,q \in Q_1 \\ \delta_{2\text{flush}}(p,q) & \text{if } p,q \in Q_2 \end{cases}$$

$\forall p, q \in Q, a \in \Sigma$;

Hence, since the sets of states of the two automata are disjoint and Q equals their union, and $\mathcal{A}_1$ and $\mathcal{A}_2$ have the same OPM as $\mathcal{A}$, then for every $x \in \Sigma^\omega$ there exists a successful run in $\mathcal{A}$ iff there exists a successful run of $\mathcal{A}_1$ on x or a successful run of $\mathcal{A}_2$ on x. □

Notice that if the OPM of the two automata are not filled to $M = M_1 \cup M_2$ and the OPM of $\mathcal{A}$ is naturally defined as $M = M_1 \cup M_2$, then the language recognized by $\mathcal{A}$ is not necessarily $L_1 \cup L_2$, since the presence of precedence relations in $M_1 \cup M_2$ which are not included in M_1 (or M_2) may allow successful runs in $\mathcal{A}$ on some words which are, instead, not accepting in $\mathcal{A}_1$ and in $\mathcal{A}_2$.

5.2 Closure properties of BFAD

As regards BFAD, some negative results can be presented. In particular, the class of languages they recognize is not closed under complement and under concatenation with a language of finite words accepted by a FA. In particular:

Theorem 5.3 *Let L be an ω-language $L \subseteq \Sigma^\omega$ accepted by a BFAD with OPM M. Let L_M be defined as in Section 3.4. Then there does not necessarily exist a BFAD recognizing the complement of L w.r.t. L_M.*

Proof This theorem can be proved considering that, given the alphabet $\Sigma = \{a, b\}$, the language $L = \{\alpha \in \Sigma^\omega : \alpha$ contains an infinite number of letters a $\}$ can be recognized by a BFAD $\mathcal{B} = \langle \Sigma, M, Q, q_0, F, \delta \rangle$ with OPM and graph as in the figure below (Figure 5.1).

	a	b
a	$\lessdot$	$\lessdot$
b	$\lessdot$	$\lessdot$
#	$\lessdot$	$\lessdot$

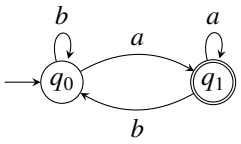


Figure 5.1: BFAD $\mathcal{B}$, with its OPM, of Theorem 5.3.

The automaton visits infinitely often the final state q_1 iff it reads an infinite number of letters a and, thus, it accepts exactly L.
However, there's no BFAD that can recognize the complement of this language w.r.t. L_M, i.e. the language $\neg L = \{\alpha \in \Sigma^\omega : \alpha$ contains finitely many letters a $\}$, as it has been proved in Section 4.2.2. □

Notice that the same result holds also for classical ω-regular languages.
Furthermore, another negative result for BFAD is :

Theorem 5.4 *Let $L_1 \subseteq \Sigma^\omega$ be an ω-language recognized by a BFAD and let $L_2 \subseteq \Sigma^*$ be a language (of finite words) recognized by a FA with a compatible precedence matrix. Then the ω-language defined by the product $L_2 \cdot L_1$ is not necessarily recognizable by a BFAD.*

Proof Given the alphabet $\Sigma = \{a, b\}$, the language

$$L = \{\alpha \in \Sigma^\omega : \alpha \text{ contains finitely many letters a }\}$$

presented in Section 4.2.2, can be seen as the concatenation $L = L_2 \cdot L_1$ of a language of finite words L_2 and an ω-language L_1, with compatible precedence matrices, defined as follows:
$L_2 \subseteq \Sigma^*, \qquad L_2 = \{a, b\}^*$
i.e. L_2 is the set of all finite words on Σ, while
$L_1 \subseteq \Sigma^\omega, \qquad L_1 = \{b^\omega\}$
is the set of infinite words without any letter a.
Language L_2 is recognized by the FA with the OPM and state-graph in Figure 5.2:

	a	b	#
a	$\lessdot$	$\lessdot$	$\gtrdot$
b	$\lessdot$	$\lessdot$	$\gtrdot$
#	$\lessdot$	$\lessdot$	$\doteq$

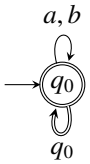


Figure 5.2: FA for L_2 of Theorem 5.4.

and language L_1 is recognized by the BFAD in Figure 5.3:

	b
b	$\lessdot$
#	$\lessdot$

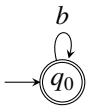


Figure 5.3: BFAD for L_1 of Theorem 5.4.

Since language L cannot be recognized by a BFAD, then the class of languages L_{BFAD} is not closed w.r.t concatenation. □

5.3 Closure properties of MFAD

This section deals with the closure properties of FLs recognized by MFAD w.r.t some traditional operations (Boolean and concatenation). Closure properties are investigated only as regards languages recognized by deterministic Muller automata, since similar properties have been likewise analyzed for BFA, and BFA are equivalent to MFA.

A first property enjoyed by MFAD is that, without loss of generality, a MFAD may be considered complete as regards its function δ, since one may always associate an automaton equivalent to it and with a total transition function, as the following statement shows.
Remind that a ω-FA complete w.r.t its transition function has been defined in Section 3.4.1.

Proposition 5.5 *Any ω-language $L \subseteq \Sigma^\omega$ recognized by a MFAD can be recognized by a complete MFAD.*

Proof Let $\mathcal{A} = \langle \Sigma, M, Q, q_0, \mathcal{T}, \delta \rangle$ be a MFAD recognizing a language $L \subseteq \Sigma^\omega$. If $\mathcal{A}$ is not complete, then add a new state (the "sink" state) and make $\mathcal{A}$ complete by defining a transition corresponding any letter in Σ for any state where there is no such move. More precisely, define $\tilde{\mathcal{A}} = \langle \Sigma, M, \tilde{Q}, q_0, \mathcal{T}, \tilde{\delta} \rangle$ where

- $\tilde{Q} = Q \cup \{p\}$, where p is the "sink" state,
- $\tilde{\delta} : \tilde{Q} \times (\Sigma \cup \tilde{Q}) \to \tilde{Q}$ is the transition function defined as follows. The push transition $\tilde{\delta}_{\text{push}} : \tilde{Q} \times \Sigma \to \tilde{Q}$ is expressed as:

$$\tilde{\delta}_{\text{push}}(q, a) = \begin{cases} \delta_{\text{push}}(q, a) & \text{where } \delta_{\text{push}}(q, a) \in Q \\ p & \text{where } \delta_{\text{push}}(q, a) \text{ is undefined} \end{cases}$$
$$\tilde{\delta}_{\text{push}}(p, a) = p$$

  $\forall q \in Q, a \in \Sigma$;
  The flush transition $\tilde{\delta}_{\text{flush}} : \tilde{Q} \times \tilde{Q} \to \tilde{Q}$ is defined as:

$$\tilde{\delta}_{\text{flush}}(q_1, q_2) = \begin{cases} \delta_{\text{flush}}(q_1, q_2) & \text{where } \delta_{\text{flush}}(q_1, q_2) \in Q \\ p & \text{where } \delta_{\text{flush}}(q_1, q_2) \text{ is undefined} \end{cases}$$
$$\tilde{\delta}_{\text{flush}}(p, q) = p$$

  $\forall q_1, q_2 \in Q, \forall q \in Q \cup \{p\}$.

$\tilde{\mathcal{A}}$ recognizes the same language as $\mathcal{A}$, since each successful run in $\mathcal{A}$ (which visits infinitely often all the states in a given set $T \in \mathcal{T}$) is still valid in $\tilde{\mathcal{A}}$, and conversely any successful run in $\tilde{\mathcal{A}}$ cannot follow any of the added transitions because, otherwise, the run would visit infinitely often only the sink state which, however, does not belong to any of the sets $T \in \mathcal{T}$.

$\tilde{\mathcal{A}}$ is complete, by construction, and it is deterministic. Notice, in fact, that a peculiarity of Floyd automata is that their state-graph (and transition function) can apparently seem nondeterministic, as for instance there can be different flush edges departing from the same state and labeled with distinct states: indeed, determinism in the moves is guaranteed by the content of the stack. □

Notice also that this construction to complete a MFAD might be naturally adapted to complete the transition function of automata recognizing other families of Floyd languages.

Example 5.1 *As an example, consider the MFAD $\mathcal{A} = \langle \Sigma, M, Q, q_0, \mathcal{T}, \delta \rangle$ on $\Sigma = \{a, b, c\}$ with $\mathcal{T} = \{\{q_1, q_2\}, \{q_3\}\}$. The graph and the OPM of the automaton are shown in Figure 5.4.*
The automaton recognizes the language $L(\mathcal{A}) = \{(b^+c)^\omega\} \cup \{a(a|b)^\omega\}$.
$\mathcal{A}$ can be completed as in Figure 5.5.

Remark *Notice that $\tilde{\mathcal{A}}$ admits only paths with labels compatible with the OPM M: for instance, even though a transition with letter c, from q_0 to the sink p, is added to the original automaton $\mathcal{A}$, however no ω-word beginning with c is a label for a path in $\tilde{\mathcal{A}}$, since this move can never be performed. In fact, the OPM does not include a precedence relation between the symbols # and c, and this is consistent with the fact that $\mathcal{A}$ has not been completed with paths for every word in Σ^ω, but only for those ω-words whose syntax tree is compatible with M (see Section 3.4).*

	a	b	c
a	$\lessdot$	$\lessdot$	
b	$\lessdot$	$\doteq$	$\lessdot$
c		$\gtrdot$	
#	$\lessdot$	$\lessdot$	

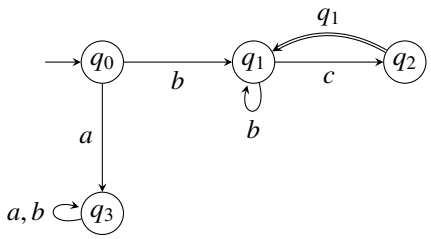


Figure 5.4: Non complete MFAD $\mathcal{A}$ of Example 5.1.

	a	b	c
a	$\lessdot$	$\lessdot$	
b	$\lessdot$	$\doteq$	$\lessdot$
c		$\gtrdot$	
#	$\lessdot$	$\lessdot$	

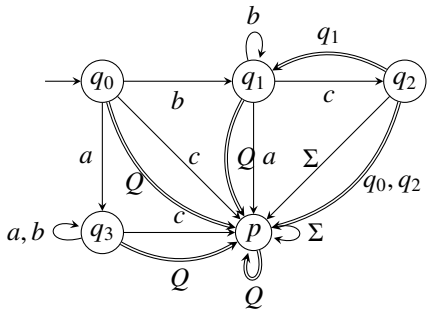


Figure 5.5: Complete MFAD $\tilde{\mathcal{A}}$ of Example 5.1.

Now, the following results show that the class of languages recognized by MFAD is closed under all Boolean operations.
Let L_M be defined as in Section 3.4.

Theorem 5.6 *For every precedence matrix M, the class of languages with OPM M and recognizable by MFAD is closed under complement w.r.t L_M.*

Proof Let $\mathcal{A} = \langle \Sigma, M, Q, q_0, \mathcal{T}, \delta \rangle$ be a MFAD, that without loss of generality can be assumed to be complete with respect to M, as regards its transition function δ.
The MFAD $\mathcal{A}' = \langle \Sigma, M, Q, q_0, 2^Q \setminus \mathcal{T}, \delta \rangle$, whose table is defined turning the table of $\mathcal{A}$ into its complement $2^Q \setminus \mathcal{T}$, recognizes $L_M \setminus L(\mathcal{A})$.
Indeed, since $\mathcal{A}$ is complete, every word in L_M labels one and only one path in $\mathcal{A}$, starting form the initial state.
If a path $\mathcal{S}$ visits infinitely often only each state of a set $T \in \mathcal{T}$ (i.e. $In(\mathcal{S}) = T \in \mathcal{T}$), then the path is successful for $\mathcal{A}$ but not for $\mathcal{A}'$. Conversely, if a word labels a path $\mathcal{S}$ with $In(\mathcal{S}) \notin \mathcal{T}$, then the ω-word is recognized by $\mathcal{A}'$ but is not accepted by $\mathcal{A}$. □

Theorem 5.7 *Let L_1 and L_2 be ω-languages with precedence compatible matrices M_1 and M_2 respectively, recognized by two MFAD. Then $L = L_1 \cap L_2$ is recognized by a MFAD with OPM $M = M_1 \cap M_2$.*

Proof Let $\mathcal{A}_1 = \langle \Sigma, M_1, Q_1, q_{01}, \mathcal{T}_1, \delta_1 \rangle$ and $\mathcal{A}_2 = \langle \Sigma, M_2, Q_2, q_{02}, \mathcal{T}_2, \delta_2 \rangle$ be two MFAD recognizing respectively languages $L(\mathcal{A}_1) = L_1$ and $L(\mathcal{A}_2) = L_2$. Assume without loss of generality that they are both complete as regards their transition function.

A MFAD $\mathcal{A}$ with OPM $M = M_1 \cap M_2$ recognizing $L = L_1 \cap L_2$ may be defined adopting the usual product construction for ω-regular automata, requiring that a successful path in $\mathcal{A}$ corresponds to paths that visit infinitely often sets in the table $\mathcal{T}_1$ and $\mathcal{T}_2$ respectively of the two automata, and are then successful for them. More precisely let $\mathcal{A} = \langle \Sigma, M, Q, q_0, \mathcal{T}, \delta \rangle$ where

- $Q = Q_1 \times Q_2$,
- $q_0 = (q_{01}, q_{02})$,
- Define π_i (i = 1, 2) as the projection from $Q_1 \times Q_2$ on Q_i, that can also be naturally extended to define projections on paths of the automata, and let
  $\mathcal{T} = \{P \subseteq Q_1 \times Q_2 \mid \pi_1(P) \in \mathcal{T}_1 \text{ and } \pi_2(P) \in \mathcal{T}_2\}$,
- The transition function $\delta : Q \times (\Sigma \cup Q) \to Q$ is defined as follows. The push transition $\delta_{\text{push}} : Q \times \Sigma \to Q$ is expressed as:
  $\delta_{\text{push}}((q_1, q_2), a) = (q_1', q_2') \qquad \text{if } \delta_{1\text{push}}(q_1, a) = q_1' \text{ and } \delta_{2\text{push}}(q_2, a) = q_2'$

  $\forall q_1, q_2 \in Q, a \in \Sigma$;
  The flush transition $\delta_{\text{flush}} : Q \times Q \to Q$ is defined as:
  $\delta_{\text{flush}}((q_1, q_2), (p_1, p_2)) = (r_1, r_2) \quad \text{if } \delta_{1\text{flush}}(q_1, p_1) = r_1 \text{ and } \delta_{2\text{flush}}(q_2, p_2) = r_2$
  $\forall q_1, q_2, p_1, p_2 \in Q$;

Notice that the definition of the transition function is well-founded since $\mathcal{A}_1$ and $\mathcal{A}_2$ have precedence compatible matrices, and then, if a precedence relation between two letters is defined in both matrices, it is the same. Therefore, during the parsing of the input string, they either perform the same type of move (push/mark/flush), if the precedence relation between the last character read and the current symbol is defined in both M_1 and M_2 (and this relation is the same, being M_1 and M_2 compatible), or at least one of the two automata stops without accepting the word in input since its precedence matrix is undefined for the current symbol.
The theorem now follows noting that, being the set of states Q_1 and Q_2 of the two automata finite, then the first components of the pairs visited infinitely often by $\mathcal{A}$ along a run $\mathcal{S}$ on an ω-word x equal the states visited infinitely often by the automaton $\mathcal{A}_1$ along a run $\mathcal{S}_1$ which is the projection of $\mathcal{S}$ by π_1, and a dual result holds also for the second components of the set of the infinitely often visited pairs.
Then, let $\mathcal{S}$ be a successful path of $\mathcal{A}$, starting in the initial state $q_0 = (q_{01}, q_{02})$: since it is accepting, the set $In(\mathcal{S}) = P \in \mathcal{T}$. Thus, by definition of $\mathcal{T}$ and following the previous consideration, the paths $\mathcal{S}_1$ and $\mathcal{S}_2$, that are the projection of $\mathcal{S}$ on the set of states of $\mathcal{A}_1$ and $\mathcal{A}_2$ respectively, have $In(\mathcal{S}_1) = \pi_1(P) \in \mathcal{T}_1$ and $In(\mathcal{S}_2) = \pi_2(P) \in \mathcal{T}_2$: whence $\mathcal{S}_1$ and $\mathcal{S}_2$ are successful paths for the two automata, and x belongs to $L(\mathcal{A}_1) \cap L(\mathcal{A}_2)$.
In order to prove that an ω-word $x \in L(\mathcal{A}_1) \cap L(\mathcal{A}_2)$ is recognized by $\mathcal{A}$, consider that if x labels two successful paths $\mathcal{S}_1$ and $\mathcal{S}_2$ for the two automata, then $In(\mathcal{S}_1) \in \mathcal{T}_1$ and $In(\mathcal{S}_2) \in \mathcal{T}_2$. The path $\mathcal{S}$ of $\mathcal{A}$ which visits the pairs of states of the two automata, performing the same type of move they perform for each input symbol, is defined so as $\pi_1(In(\mathcal{S})) = In(\mathcal{S}_1) \in \mathcal{T}_1$ and $\pi_2(In(\mathcal{S})) = In(\mathcal{S}_2) \in \mathcal{T}_2$. Therefore, by definition of $\mathcal{T}$, $\mathcal{S}$ is a successful path for $\mathcal{A}$. □

Finally, even the union of two languages with precedence compatible matrices M_1 and M_2 recognizable by MFAD is a language recognizable by a MFAD having an OPM

compatible with M_1 and M_2. This statement follows from the closure of this class of languages under complement (Theorem 5.6), intersection (Theorem 5.7) and by De Morgan identity. More precisely:

Theorem 5.8 *Let L_1 and L_2 be ω-languages with precedence compatible matrices M_1 and M_2 respectively, recognized by two MFAD. Then $L = L_1 \cup L_2$ is recognized by a MFAD with OPM $M = M_1 \cup M_2$.*

Proof Let $\mathcal{A}_1 = \langle \Sigma, M_1, Q_1, q_{01}, \mathcal{T}_1, \delta_1 \rangle$ and $\mathcal{A}_2 = \langle \Sigma, M_2, Q_2, q_{02}, \mathcal{T}_2, \delta_2 \rangle$ be two MFAD which recognize respectively languages $L(\mathcal{A}_1) = L_1$ and $L(\mathcal{A}_2) = L_2$, with precedence matrices M_1 and M_2.
Since M_1 and M_2 are compatible, then $M = M_1 \cup M_2$ is conflict-free and, applying the normalization described in Section 3.4.2, the matrices of the two automata may be completed to become equal to $M = M_1 \cup M_2$, obtaining the automata $\mathcal{A}'_1 = \langle \Sigma, M_1 \cup M_2, Q'_1, q'_{01}, \mathcal{T}'_1, \delta'_1 \rangle$ and $\mathcal{A}'_2 = \langle \Sigma, M_1 \cup M_2, Q'_2, q'_{02}, \mathcal{T}'_2, \delta'_2 \rangle$ which recognize the same languages as the original MFADs, $L(\mathcal{A}'_1) = L_1$ and $L(\mathcal{A}'_2) = L_2$.

Consider the MFADs which recognize the complement of these languages w.r.t $L_{M_1 \cup M_2}$: let $\mathcal{B}_i$ be a MFAD which accepts $L_{M_1 \cup M_2} \backslash L_i$, $(i = 1, 2)$ with OPM $M_1 \cup M_2$. Notice that these automata are also MFAD, since the class of languages recognized by MFAD is closed under complement (Theorem 5.6).
Then, considering that $\mathcal{B}_1$ and $\mathcal{B}_2$ have the same matrix $M_1 \cup M_2$ (and thus their OPM are compatible), the intersection of the languages recognized by these two automata, $L(\mathcal{B}_1) \cap L(\mathcal{B}_2)$ may be also recognized by a MFAD with the same matrix, $M_1 \cup M_2$.
Finally, the complement of this language w.r.t $L_{M_1 \cup M_2}$:

$$L_{M_1 \cup M_2} \backslash [L(\mathcal{B}_1) \cap L(\mathcal{B}_2)] = L_{M_1 \cup M_2} \backslash [(L_{M_1 \cup M_2} \backslash L_1) \cap (L_{M_1 \cup M_2} \backslash L_2)]$$

equals exactly $L_1 \cup L_2$ and, by Theorem 5.6 (on closure of MFAD w.r.t complement), it is indeed recognizable by a MFAD with OPM $M_1 \cup M_2$. □

Notice that the matrices of the two automata $\mathcal{A}_1$ and $\mathcal{A}_2$ must be completed to become equal to $M_1 \cup M_2$ in order to apply De Morgan identity. If one considers, instead, automata with compatible but not equal matrices, the identity would not hold: in fact, the expression

$$L_1 \cup L_2 = L_{M_1 \cap M_2} \backslash [(L_{M_1} \backslash L_1) \cap (L_{M_2} \backslash L_2)]$$

is not valid, unless $M_1 = M_2$.

Hence, the previous results for the class of languages recognized by MFAD lead to the following theorem:

Theorem 5.9 *For every precedence matrix M, the class of languages with OPM compatible with M and recognizable by MFAD is a Boolean algebra.*

There is also a negative result for the class L_{MFAD} : indeed, this class is not closed under concatenation with a language of finite words accepted by a FA. More precisely:

Theorem 5.10 *Let $L_1 \subseteq \Sigma^\omega$ be an ω-language recognized by a MFAD and let $L_2 \subseteq \Sigma^*$ be a language (of finite words) recognized by a FA with a compatible precedence matrix. Then the ω-language defined by the product $L_2 \cdot L_1$ is not necessarily recognizable by a MFAD.*

Proof Given the alphabet $\Sigma = \{c, r\}$, let L_{repbdd} be the language consisting of ω-words n on the alphabet Σ such that n has only finitely many *pending calls* (i.e. letters c not matched by any subsequent corresponding letter r). This language has already been described in Section 4.2.4 and cannot be accepted by any MFAD.

Now, consider the finite FA $\mathcal{A}_2 = \langle \Sigma, M_2, Q_2, I_2, F_2, \delta_2 \rangle$ that accepts the language L_2 of words of finite length on Σ^* depicted in Figure 5.6. These words have necessarily a finite number of pending calls, since they have finite length.

	c	r	#
c	$\lessdot$	$\doteq$	$\gtrdot$
r	$\gtrdot$	$\gtrdot$	$\gtrdot$
#	$\lessdot$		$\doteq$

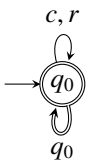


Figure 5.6: FA for L_2 of Theorem 5.10.

Moreover, let $\mathcal{A}_1$ be a MFAD that recognizes the ω-language L_1 whose words are infinite sequences of well-parenthesized finite-length strings (i.e. finite-length strings belonging to the Dyck language on Σ, and thus they have no pending calls).

$$L_1 = (L_{Dyck}(c, r))^\omega$$

$\mathcal{A}_1$ is depicted in the figure below (Figure 5.7), with $\mathcal{T} = \{\{q_0, q_1\}\}$.

	c	r
c	$\lessdot$	$\doteq$
r	$\gtrdot$	$\gtrdot$
#	$\lessdot$	

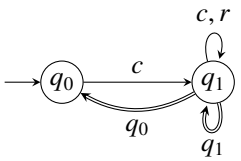


Figure 5.7: MFAD for L_1 of Theorem 5.10.

Incidentally, notice that L_1 and L_2 have compatible matrices.
The product ω-language $L_2 \cdot L_1$ is exactly the set of ω-words with a finite number of pending calls, i.e. it equals L_{repbdd}. But, by Theorem 4.7, it cannot be recognized by a MFAD, then the class of languages L_{MFAD} is not closed w.r.t concatenation. □

5.4 Comparison of closure properties

The closure properties of the families of ω-FLs that have been analyzed in this chapter are summarized in the following table.
Empty entries refer to properties that have not been proved or disproved yet.

	BFAD	MFAD	$BFA \equiv MFA$	Büchi VPA [2]
Intersection		Yes	Yes	Yes
Union		Yes	Yes	Yes
Complement	No	Yes		Yes
$L_2 \cdot L_1$	No	No		Yes

Table 5.1: Closure properties of families of ω-languages.

where L_1 is an ω-language and L_2 is a language of finite-length words.
The hypotheses under which the closures listed in the table are valid have been already discussed in the previous sections of this chapter or are presented in [2] as regards ω-VPA.

Chapter 6

Conclusions

This thesis dealt with Floyd omega languages extending, to the world of languages of infinite words, the traditional class of finite Floyd languages (FL).

Floyd languages have been investigated for the first time by Robert Floyd in 1963 with the aim of defining deterministic (and efficient) algorithms for the parsing of programming languages. However, during the following decades, they have been eclipsed by the advent of classical powerful families of parsers as LALR and LR parsers. Rediscovered again, along a line of research motivated by arising interest in fields as Natural language Processing (NLP), software verification and querying and analysis of structured web documents, they have been finally completely characterized in the recent papers [8] and [17]. This last work, in particular, laid the basis for the subject matter of this work of thesis, as it formalized FLs with a perfectly equivalent class of infinite-state abstract machines, named Floyd automata (FA). This considerable step in research of FLs allows to generalize this class of languages with the classical notions of ω-language theory, quite interesting a field in these days when nonterminating systems, that can be naturally modeled with words of infinite sequences of symbols, appear in so distant application contexts as operating systems, web document processing and so on.

Along this line of research, the main contribution of this work of thesis to classical theory on Floyd and ω-languages has been the formalization of suitable operational formalisms for automata able to recognize Floyd languages of infinite words. Different classes of ω-automata have been analyzed, on the basis of the acceptance conditions of infinite-length words they are endowed with, as for instance Büchi automata with various recognizing modes and Muller automata. The expressive power of these classes of ω-automata has been further investigated to take into account deterministic and nondeterministic computations of these abstract devices; and this study has allowed to define a hierarchy of ω-automata for FLs, also placing it in relation with other classical families of automata on infinite words, as those recognizing ω-VPL [2] and ω-regular languages ([24] and [20]).
Moreover, ω-Floyd automata have been further characterized by the analysis of the closure properties w.r.t classical operations on ω-languages they enjoy.

6.1 Perspectives for future work

Naturally, since Floyd languages represent an open field of study, this thesis could not exhaust all the research on this subject, and further interesting themes may be investigated.

A fundamental direction to enhance the results of this thesis consists in the analysis of other basic properties of ω-FAs, as the closure property w.r.t complementation of nondeterministic Büchi automata and the verification of the decidability of the emptiness problem, which is crucial for applications to model-checking. Indeed, model-checking is the leading application field that motivates the study carried on in this work of thesis.

Model-checking is a well-known verification technique introduced in the 1980s and adopted for automatic or semi-automatic validation of properties of systems, specified by means of suitable formalisms. Actually, model checking cannot guarantee the absolute correspondence between the real behavior of a system and the desired specification of its properties, but it appeared to be a powerful approach to assess consistency between the wished features of the system and those that actually hold. Model-checking, thus, acquires a significant importance especially in these days, where most modern computing systems have reached such a level of complexity and pervasiveness in critical fields that their verification becomes of the utmost importance to identify potential vulnerabilities and to prevent possible failures.
However, it is naturally impossible to guarantee that a system is devoid of errors, so model-checking attempts only to compare an available model of the system with respect to a given specification of its required properties, and verification of consistency is performed algorithmically. Algorithmic verification, though, implies some typical challenges in model-checking domain of research, since it is fundamental to deal with the analysis of classes of systems which admit a decidable model checking problem, and furthermore the computational complexity of the verification process is critical for practical applications.

Within this context, FLs appear to be a promising formalism for infinite-state model-checking and further investigation on this class of languages is motivated by several reasons. First, as model-checking is actually based on the closure w.r.t. Boolean operations and the decidability of the emptiness problem, further reasearch of theoretical properties along this direction is fundamental.
Moreover, the specification of the properties of a system to be verified has been traditionally expressed resorting to logic notations as Linear Temporal Logic (LTL) and Computational Tree Logic (CTL), both generalized by the logic CTL*, and it might be interesting to characterize directly ω-FLs in terms of an equivalently expressive logic (based on LTL or monadic second order logics). An analogous characterization has been defined for ω-VPLs in the paper [2], which proved that the monadic second order logic of nested words has the same expressiveness as VPAs, but a formalization in logic of ω-FLs might significantly extend the expressive power of this former formalism, yet maintaining a reasonable complexity for the decision procedure. Indeed, some peculiar properties of FLs, as the fact that they can be parsed without applying a strictly left-to-right order, open up the perspective of exploiting parallel parsing algorithms to increase the speed of verification, bounding algorithmic complexity.

From this perspective, a further assessment of current investigations on Floyd languages on infinite words and the analysis of the theoretical properties enjoyed by ω-FLs represent a fundamental step and the natural basis for the design of a model checker for infinite-state systems for this class of languages, that could fit for verifying practical

and realistic software systems.

A prototype for parsing of finite Floyd languages has been already presented in [17], and the implementation of a model-checker based on FLs might be the natural prosecution of this work, blending the classical results of ω-language theory concerning the model checking scenario and the results on operational formalisms suitable for ω-FLs that this thesis has described.

Bibliography

[1] R. Alur and P. Madhusudan. Visibly pushdown languages. In *STOC: ACM Symposium on Theory of Computing (STOC)*, 2004.

[2] R. Alur and P. Madhusudan. Adding nesting structure to words. *Journ. ACM*, 56(3), 2009.

[3] Rajeev Alur and David L. Dill. A theory of timed automata. *Theoretical Computer Science*, 126(2):183 – 235, 1994.

[4] J. Berstel and L. Boasson. Balanced grammars and their languages. In W. Brauer et al., editor, *Formal and Natural Computing*, volume 2300 of *LNCS*, pages 3–25. Springer, 2002.

[5] J. Richard Büchi. Weak second-order arithmetic and finite automata. *Mathematical Logic Quarterly*, 6(1-6):66–92, 1960.

[6] Karlis Cerans. Deciding properties of integral relational automata. In *Proceedings of the 21st International Colloquium on Automata, Languages and Programming*, ICALP '94, pages 35–46, London, UK, 1994. Springer-Verlag.

[7] E. M. Clarke, E. A. Emerson, and A. P. Sistla. Automatic verification of finite-state concurrent systems using temporal logic specifications. *ACM Trans. Program. Lang. Syst.*, 8:244–263, April 1986.

[8] S. Crespi Reghizzi and D. Mandrioli. Operator precedence and the visibly pushdown property. In A. Horia Dediu, H. Fernau, and C. Martín-Vide, editors, *LATA*, volume 6031 of *LNCS*, pages 214–226. Springer, 2010.

[9] S. Crespi Reghizzi, D. Mandrioli, and D. F. Martin. Algebraic properties of operator precedence languages. *Information and Control*, 37(2):115–133, May 1978.

[10] E.A. Emerson and K.S. Namjoshi. On model checking for non-deterministic infinite-state systems. In *Logic in Computer Science, 1998. Proceedings. Thirteenth Annual IEEE Symposium on*, pages 70 –80, jun 1998.

[11] M. J. Fischer. Some properties of precedence languages. In *STOC '69: Proc. first annual ACM Symp. on Theory of Computing*, pages 181–190, New York, NY, USA, 1969. ACM.

[12] R. W. Floyd. Syntactic analysis and operator precedence. *Journ. ACM*, 10(3):316–333, 1963.

[13] D. Grune and C. J. Jacobs. *Parsing techniques: a practical guide*. Springer, New York, 2008.

[14] M. A. Harrison. *Introduction to Formal Language Theory*. Addison Wesley, Reading, MA, 1978.

[15] Oscar H. Ibarra. Reversal-bounded multicounter machines and their decision problems. *J. ACM*, 25:116–133, January 1978.

[16] Violetta Lonati, Dino Mandrioli, and Matteo Pradella. Precedence automata and languages. *CoRR*, abs/1012.2321, 2010.

[17] Violetta Lonati, Dino Mandrioli, and Matteo Pradella. Precedence automata and languages. In Alexander Kulikov and Nikolay Vereshchagin, editors, *Computer Science - Theory and Applications*, volume 6651 of *Lecture Notes in Computer Science*, pages 291–304. Springer Berlin / Heidelberg, 2011.

[18] R. McNaughton. Parenthesis grammars. *Journ. ACM*, 14(3):490–500, 1967.

[19] David E. Muller. Infinite sequences and finite machines. In *Proceedings of the 1963 Proceedings of the Fourth Annual Symposium on Switching Circuit Theory and Logical Design*, SWCT '63, pages 3–16, Washington, DC, USA, 1963. IEEE Computer Society.

[20] D. Perrin and J.-E. Pin. *Infinite words*, volume 141 of *Pure and Applied Mathematics*. Elsevier, Amsterdam, 2004.

[21] M.O. Rabin. *Automata on infinite objects and Church's problem*. Regional conference series in mathematics. Published for the Conference Board of the Mathematical Sciences by the American Mathematical Society, 1972.

[22] A. K. Salomaa. *Formal Languages*. Academic Press, New York, NY, 1973.

[23] Robert S. Streett. Propositional dynamic logic of looping and converse is elementarily decidable. *Information and Control*, 54(1-2):121 – 141, 1982.

[24] Wolfgang Thomas. Handbook of theoretical computer science (vol. b). chapter Automata on infinite objects, pages 133–191. MIT Press, Cambridge, MA, USA, 1990.