

LFApp: Um Aplicativo Móvel para o Ensino de Linguagens Formais e Autômatos

Juventino Neto, Ricardo Terra

Departamento de Ciência da Computação,
Universidade Federal de Lavras (UFLA), Brasil

jneto@computacao.ufla.br, terra@dcc.ufla.br

Resumo. *Linguagens Formais e Autômatos (LFA) é uma importante área que aborda modelos matemáticos que possibilitam a especificação e reconhecimento de linguagens, suas propriedades e características. Embora o conhecimento sólido em LFA seja de extrema importância para a formação de um bacharel em Ciência da Computação e áreas afins, os algoritmos e técnicas abordadas na disciplina são complexos e de difícil assimilação. Diante disso, este trabalho apresenta LFApp, um aplicativo móvel para ensino de LFA. O aplicativo – desenvolvido para celulares e tablets com sistema operacional Android – provê aos alunos não somente a resolução de problemas envolvendo Linguagens Regulares e Linguagens Livres de Contexto, mas também, uma interface com caráter acadêmico que descreve e ilustra cada etapa da execução dos algoritmos de modo a apoiar os alunos no processo de aprendizagem.*

Abstract. *Formal Languages and Automata (FLA) deals with mathematical models that enable the specification and recognition of languages, their properties and characteristics. Although solid knowledge in FLA is extremely important for a B.Sc. degree in Computer Science and similar fields, the algorithms and techniques covered in the course are complex and difficult to assimilate. Therefore, this paper presents LFApp, a mobile application for teaching FLA. The application—developed for mobile phones and tablets running Android—provides students not only the solving of problems involving Regular and Context-free Languages, but also an Educational environment that describes and illustrates each step of the algorithms in order to support students in the learning process.*

1. Introdução

Linguagens Formais e Autômatos (LFA) é uma importante área da computação que aborda modelos matemáticos que possibilitam a especificação e reconhecimento de linguagens, suas propriedades e características [10]. Considerando a importância que um conhecimento sólido de LFA tem sobre o perfil de um bacharel da área de Ciência da Computação e observando um nível de reprovação de quase 50% na disciplina [11], foi proposta uma plataforma que possa auxiliar os alunos no processo de aprendizagem.

Diante disso, este artigo apresenta LFApp, um aplicativo móvel para ensino de LFA que provê aos alunos não somente a resolução de problemas envolvendo Linguagens Regulares e Linguagens Livres de Contexto, mas também uma interface com caráter acadêmico – sendo essa a principal contribuição deste trabalho – que descreve e ilustra cada etapa da execução dos algoritmos de modo a apoiar os alunos no processo de

aprendizagem. É importante mencionar que o aplicativo foi desenvolvido para celulares e *tablets* com sistema operacional (SO) Android e possui o código aberto visando a ampla divulgação, uma vez que o Android é o SO mais utilizado por dispositivos móveis [7].

Em estudo prévio, notou-se que estudantes apresentavam baixo rendimento em algoritmos relacionados a gramáticas [11]. No intuito de cobrir tal deficiência, LFApp foca em Linguagens Livres de Contexto, implementando as seguintes funcionalidades: identificação do tipo de gramática, derivação, remoção de recursão no símbolo inicial, remoção de produções vazias, remoção de regras de cadeia, remoção de símbolos não-terminais, remoção de símbolos não alcançáveis, Forma Normal de Chomsky, remoção de recursão direta à esquerda, remoção de recursão indireta à esquerda, Forma Normal de Greibach e algoritmo de reconhecimento CYK (Cocke-Younger-Kasami) [10].

Este artigo está organizado como a seguir. A Seção 2 apresenta a análise do contexto em que o aplicativo melhor se adaptaria e as motivações para a sua criação, enquanto que a Seção 3 apresenta o aplicativo desenvolvido e o que espera-se com seu uso. A Seção 4 apresenta seu projeto, implementação e ilustra os testes de unidade realizados. A Seção 5 lista os trabalhos relacionados. Por fim, a Seção 6 conclui relatando as principais contribuições e enumerando trabalhos futuros.

2. Contextualização

O aplicativo foi desenvolvido com a motivação de apoiar os alunos no processo de aprendizagem de algoritmos relacionados a gramáticas, os quais os estudantes mais se queixavam [11]. Dessa forma, o aplicativo – como principal elemento pedagógico – reflete exatamente os passos seguidos pelo professor em sala de aula. A seguir, destacam-se conceitos importantes para a compreensão correta deste trabalho.

Hierarquia de Chomsky: Proposta por Noam Chomsky em 1956, a Hierarquia de Chomsky constitui uma classificação para linguagens, gramáticas e autômatos [2] que é composta por quatro categorias, sendo cada uma delas um subconjunto próprio da categoria superior. As quatro categorias são: Linguagens Recursivamente Enumeráveis (0), Linguagens Sensíveis ao Contexto (1), Linguagens Livres de Contexto (2) e Linguagens Regulares (3). Os níveis 2 e 3 são amplamente empregados na descrição de linguagens de programação e na implementação de interpretadores e compiladores [1]. Este trabalho mantém como foco a categoria de nível 2, Linguagens Livres de Contexto, empregando algoritmos e técnicas que atuam sobre esse nível da hierarquia.

Gramática: É um conjunto de regras de produções no seguinte formato $\mu \rightarrow \nu$ que permitem gerar uma linguagem. A definição formal de uma gramática é uma quádrupla (V, Σ, P, S) , onde V representa o conjunto de símbolos não-terminais (variáveis), Σ o conjunto de símbolos terminais, P o conjunto de regras e S o símbolo não-terminal inicial [10]. Segundo a Hierarquia de Chomsky, a Tabela 1 descreve os quatro tipos de gramáticas, em que cada tipo representa uma categoria.

Tabela 1. Hierarquia de Chomsky

Tipo	Linguagem	Gramática	Regras $\mu \rightarrow \nu$
0	Rec. Enumerável	Irrestrita (GI)	$\mu \in (V \cup \Sigma)^+, \nu \in (V \cup \Sigma)^*$
1	Recursiva	Sensível ao Contexto (GSC)	$\mu \in (V \cup \Sigma)^+, \nu \in (V \cup \Sigma)^+, \mu \leq \nu $
2	Livre de Contexto	Livre de Contexto (GLC)	$\mu \in V, \nu \in (V \cup \Sigma)^*$
3	Regular	Regular (GR)	$\mu \in V, \nu \in \lambda \mid \Sigma \mid \Sigma V$

Forma Normal: GLCs são muito flexíveis na formação de suas regras, o que pode ser vantajoso no processo de criação de uma gramática. No entanto, essa flexibilidade não

define uma estrutura padrão para esse tipo de gramática, o que não permite estabelecer relações gerais sobre a gramática, derivações e linguagens [10]. Com o objetivo de tornar as GLCs mais concisas, formas normais impõem regras e simplificam a representação, sem alterar o poder de expressão da linguagem. A Tabela 2 descreve a Forma Normal de Chomsky (FNC) e Greibach (FNG), e o formato de suas regras.

Tabela 2. Formais Normais

Forma Normal de Chomsky (FNC)		Forma Normal de Greibach (FNG)	
$A \rightarrow BC$	onde $B, C \in V$	$A \rightarrow aA_1A_2...A_n$	onde $A_i \in V - \{S\}$
$A \rightarrow a$	onde $a \in \Sigma$	$A \rightarrow a$	onde $a \in \Sigma$
$S \rightarrow \lambda$		$S \rightarrow \lambda$	

Derivação: Consiste no processo de substituição dos símbolos lidos em uma dada palavra pelas regras de uma gramática. Dada uma gramática G e uma palavra p , o objetivo é verificar se a partir de seu símbolo inicial S deriva-se p , i.e., $S \Rightarrow^* p$. As derivações podem ser mais à esquerda ou mais à direita. A derivação mais à esquerda consiste em substituir os símbolos mais à esquerda da forma sentencial, enquanto que, de forma análoga, a mais à direita consiste em substituir os símbolos mais à direita. O processo de derivação ocorre em todas as categorias das linguagens presentes na Hierarquia de Chomsky [10].

3. Solução Proposta

Esta seção apresenta LFApp, um aplicativo que provê a implementação de diversas funcionalidades em um ambiente de caráter acadêmico. O objetivo é prover o aluno com uma descrição sobre o processo de resolução de forma a estimular seu aprendizado. Nas subseções seguintes são descritas e ilustradas as funcionalidades implementadas. Para ilustrar as funcionalidades, adota-se a seguinte GLC, a qual foi projetada de forma a ser possível observar as transformações propostas pela maioria dos algoritmos implementados:

$$\begin{array}{lll}
 S \rightarrow aABC \mid a \mid S & B \rightarrow bcB \mid bc & D \rightarrow abc \mid E \\
 A \rightarrow aA \mid \lambda & C \rightarrow cC \mid c & E \rightarrow Ea \mid a
 \end{array}$$

Basicamente, a interação entre o usuário (aluno) e a ferramenta ocorre conforme ilustrado pela Figura 1. O aluno informa a gramática e todos os algoritmos são computados.

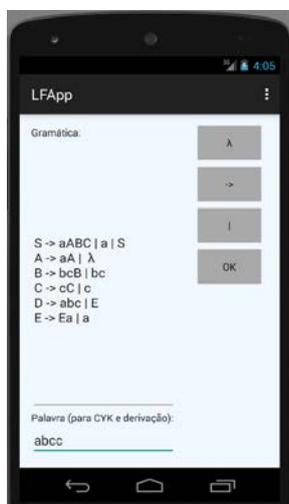


Figura 1. Tela inicial do LFApp

Identificação da gramática

A classificação de uma gramática é feita pelo tipo de suas regras ($u \rightarrow v$). A tabela abaixo mostra o formato de regras características de cada nível:

(3) GR	$u \in V$	$v \in \lambda \mid \Sigma \mid \Sigma V$
(2) GLC	$u \in V$	$v \in (V \cup \Sigma)^*$
(1) GSC	$u \in (V \cup \Sigma)^+$	$v \in (V \cup \Sigma)^+$
(0) GI	$u \in (V \cup \Sigma)^+$	$v \in (V \cup \Sigma)^*$

Resultado:
 - Na gramática informada, a regra $S \rightarrow aABC$ não pertence ao conjunto das gramáticas regulares.
 Logo, a gramática inserida é uma Gramática Livre de Contexto (GLC).

Figura 2. Identificação do tipo da gramática

Identificação do tipo de gramática: Essa funcionalidade consiste em identificar em qual nível da hierarquia de Chomsky a gramática se encontra. A Figura 2 ilustra o resultado dessa funcionalidade na gramática de referência. Basicamente, o algoritmo analisa o formato das regras da gramática e a classifica no nível mais restrito possível. Nesse exemplo, além de classificar a gramática como GLC, proveu-se uma tabela de classificação de regras e um exemplo de uma regra que não se enquadra na definição do nível superior (GR).

Derivação: Essa funcionalidade realiza o processo de derivação mais à esquerda, i.e., tenta gerar uma dada palavra a partir de uma dada gramática. Essa funcionalidade também avalia se a gramática é ambígua. Verificar se uma gramática é ambígua é considerado um problema indecidível, logo, são realizadas até 1.000 tentativas de encontrar formas diferentes de gerar a mesma sentença. Em caso positivo, a gramática é classificada como ambígua, caso contrário, nada se conclui. A Figura 3 ilustra o processo de derivação sobre a gramática de referência e a palavra *abcc*.

Derivação

Resultado:

```
S => aABC
=> aλBC
=> abcC
=> abcc
```

Palavra: abcc

Ambiguidade: Uma Gramática Livre de Contexto é ambígua caso possua mais de uma derivação mais à esquerda. Para a gramática inserida foram realizadas até 1000 tentativas de encontrar uma ambiguidade, porém, não foi encontrada outra forma de gerar a palavra abcc.

Figura 3. Derivação

Remoção de Recursão no Símbolo Inicial

Resultado:

```
S' -> S
S -> a | aABC | S
A -> λ | aA
B -> bc | bcB
C -> c | cC
D -> abc | E
E -> Ea
```

Algoritmo:
O símbolo inicial deve se limitar a iniciar derivações, não podendo ser uma variável recursiva. Logo, não deve ser possível ter derivações do tipo $S \Rightarrow^* \alpha S \beta$.

A gramática $G = (V, \Sigma, P, S)$ possui o símbolo inicial S recursivo. Logo, existe uma GLC $G' = (V \cup \{S'\}, \Sigma, P \cup \{S' \rightarrow S\}, S')$, tal que $L(G) = L(G')$ e o novo símbolo inicial S não é recursivo.

Figura 4. Remoção de recursão no símbolo inicial

Remoção de recursão no símbolo inicial: Essa funcionalidade consiste em transformar a gramática de forma que o símbolo inicial se limite a iniciar derivações. A Figura 4 ilustra o resultado dessa funcionalidade aplicada à gramática de referência. O algoritmo para remoção de recursão no símbolo inicial consiste em verificar se a gramática gera uma derivação do tipo $S \Rightarrow^* \alpha S \beta$. Caso possua, é criado um novo símbolo não-terminal S' e inserida uma nova regra do tipo $S' \rightarrow S$, removendo então a recursão no símbolo inicial. Caso contrário, a gramática não possui recursão no símbolo inicial e não é alterada. Nesse exemplo, além da transformar a gramática, proveu-se uma explicação do problema e do algoritmo descrito acima [10].

Remoção de produções vazias: Durante o processo de derivação de uma palavra, a forma sentencial pode conter variáveis que não contribuem para a formação de uma sentença, as quais são classificadas como produções vazias. Essa funcionalidade remove as produções vazias em uma GLC, garantindo a não contratibilidade da gramática. Para a realização dessa transformação, implementou-se o algoritmo λ -rules [10] que torna a gramática não-contrátil ou essencialmente não-contrátil. A Figura 5 ilustra o resultado da execução dessa funcionalidade na gramática de referência. Além da transformação realizada sobre a gramática, o aplicativo proveu uma explicação do processo de determinação das variáveis anuláveis (1), da substituição das variáveis (2) e da remoção dos símbolos λ (3).

Remoção de regras de cadeia: Ocasionalmente ocorrem em GLCs regras com o formato $A \rightarrow B$, as quais são denominadas regras de cadeia. Essas regras não contribuem de forma direta para o processo de formação de uma palavra e devem ser removidas por meio de um algoritmo que localize as regras de cadeia e as substitua pelas produções re-

Remoção de Produções Vazias

Resultado:

```
S -> aBC | a | aABC | S
A -> aA | a
B -> bcB | bc
C -> cC | c
D -> abc | E
E -> a | Ea
```

Algoritmo:

O algoritmo para remoção de regras λ consiste em 3 passos:

(1) Determinar o conjunto das variáveis anuláveis.

$NULL = \{A \mid \langle A \rightarrow \lambda \rangle \in P\}$

repita

PREV = NULL

para cada $A \in V$ faça

se $A \rightarrow w$ e $w \in PREV^*$ faça

NULL = NULL \cup $\{A\}$

até NULL == PREV

	NULL	PREV
(0)	{A}	\emptyset
(1)	{A}	{A}

(2) Adicionar regras em que as ocorrências de variáveis nulas são omitidas. Por exemplo, assumamos a regra $A \rightarrow BABA$ e B é uma variável anulável. Logo, são inseridas as seguintes regras: $A \rightarrow ABA$, $A \rightarrow BAa$ e $A \rightarrow Aa$.

S -> aBC | a | aABC | S

A -> λ | aA | a

B -> bcB | bc

C -> cC | c

D -> abc | E

E -> a | Ea

(3) Remover as regras λ .

S -> aBC | a | aABC | S

A -> λ | aA | a

B -> bcB | bc

C -> cC | c

D -> abc | E

E -> a | Ea

Figura 5. Remoção de produções vazias

ferentes a variável destino da cadeia [10]. Por exemplo, em uma gramática que possua a seguinte regra $A \rightarrow B$, será realizada a substituição de B em A por todas as produções existentes em B . A Figura 6 ilustra o processo de remoção de regras de cadeia na gramática de referência. Além da transformação realizada sobre a gramática, o aplicativo proveu o conjunto de cadeias de cada variável (1), o destaque das regras de cadeia (2) e uma explicação do processo de substituição das mesmas (3).

Remoção de Regras de Cadeia

Resultado:

```
S -> a | aABC
A ->  $\lambda$  | aA
B -> bc | bcB
C -> c | cC
D -> a | Ea | abc
E -> a | Ea
```

Algoritmo:

A remoção de regras de cadeia substitui as ocorrências de uma cadeia diretamente pelas regras da variável renomeada.

(1) O primeiro passo do algoritmo é montar as cadeias de cada variável.

CHAIN(A) = {A}

PREV = \emptyset

repita

NEW = CHAIN(A) - PREV

PREV = CHAIN(A)

para cada $B \in NEW$ faça

para cada $B \rightarrow C$ faça

CHAIN(A) = CHAIN(A) \cup {C}

até CHAIN(A) == PREV

Variável Cadeia

(S)	{S}
(A)	{A}
(B)	{B}
(C)	{C}
(D)	{D, E}
(E)	{E}

(2) Destacar as cadeias encontradas.

S -> a | aABC | S

A -> λ | aA

B -> bc | bcB

C -> c | cC

D -> abc | E

E -> a | Ea

(3) Substituir as cadeias encontradas:

S -> a | aABC

A -> λ | aA

B -> bc | bcB

C -> c | cC

D -> a | Ea | abc

E -> a | Ea

Figura 6. Remoção de regras de cadeia

Remoção de variáveis que não geram terminais: Essa funcionalidade consiste em remover as variáveis que não geram terminais da gramática. Para ilustrar o funcionamento dessa funcionalidade sem grandes alterações na gramática de referência, a regra $E \rightarrow a$ foi removida, fazendo com que a variável E não gere terminais. A Figura 7 ilustra o processo de remoção de variáveis que não geram terminais na gramática de referência. Além da transformação realizada sobre a gramática, o aplicativo apresenta a construção do conjunto $TERM$ que possui as variáveis que produzem terminais diretamente ou indiretamente (1), e o processo de remoção das variáveis que não estão em $TERM$ (2).

Remoção de símbolos não alcançáveis: Essa funcionalidade remove símbolos não alcançáveis a partir do símbolo inicial S , i.e., aqueles não-terminais A que não aparecem em uma derivação da forma $S \Rightarrow^* \alpha A \beta$. A Figura 8 ilustra o processo de remoção de

Remoção de variáveis que não geram terminais

Resultado:
 $S \rightarrow a \mid aABC \mid S$
 $A \rightarrow \lambda \mid aA$
 $B \rightarrow bc \mid bcB$
 $C \rightarrow c \mid cC$
 $D \rightarrow abc$

Algoritmo:
 Remove as regras que não geram terminais. Consiste de dois passos:
 (1) Determinar quais variáveis geram terminais direta e indiretamente.

$TERM = \{A \mid \text{existe uma regra } A \rightarrow w \in P, \text{ com } w \in \Sigma^+\}$
repita
 $PREV = TERM$
para cada $A \in V$ **faça**
 se $A \rightarrow w \in P$ e $w \in (PREV \cup \Sigma)^*$ então
 $TERM = TERM \cup \{A\}$
até $PREV == TERM$

	TERM	PREV
(0)	{D, A, S, B, C}	\emptyset
(1)	{D, A, S, B, C}	{D, A, S, B, C}

(2) Remover as variáveis que não estão em {S, A, B, C, D}, i.e., {E}.
 $S \rightarrow a \mid aABC \mid S$
 $A \rightarrow \lambda \mid aA$
 $B \rightarrow bc \mid bcB$
 $C \rightarrow c \mid cC$
 $D \rightarrow abc \mid E$
 $E \rightarrow Ea$

Figura 7. Remoção de símbolos não-terminais

Remoção de símbolos não alcançáveis

Resultado:
 $S \rightarrow a \mid aABC \mid S$
 $A \rightarrow \lambda \mid aA$
 $B \rightarrow bc \mid bcB$
 $C \rightarrow c \mid cC$

Algoritmo:
 Remover as variáveis não alcançáveis no processo de derivação de uma palavra.
 (1) Determinar quais variáveis são alcançáveis a partir do símbolo inicial S.

$REACH = \{S\}$
 $PREV = \emptyset$
repita
 $NEW = REACH - PREV$
 $PREV = REACH$
para cada $A \in NEW$ **faça**
para cada $A \rightarrow w$ **faça**
 adicione as variáveis de w em $REACH$ |
até $REACH == PREV$

	REACH	PREV	NEW
(0)	{S}	\emptyset	\emptyset
(1)	{A, S, B, C}	{S}	{S}
(2)	{A, S, B, C}	{A, S, B, C}	{A, B, C}

(2) Remover as variáveis que não estão em {S, A, B, C}, i.e., {D, E}.
 $S \rightarrow a \mid aABC \mid S$
 $A \rightarrow \lambda \mid aA$
 $B \rightarrow bc \mid bcB$
 $C \rightarrow c \mid cC$
 $D \rightarrow abc \mid E$
 $E \rightarrow a \mid Ea$

Figura 8. Remoção de símbolos não alcançáveis

variáveis inalcançáveis na gramática de referência [10]. Além da transformação realizada, o aplicativo apresenta o construção do conjunto *REACH* que possui as variáveis alcançáveis (1) e o processo de remoção das variáveis que não estão em *REACH* (2).

Forma Normal de Chomsky (FNC): Essa funcionalidade padroniza as regras da gramática em um dos formatos da FNC, como descrito na Seção 2. O algoritmo requer que a GLC de entrada não possua símbolo inicial recursivo, produções vazias, regras de cadeia e símbolos inúteis (variáveis que não geram terminais e símbolos não alcançáveis), logo, a ordem da aplicação dos algoritmos é importante. Portanto, o aplicativo pode automaticamente transformar a gramática para satisfazer as pré-condições do algoritmo utilizado [10]. A Figura 9 ilustra o processo de transformação da gramática de referência para a FNC. Além da transformação realizada sobre a gramática, o aplicativo destaca as regras não válidas na FNC (1) e as novas regras criadas (2).

Forma Normal de Chomsky

Remoção de recursão no símbolo inicial:

Remoção de produções vazias:

Remoção de regras de cadeia:

Remoção de símbolos não terminais:

Remoção de símbolos não alcançáveis:

Forma Normal de Chomsky:

Resultado:
 $S' \rightarrow I_3 I_6 \mid I_3 I_4 \mid a$
 $A \rightarrow a \mid T_3 A$
 $B \rightarrow T_1 T_5 \mid T_1 T_2$
 $C \rightarrow T_2 C \mid c$
 $I_1 \rightarrow b$
 $T_2 \rightarrow c$
 $T_3 \rightarrow a$
 $T_4 \rightarrow BC$
 $T_5 \rightarrow T_2 B$
 $T_6 \rightarrow AT_7$
 $T_7 \rightarrow BC$

Algoritmo:

Uma GLC $G = (V, \Sigma, P, S)$ está na Forma Normal de Chomsky se suas regras tem uma das seguintes formas

- $A \rightarrow BC$ onde $B, C \in V - \{S\}$

- $A \rightarrow a$ onde $a \in \Sigma$

- $S \rightarrow \lambda$

(1) Identificar as regras que não estão na Forma Normal de Chomsky.

$S' \rightarrow aBC \mid a \mid aABC$

$A \rightarrow aA \mid a$

$B \rightarrow bc \mid bcB$

$C \rightarrow c \mid cC$

(2) Transformar tais regras em um dos formatos válidos

$S' \rightarrow I_3 I_6 \mid I_3 I_4 \mid a$

$A \rightarrow a \mid T_3 A$

$B \rightarrow T_1 T_5 \mid T_1 T_2$

$C \rightarrow T_2 C \mid c$

$T_1 \rightarrow b$

$T_2 \rightarrow c$

$T_3 \rightarrow a$

$T_4 \rightarrow BC$

$T_5 \rightarrow T_2 B$

$T_6 \rightarrow AT_7$

$T_7 \rightarrow BC$

Figura 9. Forma Normal de Chomsky

Remoção de recursão direta à esquerda: Essa funcionalidade remove recursões diretas à esquerda, as quais podem produzir *loops* infinitos em analisadores sintáticos descendentes [1]. A Figura 10 ilustra o processo de remoção de recursão direta à esquerda na gramática de referência. Além da transformação realizada sobre a gramática, o aplicativo identifica as regras com recursão (1) e o processo de remoção de tais regras (2).

Remoção de Recursão Direta à Esquerda

Resultado:

```
S -> a | aABC
A -> λ | aA
B -> bcB | bc
C -> cC | c
D -> abc | E
E -> a | aZ1
Z1 -> aZ1 | a
```

Algoritmo:

Recursividade direta à esquerda pode produzir "loops infinitos" em analisadores sintáticos descendentes (top-down).

(1) O primeiro passo é identificar a recursão.

Suponha a regra genérica diretamente recursiva à esq.:

$$A \rightarrow A\mu_1 | A\mu_2 | \dots | A\mu_m | v_1 | v_2 | \dots | v_n$$

Regra equivalente não-recursiva à esquerda:

$$A \rightarrow v_1 | v_2 | \dots | v_n | v_1Z | v_2Z | \dots | v_nZ$$

$$Z \rightarrow \mu_1Z | \mu_2Z | \dots | \mu_mZ | \mu_1 | \mu_2 | \dots | \mu_m$$

Remoção de Recursão Direta e Indireta à Esquerda

Resultado:

```
S -> a | aABC | S
A -> λ | aA
B -> bc | bcB
C -> c | cC
D -> abc | E
E -> a | Ea
```

Algoritmo:

A gramática inserida possui ciclos. A regra S -> S é um ciclo.

Figura 10. Remoção de recursão direta à esquerda

Figura 11. Remoção de recursão indireta à esquerda

Remoção de recursão indireta à esquerda: Essa funcionalidade remove recursões indiretas à esquerda. A Figura 11 ilustra o processo de remoção de recursão indireta à esquerda na gramática de referência. No entanto, o algoritmo utilizado [10] requer que a gramática inserida não possua ciclos, o que impede a execução da funcionalidade na gramática de referência. Caso as pré-condições sejam satisfeitas, além da transformação realizada sobre a gramática, o aplicativo ordena as variáveis (1), localiza as recursões (2) e descreve o processo de alteração das regras (3).

Forma Normal de Greibach (FNG): Essa funcionalidade padroniza as regras da gramática em um dos formatos da FNG, como descrito na Seção 2. O objetivo é garantir que não ocorra recursões diretas e indiretas à esquerda. A Figura 12 ilustra o processo de transformação da gramática de referência para a FNG. Além da transformação realizada sobre a gramática – que usualmente envolve a remoção de recursão à esquerda direta e indireta – o aplicativo destaca as regras não válidas na FNG (1) e as novas regras criadas (2).

Algoritmo de reconhecimento CYK: Essa funcionalidade verifica se uma palavra pode ser gerada por uma GLC. O algoritmo CYK utiliza de uma abordagem *bottom-up* com o intuito de analisar a sentença associando-a com as regras da gramática. A Figura 13 ilustra o processo de reconhecimento da palavra *abcc* na gramática de referência. Além da construção da matriz triangular inferior, o aplicativo ilustra o passo-a-passo de sua construção. Nesse exemplo, o símbolo inicial da gramática (S') encontra-se no topo da matriz, logo pode-se concluir que a palavra *abcc* pertence a gramática.

4. Projeto, Implementação e Testes

Com o intuito de se tornar um recurso educacional aberto, LFApp apoia-se em soluções livres—IDE Android Studio e Android SDK—e utiliza apenas bibliotecas nativas do Android. Mais importante, o aplicativo é classificado como software livre, sendo disponibilizado desde o início de seu desenvolvimento em um repositório público no GitHub.¹ A

¹<https://github.com/rterrabh/LFApp>

Forma Normal de Greibach

Remoção de recursão à esquerda:

Resultado:

$S' \rightarrow T_3T_6 | T_3T_4 | a$
 $A \rightarrow a | T_3A$
 $B \rightarrow T_1T_5 | T_1T_2$
 $C \rightarrow T_2C | c$
 $T_1 \rightarrow b$
 $T_2 \rightarrow c$
 $T_3 \rightarrow a$
 $T_4 \rightarrow bT_5C | bT_2C$
 $T_5 \rightarrow cB$
 $T_6 \rightarrow aT_7 | aAT_7$
 $T_7 \rightarrow bT_2C | bT_5C$

Algoritmo:

A remoção de recursão à esquerda consiste em ordenar as variáveis da gramática e organizar as regras da forma que a variável do lado esquerdo sempre possua valor menor do que a variável do lado direito.

(1) Ordenar as variáveis da gramática.

Variável	Valor
S'	1
A	2
B	3
C	4
T1	5
T3	6
T2	7
T6	8
T7	9
T4	10
T5	11

(2) Localizar as recursões.

$S' \rightarrow T_3T_6 | T_3T_4 | a$
 $A \rightarrow a | T_3A$
 $B \rightarrow T_1T_5 | T_1T_2$
 $C \rightarrow T_2C | c$
 $T_1 \rightarrow b$
 $T_2 \rightarrow c$
 $T_3 \rightarrow a$
 $T_4 \rightarrow BC$
 $T_5 \rightarrow T_2B$
 $T_6 \rightarrow AT_7$
 $T_7 \rightarrow BC$

(3) Alterar as regras

$S' \rightarrow T_3T_6 | T_3T_4 | a$
 $A \rightarrow a | T_3A$
 $B \rightarrow T_1T_5 | T_1T_2$
 $C \rightarrow T_2C | c$
 $T_1 \rightarrow b$
 $T_2 \rightarrow c$
 $T_3 \rightarrow a$
 $T_4 \rightarrow bT_5C | bT_2C$
 $T_5 \rightarrow cB$
 $T_6 \rightarrow aT_7 | aAT_7$
 $T_7 \rightarrow bT_2C | bT_5C$

Forma normal de Greibach:

Resultado:

$S' \rightarrow aT_4 | a | aT_6$
 $A \rightarrow aA | a$
 $B \rightarrow bT_2 | bT_5$
 $C \rightarrow cC | c$
 $T_1 \rightarrow b$
 $T_2 \rightarrow c$
 $T_3 \rightarrow a$
 $T_4 \rightarrow bT_5C | bT_2C$
 $T_5 \rightarrow cB$
 $T_6 \rightarrow aT_7 | aAT_7$
 $T_7 \rightarrow bT_2C | bT_5C$

Algoritmo:

Uma GLC $G = (V, \Sigma, P, S)$ está na FN de Greibach se suas regras têm uma das seguintes formas:

$A \rightarrow aA_1A_2A_3 \dots A_n$ onde $a \in \Sigma$ e $A_1 \dots A_n \in V - \{S\}$
 $A \rightarrow a$ onde $a \in \Sigma$
 $A \rightarrow \lambda$

(1) Localizar as regras que não estão na Forma Normal de Greibach.

$S' \rightarrow T_3T_6 | T_3T_4 | a$
 $A \rightarrow a | T_3A$
 $B \rightarrow T_1T_5 | T_1T_2$
 $C \rightarrow T_2C | c$
 $T_1 \rightarrow b$
 $T_2 \rightarrow c$
 $T_3 \rightarrow a$
 $T_4 \rightarrow bT_5C | bT_2C$
 $T_5 \rightarrow cB$
 $T_6 \rightarrow aT_7 | aAT_7$
 $T_7 \rightarrow bT_2C | bT_5C$

(2) Transformar tais regras em um dos formatos válidos

$S' \rightarrow aT_4 | a | aT_6$
 $A \rightarrow aA | a$
 $B \rightarrow bT_2 | bT_5$
 $C \rightarrow cC | c$
 $T_1 \rightarrow b$
 $T_2 \rightarrow c$
 $T_3 \rightarrow a$
 $T_4 \rightarrow bT_5C | bT_2C$
 $T_5 \rightarrow cB$

Figura 12. Forma Normal de Greibach

CYK

Resultado:

				{S, T ₆ }
			{T ₄ , T ₇ }	
		(B)	(C)	
	(A, S, T ₃)	(T ₁)	(C, T ₂)	(C, T ₂)
a	b	c	c	

Algoritmo:

(1) O primeiro passo do algoritmo é adicionar à tabela as variáveis que produzem os respectivos terminais diretamente.

Há alguma regra que gere a diretamente? Sim.

$T_3 \rightarrow a, A \rightarrow a, S' \rightarrow a$

			{A, S, T ₃ }
		(T ₁)	(C, T ₂)
a	b	c	c

(2) O segundo passo do algoritmo é adicionar à tabela as variáveis que produzem as sentenças de tamanho dois. Há alguma regra que gere AT₁, ST₁, T₃T₁? Não.

		(B)	(C)
	{A, S, T ₃ }	(T ₁)	(C, T ₂)
a	b	c	c

(3) O terceiro passo do algoritmo é adicionar à tabela as variáveis que produzem as sentenças de tamanho três. Há alguma regra que gere AB, S'B, T₃B, -C, -T₂? Não.

				{T ₄ , T ₇ }
		(B)	(C)	
	{A, S, T ₃ }	(T ₁)	(C, T ₂)	(C, T ₂)
a	b	c	c	

E assim por diante.

Figura 13. Reconhecimento CYK

Figura 14 apresenta as duas principais classes do aplicativo: Grammar que representa uma gramática de qualquer nível da Hierarquia de Chomsky que é composta por um conjunto de Rule que representa uma regra do tipo $\mu \rightarrow \nu$. Objetos Grammar são imutáveis, logo todas as funcionalidades sempre retornam um novo objeto Grammar como resposta.

Com o objetivo de evitar erros de codificação, todos os algoritmos desenvolvidos foram testados por meio de testes de unidade, que verificam cada unidade da aplicação [8], garantindo o correto funcionamento do LFA. Portanto, utilizando o *framework* JUnit², foram desenvolvidos 39 casos de testes, os quais são executados antes da publicação de uma nova versão do aplicativo. A Figura 15 ilustra um *screenshot* da execução de um teste de regressão no aplicativo. O teste de regressão consiste em submeter as versões mais recentes do software aos mesmos testes que as versões anteriores já foram submetidas [8]. O objetivo é garantir que as funcionalidades previamente existentes permanecem corretas, i.e., o software não regrediu.

5. Trabalhos Relacionados

Existem diversas ferramentas similares à proposta neste artigo. Popular no âmbito de Teoria da Computação e LFA, a ferramenta JFLAP disponibiliza funcionalidades com

²http://www.junit.org

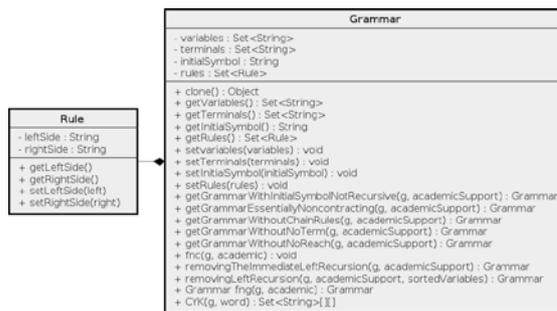


Figura 14. Diagrama de classes



Figura 15. Testes de unidade

foco principal em autômatos. Para a classe das LLCs, o JFLAP disponibiliza quatro tipos de transformações, sendo elas: conversão de APs para GLCs e mais três algoritmos para conversão de GLCs em APs [5]. Mesmo possuindo foco na classe das LLCs, ao contrário de LFApp, o JFLAP não apresenta como foco principal as transformações de GLCs, o que é amplamente abordado neste trabalho.

Yandre et al. [4] propuseram a ferramenta SCTMF (Software para a Criação e Teste de Modelos Formais) com o objetivo de auxiliar o ensino de LFA. A ferramenta consiste em um software para a criação e testes de modelos formais em máquinas aceitadoras da hierarquia de Chomsky, tais como Autômatos Finitos (AFs), Autômatos com Pilha (APs), Autômatos Linearmente Limitados (ALLs) e Máquinas de Turing (MTs). Em contraste com LFApp, seu principal foco é em máquinas aceitadoras, sem possuir algoritmos sobre gramáticas.

Menezes et al. [12] propuseram *Language Emulator*, um ambiente com foco em Linguagens Regulares (LRs) que apoia o ensino de LFA. Dentre suas funcionalidades estão algoritmos sobre Expressão Regular (ER), Gramática Regular (GR), AFs, e máquinas de Mealy e Moore. A ferramenta é completa no que se refere à classe das LRs, implementando as técnicas e algoritmos abordados na área de conhecimento. No entanto, em contraste com LFApp, não possui implementação de modelos referentes à classe das LLCs.

Jukemura et al. [6] descrevem GAM, uma ferramenta que simula diferentes tipos de máquinas abstratas para apoiar o ensino de Teoria da Computação. Dentre suas funcionalidades estão simulação de AFs, transformação de AFND para AFD, transformação de AFD para ER, transformação de ER para AFND- λ , simulação de APs e simulação de MTs. Mesmo possuindo funcionalidades no nível de LLCs (e.g., simulação de AP), o qual é o foco de LFApp, GAM não possui funcionalidades que lidam com transformações de gramáticas e formas normais.

Ainda é importante mencionar alguns jogos educacionais. Morais et al. [3] descrevem o projeto de um jogo educativo baseado em *mobile learning*, visando melhorar o ensino de alguns tópicos básicos em disciplinas introdutórias de Teoria da Computação. Silva et al. [9] descrevem o jogo educacional *Automata Defense 2.0* e a experiência de sua utilização como apoio pedagógico em LFA. Os autores ainda apresentam os resultados de testes de usabilidade e uma avaliação preliminar da sua eficácia pedagógica.

6. Considerações Finais

Linguagens Formais e Autômatos (LFA) é uma importante disciplina da computação que aborda modelos matemáticos que possibilitam a especificação e reconhecimento de lin-

guagens, suas propriedades e características. No entanto, os algoritmos e técnicas abordadas na disciplina são complexos e de difícil assimilação, o que implica em um alto índice de reprovações. Diante disso, este trabalho apresentou LFApp, um aplicativo – desenvolvido para celulares e *tablets* com SO Android – que provê aos alunos não somente a resolução de problemas envolvendo Linguagens Regulares e Linguagens Livres de Contexto, mas também, uma interface com alto caráter acadêmico que descreve e ilustra cada etapa da execução dos algoritmos de modo a apoiar os alunos no processo de aprendizagem.

Como principais contribuições do LFApp, espera-se (i) prover um ambiente de estudos interativo para os alunos de forma a auxiliar no processo de ensino e aprendizagem de LFA e (ii) promover a ideia do uso de aplicativos móveis educacionais. Como trabalhos futuros, pretende-se (i) avaliar o aplicativo no apoio ao ensino da disciplina de LFA, (ii) estender LFApp para máquinas aceitadoras desenvolvendo, por exemplo, algoritmos para simulação de AF/AP/ALL/MT, conversão de AFND para AFD, minimização de AFD, etc., (iii) expandir a ferramenta para abordar todos os níveis da Hierarquia de Chomsky e (iv) desenvolver versões para outras plataformas (iOS e Windows Mobile).

A ferramenta LFApp e seu código fonte estão publicamente disponíveis em:

http://www.dcc.ufla.br/~terra/papers/2016_wei_lfapp

Agradecimentos: Este trabalho foi apoiado pela FAPEMIG, CAPES e CNPq.

Referências

- [1] Alfred V Aho, Ravi Sethi, and Jeffrey D Ullman. *Compiladores: Princípios, técnicas e ferramentas*. LTC, 2 edition, 2008.
- [2] Noam Chomsky. Three models for the description of language. *IRE Transactions on Information Theory*, 2(3):113–124, 1956.
- [3] Dyego Carlos Sales de Moraes, Antônio DPC Alencar, and Rodrigo de Souza. Jogo baseado em m-learning e aprendizado tangencial para auxílio ao ensino de teoria da computação. In *XXII Simpósio Brasileiro de Informática na Educação*, pages 554–557, 2011.
- [4] Yandre M. e G. da Costa, Rafael C. de Meneses, and Flávio R. Uber. Uma ferramenta para auxílio didático no ensino de teoria da computação. In *XVI Workshop sobre Educação em Computação (WEI)*, pages 208–217, 2008.
- [5] Eric Gramond and Susan H Rodger. Using JFLAP to interact with theorems in automata theory. *ACM SIGCSE Bulletin*, 31(1):336–340, 1999.
- [6] Anibal S. Jukemura, Hugo A. D. do Nascimento, and Joaquim Q. Uchôa. GAM: Um simulador para auxiliar o ensino de linguagens formais e de autômatos. *XIII Workshop sobre Educação em Computação (WEI)*, pages 2432–2443, 2005.
- [7] Farhad Manjoo. A murky road ahead for Android, despite market dominance. *The New York Times*, 2015.
- [8] Roger S Pressman. *Engenharia de software*. Bookman, 7 edition, 2011.
- [9] Rômulo César Silva, Ricardo Luis Binsfeld, Izaura Maria Carelli, and Rodrigo Watanabe. Automata defense 2.0: reedição de um jogo educacional para apoio em linguagens formais e autômatos. In *XXI Simpósio Brasileiro de Informática na Educação*, 2010.
- [10] Thomas A. Sudkamp. *Languages and machines: an introduction to the theory of Computer Science*. Addison-Wesley, 3 edition, 2005.
- [11] Ricardo Terra. Dados da disciplina de Linguagens Formais e Autômatos. Technical report, Universidade Federal de Lavras (UFLA), 2016.
- [12] Luiz Filipe M. Vieira, Marcos Augusto M. Vieira, and Newton J. Vieira. Language emulator, a helpful toolkit in the learning process of computer theory. In *35th SIGCSE Technical Symposium on Computer Science Education (SIGCSE)*, pages 135–139, 2004.