Visualizing Relations between Computational Models and Grammars of Context-Free Languages

M. AYALA-RINCÓN¹,² H. W. POUBEL¹, R. B. NOGUEIRA*³, ¹Grupo de Teoría da Computação, Departamento de Matemática and ²Departamento de Engenharia Mecânica, Universidade de Brasília, 70900-010 Brasília D.F., Brasil ayala@mat.unb.br haydee@mat.unb.br

Abstract. We present an extension of the animation tool SAGEMoLiC, which enalrs it for the treatment of the equivalences between context-free languages (grammars) and pushdown automata, that are their machine models. The main difference between our system and many other animation systems is the possibility of visualization, during the translation of a model (and/or grammatical representation) to other, the key notions which guarantee the correctness of these transformations. Consequently, our tool is a truly educational system and not simply a translation tool between models. One can, for instance, develop step by step the transformation of a grammar to its Greibach normal form and then, from the production rules of the grammar, one can obtain the transitions of the corresponding pushdown automaton.

Keywords: Automata theory, Formal languages, Visualization, Algorithm Animation, Pushdown automata, Context-free Grammars

1. Introduction

We had introduced a new methodology of visualization of the relations between finite automata, regular grammars and expressions in SAGEMoLiC⁴ [5, 1]. Our motivation to develop this system grew with our experience in teaching Automata Theory and Formal Languages at the Universidade de Brasília. Most of the basic concepts studied in this area can be easily understood by interacting with computer simulations of the involved machine models. For instance, the mechanics of finite state automata and Turing machines can be quickly assimilated after some simulations of simple examples of these machines with systems like Deus ex Machina [15], JFLAP [9], Turing’s World [3] as well as SAGEMoLiC. Simulation of

¹Research partially supported by Grant 47488/01-6 of CNPq Brazilian Council
²Partially supported by CNPq Brazilian Council
³Supported by CNPq Brazilian Council
⁴Available at www.mat.unb.br/~ayala/70group/SAGEMoLiC. SAGEMoLiC is a Portuguese acronym for Sistema de Animação Gráfica de Teoremas de Equivalência entre Modelos e Línguagens Computacionais.
specific machines is a very productive exercise in order to understand each kind of
machine, but the most important concepts of formal language and automata theory
are the ones that establish the relations between these models, their grammars and
languages.

These relations are established by constructive proofs that allow, from a specific
instance of a model or language representation as input, the construction of the
related model or language representation as output.

For example, a non-deterministic finite state automaton can be transformed into
a deterministic finite state automaton; a context-free grammar into a pushdown
automaton; etc. Implementation of one of the related translations or conversions is
an easy programming exercise, that surely will make clear the subjacent equivalence
relation (at least for the programmer him/herself). But experimentation with the
direct transformations generated by these kind of implementations is not the best
way to make clear to students these relations, since in many cases very important
factors as the correctness of the related translation algorithms are not evident.

In this work we report how this system has been extended for visualizing the
relations between context-free languages related machine models and grammatical
representations. As for the case of regular language models and grammars, SAGE-
MoLiC remains a full active visualization tool or animator of finite state and
pushdown automata that allows for direct transformations or conversions between
the involved machine models and their language representations. For active we
mean that the user has to provide his/her own machine models and/or language
representations for being further executed and simulated according to appropriate
inputs given also for him/her. Consequently, SAGEMoLiC is adequate for being
used for a person with good knowledge of the field, who is able to select adequate
eamples to explain specific situations and notions related with the theory. The
tool was made as flexible and dynamic as possible: for instance, during a simula-
tion of execution of a pushdown automaton, the user can interrupt the simulation,
return to the beginning of the execution, change the input, speed up or slow down
the simulation, etc. This flexibility makes very easy its use and adaptation for the
explanation of dependent notions after some basic concepts are understood via slow
simulations or the use of very simple examples. This makes the tool adequate for
explanation of a sequence of concepts of increasing complexity, that is typical from
the area. Finally, other of the nice properties of SAGEMoLiC is its portability: being
it a Java applet, the system can be used independently of computer architecture
and operational system by means of any web browser on the net.

2. Background

We assume familiarity with the notions of formal language and automata theory as
presented in the textbooks [8, 12, 6, 10, 11, 14, 13, 15]. In the next section we will
present the minimal needed notions and notations on context-free languages.

It is important to remark that most of the currently known systems, as for instanc
Deus ex Machina [15], Turing’s World [3] and JFLAP [4], for machine model
and related language visualization are simple simulators and are not concerned with the visualization of relations between machine models and their languages. From this point of view Rodger’s system JFLAP is surely the most interesting since it also allows for some direct conversions or translations between machine and grammatical language representations. Moreover, in the last papers describing the current evolution of this system it is aimed to granulate the steps in the transformations to make clearer, at least, these transformations [7, 9]. In contrast to these simulators, SAGEmoLiC has been developed for allowing the visualization of relations between machine models and grammatical representations additionally. For the case of regular languages, this system allows for the direct conversions or transformations presented in Figure 1 where as usual the acronyms λ-NFA, NFA, DFA, min-DFA, RE and RG which are usually standing for Non-Deterministic Finite Automata with λ-transitions, Non-Deterministic Finite Automata, Deterministic Finite Automata, Minimal Deterministic Finite Automata, Regular Expressions and Regular Grammars, respectively. Once these transformations have been implemented a good tentative to illustrate the relations between the involved models or language representations is to granulate, as proposed by Rodger, for different examples, the steps of the conversion. This works well in the very simple cases such as construction of a RG from a DFA or construction of a λ-NFA from a RE, where the equivalence relation is straightforward. But doing that for the more complex and interesting cases doesn’t work at all. In these cases, because of the complexity of the transformation, students (and teachers too) have the tendency to focus on the correct execution of the algorithm (and their particular data structure) missing the most important that is the equivalence relation between the models and language representations involved. Observe that this kind of exercise can be done by use of sophisticated debuggers or program animators, but this should not be the objective of a system for illustrating the equivalence relations as the ones we want. The development of dedicated systems to animate these conversion algorithms as it has successfully been done in other contexts such as the one of string matching algorithms (see for example [2]) results inappropriate too. The cause of this is that all these conversion algorithms are based on non graphical representations of the involved machine
models and grammars. In [1] we reported how the interesting cases (which correspond to remarked arrows of the Figure 1) of minimization of finite automata, equivalence between regular expressions and finite automata, inexpressiveness of $\lambda$-transitions and of non-determinism in finite automata have been visualized (and how these visualizations have been justified) in our system. The other cases are easily explained by simple granulated translations as suggested by Rodgers for her system $JFLAP$. For the case of context-free languages - visualization of the relations between context-free grammars and pushdown automata -, this perspective has been maintained. We have developed modules not only for simulation and direct translations but also for visualizing

- grammatical derivations,
- normalizations of context-free grammars and
- their equivalence with pushdown automata.

The second and third visualized relations are illustrated in the Figure 2, where CFG, CNF, GNF and PDA are acronyms for Context-Free Grammar, Chomsky and Greibach Normal Forms and Pushdown Automata, respectively.

For instance, for the case of visualization of the correspondence between context-free grammars and pushdown automata, our system explains how the grammar is translated to its Greibach Normal Form and after that, graphically, how each production corresponds to an extended transition of a pushdown automaton.

\begin{figure}
\centering
\includegraphics[width=0.5\textwidth]{figure2}
\caption{Visualizations with SAGEMoLiC for context-free languages}
\end{figure}

As for the case of regular languages, trying to explain these relations by (granulation) of translation algorithms is wrong, since it is restricted to the peculiarities of the given conversion. The sophisticated and economic software technology from the current personal computers makes it possible more accurate graphical tools that really help to explain these relations as our framework does.

### 3. Visualization of grammar normalizations and grammatical derivations

The needed notions and notations on context-free languages are presented.

For a finite set $A$, $2^A$ denotes its subsets and for $A$ and a binary relation $R$, $A^* = A^+ = R^*$, $R^+$ being respectively, the set of words build concatenating elements
of $A$, including the empty word, and the set of words without the empty word \((A^* \setminus \{\lambda\})\) and, the reflexive-transitive and transitive closure of $R$. Context-Free Grammars (CFG) are denoted by tuples \(G = (V, \Sigma, P, S)\), where $V$ is a finite set of variables, $\Sigma$ the alphabet: a finite set of symbols, $P$ is a finite set of grammatical rules and $S$ is the distinguished initial symbol which belongs to $V$. A rule in $P$ is an element of $V \times (V \cup \Sigma)^*$ denoted by \(A \to w\), where \(A \in V\) and \(w \in (V \cup \Sigma)^*\). The application of a rule \(A \to w \in P\) in a \(uAv \in (V \cup \Sigma)^*\) gives the string $uwv$, which is denoted by \(uAv \Rightarrow uwv\). The derivability relation is $\Rightarrow^*$. A word $w \in \Sigma^*$ is said to belong to the language of the grammar $G$, denoted by $L(G)$, whenever there exists a derivation starting from the initial symbol and finishing in $w$: $S \Rightarrow^* w$. The length of a derivation $S \Rightarrow^* w$ is the number of rules applied and for that we use the more specific notation $S \Rightarrow^n w$.

A CFG $G = (V, \Sigma, P, S)$ is said to be in Chomsky Normal Form (CNF) whenever its rules are restricted to be in one of the following three forms: $S \to \lambda$, $A \to BC$ and $A \to a$, where $A, B, C \in V$ and $a \in \Sigma$. $G$ is said to be in Greibach Normal Form (GNF) if and only if its rules are restricted to be in one of the following two forms: $S \to \lambda$ and $A \to aw$, where $A \in V, a \in \Sigma$ and $w \in \Sigma^*$.

A PDA is a sextuple of the form $M = (Q, \Sigma, \Gamma, \delta, q_0, F)$, where $Q$ is a finite set of states, $\Sigma$ and $\Gamma$ are finite alphabets of input and stack symbols, respectively, $q_0 \in Q$ is the initial state and $F \subseteq Q$ is the set of final states and $\delta : Q \times (\Sigma \cup \{\lambda\}) \times (\Gamma \cup \{\lambda\}) \to 2^{Q \times \Gamma}$ is a transition function. $[q_j, B] \in \delta(q_i, a, A)$ indicates a transition of the automaton moving from the state $q_i$ to the state $q_j$, processing the input symbol $a$; removing $A$ from the top stack (pop the stack) and pushing $B$ onto the stack. As finite automata PDAs are depicted by SAGEMoLiC by their state diagrams. The transitions of extended PDAs are functions from the same domain into $2^{Q \times \Gamma}$: transitions of extended PDAs push words over the work alphabet into the stack. A configuration of a PDA is a triple $[q_i, v, \alpha]$, $v \in \Sigma^*$, consisting of the current state, the unprocessed input word and the contents of the stack, respectively. A PDA processes an input word $v$ starting from the initial configuration $[q_0, v, \lambda]$. The notation $[q_i, u, \alpha] \vdash [q_j, v, \beta]$ indicates that the second configuration can be obtained from the first one by applying a transition of the automaton. A computation with a PDA starts from the initial configuration following the relation $\vdash^*$. We will say that the PDA accepts $v$, denoted by $v \in L(M)$, whenever there exists a computation $[q_0, v, \lambda] \vdash^* [q_f, \lambda', \lambda]$ finishing in a final state $q_f \in F$. We define the length of computations similarly to the length of derivations.

3.1. Visualizing the derivability relation

After specifying a grammar SAGEMoLiC allows for dynamical visualization of the application of grammatical rules. This ability of our system enables the explanation of essential notions such as the derivability relation over interesting CFG examples, which is not unusual when the instructor has to specify a grammar over the blackboard. For instance, an interesting built-in example of a CFG for Wirth's programming language Pascal is available. But the instructor could define other of his/her interest. This ability is not only of importance for illustrating the real ex-
pressiveness of CFG to the students (avoiding the use of uninteresting toy grammars and languages, which are the sole possibility when working over the blackboard), but also of great importance for explaining more sophisticated properties of the theory of CFG such as the pumping lemma. In fact, the key notion which implies this periodicity property of CFGs, can be visualized for any specific CFG, \( G \), and derivation tree of depth greater than the number of rules in \( G \). This visualization is possible because our system allows, for free derivations starting from the initial symbol of the grammar, the flexibility to turn back to previous expressions.

### 3.2. Visualization of grammar normalizations

To transform a CFG in its graphical model, the pushdown automata, it is usual to pass through several steps that simplify the grammar until we reach one of its normal forms. From a CFG in Greibach Normal Form one can construct its corresponding extended PDA as we explain in the next subsection.

Our system exhibits the atomic steps involved in the following grammar simplifications/transformations:

1. elimination of recursion of the initial symbol (rules of the form \( S \rightarrow uSv \), \( u, v \in (\Sigma \cup \Gamma)^* \));
2. elimination of nullable rules (of the form \( A \rightarrow \lambda \), where \( A \in V \setminus S \));
3. elimination of unit or chain rules (of the form \( A \rightarrow B \), where \( A, B \in V \));
4. elimination of useless symbols (that cannot be derived from the start symbol or from which there are no derivation to terminal words) and
5. elimination of direct left recursion (rules of the form \( A \rightarrow Au \), where \( A \in V \) and \( u \in (\Sigma \cup \Gamma)^* \)).

These simplifications are elemental and their visualizations resume to the granulation of the related translation algorithms and, however our tool allows it, they are not explained separately here. In the theory these simplifications are profit as a sequence of prelminar requisites (lemmas) needed for the formal justification of the construction of the CNF and GNF of CFGs. Our system explains these simplifications independently as well as during the visualization of the transformation of a CFG to its Chomsky and/or Greibach Normal Form.

The standard algorithm to translate the production rules of a CFG in its CNF can be described in three steps: firstly, apply the grammar simplifications 1-4; afterwards, replace any terminal symbol \( a \) occurring on the right-hand side, with length greater than or equal to two, of a production by a new variable \( A' \) and include a new (admissible) production \( A' \rightarrow a \); finally, using new variables again, break iteratively all productions whose right-hand side lengths are greater than or equal to two into new productions until all right-hand sides of the productions are of the desired form (that is either consisting of a unique terminal or of two variable symbols).
The correction of this translation algorithm, that is the equivalence of the original and the generated grammar in CNF, is easy to prove, but in the blackboard there is no possibility to capture the intuition of this mechanism naturally. In contrast this is possible with SAGEMoLiC because it permits the visualization of the atomic steps involved in this translation providing during the animated translation messages to the user that explain what is being done.

**Theorem 3.1 (Correctness CFG vs Chomsky Normal Form).** Let $G$ be a CFG. SAGEMoLiC build a CFG $G_C$ in CNF such that $L(G_C) = L(G)$.

**Proof.** Since the animation given by our system is based on the well-known standard transition algorithm mentioned previously, it is only necessary to show the correctness of the three steps involved in it. We suppose simplifications 1-4 are correct. The second step is also sound because what is done is to enlarge derivations with intermediate steps where terminal symbols $a$ are generated through new auxiliary rules $A' \rightarrow a$. Finally, rules of the form $A \rightarrow B_1 \ldots B_n$ are changed by $n - 1$ rules $A \rightarrow B_1 C_1, C_1 \rightarrow B_2 C_2, \ldots, C_{n-2} \rightarrow B_{n-1} B_n$. This step of the translation changes one application of the original rule in the derivation by equivalent derivations of length $n - 1$.

The standard algorithm to translate a CFG in its GNF can be described in six steps: 1. apply the grammar simplifications 1-4; 2. rename the variables of the grammar using a new sequence of variables in increasing order, such as $A_1, A_2, \ldots, A_n$ where $n$ is the number of variables of the grammar; 3. place the productions in the form $A_i \rightarrow A_j\alpha$, where $i \leq j$. This is possible because the set of variables is finite and then, there exists a limit for the increasing productions, i.e., we reach productions of the forms $A_i \rightarrow \alpha\alpha$ or left recursions $A_i \rightarrow A_i\alpha$; 4. eliminate the recursions of the form $A_i \rightarrow A_i\alpha$ introducing new variables and (possibly) recursions on the right (apply simplification 5); 5. place a terminal at the beginning of the right-hand side of each production. This can be done because the $A_{n'}$-productions have a terminal at the beginning of their right-hand side; 6. substitute all terminals (except the first one) on the right-hand side of the productions by new variables and add the new needed productions.

As for the CNF translation SAGEMoLiC allows the (direct translation as well as the) visualization of these steps, giving messages to the user with respect to what is being done.

**Theorem 3.2 (Correctness CFG vs Greibach Normal Form).** Let $G$ be a CFG. SAGEMoLiC build $G_G$ in GNF such that $L(G_G) = L(G)$.

**Proof.** Each of the six simplification steps should be proved sound. The first step coincides with the one for CNF and the second is trivial because the set of variables is finite. The third and fourth steps consist of removing all productions of the form $A_i \rightarrow A_j\alpha$, where $i \leq j$. This is done putting in its place $A_i \rightarrow \alpha\beta$. For all productions of the form $A_j \rightarrow \beta$ and $A_i \rightarrow A_i\alpha$. For doing the last, two kinds of productions are placed: $B_i \rightarrow \alpha$ and $B_i \rightarrow \alpha\beta$ (where $B_i$ are new variables), and also $A_i \rightarrow \delta B_i$ whenever we have $A_i \rightarrow \delta$. For the fifth step, to place a terminal
at the beginning of the right-hand side of each production, we remove all productions of the form \( A_i \rightarrow A_j \alpha \) and put in its place \( A_i \rightarrow \beta \alpha \) for all \( A_j \rightarrow \beta \) of the previous step. The sixth step corresponds to the second of the CNF translation. □

It should be stressed here that these proofs correspond at all to the well-known classical proofs in formal language theory. But what is important is that these proofs justify the explanation given by our system. In fact SAGEMoLiC displays the execution of these translations all the relevant details related with the simplifications and constructions which guarantee a good understanding of the soundness of the involved grammatical relations.

4. CFGs and pushdown automata

The visualization of the equivalence between CFGs and pushdown automata is perhaps the more interesting relation from context-free languages. SAGEMoLiC, in its current status, allows for transformation of a CFG into its associated extended pushdown automaton as well as for visualization of the key notions which guarantee this relation. The series of transformations and normalizations from the previous sections allows us to translate step by step any CFG to its Greibach Normal Form. And from this normalized grammar each transition corresponds to an extended transition of a pushdown automaton. The visualization which our system allows do not differ from the dynamic exhibition over the state diagram of the construction one can find in textbooks and is based on the following theorem.

**Theorem 4.1 (Correctness CFG vs Pushdown Automata).** Let \( G \) be a CFG, SAGEMoLiC builds an extended pushdown automaton \( M \) such that \( L(M) = L(G) \).

**Proof.** By the results of the previous section we can suppose that \( G \) is in GNF. Then each production in \( P \) is either of the form \( S \rightarrow \lambda \) or \( A \rightarrow aw \), where \( a \in \Sigma, A \in V \) and \( w \in V^* \). SAGEMoLiC exhibits the construction of an extended pushdown automaton \( M = (Q, \Sigma, \Gamma, \delta, q_0, F) \) which consists of two states \( Q := \{ q_0, q_1 \} \) being \( F := \{ q_1 \} \) and whose transition function \( \delta \) is built as follows:

\[
\begin{align*}
\delta(q_0, a, \lambda) & := \{[q_1, w] \mid \text{if } S \rightarrow \lambda \in P \} \\
\delta(q_1, a, A) & := \{[q_1, w] \mid \text{if } A \rightarrow aw \in P \} \\
\delta(q_0, \lambda, \lambda) & := \{[q_1, \lambda] \mid \text{if } S \rightarrow \lambda \in P \}
\end{align*}
\]

The correctness of this construction/visualization is proved by showing that for any \( v \in L(G) \) we get \( v \in L(A) \) and vice versa. The former is showed by induction on the length of the derivations \( S \Rightarrow^* v \) and the last by induction on the length of computations. □

5. Conclusions and future work

SAGEMoLiC has been extended, from the case of regular languages reported in [1], to the case of visualization of equivalence relation between machine models...
and grammatical representations of context-free languages. The tool allows for a
detailed explanation of concepts which justify the equivalence of restricted context-
free grammatical representations and/or normalizations, by showing step by step
how productions of the grammars can be replaced into others correspondingly. In
particular, the important cases of Chomsky and Greibach normal forms are visuali-
zed. Also it is possible to illustrate the translation from grammars to corresponding
pushdown automata. Additionally, the system allows for a visualization of the gram-
matical derivation process which results useful for explaining relevant concepts such
as the pumping lemma for context-free languages. As current work the context-free
language module is being extended with a visualization of the generation of gram-
mars from pushdown automata.

The users of our system should have experience with formal language theory to
be able to provide interesting examples of machines and language representations.
For obtaining a tutorial system that could be used directly for students without
knowledge about this field, the system is being extended with a passive module,
where the user will find very well selected built-in examples.

One could argue that SAGEMoLiC formulates a very particular way to explain
some theoretical concepts in the field of formal languages and automata theory, but
what we conclude is that explaining these relations by simple granulation of specific
translation algorithms is wrong, since this is restricted to the peculiarities of the
given conversion. These classical explanation induces students as well as instructors
to forget the grade of generality of the studied equivalence relations and to be
focused (and confused) on properties of the algorithm itself, such as its terminating
property, its correctness and its completeness. Of course, granulating translation
algorithms is useful. But that was yet possible with the technology available some
years ago. Today, the current sophisticated and very cheap software technology from
simple personal computers makes it possible more accurate graphical educational
tools that really help instructors to show and explain these relations to students as
our proposed tool does.

Resumo. Apresentamos uma extensão da ferramenta de animação SAGEMoLiC,
acrescentando o tratamento das equivalências entre linguagens (gramáticas) livres
de contexto e automatos de pilha, que são seus modelos gráficos. A principal diferença
entre o nosso sistema e vários outros de animação continua sendo a possibilidade de
visualizar, durante o processo de transformação de um modelo (e/ou representação
gramatical) no outro, as principais idéias que garantem a correção das provas dessas
transformações. Portanto, trata-se de uma ferramenta de apoio educacional e não
um sistema que simplesmente realiza a transformação entre os modelos. Pode-se,
por exemplo, realizar passo a passo a transformação da gramática para sua forma
normal de Greibach e em seguida, a partir das regras de produção da gramática,
obter as transições do automato à pilha correspondente.

Referências

work to Visualize Equivalences Between Computational Models of Regular


