Information Extraction Grammars

Mónica Marrero[1] and Julián Urbano[2]

[1] Barcelona Supercomputing Center, Spain
`monica.marrero@bsc.es`
[2] Universitat Pompeu Fabra, Barcelona, Spain
`julian.urbano@upf.edu`

Abstract. Formal grammars are extensively used to represent patterns in Information Extraction, but they do not permit the use of several types of features. Finite-state transducers, which are based on regular grammars, solve this issue, but they have other disadvantages such as the lack of expressiveness and the rigid matching priority. As an alternative, we propose Information Extraction Grammars. This model, supported on Language Theory, does permit the use of several features, solves some of the problems of finite-state transducers, and has the same computational complexity in recognition as formal grammars, whether they describe regular or context-free languages.

1 Introduction

Information Extraction (IE) seeks to identify entities in a text and the relations among them, altogether satisfying the user search needs. Named Entity Recognition (NER) is one of the main areas in IE, needed to identify entities of interest such as persons, locations and dates. It is generally accepted that IE systems should be capable of adapting to different entities and domains [4], but patterns to recognize named entities usually respond to different lexicons and grammatical structures, and they often require high-level features (e.g., lemma, letter case, gazetteers). Besides machine learning-based models, grammars are also widely used to represent these patterns, especially regular expressions. However, they can only recognize entities that respond to one type of feature (usually characters), so their use is restricted to entities that follow a simple pattern.

Cascade grammars are used to overcome this limitation. They are built with transducers (finite-state automata that also generate an output language) concatenated such that the output alphabet from one transducer is the input alphabet to the next one [5]. This way, it is possible to use several features in the same pattern. The Common Pattern Specification Language (CPSL) [2] standardizes this representation, although it presents several drawbacks, as detailed in [3].

First, and because they are in cascade, each of the finite-state automata is independent of the others, which can lead to ambiguities and the application of incorrect rules early in the recognition process. For example, we can have a cascade grammar to recognize person names with two transducers, $P1$ and $P2$. The input alphabet to $P1$ are the tokens, and the resulting alphabet contains tags from two gazetteers that identify first and last person names: F and L, respectively. $P2$ identifies full names of persons from these tags by recognizing one of three rules: F, FL or FLL. However, if a text chunk can be tagged as F and L in $P1$, until $P2$ we can not decide which one is better. Consider for instance the text *"Lisa Brown Smith"*, where *"Smith"* can be both F and L. This language is ambiguous, but our domain knowledge may help us disambiguate it by assigning more priority to rule FLL than to FL and F. However, even though the language would no longer be ambiguous, the pattern still would, because $P1$ must decide between F and L *before* executing $P2$.

Second, and because they are as expressive as regular grammars, they can not be used to describe complex languages. Context-free grammars are especially useful when we face markup languages, because it is common to find nested structures that can not be recognized with regular grammars. Wrappers are often used in these cases, but they are task-specific.

To partially overcome the first problem, CPSL has been customized to different extents by different platforms. GATE (`http://gate.ac.uk`) is one of the most successful ones with its JAPE notation. In this scenario we propose Information Extraction Grammars (IEG) as an alternative to represent patterns for entity recognition. It solves the ambiguity issue of cascade grammars, has the expressiveness of context-free grammars, and at the same time it provides more flexibility than wrappers. Furthermore, the main advantage of our proposal is that it contributes to the development of pattern generation methods that can work independently of the kind of features used and the expressiveness of the language to recognize [7].

2 Information Extraction Grammars

Context-free grammars are defined with a tuple $G = (\mathcal{V}, S, \Sigma, \mathcal{P})$, where $\mathcal{V}$ is the set of non-terminal symbols, $S \in \mathcal{V}$ is the initial symbol, Σ is the set of terminal symbols making up the input alphabet, and $\mathcal{P}$ is the set of production rules which recursively define the language recognized by G. Productions are defined by a non-terminal, followed by the production symbol $\rightarrow$ and a sequence of terminals and non-terminals—the production body. The language recognized by G is the set of strings of terminal symbols that can be derived from S:

$$L(G) = \{\omega \in \Sigma^* \mid S \overset{*(G)}{\Longrightarrow} \omega\}$$

where $\overset{*(G)}{\Longrightarrow}$ represents derivations in zero or more steps, that is, replacements of the non-terminals with the body of one of their productions, sequentially from the initial symbol until we reach strings with terminal symbols alone.

These grammars do not support the recognition of more than one input alphabet at the same time. For example, it is not possible to recognize the syntax of a text and whether its tokens are included in gazetteers or not. To solve this problem we have associated conditions to the non-terminal symbols, so that for an input string $\omega \in \Sigma^*$ and a non-terminal $A \in \mathcal{V}$ such that $A \overset{*(G)}{\Longrightarrow} \omega$,

$$\begin{array}{lll} S \to FLL \mid FL \mid F & F \to T & \mathcal{C}_F = \{(FirstGaz, true)\} \\ T \to [\texttt{a-zA-Z0-9}]+ & L \to T & \mathcal{C}_L = \{(LastGaz, true)\} \end{array}$$

Fig. 1. IEG for the recognition of full person names.

ω will be recognized by A only if it meets all conditions associated to A. Each condition is defined with a tuple (f, y), where $f : \Sigma^* \to \mathcal{Y}_f$ is a feature function that receives a string ω and is expected to return $y \in \mathcal{Y}_f$. Note that the set $\mathcal{Y}_f$ of possible values returned by f depends on the particular type of feature. For example, if f returned the lemma of a term, we would have $\mathcal{Y}_f \subset \Sigma^*$; if f returned its length, we would have $\mathcal{Y}_f = \mathbb{N}$. We thus define an IE grammar as $IEG = (G, \mathcal{C})$, where $\mathcal{C}$ is the set of all condition sets assigned to non-terminals. All derivations must therefore meet:

$$A \overset{*(IEG)}{\Longrightarrow} \omega := A \overset{*(G)}{\Longrightarrow} \omega \text{ and } \forall (f, y) \in \mathcal{C}_A \ : \ f(\omega) = y$$

That is, each and every condition associated to A must return the expected value. Figure 1 shows an IEG to recognize person names as in our previous example. To solve the ambiguity of the pattern, we can assign priorities to the different productions of a non-terminal, as we did: $S \to FLL$ first, followed by $S \to FL$ and $S \to F$. At the same time, we could add new rules indicating that the person names have to appear inside specific HTML tags, or new conditions forcing the matching with other features such as POS tagging or font family.

We note that an IEG is similar in definition to an S-attributed grammar, widely used in compilers and language translators [1]. In the latter, any symbol of the grammar may have a set of attributes, and the attributes of the non-terminals are computed in a bottom-up fashion by semantic rules associated to their productions. In particular, these rules are applied when the production reduces an input substring or after the whole input is parsed. However, semantic rules are only used to incorporate semantics to the parse tree; they do *not* intervene in the syntactic analysis. This is precisely the purpose of our conditions: to avoid applying a production, *during* syntactic analysis, when the conditions are not met by the substring.

3 Analysis of Computational Complexity

Text recognition with regular grammars is often performed with automata; for context-free grammars, the Cocke-Younger-Kasami (CYK) is one of the most common algorithms used [6]. But the introduction of conditions to a formal grammar requires modifying these algorithms, which could have an impact in terms of efficiency. Next, we show that the time complexity does not increase as long as the conditions meet some requirements.

3.1 Regular Grammars

Regular grammars are usually represented with regular expressions. A regular expression of length r can be converted to an ε-NFA (non-deterministic finite

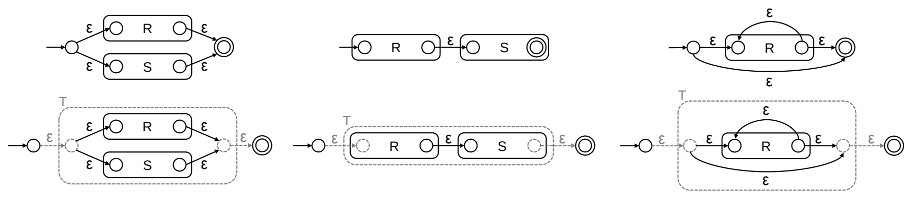

Fig. 2. ε-NFA automata for the union, concatenation and Kleene closure operations (top), and equivalent automata with additional ε-transitions (bottom).

state automaton with ε-transitions) in $\mathcal{O}(r)$ time. Given an input string of length n, an ε-NFA takes $\mathcal{O}(ns^2)$ in the worst case to recognize it, where $s \leq 2r$ is the number of states. This automaton can be transformed into a DFA (deterministic finite state automaton), which takes $\mathcal{O}(n)$ to recognize an input string [6].

We can represent every rule of the IEG with an associated condition as a T-automaton with two extra states and ε-transitions (see Figure 2, bottom). The extra ε-transition at the beginning is used to save the position in the input string; the one at the end is used to check all conditions associated to the regular expression represented by the T-automaton. It will only continue to the acceptance state if all conditions in $\mathcal{C}_T$ are met with the current substring (see Figure 3). Since the union, concatenation and Kleene closure of a regular language is also a regular language (see Figure 2, top), we can concatenate all T-automata to obtain an ε-NFA. For each symbol in the input string, we have to check at most m conditions in each of the t T-automatons, each taking d time. Therefore, the time complexity of recognition remains linear with respect to n. By using appropriate indexing mechanisms, d can be $\mathcal{O}(1)$, so the time complexity would be $\mathcal{O}(n(tm+s^2))$ in the worst case. In practice though, the conditions reduce the number of active states in the ε-NFA, reducing the s^2 factor.

A cascade grammar with $\mathcal{O}(m)$ transducers would have a recognition time of $\mathcal{O}(mns^2)$ if using ε-NFA. If each transducer is converted to a DFA, it can take $\mathcal{O}(mn)$. However, that conversion requires $\mathcal{O}(2^r)$ for each transducer, so it can be simpler and more efficient to use ε-NFA directly [6].

3.2 Context-free Grammars

For an arbitrary string $\omega = \alpha_1\alpha_2\ldots\alpha_n$, the CYK algorithm builds a triangular table X. Each cell X_{ij} in the table contains those non-terminals capable of deriving the substring $\alpha_i\alpha_{i+1}\ldots\alpha_j$ in one or more steps. The table is filled bottom-up, so that in the bottom row we will have those non-terminals that directly derive each of the terminals in ω. In the row above we will have those

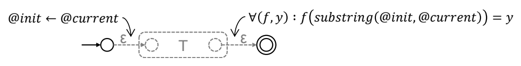

Fig. 3. T-automaton with associated conditions.

$$\begin{array}{ll} S \to \alpha A \beta & S \to \alpha\beta \mid \alpha A \beta \\ A \to \gamma \mid \varepsilon & A \to \gamma \cancel{\mid \varepsilon} \\ \mathcal{C}_S = \{(f_{S1}, y_{S1}), \dots\} & \mathcal{C}_S = \{(f_{S1}, y_{S1}), \dots\} \\ \mathcal{C}_A = \{(f_{A1}, y_{A1}), \dots\} & \mathcal{C}_A = \{(f_{A1}, y_{A1}), \dots\} \end{array}$$

Fig. 4. IEG with ε-productions (left) and its equivalent without ε-productions (right).

non-terminals whose production bodies contain the non-terminals below to form the corresponding substring, and so on. The algorithm stops if in the uppermost cell there is a non-terminal capable of generating the whole input string ω.

This algorithm has a time complexity of $\mathcal{O}(n^3)$, because it takes $\mathcal{O}(n)$ to compute any cell in the table and there are $n(n+1)/2 = \mathcal{O}(n^2)$ cells. If the non-terminals have associated conditions, then the time to check them is added to the time taken by the algorithm to fill up each cell. Again, if the time needed to check each condition is independent of the input string, the overall complexity is kept. But there are two prerequisites for the application of the CYK algorithm: the grammar can not have ε-productions, where the right side contains just the empty string ε; and it must be defined in Chomsky Normal Form (CNF).

Removal of ε-productions. Even though the use of ε-productions may facilitate the design of grammars, they are not essential for any language other than the empty string. Thus, if a language L has a grammar, then $L - \{\varepsilon\}$ has one too without ε-productions [6]. The algorithm for this transformation identifies the nullable non-terminals. A non-terminal A is nullable if $A \overset{*}{\Longrightarrow} \varepsilon$. Whenever A appears in a production body, we make two versions of the production: one with A and one without it, thus removing all productions whose right part is ε.

In our case we need to determine what happens when those nullable non-terminals, or the non-terminals which contain them, have associated conditions. Non-terminals whose production body is just ε can be removed altogether, because we know beforehand whether the conditions are met or not for ε. On the other hand, we have to create two versions for the non-terminals containing them in their production bodies. This does not pose any problem either for non-terminals that contain these nullable symbols, because the conditions are applied upon the resulting substring, regardless of how it is reduced (see Figure 4).

Transformation to CNF. A grammar is in Chomsky Normal Form when all its production rules have one of these forms [6]: (i) $A \to BC$, where A, B and C are non-terminals, or (ii) $A \to \alpha$, where A is a non-terminal and α is a terminal. To achieve this, after removing all ε-productions we need to (i) if there are two or more terminals in a production body, replace each of them with a new non-terminal that produces the terminal itself, and (ii) iteratively reduce production bodies so that they contain two non-terminals at most, by creating again new non-terminals and production rules for them.

In both cases we are adding sublanguages of the language recognized by a non-terminal A to the new non-terminals in its production body. Because all conditions apply to the language recognized by A, and not to its sublanguages, the result is not affected when A has conditions (see Figure 5).

$$\begin{array}{lll}
S \to \alpha\beta\gamma & \text{step 1: } S \to ABC & \text{step 2: } S \to RC \\
\mathcal{C}_S = \{(f_{S1}, y_{S1}), \ldots\} & \quad A \to \alpha & \quad R \to AB \\
 & \quad B \to \beta & \quad A \to \alpha \\
 & \quad C \to \gamma & \quad B \to \beta \\
 & \quad \mathcal{C}_S = \{(f_{S1}, y_{S1}), \ldots\} & \quad C \to \gamma \\
 & & \quad \mathcal{C}_S = \{(f_{S1}, y_{S1}), \ldots\}
\end{array}$$

Fig. 5. Transformation of an IEG into CNF.

4 Conclusions and Future Work

The complexity of patterns for Named Entity Recognition often requires automatic learning methods. Grammar-based models currently used to represent patterns are limited in the features they support and the expressiveness of the languages they recognize. As a consequence, different learning methods are developed for different representation models. For end users, this often requires a previous analysis of the entities to choose a model and learning method.

We propose Information Extraction Grammars as a common model to represent patterns. This model supports a custom set of features, it has the expressiveness of context-free grammars, and it avoids the ambiguity issue of cascade grammars, thus facilitating the development of portable learning methods. An analysis of the time complexity in recognition shows that it is competitive when used by standard recognition algorithms, though further research is needed to optimize them. Another extension may be the use of probabilities in the feature functions, allowing us to select the most likely parse tree for an ambiguous input.

Acknowledgments. Work partially supported by an A4U postdoctoral grant. We thank the reviewers for their comments.

References

1. Aho, A.V., Lam, M.S., Sethi, R., Ullman, J.D.: Compilers: Principles, Techniques, and Tools. Addison-Wesley, 2rd edn. (2006)
2. Appelt, D., Onyshkevych, B.: The common pattern specification language. In: TIPSTER Workshop, 1998. pp. 23–30 (1998)
3. Chiticariu, L., Krishnamurthy, R., Li, Y., et al.: Systemt: An algebraic approach to declarative information extraction. In: ACL. pp. 128–137 (2010)
4. Freitag, D.: Toward general-purpose learning for information extraction. In: ACL. pp. 404–408 (1998)
5. Hobbs, J.R., Riloff, E.: Information Extraction. In: Indurkhya, N., Damerau, F. (eds.) Handbook of Natural Language Processing, pp. 511–532. CRC Press (2010)
6. Hopcroft, J., Motwani, R., Ullman, J.: Introduction to automata theory, languages, and computation. Addison-Wesley, 3rd edn. (2006)
7. Marrero, M.: Diseño y generación semi-automática de patrones adaptables para el reconocimiento de entidades. Journal of the SEPLN 52, 87–90 (2014)