# Regular Approximation of Link Grammar

Filip Ginter, Sampo Pyysalo, Jorma Boberg, and Tapio Salakoski

Turku Centre for Computer Science (TUCS)
and Department of IT, University of Turku
Lemminkäisenkatu 14 A
20520 Turku, Finland
first.last@it.utu.fi

**Abstract.** We present a regular approximation of Link Grammar, a dependency-type formalism with context-free expressive power, as a first step toward a finite-state joint inference system. The approximation is implemented by limiting the maximum nesting depth of links, and otherwise retains the features of the original formalism. We present a string encoding of Link Grammar parses and describe finite-state machines implementing the grammar rules as well as the planarity, connectivity, ordering and exclusion axioms constraining grammatical Link Grammar parses. The regular approximation is then defined as the intersection of these machines. Finally, we implement two approaches to finite-state parsing using the approximation and discuss their feasibility. We find that parsing in the intersection grammars framework using the approximation is feasible, although inefficient, and we discuss several approaches to improve the efficiency.

## 1 Introduction

Finite-state techniques provide simple and efficient models in natural language processing. They have been successfully applied to many basic problems such as tokenization, phonological and morphological analysis, parsing, and language modeling [1]. With the well-studied mathematical apparatus for combining and transforming finite-state machines, including the standard algorithms for intersection, composition, determinization and minimization, it is possible to build large efficient systems by combining many simple, small machines. Moreover, weighted formulations of finite-state machines allow for probabilistic models.

In natural language parsing, context-free parsing algorithms currently form the basis for almost all full parsing approaches producing either a hierarchical phrase structure or a full dependency structure, while finite-state techniques are most commonly applied in shallow parsing which produces no, or very limited, hierarchical structure [2]. There is, however, a growing interest in the application of finite-state techniques to full parsing. Although finite-state models are weaker in terms of expressive power, they are applicable in practical cases, for example as approximations of context-free grammars. Such an approximation recognizes a regular subset or superset of the original context-free language. Approximation approaches have primarily been developed in the context of phrase

T. Salakoski et al. (Eds.): FinTAL 2006, LNAI 4139, pp. 564–575, 2006.

structure grammars (see Nederhof [3] for a detailed discussion). For instance, Grimley Evans [4] obtains a finite-state approximation in an intersection framework, where finite-state representation of the dotted rules in phrase structure parsing is intersected with regular languages expressing constraints on their usage. The result is a finite-state automaton that recognizes the language as sequence of terminals, but does not encode the parse trees. Further, Johnson [5] implements the approximation through left-corner grammar transforms, allowing unlimited left and right recursion without increasing stack depth. By contrast with these phrase-structure based approaches, we focus on dependency grammars. We also have the additional requirement of obtaining a representation of the full dependency analysis.

Several approaches have been introduced for finite-state dependency parsing. Oflazer [6] has developed a robust finite-state full dependency parser as a transducer that is iteratively applied to the input string, each time producing one level of analysis. Elworthy [7] presents a parser with dependency output based on deterministic finite-state transducers. In the framework of finite-state intersection grammars [8], parsing is treated as an intersection problem. For each sentence, a finite-state automaton (FSA) is built that generates the sentence together with all syntactic hypotheses allowed by the grammar for the individual words. This FSA is then intersected with automata that implement the grammatical constraints. The result of the intersection is an FSA that describes all grammatical analyses of the sentence. Yli-Jyrä has recently advanced the finite-state intersection grammars to allow full tree structures and resolve all structural ambiguities. This paper is mainly related to Yli-Jyrä [9], where a finite-state approximation of Hays and Gaifman dependency grammars [10,11] is introduced.

In this paper, we present a finite-state approximation of Link Grammar (LG) [12], a dependency-type formalism with context-free expressive power. LG and its parser[1] represent one of the major computational dependency grammar implementations with a wide coverage of general English. Recently, there has been an increased interest in applications of LG in NLP tasks such as information extraction.

One of the key motivations for introducing a finite-state approximation of link grammar is to facilitate the integration of the parser into a finite-state system which identifies the globally optimal solution through joint inference across all different levels of linguistic analysis in Information Extraction [13]. In such an integrated model, each level of analysis produces a set of alternate hypotheses encoded as a finite-state automaton, and the intersection of these automata then encodes the set of all possible analyses structurally compatible with all levels in the system. Subsequently, the globally optimal solution can be identified by a search through this unified automaton, taking into account the local preferences of the individual levels of analysis [14]. The finite-state approximation of LG presented in this paper is a first step toward a finite-state joint inference system.

For the purpose of integration into a unified model, each component must be developed in the context of the whole system. Most importantly, each of the

[1] Available at `http://link.cs.cmu.edu`


**Fig. 1.** Example LG linkage

components must share a representation that allows intersection, in this case, that of finite-state machines. Thus, the representation constrains the implementation, and, at least initially, takes priority over considerations of the efficiency and expressive power of the parser.

Additionally, a finite-state formulation of LG could, for example, in combination with FSA weighting and unsupervised FSA weight estimation algorithms, provide means for developing a statistical model of LG in terms of weighted finite-state machines.

## 2 Link Grammar

The LG formalism is closely related to dependency grammars. An LG parse of a sentence, termed a *linkage*, consists of a set of undirected, typed *links* connecting pairs of words of the sentence (see Figure 1). The links connecting each word to others must fulfill the *linking requirements* given to the word in the grammar: for example, verbs could require a `S` link to the left to connect to their subject.

LG is highly lexical: the grammar rules are only expressed through the linking requirements assigned to individual words. LG differs from traditional dependency grammars in that linkages are unrooted and links do not explicitly identify which word is the governor and which the dependent.

Linkages must further fulfill a set of axioms (termed *meta-rules* by Sleator and Temperley) which, together with the linking requirements of the words, specify the set of grammatical sentences and their analyses. The following sections describe the specification of the linking requirements and the linkage axioms.

### 2.1 Linking Requirements

The linking requirements of each word in the grammar are specified by a formula of *connectors*, each of which has a *type* and a *direction*. The type of a connector is specified by a string of characters, and the direction is either `-` for left or `+` for right. LG linking requirement formulas are built of connectors joined by the *and* and *or* operators (written `&` and `or`). Parentheses are used to specify precedence in formulas. Connectors or larger parts of linking requirement formulas can be made optional by enclosing them in curly brackets (e.g. `{MV+}`), and connectors can be allowed to repeat one or more times by prepending the `@` character. In parsing, links must be formed by connecting left and right connectors of matching types so that all non-optional connectors participate in a link.

As an example, consider the following grammar:

```
colorless red green: A+
ideas theories proofs: {@A-} & (O- or S+)
```

```
sleep dream rest: S- & {O+} & {MV+}
furiously symbolically: MV-
```

The language specified by this grammar requires that the adjectives take an `A` connector to the right, the nouns take any number of `A` connectors to the left and either an `O` connector further to the left or an `S` connector to the right, the verbs take an `S` connector to the left and optionally `O` and `MV` connectors to the right, and the adverbs require an `MV` connector to the left. This language thus includes linkages such as that shown in Figure 1.

### 2.2 Connector Matching

LG connector type strings can consist of any sequence of capital letters, followed by a *subscript* containing any sequence of lowercase letters and a wild-card character. When comparing connectors, shorter subscripts are (conceptually) padded with wild-cards. Two connectors are defined to match if they are equal when wild-cards are considered equal to any lowercase character.

### 2.3 Linkage Axioms

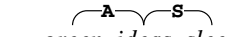
The following four axioms constrain the set of grammatical LG linkages. The *planarity axiom* states that the links of a linkage must not cross when drawn above the sentence. This axiom is closely related to projectivity constraints of dependency grammars. The *connectivity axiom* requires that, when considered as an undirected graph with the words as nodes and the links as edges, the linkage must be connected. The *ordering axiom* specifies that when traversing the connectors of the linking requirement formula of a word from left to right, the words which the connectors link to proceed from near to far. The *exclusion axiom* states that any two words are directly connected by at most one link.

## 3 The Approximation

We now present the components of the regular LG approximation. Note that we approximate the LG grammar, and grammar-external LG features implemented in the parser code, such as the special treatment of coordination, the post-processing mechanism and the robust parsing algorithm [15], are not considered.

### 3.1 String Encoding

We define a string encoding of linkages as follows. The words of the sentence are preceded by a word boundary marker #. Each word is followed by a $\Downarrow$ character separating it from the links *closing* at the word, i.e. links connecting to the word from the left. These are in turn separated by a $\Uparrow$ character from the links *opening* at the word. Opening links are represented by an opening angle bracket character ($<$) followed by the connector type string, and closing links are represented by the type string followed by a closing angle bracket ($>$). Finally, the sentence is terminated by a # character. An example is given in Figure 2.




#green⇓⇑<A#ideas⇓A>⇑<S#sleep⇓S>⇑ #

**Fig. 2.** Example linkage with string encoding

### 3.2 Lexicon Language

The linking requirement formulas of words can be expressed in equivalent *disjunctive forms*. For example, the formula {@A-} & (O- or S+) has the disjunctive form ({@A-} & S+) or ({@A-} & O-). In the LG terminology, the elementary conjunctions in the disjunctive form are referred to as *disjuncts*. Each disjunct represents a particular way of satisfying the linking requirements of the word. We define disjuncts in terms of regular expressions that describe their string representation: for example, the conjunction operator & corresponds to concatenation, the optionality operator {} corresponds to disjunction with an empty string, and optional repetition {@} corresponds to the Kleene star operator. The order of concatenation respects the interpretation of the formula according to the ordering axiom as well as the grouping by connector direction according to the string encoding whereby left connectors are separated from right connectors by the ⇑ symbol.

The regular expression for the whole formula is then a disjunction of the regular expressions of its disjuncts, prefixed with the symbol ⇓. For the disjunctive form above, the corresponding expression is ⇓(((A>)*⇑<S)|((A>)*O>⇑)).

Let $R_f$ be the regular expression corresponding to the linking requirement formula $f$. A grammar entry for a word $w$ with requirement formula $f$ is then represented by the regular expression $R_w$

$$R_w = \#wR_f$$

For example, for the word *ideas* with a grammar entry ideas: {@A-} & (O- or S+), the language $R_{ideas}$ contains the following strings:

| | |
|---|---|
| #ideas⇓O>⇑ | #ideas⇓⇑<S |
| #ideas⇓A>O>⇑ | #ideas⇓A>⇑<S |
| #ideas⇓A>A>O>⇑ | #ideas⇓A>A>⇑<S |
| ... | ... |

Let further $L$ be the *lexicon language* defined by the regular expression

$$L = (R_{w_1}|\ldots|R_{w_n}) + \#$$

where $w_1 \ldots w_n$ are the words defined in the grammar. The lexicon language $L$ thus consists of sequences of words with linking requirements, terminated with the boundary symbol #. $L$ contains strings such as

#green⇓⇑<A#ideas⇓A>⇑<S#sleep⇓S>⇑ #
#sleep⇓S>⇑ #ideas⇓A>⇑<S#green⇓⇑<A#
#sleep⇓S>⇑<MV#furiously⇓MV>⇑ #

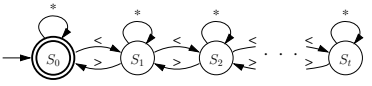



**Fig. 3.** Untyped balanced bracketing FSA $B_t$. The states $S_0$–$S_t$ serve as a memory of the number of currently open brackets.

#green⇓⇑<A#ideas⇓A>O>⇑ #
...

Note that of the strings above, only the first string encodes a valid LG linkage.

### 3.3 Planarity and the Nature of the Approximation

Let us define a *typed balanced bracketing* in the context of the strings encoding LG linkages as a bracketing where brackets do not cross and the connector types associated with the corresponding opening and closing brackets match.

By definition, an LG linkage is planar[2] if and only if its links when drawn above the sentence do not cross. As crossing links directly translate to crossing typed brackets and vice versa, it is easy to see that the string representation contains a typed balanced bracketing if and only if the linkage it encodes does not contain crossing links. Thus, an LG linkage is planar if and only if its encoding string contains a typed balanced bracketing. Enforcing the planarity axiom is thus equivalent to enforcing a typed balanced bracketing in the string representation.

Let us consider the untyped bracketing case, disregarding the connector type matching. It is a well-known fact that a balanced bracketing with unrestricted depth is not a regular language. A balanced bracketing with a finite fixed maximum depth $t \in \mathbb{N}$, however, is a regular language and can be defined by the simple $t+1$-state FSA $B_t$ illustrated in Figure 3. Following Yli-Jyrä [9], we limit the maximum depth of the bracketing, thus approximating LG. The maximum bracketing depth $t$ is a parameter of the approximation.

In order to enforce the planarity axiom, it is necessary to enforce the LG connector type matching in addition to the untyped balanced bracketing. Let $\Delta$ be the alphabet of all connector types used in right connectors in a particular LG grammar. Similarly, $\nabla$ is the alphabet of left connector types[3]. Let further $M(c) \subseteq \nabla$ be the set of all connector types in $\nabla$ matching a connector type $c \in \Delta$. For each bracketing depth $d$, we define an FSA $P_{d,t}$ (Figure 4) which accepts a string only if for each link opened with a connector type $c \in \Delta$ at the depth $d$, the link is closed with a connector type $c' \in M(c)$. This approach implements the connector type matching algorithm by simple enumeration, since in any given grammar, the number of unique connector types is finite. The intersection FSA

[2] More correctly, the term *semi-planar* is often used.
[3] Roughly, $\Delta$ corresponds to the alphabets $B_L$ and $B_l$ in [9] and $\nabla$ corresponds to $B_R$ and $B_r$.

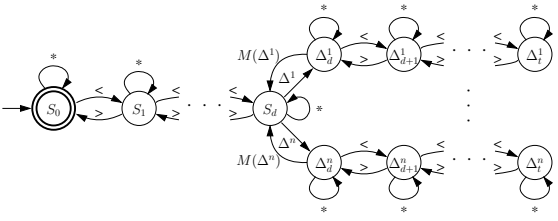



**Fig. 4.** Planarity FSA $P_{d,t}$. The states $S_0$–$S_d$ maintain a balanced bracketing up to the depth $d$. From the state $S_d$, there is an outgoing edge and a new state $\Delta^i_d$ for every connector type $\Delta^1 \dots \Delta^n \in \Delta$. For every such state $\Delta^i_d$, there is a sequence of states $\Delta^i_d \dots \Delta^i_t$ that maintain a balanced bracketing up to the total depth $t$. Further, there is an edge from $\Delta^i_d$ to $S_d$ for each connector type in $M(\Delta^i)$. Consequently, the states $\Delta^1_d \dots \Delta^n_d$ serve as a memory of which right connector type was used to open the link at depth $d$. A transition back to the state $S_d$ is only possible through a matching left connector type.

$$P_t = \bigcap_{d=1}^{t} P_{d,t}$$

then accepts a language where all connector types associated with corresponding open/close bracket pairs match. Any string from $L \cap P_t$ thus encodes a planar LG linkage graph.

### 3.4 Exclusion

The exclusion FSA $E_{d,t}$ accepts only strings such that if the bracketing depths $d$ and $d+1$ are opened by the same word, then they are closed by two different words, thus enforcing the exclusion axiom at the two adjacent bracketing depths. It is easy to see that the intersection FSA

$$E_t = \bigcap_{d=1}^{t-1} E_{d,t}$$

then accepts only strings where no two words are connected by more than one link. The FSA $E_{d,t}$ is detailed in Figure 5.

### 3.5 Connectivity

Words in a linkage that do not connect to any other words to the left (resp. right) are called *left-bare words* (resp. *right-bare words*). Further, an *island* in a linkage is a connected component of the linkage graph. The connectivity axiom requires that a linkage only has one connected component. Trivially, any island starts with a left-bare word and ends with a right-bare word. By the projectivity axiom, any island consists of a balanced bracketing. The depth $d$ at the boundary symbol preceding the first word of an island is therefore equal to the depth at the

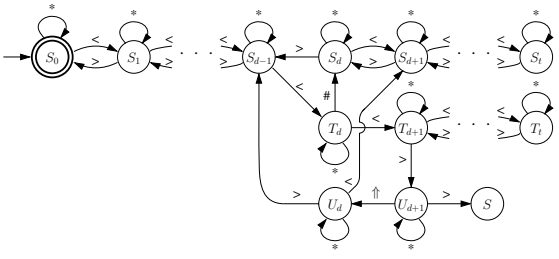



**Fig. 5.** Exclusion FSA $E_{d,t}$. The states $S_0$–$S_t$ maintain balanced bracketing up to the depth $t$. If a link is opened at the depth $d-1$, thus opening the depth $d$, the FSA reaches the state $T_d$. If another link is opened by the same word, the state $T_{d+1}$ is reached, otherwise the FSA continues in $S_d$. When the depth $d+1$ is closed the FSA reaches the state $U_{d+1}$. If the depth $d$ is closed by the same word, the sink state $S$ is reached and the string is rejected. Otherwise the states $S_{d-1}$ or $S_{d+1}$ are reached through $U_d$.

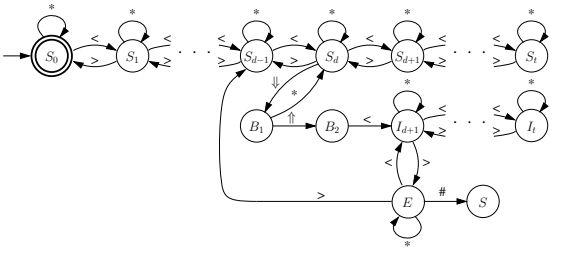



**Fig. 6.** Connectivity FSA $C_{d,t}$. The states $S_0$–$S_d$ maintain balanced bracketing up to the depth $d$. If the current word is a left-bare word at depth $d$, the FSA reaches state $B_2$, otherwise it returns to state $S_d$. States $I_{d+1}$–$I_t$ maintain a balanced bracketing from the left-bare word onward and the state $E$ is reached upon closing the depth $d$. If, at this point, also the depth $d-1$ is closed, it means that the left-bare word did not open an island since the current depth is smaller than $d$. The FSA then returns to state $S_{d-1}$. If, on the other hand, the FSA proceeds from the state $E$ to $S$, it has reached a right-bare word and the depth within the island was never been smaller than $d$. Therefore, an island was found and since $S$ is a sink state, the string is rejected.

boundary symbol following the last word of the island. Moreover, at any point within the island, the depth is $\geq d$. These properties are used in the construction of an FSA $C_{d,t}$ that only accepts strings that do not have islands opened at the depth $d$. The FSA $C_{0,t}$ must, however, accept the single island comprising the whole linkage. The intersection FSA

$$C_t = \bigcap_{d=0}^{t-1} C_{d,t}$$

then accepts strings encoding connected linkages. The FSA $C_{d,t}$ is detailed in Figure 6. Note that the FSA $C_{0,t}$ cannot be constructed as in the figure. Its construction is, however, trivial.

### 3.6 Approximation Language

The approximation language $\mathcal{L}_t$ is defined as an intersection of the lexicon language $L$ with the languages $P_t$, $E_t$, and $C_t$ which implement the LG axioms of planarity, exclusion, and connectivity. The LG axiom of ordering is implicit in $L$.

$$\mathcal{L}_t = L \cap P_t \cap E_t \cap C_t$$

The definition of $\mathcal{L}_t$ concludes the construction of the regular LG approximation.

## 4 Parsing with the Finite-State LG Approximation

We now introduce two ways to implement a parser based on the finite-state approximation of LG.

### 4.1 LG as a Monolithic Transducer

Let us consider $\mathcal{L}_t$ as an identity finite-state transducer (FST) with edge input:output symbol pairs $z : z$. Let us define a FST $\mathcal{T}_t$ based on $\mathcal{L}_t$ such that every edge symbol pair $z : z$ in $\mathcal{L}_t$ where $z \in \Delta \cup \nabla \cup \{\Uparrow, \Downarrow, <, >\}$ is replaced with $\epsilon : z$, where $\epsilon$ is the empty transition symbol. The FST $\mathcal{T}_t$ thus generates the parse encoding on the output string via $\epsilon$-transitions on the input string.

### 4.2 LG as a Finite-State Intersection Grammar

We now cast LG finite-state parsing as an intersection problem within the framework of finite-state intersection grammars [8].

For a sentence $s = w_1 w_2 \ldots w_n$ we define the language $R_s$ as the regular expression

$$R_s = R_{w_1} R_{w_2} \ldots R_{w_n} \#$$

that is, the concatenation of the regular languages representing the grammar entries for the individual words as defined in Section 3.2. The sentence $s$ is then parsed by computing the intersection

$$R_s \cap P_t \cap E_t \cap C_t$$

The resulting language encodes all parses of $s$ with bracketing depth $\leq t$.

Note that for a sentence $s$ with $n$ words, the planarity and exclusion axioms imply that the maximum possible bracketing depth is $n - 1$. A *perfect approximation*, where the set of parses in the regular language $R_s$ is exactly the set of all parses possible for the sentence in the context-free language, can therefore theoretically be achieved by setting $t = n - 1$.

## 5 Practical Considerations

We have created proof-of-concept implementations of the two finite-state LG parsers. We have implemented the automata with the FSA utilities [16] and, for efficiency reasons, executed the automata and operations such as intersections and minimizations in the AT&T FSM utilities [17].

The efficiency of the intersection algorithm critically depends on the size of the intersected FSAs. When sequentially intersecting several FSAs, the size of the intermediate automata has a strong influence on the efficiency of the computation. For illustration, we list the sizes (as the number of states/transitions) of several determinized and minimized automata constructed based on a broad-coverage English LG grammar[4] with 47K words: $P_{1,6}$ (1.1K/465K), $C_6$ (159/40K), $E_6$ (157/52K), $C_6 \cap E_6$ (1.2K/337K), $L$ (45K/110K).

Given the size of the automata, it is not surprising that building the monolithic FST has proven impractical. Even for a very simple grammar[5] and $t = 3$ the FSA $\mathcal{L}_t$ has over 20M states. The explicit computation of the monolithic parser for a broad-coverage grammar is thus infeasible.

Parsing within the framework of finite-state intersection grammars is more practical. For illustration, when considering a complex sentence with 41 words and depth up to 6, $R_s$ has 2.2K states and 4K transitions. The parsing of this complex sentence, however, takes on the order of minutes, while the LG parser takes on the order of seconds. Hence, unlike the monolithic approach, LG parsing as a finite-state intersection grammar is feasible, but inefficient due to the computation of intermediate results, where, although the result FSA has 130K states, the largest intermediate result has 1M states.

The problem of the large size of intermediate results is inherent to the finite-state intersection grammars. Several approaches to alleviate the problem are discussed, for example, by Tapanainen in [1]. For instance, the size of intermediate results strongly depends on the order in which the FSAs are intersected and optimizing the order can result in improved efficiency. We first compute $R_s \cap E_t \cap C_t$ and then intersect sequentially with $P_{d,t}$ for the individual depths $d$. We observed that intersecting the FSAs $P_{d,t}$ in the inverse order, starting with $P_{t,t}$, leads to several times faster intersection (largest intermediate result has 1M states) than intersecting $P_{1,t}$ first and $P_{t,t}$ last (largest intermediate result has 2.5M states). Other approaches discussed by Tapanainen include using a parallel intersection algorithm, or alternatively avoiding the explicit computation of the intersection by using a depth-first search through the FSA $R_s$, backtracking any time an axiom FSA rejects the string.

The efficiency of the original LG parser depends critically on *pruning*, where, prior to execution of the main parsing algorithm, the set of disjuncts assigned to each word is pruned so that, for example, if a disjunct contains a right connector and no disjunct of any following word contains a matching left connector, the disjunct cannot be satisfied and can thus be discarded [12]. The decrease in

[4] `4.0.dict` in the LG distribution
[5] `tiny.dict` in the LG distribution

the number of disjuncts and hence the decrease in the size of the search space is very substantial, often several orders of magnitude. As the number of disjuncts directly relates to the number of paths through the machine, pruning, once implemented, should result in a substantial decrease in parsing time also for the finite-state approximation of LG. Additionally, Yli-Jyrä [9] proposed several techniques which, through extension of the internal alphabets, achieve local testability of some of the linking axioms — with a corresponding positive effect on the size of intermediate results. Adapting Yli-Jyrä's techniques to the current implementation is thus another potential direction of research.

The explosion in the number of states can also potentially be avoided by the use of *extended* finite-state approaches, where the finite-state formalism is augmented in order to allow for more compact machines. Commonly, for every extended FSM there exists an equivalent, but considerably larger, pure FSM. However, some extended FSM techniques result in non-regular languages. An example of a practical application of an extended finite-state approach to parsing is that of Oflazer [6]. Additionally, *lazy evaluation*, supported for example by the AT&T FSM library, avoids explicitly expanding the machines in terms of atomic states and transitions and could result in a further decrease in parsing time.

Optimizing the computation of the intersection through the techniques discussed above or, alternatively, avoiding its explicit computation with an extended finite-state approach can be expected to increase the practicability and efficiency of finite-state LG parsing.

## 6 Conclusions

In this study, we have introduced a finite-state approximation of Link Grammar. The regular language approximating a given LG grammar was constructed by intersecting finite-state machines implementing the LG grammar and the LG axioms that constrain the set of grammatical parses. The approximation language is a subset of the corresponding context-free LG language with a limited maximum nesting depth of links.

Further, as a preliminary study of the practical applicability of the presented approximation, we have implemented finite-state LG parsers in terms of a monolithic transducer and in the framework of finite-state intersection grammars.

We have shown that a finite-state approximation of LG can be constructed and that finite-state parsing based on the approximation is feasible. Whether efficiency comparable to that of the original LG parser can be achieved using intersection optimization techniques or extended finite-state approaches remains a question for future research.

## Acknowledgments

This work has been supported by Tekes, the Finnish Funding Agency for Technology and Innovation. We would like to thank the anonymous reviewers for their detailed and enlightening comments.

## References


1. Roche, E., Schabes, Y., eds.: Finite-State Language Processing. MIT Press (1997)
2. Carroll, J.: Parsing. In Mitkov, R., ed.: The Oxford Handbook of Computational Linguistics. Oxford University Press (2003) 233–248
3. Nederhof, M.J.: Practical experiments with regular approximation of context-free languages. Computational Linguistics **26**(1) (2000) 17–44
4. Grimley Evans, E.: Approximating context-free grammars with a finite-state calculus. In: Proceedings of ACL/EACL '97, Association for Computational Linguistics (1997) 452–459
5. Johnson, M.: Finite-state approximation of constraint-based grammars using left-corner grammar transforms. In: Proceedings of ACL/COLING '98, Association for Computational Linguistics (1998) 619–623
6. Oflazer, K.: Dependency parsing with an extended finite-state approach. Computational Linguistics **29**(4) (2003) 515–544
7. Elworthy, D.: A finite state parser with dependency structure output. In: Proceedings of the Sixth International Workshop on Parsing Technologies IWPT 2000, Trento, Italy. (2000)
8. Koskenniemi, K.: Finite-state parsing and disambiguation. In Karlgren, H., ed.: Proceedings of the 13th International Conference on Computational Linguistics COLING 90, Helsinki, Finland, ACL (1990) 229–232
9. Yli-Jyrä, A.M.: Approximating dependency grammars through intersection of star-free regular languages. International Journal of Foundations of Computer Science **16**(3) (2005) 565–579
10. Hays, D.G.: Dependency theory: A formalism and some observations. Language **40** (1964) 511–525
11. Gaifman, H.: Dependency systems and phrase-structure systems. Information and Control **8** (1965) 304–337
12. Sleator, D.D., Temperley, D.: Parsing English with a Link Grammar. In: Proceedings of the Third International Workshop on Parsing Technologies IWPT 93, Tilburg, Netherlands. (1993)
13. Miller, S., Fox, H., Ramshaw, L., Weischedel, R.: A novel use of statistical parsing to extract information from text. In: Proceedings of NAACL '00, Morgan Kaufmann (2000) 226–233
14. Ginter, F., Mylläri, A., Salakoski, T.: A probabilistic search for the best solution among partially completed candidates. In: Proceedings of the HLT/NAACL'06 Workshop on Computationally Hard Problems and Joint Inference in Speech and Language Processing, Association for Computational Linguistics (2006) 33–40
15. Grinberg, D., Lafferty, J., Sleator, D.D.: A robust parsing algorithm for link grammars. In: Proceedings of the Fourth International Workshop on Parsing Technologies IWPT 95, Prague, Czech Republic. (1995)
16. van Noord, G.: FSA utilities: A toolbox to manipulate finite-state automata. In Wood, D., Darrell, R., Yu, S., eds.: Automata Implementation. Volume 1260 of Lecture Notes in Computer Science (LNCS)., Springer, Heidelberg (1997) 87–108
17. Mohri, M., Pereira, F.C.N., Riley, M.: A rational design for a weighted finite-state transducer library. In Wood, D., Yu, S., eds.: Proceedings of the Second International Workshop on Implementing Automata WIA 97, London, Canada. Volume 1436 of Lecture Notes in Computer Science (LNCS)., Springer, Heidelberg (1998) 144–158