Parsing and Generation as Datalog Queries

Makoto Kanazawa
National Institute of Informatics
2–1–2 Hitotsubashi, Chiyoda-ku, Tokyo, 101–8430, Japan
kanazawa@nii.ac.jp

Abstract

We show that the problems of parsing and surface realization for grammar formalisms with "context-free" derivations, coupled with Montague semantics (under a certain restriction) can be reduced in a uniform way to Datalog query evaluation. As well as giving a polynomial-time algorithm for computing all derivation trees (in the form of a shared forest) from an input string or input logical form, this reduction has the following complexity-theoretic consequences for all such formalisms: (i) the decision problem of recognizing grammaticality (surface realizability) of an input string (logical form) is in LOGCFL; and (ii) the search problem of finding one logical form (surface string) from an input string (logical form) is in functional LOGCFL. Moreover, the generalized supplementary magic-sets rewriting of the Datalog program resulting from the reduction yields efficient Earley-style algorithms for both parsing and generation.

1 Introduction

The representation of context-free grammars (augmented with features) in terms of definite clause programs is well-known. In the case of a bare-bone CFG, the corresponding program is in the function-free subset of logic programming, known as *Datalog*. For example, determining whether a string John found a unicorn belongs to the language of the CFG in Figure 1 is equivalent to deciding whether the Datalog program in Figure 2 together with the *database* in (1) can derive the *query* "?– S(0, 4)."

(1) John(0, 1). found(1, 2). a(2, 3). unicorn(3, 4).

S → NP VP	V → found
VP → V NP	V → caught
V → V Conj V	Conj → and
NP → Det N	Det → a
NP → John	N → unicorn

Figure 1: A CFG.

$\mathsf{S}(i, j)$:– $\mathsf{NP}(i, k), \mathsf{VP}(k, j)$.	$\mathsf{V}(i, j)$:– $\mathsf{found}(i, j)$.
$\mathsf{VP}(i, j)$:– $\mathsf{V}(i, k), \mathsf{NP}(k, j)$.	$\mathsf{V}(i, j)$:– $\mathsf{caught}(i, j)$.
$\mathsf{V}(i, j)$:– $\mathsf{V}(i, k), \mathsf{Conj}(k, l), \mathsf{V}(l, j)$.	$\mathsf{Conj}(i, j)$:– $\mathsf{and}(i, j)$.
$\mathsf{NP}(i, j)$:– $\mathsf{Det}(i, k), \mathsf{N}(k, j)$.	$\mathsf{Det}(i, j)$:– $\mathsf{a}(i, j)$.
$\mathsf{NP}(i, j)$:– $\mathsf{John}(i, j)$.	$\mathsf{N}(i, j)$:– $\mathsf{unicorn}(i, j)$.

Figure 2: The Datalog representation of a CFG.

By *naive* (or *seminaive*) bottom-up evaluation (see, e.g., Ullman, 1988), the answer to such a query can be computed in polynomial time in the size of the database for any Datalog program. By recording rule instances rather than derived facts, a packed representation of the complete set of Datalog derivation trees for a given query can also be obtained in polynomial time by the same technique. Since a Datalog derivation tree uniquely determines a grammar derivation tree, this gives a reduction of context-free recognition and parsing to query evaluation in Datalog.

In this paper, we show that a similar reduction to Datalog is possible for more powerful grammar formalisms with "context-free" derivations, such as *(multi-component) tree-adjoining grammars* (Joshi and Schabes, 1997; Weir, 1988), *IO macro grammars* (Fisher, 1968), and *(parallel) multiple context-free grammars* (Seki et al., 1991). For instance, the TAG in Figure 3 is represented by the Datalog program in Figure 4. Moreover, the method of reduc-

Proceedings of the 45th Annual Meeting of the Association of Computational Linguistics, pages 176–183,
Prague, Czech Republic, June 2007.

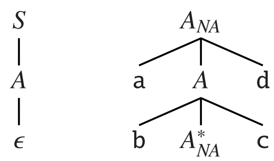

Figure 3: A TAG with one initial tree (left) and one auxiliary tree (right)

$S(p_1, p_3) :\!- A(p_1, p_3, p_2, p_2).$
$A(p_1, p_8, p_4, p_5) :\!- A(p_2, p_7, p_3, p_6), \mathsf{a}(p_1, p_2), \mathsf{b}(p_3, p_4), \mathsf{c}(p_5, p_6), \mathsf{d}(p_7, p_8).$
$A(p_1, p_2, p_1, p_2).$

Figure 4: The Datalog representation of a TAG.

tion extends to the problem of tactical generation (surface realization) for these grammar formalisms coupled with Montague semantics (under a certain restriction). Our method essentially relies on the encoding of different formalisms in terms of *abstract categorial grammars* (de Groote, 2001).

The reduction to Datalog makes it possible to apply to parsing and generation sophisticated evaluation techniques for Datalog queries; in particular, an application of *generalized supplementary magic-sets* rewriting (Beeri and Ramakrishnan, 1991) automatically yields Earley-style algorithms for both parsing and generation. The reduction can also be used to obtain a tight upper bound, namely *LOGCFL*, on the computational complexity of the problem of recognition, both for grammaticality of input strings and for surface realizability of input logical forms.

With regard to parsing and recognition of input strings, polynomial-time algorithms and the LOGCFL upper bound on the computational complexity are already known for the grammar formalisms covered by our results (Engelfriet, 1986); nevertheless, we believe that our reduction to Datalog offers valuable insights. Concerning generation, our results seem to be entirely new.[1]

2 Context-free grammars on λ-terms

Consider an augmentation of the grammar in Figure 1 with Montague semantics, where the left-hand side of each rule is annotated with a λ-term that tells how the meaning of the left-hand side is composed from the meanings of the right-hand side nonterminals, represented by upper-case variables $X_1, X_2, \ldots$ (Figure 5).[2]

$\mathsf{S}(X_1 X_2) \to \mathsf{NP}(X_1)\ \mathsf{VP}(X_2)$
$\mathsf{VP}(\lambda x.X_2(\lambda y.X_1 y x)) \to \mathsf{V}(X_1)\ \mathsf{NP}(X_2)$
$\mathsf{V}(\lambda y x.X_2(X_1 y x)(X_3 y x)) \to \mathsf{V}(X_1)\ \mathsf{Conj}(X_2)\ \mathsf{V}(X_3)$
$\mathsf{NP}(X_1 X_2) \to \mathsf{Det}(X_1)\ \mathsf{N}(X_2)$
$\mathsf{NP}(\lambda u.u\, \mathbf{John}^{e}) \to \mathsf{John}$
$\mathsf{V}(\mathbf{find}^{e \to e \to t}) \to \mathsf{found}$
$\mathsf{V}(\mathbf{catch}^{e \to e \to t}) \to \mathsf{caught}$
$\mathsf{Conj}(\wedge^{t \to t \to t}) \to \mathsf{and}$
$\mathsf{Det}(\lambda uv.\exists^{(e \to t) \to t}(\lambda y.\wedge^{t \to t \to t}(uy)(vy))) \to \mathsf{a}$
$\mathsf{N}(\mathbf{unicorn}^{e \to t}) \to \mathsf{unicorn}$

Figure 5: A context-free grammar with Montague semantics.

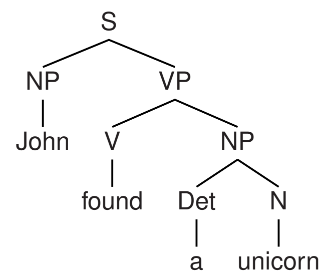

Figure 6: A derivation tree.

The meaning of a sentence is computed from its derivation tree. For example, John found a unicorn has the derivation tree in Figure 6, and the grammar rules assign its root node the λ-term

$$(\lambda u.u\ \mathbf{John})(\lambda x.(\lambda uv.\exists(\lambda y.\wedge(uy)(vy)))\ \mathbf{unicorn}\ (\lambda y.\mathbf{find}\ y\ x)),$$

which β-reduces to the λ-term

(2) $\exists(\lambda y.\wedge(\mathbf{unicorn}\ y)(\mathbf{find}\ y\ \mathbf{John}))$

encoding the first-order logic formula representing the meaning of the sentence (i.e., its logical form).

Thus, computing the logical form(s) of a sentence involves parsing and λ-term normalization. To find a sentence expressing a given logical form, it suffices

[1]We only consider *exact generation*, not taking into account the problem of logical form equivalence, which will most likely render the problem of generation computationally intractable (Moore, 2002).

[2]We follow standard notational conventions in typed λ-calculus. Thus, an application $M_1 M_2 M_3$ (written without parentheses) associates to the left, $\lambda x.\lambda y.M$ is abbreviated to $\lambda xy.M$, and $\alpha \to \beta \to \gamma$ stands for $\alpha \to (\beta \to \gamma)$. We refer the reader to Hindley, 1997 or Sørensen and Urzyczyn, 2006 for standard notions used in simply typed λ-calculus.

$$\mathsf{S}(X_1X_2) :- \mathsf{NP}(X_1), \mathsf{VP}(X_2).$$
$$\mathsf{VP}(\lambda x.X_2(\lambda y.X_1yx)) :- \mathsf{V}(X_1), \mathsf{NP}(X_2).$$
$$\mathsf{V}(\lambda yx.X_2(X_1yx)(X_3yx)) :- \mathsf{V}(X_1), \mathsf{Conj}(X_2), \mathsf{V}(X_3).$$
$$\mathsf{NP}(X_1X_2) :- \mathsf{Det}(X_1), \mathsf{N}(X_2).$$
$$\mathsf{NP}(\lambda u.u\ \mathbf{John}^e).$$
$$\mathsf{V}(\mathbf{find}^{e\to e\to t}).$$
$$\mathsf{V}(\mathbf{catch}^{e\to e\to t}).$$
$$\mathsf{Conj}(\wedge^{t\to t\to t}).$$
$$\mathsf{Det}(\lambda uv.\exists^{(e\to t)\to t}(\lambda y.\wedge^{t\to t\to t}(uy)(vy))).$$
$$\mathsf{N}(\mathbf{unicorn}^{e\to t}).$$

Figure 7: A CFLG.

to find a derivation tree whose root node is associated with a λ-term that β-reduces to the given logical form; the desired sentence can simply be read off from the derivation tree. At the heart of both tasks is the computation of the derivation tree(s) that yield the input. In the case of generation, this may be viewed as parsing the input λ-term with a "context-free" grammar that generates a set of λ-terms (in normal form) (Figure 7), which is obtained from the original CFG with Montague semantics by stripping off terminal symbols. Determining whether a given logical form is surface realizable with the original grammar is equivalent to recognition with the resulting *context-free λ-term grammar* (CFLG).

In a CFLG such as in Figure 7, constants appearing in the λ-terms have preassigned types indicated by superscripts. There is a mapping σ from nonterminals to their types ($\sigma = \{\mathsf{S} \mapsto t, \mathsf{NP} \mapsto (e \to t) \to t, \mathsf{VP} \mapsto e{\to}t, \mathsf{V} \mapsto e{\to}e{\to}t, \mathsf{Conj} \mapsto t{\to}t{\to}t, \mathsf{Det} \mapsto (e{\to}t){\to}(e{\to}t){\to}t, \mathsf{N} \mapsto e{\to}t\}$). A rule that has A on the left-hand side and $B_1, \dots, B_n$ as right-hand side nonterminals has its left-hand side annotated with a well-formed λ-term M that has type $\sigma(A)$ under the type environment $X_1 : \sigma(B_1), \dots, X_n : \sigma(B_n)$ (in symbols, $X_1 : \sigma(B_1), \dots, X_n : \sigma(B_n) \vdash M : \sigma(A)$).

What we have called a context-free λ-term grammar is nothing but an alternative notation for an *abstract categorial grammar* (de Groote, 2001) whose abstract vocabulary is second-order, with the restriction to *linear* λ-terms removed.[3] In the linear case, Salvati (2005) has shown the recognition/parsing complexity to be PTIME, and exhibited an algorithm similar to Earley parsing for TAGs. Second-order linear ACGs are known to be expressive enough to encode well-known mildly context-sensitive grammar formalisms in a straightforward way, including TAGs and multiple context-free grammars (de Groote, 2002; de Groote and Pogodalla, 2004).

[3] A λ-term is a *λI-term* if each occurrence of λ binds at least one occurrence of a variable. A λI-term is *linear* if no subterm contains more than one free occurrence of the same variable.

$$S(\lambda y.X_1(\lambda z.z)y) :- A(X_1).$$
$$A(\lambda xy.\mathsf{a}^{o\to o}(X_1(\lambda z.\mathsf{b}^{o\to o}(x(\mathsf{c}^{o\to o}z)))(\mathsf{d}^{o\to o}y))) :- A(X_1).$$
$$A(\lambda xy.xy).$$

Figure 8: The CFLG encoding a TAG.

For example, the linear CFLG in Figure 8 is an encoding of the TAG in Figure 3, where $\sigma(S) = o{\to}o$ and $\sigma(A) = (o \to o) \to o \to o$ (see de Groote, 2002 for details of this encoding). In encoding a string-generating grammar, a CFLG uses o as the type of string position and $o \to o$ as the type of string. Each terminal symbol is represented by a constant of type $o{\to}o$, and a string $a_1 \dots a_n$ is encoded by the λ-term $\lambda z.a_1^{o\to o}(\dots(a_n^{o\to o}z)\dots)$, which has type $o \to o$.

A string-generating grammar coupled with Montague semantics may be represented by a *synchronous CFLG*, a pair of CFLGs with matching rule sets (de Groote 2001). The transduction between strings and logical forms in either direction consists of parsing the input λ-term with the source-side grammar and normalizing the λ-term(s) constructed in accordance with the target-side grammar from the derivation tree(s) output by parsing.

3 Reduction to Datalog

We show that under a weaker condition than linearity, a CFLG can be represented by a Datalog program, obtaining a tight upper bound (LOGCFL) on the recognition complexity. Due to space limitation, our presentation here is kept at an informal level; formal definitions and rigorous proof of correctness will appear elsewhere.

We use the grammar in Figure 7 as an example, which is represented by the Datalog program in Figure 9. Note that all λ-terms in this grammar are *almost linear* in the sense that they are λI-terms where any variable occurring free more than once in any subterm must have an atomic type. Our construction is guaranteed to be correct only when this condition is met.

Each Datalog rule is obtained from the corresponding grammar rule in the following way. Let

$$
\begin{array}{l}
\mathsf{S}(p_1) :- \mathsf{NP}(p_1,p_2,p_3), \mathsf{VP}(p_2,p_3).\\
\mathsf{VP}(p_1,p_4) :- \mathsf{V}(p_2,p_4,p_3), \mathsf{NP}(p_1,p_2,p_3).\\
\mathsf{V}(p_1,p_4,p_3) :-\\
\quad \mathsf{V}(p_2,p_4,p_3), \mathsf{Conj}(p_1,p_5,p_2), \mathsf{V}(p_5,p_4,p_3).\\
\mathsf{NP}(p_1,p_4,p_5) :- \mathsf{Det}(p_1,p_4,p_5,p_2,p_3), \mathsf{N}(p_2,p_3).\\
\mathsf{NP}(p_1,p_1,p_2) :- \mathbf{John}(p_2).\\
\mathsf{V}(p_1,p_3,p_2) :- \mathbf{find}(p_1,p_3,p_2).\\
\mathsf{V}(p_1,p_3,p_2) :- \mathbf{catch}(p_1,p_3,p_2).\\
\mathsf{Conj}(p_1,p_3,p_2) :- \wedge(p_1,p_3,p_2).\\
\mathsf{Det}(p_1,p_5,p_4,p_3,p_4) :- \exists(p_1,p_2,p_4), \wedge(p_2,p_5,p_3).\\
\mathsf{N}(p_1,p_2) :- \mathbf{unicorn}(p_1,p_2).
\end{array}
$$

Figure 9: The Datalog representation of a CFLG.

M be the λ-term annotating the left-hand side of the grammar rule. We first obtain a *principal* (i.e., most general) *typing* of M.[4] In the case of the second rule, this is

$$X_1 : p_3 \to p_4 \to p_2,\ X_2 : (p_3 \to p_2) \to p_1 \vdash \lambda x.X_2(\lambda y.X_1 yx) : p_4 \to p_1.$$

We then remove $\to$ and parentheses from the types in the principal typing and write the resulting sequences of atomic types in reverse.[5] We obtain the Datalog rule by replacing X_i and M in the grammar rule with the sequence coming from the type paired with X_i and M, respectively. Note that atomic types in the principal typing become variables in the Datalog rule. When there are constants in the λ-term M, they are treated like free variables. In the case of the second-to-last rule, the principal typing is

$$\exists : (p_4 \to p_2) \to p_1,\ \wedge : p_3 \to p_5 \to p_2 \vdash \lambda uv.\exists(\lambda y.\wedge(uy)(vy)) : (p_4 \to p_3) \to (p_4 \to p_5) \to p_1.$$

If the same constant occurs more than once, distinct occurrences are treated as distinct free variables.

The construction of the database representing the input λ-term is similar, but slightly more complex. A simple case is the λ-term (2), where each constant occurs just once. We compute its principal typing, treating constants as free variables.[6]

$$\exists : (4 \to 2) \to 1,\ \wedge : 3 \to 5 \to 2,\ \mathbf{unicorn} : 4 \to 3,\ \mathbf{find} : 4 \to 6 \to 5,\ \mathbf{John} : 6 \vdash \exists(\lambda y.\wedge(\mathbf{unicorn}\ y)(\mathbf{find}\ y\ \mathbf{John})) : 1.$$

[4]To be precise, we must first convert M to its *η-long form* relative to the type assigned to it by the grammar. For example, $X_1 X_2$ in the first rule is converted to $X_1(\lambda x.X_2 x)$.

[5]The reason for reversing the sequences of atomic types is to reconcile the λ-term encoding of strings with the convention of listing string positions from left to right in databases like (1).

[6]We assume that the input λ-term is in η-long normal form.

We then obtain the corresponding database (3) and query (4) from the antecedent and succedent of this judgment, respectively. Note that here we are using $1, 2, 3, \ldots$ as atomic types, which become database constants.

(3) $\exists(1,2,4).\ \wedge(2,5,3).\ \mathbf{unicorn}(3,4).\ \mathbf{find}(5,6,4).\ \mathbf{John}(6).$

(4) $?\!-\ \mathsf{S}(1).$

When the input λ-term contains more than one occurrence of the same constant, it is not always correct to simply treat them as distinct free variables, unlike in the case of λ-terms annotating grammar rules. Consider the λ-term (5) (John found and caught a unicorn):

(5) $\exists(\lambda y.\wedge(\mathbf{unicorn}\ y)(\wedge(\mathbf{find}\ y\ \mathbf{John})(\mathbf{catch}\ y\ \mathbf{John}))).$

Here, the two occurrences of **John** must be treated as the same variable. The principal typing is (6) and the resulting database is (7).

(6) $\exists : (4 \to 2) \to 1,\ \wedge_1 : 3 \to 5 \to 2,\ \mathbf{unicorn} : 4 \to 3,\ \wedge_2 : 6 \to 8 \to 5,\ \mathbf{find} : 4 \to 7 \to 6,\ \mathbf{John} : 7, \mathbf{catch} : 4 \to 7 \to 8 \vdash \exists(\lambda y.\wedge_1(\mathbf{unicorn}\ y)(\wedge_2(\mathbf{find}\ y\ \mathbf{John})(\mathbf{catch}\ y\ \mathbf{John}))) : 1.$

(7) $\exists(1,2,4).\ \wedge(2,5,3).\ \wedge(5,8,6).\ \mathbf{unicron}(3,4).\ \mathbf{find}(6,7,4).\ \mathbf{John}(7).\ \mathbf{catch}(8,7,4).$

It is not correct to identify the two occurrences of $\wedge$ in this example. The rule is to identify distinct occurrences of the same constant just in case they occur in the same position within α-equivalent subterms of an atomic type. This is a necessary condition for those occurrences to originate as one and the same occurrence in the non-normal λ-term at the root of the derivation tree. (As a preprocessing step, it is also necessary to check that distinct occurrences of a bound variable satisfy the same condition, so that the given λ-term is β-equal to some almost linear λ-term.[7])

[7]Note that the way we obtain a database from an input λ-term generalizes the standard database representation of a string: from the λ-term encoding $\lambda z.a_1^{o\to o}(\ldots(a_n^{o\to o}z)\ldots)$ of a string $a_1 \ldots a_n$, we obtain the database $\{a_1(0,1), \ldots, a_n(n-1,n)\}$.

4 Correctness of the reduction

We sketch some key points in the proof of correctness of our reduction. The λ-term N obtained from the input λ-term by replacing occurrences of constants by free variables in the manner described above is the normal form of some almost linear λ-term N'. The *leftmost reduction* from an almost linear λ-term to its normal form must be *non-deleting* and *almost non-duplicating* in the sense that when a β-redex $(\lambda x.P)Q$ is contracted, Q is not deleted, and moreover it is not duplicated unless the type of x is atomic. We can show that the *Subject Expansion Theorem* holds for such β-reduction, so the principal typing of N is also the principal typing of N'. By a slight generalization of a result by Aoto (1999), this typing $\Gamma \vdash N' : \alpha$ must be *negatively non-duplicated* in the sense that each atomic type has at most one negative occurrence in it. By Aoto and Ono's (1994) generalization of the *Coherence Theorem* (see Mints, 2000), it follows that every λ-term P such that $\Gamma' \vdash P : \alpha$ for some $\Gamma' \subseteq \Gamma$ must be $\beta\eta$-equal to N' (and consequently to N).

Given the one-one correspondence between the grammar rules and the Datalog rules, a Datalog derivation tree uniquely determines a grammar derivation tree (see Figure 10 as an example). This relation is not one-one, because a Datalog derivation tree contains database constants from the input database. This extra information determines a typing of the λ-term P at the root of the grammar derivation tree (with occurrences of constants in the λ-term corresponding to distinct facts in the database regarded as distinct free variables):

$\textbf{John} : 6$, $\textbf{find} : 4 \to 6 \to 5$, $\exists : (4 \to 2) \to 1$,
$\wedge : 3 \to 5 \to 2$, $\textbf{unicorn} : 4 \to 3 \vdash$
$(\lambda u.u\ \textbf{John})$
$(\lambda x.(\lambda uv.\exists(\lambda y.\wedge(uy)(vy)))\ \textbf{unicorn}\ (\lambda y.\textbf{find}\ y\ x)) : 1.$

The antecedent of this typing must be a subset of the antecedent of the principal typing of the λ-term N from which the input database was obtained. By the property mentioned at the end of the preceding paragraph, it follows that the grammar derivation tree is a derivation tree for the input λ-term.

Conversely, consider the λ-term P (with distinct occurrences of constants regarded as distinct free variables) at the root of a grammar derivation tree for the input λ-term. We can show that there is a substitution θ which maps the free variables of P to the free variables of the λ-term N used to build the input database such that θ sends the normal form of P to N. Since P is an almost linear λ-term, the leftmost reduction from $P\theta$ to N is non-deleting and almost non-duplicating. By the Subject Expansion Theorem, the principal typing of N is also the principal typing of $P\theta$, and this together with the grammar derivation tree determines a Datalog derivation tree.

5 Complexity-theoretic consequences

Let us call a rule $A(M) :- B_1(X_1), \dots, B_n(X_n)$ in a CFLG an *ϵ-rule* if $n = 0$ and M does not contain any constants. We can eliminate ϵ-rules from an almost linear CFLG by the same method that Kanazawa and Yoshinaka (2005) used for linear grammars, noting that for any Γ and α, there are only finitely many almost linear λ-terms M such that $\Gamma \vdash M : \alpha$. If a grammar has no ϵ-rule, any derivation tree for the input λ-term N that has a λ-term P at its root node corresponds to a Datalog derivation tree whose number of leaves is equal to the number of occurrences of constants in P, which cannot exceed the number of occurrences of constants in N.

A Datalog program $\mathbf{P}$ is said to have the *polynomial fringe property* relative to a class $\mathcal{D}$ of databases if there is a polynomial $p(n)$ such that for every database D in $\mathcal{D}$ of n facts and every query q such that $\mathbf{P} \cup D$ derives q, there is a derivation tree for q whose fringe (i.e., sequence of leaves) is of length at most $p(n)$. For such $\mathbf{P}$ and $\mathcal{D}$, it is known that $\{ (D, q) \mid D \in \mathcal{D}, \mathbf{P} \cup D \text{ derives } q \}$ is in the complexity class *LOGCFL* (Ullman and Van Gelder, 1988; Kanellakis, 1988).

We state without proof that the database-query pair (D, q) representing an input λ-term N can be computed in logspace. By padding D with extra useless facts so that the size of D becomes equal to the number of occurrences of constants in N, we obtain a logspace reduction from the set of λ-terms generated by an almost linear CFLG to a set of the form $\{ (D, q) \mid D \in \mathcal{D}, \mathbf{P} \cup D \vdash q \}$, where $\mathbf{P}$ has the polynomial fringe property relative to $\mathcal{D}$. This shows that the problem of recognition for an almost linear CFLG is in LOGCFL.

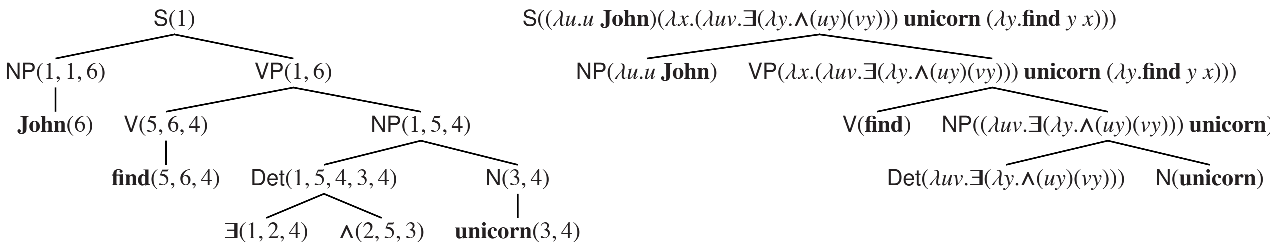

Figure 10: A Datalog derivation tree (left) and the corresponding grammar derivation tree (right)

By the main result of Gottlob et al. (2002), the related search problem of finding one derivation tree for the input λ-term is in *functional LOGCFL*, i.e., the class of functions that can be computed by a logspace-bounded Turing machine with a LOGCFL oracle. In the case of a synchronous almost linear CFLG, the derivation tree found from the source λ-term can be used to compute a target λ-term. Thus, to the extent that transduction back and forth between strings and logical forms can be expressed by a synchronous almost linear CFLG, the search problem of finding one logical form of an input sentence and that of finding one surface realization of an input logical form are both in functional LOGCFL.[8] As a consequence, there are efficient parallel algorithms for these problems.

6 Regular sets of trees as input

Almost linear CFLGs can represent a substantial fragment of a Montague semantics for English and such "linear" grammar formalisms as (multi-component) tree-adjoining grammars (both as string grammars and as tree grammars) and multiple context-free grammars. However, IO macro grammars and parallel multiple context-free grammars cannot be directly represented because representing string copying requires multiple occurrences of a variable of type $o \to o$. This problem can be solved by switching from strings to trees. We convert the input string into the regular set of binary trees whose yield equals the input string (using c as the sole symbol of rank 2), and turn the grammar into a tree grammar, replacing all instances of string concatenation in the grammar with the tree operation $t_1, t_2 \mapsto c(t_1, t_2)$. This way, a string grammar is turned into a tree grammar that generates a set of trees whose image under the yield function is the language of the string grammar. (In the case of an IO macro grammar, the result is an *IO context-free tree grammar* (Engelfriet, 1977).) String copying becomes tree copying, and the resulting grammar can be represented by an almost linear CFLG and hence by a Datalog program. The regular set of all binary trees that yield the input string is represented by a database that is constructed from a deterministic bottom-up finite tree automaton recognizing it. Determinism is important for ensuring correctness of this reduction. Since the database can be computed from the input string in logspace, the complexity-theoretic consequences of the last section carry over here.

7 Magic sets and Earley-style algorithms

Magic-sets rewriting of a Datalog program allows bottom-up evaluation to avoid deriving useless facts by mimicking top-down evaluation of the original program. The result of the *generalized supplementary magic-sets* rewriting of Beeri and Ramakrishnan (1991) applied to the Datalog program representing a CFG essentially coincides with the *deduction system* (Shieber et al., 1995) or *uninstantiated parsing system* (Sikkel, 1997) for Earley parsing. By applying the same rewriting method to Datalog programs representing almost linear CFLGs, we can obtain efficient parsing and generation algorithms for various grammar formalisms with context-free derivations.

We illustrate this approach with the program in Figure 4, following the presentation of Ullman

[8] If the target-side grammar is not linear, the normal form of the target λ-term cannot be explicitly computed because its size may be exponential in the size of the source λ-term. Nevertheless, a typing that serves to uniquely identify the target λ-term can be computed from the derivation tree in logspace. Also, if the target-side grammar is linear and string-generating, the target string can be explicitly computed from the derivation tree in logspace (Salvati, 2007).

(1989a; 1989b). We assume the query to take the form "?– $S(0, x)$.", so that the input database can be processed incrementally. The program is first made *safe* by eliminating the possibility of deriving non-ground atoms:

$$
\begin{aligned}
&S(p_1, p_3) \mathrel{:-} A(p_1, p_3, p_2, p_2).\\
&A(p_1, p_8, p_4, p_5) \mathrel{:-} A(p_2, p_7, p_3, p_6), \mathsf{a}(p_1, p_2), \mathsf{b}(p_3, p_4), \mathsf{c}(p_5, p_6), \mathsf{d}(p_7, p_8).\\
&A(p_1, p_8, p_4, p_5) \mathrel{:-} \mathsf{a}(p_1, p_2), \mathsf{b}(p_2, p_4), \mathsf{c}(p_5, p_6), \mathsf{d}(p_6, p_8).
\end{aligned}
$$

The *subgoal rectification* removes duplicate arguments from subgoals, creating new predicates as needed:

$$
\begin{aligned}
&S(p_1, p_3) \mathrel{:-} B(p_1, p_3, p_2).\\
&A(p_1, p_8, p_4, p_5) \mathrel{:-} A(p_2, p_7, p_3, p_6), \mathsf{a}(p_1, p_2), \mathsf{b}(p_3, p_4), \mathsf{c}(p_5, p_6), \mathsf{d}(p_7, p_8).\\
&A(p_1, p_8, p_4, p_5) \mathrel{:-} \mathsf{a}(p_1, p_2), \mathsf{b}(p_2, p_4), \mathsf{c}(p_5, p_6), \mathsf{d}(p_6, p_8).\\
&B(p_1, p_8, p_4) \mathrel{:-} A(p_2, p_7, p_3, p_6), \mathsf{a}(p_1, p_2), \mathsf{b}(p_3, p_4), \mathsf{c}(p_4, p_6), \mathsf{d}(p_7, p_8).\\
&B(p_1, p_8, p_4) \mathrel{:-} \mathsf{a}(p_1, p_2), \mathsf{b}(p_2, p_4), \mathsf{c}(p_4, p_6), \mathsf{d}(p_6, p_8).
\end{aligned}
$$

We then attach to predicates *adornments* indicating the free/bound status of arguments in top-down evaluation, reordering subgoals so that as many arguments as possible are marked as bound:

$$
\begin{aligned}
&S^{bf}(p_1, p_3) \mathrel{:-} B^{bff}(p_1, p_3, p_2).\\
&B^{bff}(p_1, p_8, p_4) \mathrel{:-} \mathsf{a}^{bf}(p_1, p_2), A^{bfff}(p_2, p_7, p_3, p_6), \mathsf{b}^{bf}(p_3, p_4), \mathsf{c}^{bb}(p_4, p_6),\\
&\qquad \mathsf{d}^{bf}(p_7, p_8).\\
&B^{bff}(p_1, p_8, p_4) \mathrel{:-} \mathsf{a}^{bf}(p_1, p_2), \mathsf{b}^{bf}(p_2, p_4), \mathsf{c}^{bf}(p_4, p_6), \mathsf{d}^{bf}(p_6, p_8).\\
&A^{bfff}(p_1, p_8, p_4, p_5) \mathrel{:-} \mathsf{a}^{bf}(p_1, p_2), A^{bfff}(p_2, p_7, p_3, p_6), \mathsf{b}^{bf}(p_3, p_4), \mathsf{c}^{bb}(p_5, p_6),\\
&\qquad \mathsf{d}^{bf}(p_7, p_8).\\
&A^{bfff}(p_1, p_8, p_4, p_5) \mathrel{:-} \mathsf{a}^{bf}(p_1, p_2), \mathsf{b}^{bf}(p_2, p_4), \mathsf{c}^{ff}(p_5, p_6), \mathsf{d}^{bf}(p_6, p_8).
\end{aligned}
$$

The generalized supplementary magic-sets rewriting finally gives the following rule set:

$$
\begin{aligned}
r_1 &: m\_B(p_1) \mathrel{:-} m\_S(p_1).\\
r_2 &: S(p_1, p_3) \mathrel{:-} m\_B(p_1), B(p_1, p_3, p_2).\\
r_3 &: sup_{2.1}(p_1, p_2) \mathrel{:-} m\_B(p_1), \mathsf{a}(p_1, p_2).\\
r_4 &: sup_{2.2}(p_1, p_7, p_3, p_6) \mathrel{:-} sup_{2.1}(p_1, p_2), A(p_2, p_7, p_3, p_6).\\
r_5 &: sup_{2.3}(p_1, p_7, p_6, p_4) \mathrel{:-} sup_{2.2}(p_1, p_7, p_3, p_6), \mathsf{b}(p_3, p_4).\\
r_6 &: sup_{2.4}(p_1, p_7, p_4) \mathrel{:-} sup_{2.3}(p_1, p_7, p_6, p_4), \mathsf{c}(p_4, p_6).\\
r_7 &: B(p_1, p_8, p_4) \mathrel{:-} sup_{2.4}(p_1, p_7, p_4), \mathsf{d}(p_7, p_8).\\
r_8 &: sup_{3.1}(p_1, p_2) \mathrel{:-} m\_B(p_1), \mathsf{a}(p_1, p_2).\\
r_9 &: sup_{3.2}(p_1, p_4) \mathrel{:-} sup_{3.1}(p_1, p_2), \mathsf{b}(p_2, p_4).\\
r_{10} &: sup_{3.3}(p_1, p_4, p_6) \mathrel{:-} sup_{3.2}(p_1, p_4), \mathsf{c}(p_4, p_6).\\
r_{11} &: B(p_1, p_8, p_4) \mathrel{:-} sup_{3.3}(p_1, p_4, p_6), \mathsf{d}(p_6, p_8).\\
r_{12} &: m\_A(p_2) \mathrel{:-} sup_{2.1}(p_1, p_2).\\
r_{13} &: m\_A(p_2) \mathrel{:-} sup_{4.1}(p_1, p_2).\\
r_{14} &: sup_{4.1}(p_1, p_2) \mathrel{:-} m\_A(p_1), \mathsf{a}(p_1, p_2).\\
r_{15} &: sup_{4.2}(p_1, p_7, p_3, p_6) \mathrel{:-} sup_{4.1}(p_1, p_2), A(p_2, p_7, p_3, p_6).\\
r_{16} &: sup_{4.3}(p_1, p_7, p_6, p_4) \mathrel{:-} sup_{4.2}(p_1, p_7, p_3, p_6), \mathsf{b}(p_3, p_4).\\
r_{17} &: sup_{4.4}(p_1, p_7, p_4, p_5) \mathrel{:-} sup_{4.3}(p_1, p_7, p_6, p_4), \mathsf{c}(p_5, p_6).\\
r_{18} &: A(p_1, p_8, p_4, p_5) \mathrel{:-} sup_{4.4}(p_1, p_7, p_4, p_5), \mathsf{d}(p_7, p_8).\\
r_{19} &: sup_{5.1}(p_1, p_2) \mathrel{:-} m\_A(p_1), \mathsf{a}(p_1, p_2).\\
r_{20} &: sup_{5.2}(p_1, p_4) \mathrel{:-} sup_{5.1}(p_1, p_2), \mathsf{b}(p_2, p_4).\\
r_{21} &: sup_{5.3}(p_1, p_4, p_5, p_6) \mathrel{:-} sup_{5.2}(p_1, p_4), \mathsf{c}(p_5, p_6).\\
r_{22} &: A(p_1, p_8, p_4, p_5) \mathrel{:-} sup_{5.3}(p_1, p_4, p_5, p_6), \mathsf{d}(p_6, p_8).
\end{aligned}
$$

The following version of chart parsing adds control structure to this deduction system:

1. (INIT) Initialize the chart to the empty set, the agenda to the singleton $\{m\_S(0)\}$, and n to 0.
2. Repeat the following steps:
   (a) Repeat the following steps until the agenda is exhausted:
      i. Remove a fact from the agenda, called the *trigger*.
      ii. Add the trigger to the chart.
      iii. Generate all facts that are immediate consequences of the trigger together with all facts in the chart, and add to the agenda those generated facts that are neither already in the chart nor in the agenda.
   (b) (SCAN) Remove the next fact from the input database and add it to the agenda, incrementing n. If there is no more fact in the input database, go to step 3.
3. If $S(0, n)$ is in the chart, accept; otherwise reject.

The following is the trace of the algorithm on input string `aabbccdd`:

1.	$m\_S(0)$	INIT	14.	$\mathsf{c}(4, 5)$	SCAN
2.	$m\_B(0)$	$r_1, 1$	15.	$sup_{5.3}(1, 3, 4, 5)$	$r_{21}, 12, 14$
3.	$\mathsf{a}(0, 1)$	SCAN	16.	$\mathsf{c}(6, 5)$	SCAN
4.	$sup_{2.1}(0, 1)$	$r_3, 2, 3$	17.	$sup_{5.3}(1, 3, 5, 6)$	$r_{21}, 12, 16$
5.	$sup_{3.1}(0, 1)$	$r_8, 2, 3$	18.	$\mathsf{d}(6, 7)$	SCAN
6.	$m\_A(1)$	$r_{12}, 4$	19.	$A(1, 7, 3, 5)$	$r_{22}, 17, 18$
7.	$\mathsf{a}(1, 2)$	SCAN	20.	$sup_{2.2}(0, 7, 3, 5)$	$r_4, 4, 19$
8.	$sup_{4.1}(1, 2)$	$r_{14}, 6, 7$	21.	$sup_{2.3}(0, 7, 5, 4)$	$r_5, 13, 20$
9.	$sup_{5.1}(1, 2)$	$r_{19}, 6, 7$	22.	$sup_{2.4}(0, 7, 4)$	$r_6, 14, 21$
10.	$m\_A(2)$	$r_{13}, 8$	23.	$\mathsf{d}(7, 8)$	SCAN
11.	$\mathsf{b}(2, 3)$	SCAN	24.	$B(0, 8, 4)$	$r_7, 22, 23$
12.	$sup_{5.2}(1, 3)$	$r_{20}, 9, 11$	25.	$S(0, 8)$	$r_2, 2, 24$
13.	$\mathsf{b}(3, 4)$	SCAN			

Note that unlike existing Earley-style parsing algorithms for TAGs, the present algorithm is an instantiation of a general schema that applies to parsing with more powerful grammar formalisms as well as to generation with Montague semantics.

8 Conclusion

Our reduction to Datalog brings sophisticated techniques for Datalog query evaluation to the problems of parsing and generation, and establishes a tight bound on the computational complexity of recognition for a wide range of grammars. In particular, it shows that the use of higher-order λ-terms for semantic representation need not be avoided for the purpose of achieving computational tractability.

References

Aoto, Takahito. 1999. Uniqueness of normal proofs in implicational intuitionistic logic. *Journal of Logic, Language and Information* **8**, 217–242.

Aoto, Takahito and Hiroakira Ono. 1994. Uniqueness of normal proofs in $\{\rightarrow, \wedge\}$-fragment of NJ. Research Report IS-RR-94-0024F. School of Information Science, Japan Advanced Institute of Science and Technology.

Beeri, Catriel and Raghu Ramakrishnan. 1991. On the power of magic. *Journal of Logic Programming* **10**, 255–299.

Engelfriet, J. and E. M. Schmidt. 1977. IO and OI, part I. *The Journal of Computer and System Sciences* **15**, 328–353.

Engelfriet, Joost. 1986. The complexity of languages generated by attribute grammars. *SIAM Journal on Computing* **15**, 70–86.

Fisher, Michael J. 1968. *Grammars with Macro-Like Productions*. Ph.D. dissertation. Harvard University.

Gottlob, Georg, Nicola Lenoe, Francesco Scarcello. 2002. Computing LOGCFL certificates. *Theoretical Computer Science* **270**, 761–777.

de Groote, Philippe. 2001. Towards abstract categorial grammars. In *Association for Computational Linguistics, 39th Annual Meeting and 10th Conference of the European Chapter, Proceedings of the Conference*, pages 148–155.

de Groote, Philippe. 2002. Tree-adjoining grammars as abstract categorial grammars. In *Proceedings of the Sixth International Workshop on Tree Adjoining Grammar and Related Frameworks (TAG+6)*, pages 145–150. Universitá di Venezia.

de Groote, Philippe and Sylvain Pogodalla. 2004. On the expressive power of abstract categorial grammars: Representing context-free formalisms. *Journal of Logic, Language and Information* **13**, 421–438.

Hindley, J. Roger. 1997. *Basic Simple Type Theory*. Cambridge: Cambridge University Press.

Aravind K. Joshi and Yves Schabes. 1997. Tree-adjoining grammars. In Grzegoz Rozenberg and Arto Salomaa, editors, *Handbook of Formal Languages, Vol. 3*, pages 69–123. Berlin: Springer.

Kanazawa, Makoto and Ryo Yoshinaka. 2005. Lexicalization of second-order ACGs. NII Technical Report. NII-2005-012E. National Institute of Informatics, Tokyo.

Kanellakis, Paris C. 1988. Logic programming and parallel complexity. In Jack Minker, editor, *Foundations of Deductive Databases and Logic Programming*, pages 547–585. Los Altos, CA: Morgan Kaufmann.

Mints, Grigori. 2000. *A Short Introduction to Intuitionistic Logic*. New York: Kluwer Academic/Plenum Publishers.

Moore, Robert C. 2002. A complete, efficient sentence-realization algorithm for unification grammar. In *Proceedings, International Natural Language Generation Conference*, Harriman, New York, pages 41–48.

Salvati, Sylvain. 2005. *Problèmes de filtrage et problèmes d'analyse pour les grammaires catégorielles abstraites*. Doctoral dissertation, l'Institut National Polytechnique de Lorraine.

Salvati, Sylvain. 2007. Encoding second order string ACG with deterministic tree walking transducers. In Shuly Wintner, editor, *Proceedings of FG 2006: The 11th conference on Formal Grammar*, pages 143–156. FG Online Proceedings. Stanford, CA: CSLI Publications.

Seki, Hiroyuki, Takashi Matsumura, Mamoru Fujii, and Tadao Kasami. 1991. On multiple context-free grammars. *Theoretical Computer Science* **88**, 191–229.

Shieber, Stuart M., Yves Schabes, and Fernando C. N. Pereira. 1995. Principles and implementations of deductive parsing. *Journal of Logic Programming* **24**, 3–36.

Sikkel, Klaas. 1997. *Parsing Schemata*. Berlin: Springer.

Sørensen, Morten Heine and Paweł Urzyczyn. 2006. *Lectures on the Curry-Howard Isomorphism*. Amsterdam: Elsevier.

Ullman, Jeffrey D. 1988. *Principles of Database and Knowledge-Base Systems. Volume I.* Rockville, MD.: Computer Science Press.

Ullman, Jeffrey D. 1989a. Bottom-up beats top-down for Datalog. In *Proceedings of the Eighth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*, Philadelphia, pages 140–149.

Ullman, Jeffrey D. 1989b. *Principles of Database and Knowledge-Base Systems. Volume II: The New Technologies*. Rockville, MD.: Computer Science Press.

Ullman, Jeffrey D. and Allen Van Gelder. 1988. Parallel complexity of logical query programs. *Algorithmica* **3**, 5–42.

David J. Weir. 1988. *Characterizing Mildly Context-Sensitive Grammar Formalisms*. Ph.D. dissertation. University of Pennsylvania.