右再帰規則の適用回数の制限 - LR 表への組み込み

14-01

Restriction on the Number of Applications of a Right-Recursive Rule

小田 裕子、田中 穂積、徳永 健伸

ODA yuko, TANAKA Hozumi, TOKUNAGA Takenobu

東京工業大学大学院情報理工学研究科計算工学専攻

Department of Computer Science, Tokyo Institute of Technology

There are many right recursive rules in CFG(context free grammar) used in parsing. Recursive rules are effective for improving readability and decreasing the number of CFG rules. In a case where consecutive applications of a specified recursive rule is not allowed, we want to use this restriction in a parsing process. In GLR(Generalized LR) parsing, precompilation of such a restriction into an LR table in advance is better than dynamically applying the restriction in a parsing process. We show one of the former methods. In generating an LR table from a CFG, an LR item corresponding to the specified right recursive rule isn't allowed in the closure-expansion. We also compare our method to another method, which rewrites the CFG to another CFG with the same constraints. A preliminary experiment tells that our method is better than the another method.

1 はじめに

GLR 法 [1] は経験的に最も高速な構文解析アルゴリズムとして知られるが、これは CFG の情報をすべて LR 表にプリコンパイルしておくためだと考えられる。さらに、CFG で表すと規則数の爆発を引き起こす接続制約などの局所的な制約も、CFG と同時に LR 表に組み込む手法が提案され、成果をあげている [2]。本稿では、一般に再帰規則の適用は無制限に許容されている訳ではないという制約 [3] を取り上げ、LR 表にこの制約を組み込む手法を提案し、その利点を実験結果とともに示す。

構文解析に用いる CFG には多くの再帰規則が含まれ、これらは可読性の向上や規則数の抑制に有効である。しかし、再帰規則によって CFG の生成力が必要以上に増大し、解析の曖昧性が増えることがありうる。

ところで、ある種の再帰規則は連続して複数回用いることはないにも関わらず、文法の書きやすさ・読みやすさの点から、用意される。例えば、以下のような簡単な英語用の CFG（の一部）を考える。

(0) AdjP → not so AdjP
(1) AdjP → AdvP AdjP
(2) AdjP → AdjP and AdjP
(3) AdjP → adj

ここで、(1)(2) の再帰規則は連続して用いても不自然ではない。しかし (0) を連続して複数回用いることはありえないし、文法製作者もそのようなことは考えずに文法を記述しているだろう。(0) の右辺の AdjP を展開する規則として暗黙に想定されているのは (1) か (3) の筈である。そこで本稿では「(0) の規則は連続して複数回適用することはない」という制約を与えることを考える。

この制約を与える一番簡単な方法は、解析結果として得られた構文木の中から、(0) を連続して複数回適用しているものを除くことだが、解析の最後で行う後処理のためのコストがかかる。

第二の方法として、解析途中で (0) の規則を連続して適用する時に、その候補を捨てることが考えられる。余計な候補と判った時点で更なる計算を必要としなくなるが、そのためのチェックを動的に行うぶん、コストが

連絡先：yuko@cs.titech.ac.jp

かかる。これは手続き的手法であると言える。

本稿では手続き的ではなく宣言的にこの種の制約、つまり「(0) の規則は連続して複数回適用することはない」という制約を扱う方法を提案する。これは、この制約を LR 表にプリコンパイルする方法であるため、GLR パーザを全く変更せずに済み、GLR 法の長所を活かすことができる。

一方、この種の制約を CFG と別に与えて LR 表にプリコンパイルするのではなく、制約を CFG の枠組の中で規則として記述しつくすことも考えられる。この方法よりも提案手法の方が優れていることは、4 節で述べる。

2 右再帰規則の適用回数を制限する方法

LR 表は GOTO グラフという状態遷移図から作られる。そして GOTO グラフの状態は、核となる LR アイテム集合をクロージャ展開して作られる。もし適用回数を制限したい再帰規則が右再帰規則（$\alpha \neq \varepsilon$ として $A \to \alpha A\beta$ の形）ならば、再帰規則の適用に該当するアイテムをクロージャ展開しないことで制約を LR 表に組み込める。具体的には以下のようにすれば良い。

1. 連続適用を制限したい右再帰規則を予め指定しておく。
2. 状態を作る（クロージャ展開する）ときに、もし制限したい規則 $A \to \alpha A\beta$ から作られたアイテム $[A \to \alpha \cdot A\beta]$ がその中にあるなら、$[A \to \cdot\, \alpha A\beta]$ というアイテムは生成しない。
   その結果、$A \to \alpha A\beta$ を連続適用することに対応する GOTO グラフ上のパスは生成されない。

3 本手法の限界

制約が過度にはたらく場合　本手法では、$A \to \alpha A\beta$ の連続適用を制限した結果、副作用を生じることがあるので注意がいる（ただし $\alpha \neq \varepsilon$）。

case 1　もし $A \to B\gamma$ なる規則があり、$A \in \mathrm{LC}(B)$ なら[1]、図 1 のように、アイテム $[A \to \alpha \cdot A\beta]$ から直接 $A \to \alpha A\beta$ を使って展開せずとも、間接的に $[A \to \cdot\, \alpha A\beta]$ というアイテムが生成される。本手法ではこのような間接的再帰による $[A \to \cdot\, \alpha A\beta]$ も一律に削除している。なぜなら LR(0) アイテムにより GOTO グラフを作る場合[2]には、直接的再帰か間接的再帰か区別できないからである。

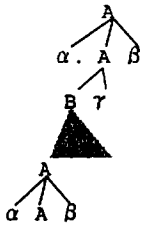

図 1: 間接的再帰

case 2　$A, X \in \mathrm{LC}(Y)$, $A \in \mathrm{LC}(Z)$ のとき、GOTO グラフの一部として図 2 が考えられる。ここで右側の状態で * のアイテムを生成するかどうかが問題となる。なぜなら、$[A \to \alpha \cdot A\beta]$ からの展開では生成してはならず、$[X \to \alpha \cdot Z\gamma]$ からの展開では生成するため、矛盾してしまうからである。現状では * は生成しないことにしている。

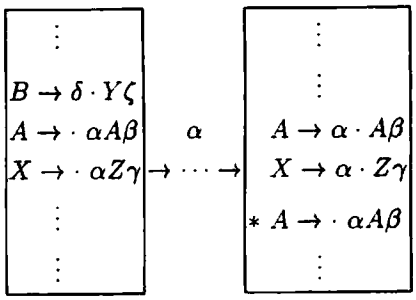

図 2: 再帰でない導出も制限する場合

先読み記号　上記の二つの問題は、同じ規則によって展開され生成された LR(0) アイテムは区別できないために起こる。しかし、CLR ではアイテムは先読み記号を持っているので、文法によっては、どのアイテムから展開して生成されたアイテムなのか、先読み記号によって区別することも可能である。すると、直接的な再帰と間接的な再帰を区別したり、図 2 の * に相当する LR(1) アイテムのうち、再帰規則の適用に該当するものは生成せず、そうでないものは生成することができる。

しかし、区別できるかどうかは文法に依存するため、制限したい再帰規則を指定する際にそこまで考えて指定しなくてはならないのはシステムとして使い勝手が悪い。そこで現段階では、意図したよりもきつい制約になってしまう場合には警告を出すだけにとどめ、先読み記号による区別は全く考えていない。

[1] LC は Left Corner の略で、$\mathrm{LC}(X) = \{Y \mid X$ から 0 回以上の導出により $Y\delta$ が得られる $\}$ と定義する。ただし X, Y は非終端記号、δ は 0 個以上の記号列。

[2] SLR 法の場合は勿論だが、LALR 法の場合も普通は LR(0) アイテムで GOTO グラフを作り、後から先読み記号を計算するので、これに該当する。

勿論これにも利点はある。例えば1節の例で (0) を制限したとする。ここで (0) の右辺の AdjP に (2), (0) の順で規則を適用すると、"AdjP" から "not so not so AdjP and AdjP" が導出されるが、これが好ましくないのは、直接的再帰による "not so not so AdjP" が好ましくないのと全く同様であり、制約が過度にはたらいたのが結果的に良かったことになる。

左再帰規則　左再帰規則 $A \rightarrow A\alpha$ の連続適用を制限したいとしても、本手法では無理である。なぜなら、$[A \rightarrow \cdot A\alpha]$ というアイテムからこの規則を使って展開してできるアイテムはこのアイテム自身なので、削除する訳にいかないからである。もちろん CLR の場合には先読み記号によって区別できる場合もあるが、常に区別できる訳ではないため、制約を組み込む方法としては良くない。

4　予備実験

4.1　結果

実験には ATR コーパスで正解木の付与された 9794 文を用い、細品詞を終端記号とする文法 [4]（終端記号数 442、非終端記号数 173、規則数 860）を使って、LR 表を作る際には終端記号どうしの接続表を組み込んだ。

まず、文法中の右再帰規則 64 個のうち、連続して適用することはないと考えられる 14 個を選び、それを (a) 制限しない場合と (b) 制限した場合で LR 表を作り、それぞれ訓練データで学習して確率つき LR 表を得る[3]。そして、テストデータの文を解析し、PGLR モデル [5] で上位 1 位の木を正解として両者を比較した。

また、参考データとして、CFG を書き換えて再帰規則の適用回数の制限を表す別の CFG にした場合の LR 表も作った。(c) 直接的な再帰のみ制限するよう書き換えた場合と、(d) 間接的再帰まで制限するよう書き換えた場合の 2 通りを試したが、これらは非終端記号が増加してコーパス中の正解木がそのままでは使えないので、解析実験はしていない。

各手法による文法と LR 表について表 1 に示す（NT は非終端記号数、LA は先読み記号の予測数の平均である）。また、(a)(b) の解析実験（10 分割クロスバリデーション）の結果を表 2 に示す。

手法	(a)	(b)	(c)	(d)
規則	860	860	874	3834
NT	173	173	187	3157
状態	828	836	842	3766
shift	8440	9104	7586	8863
reduce	64861	65553	66800	418953
goto	8556	8984	8236	90536
LA	60.0	60.6	61.0	107.0

表 1: 実験結果

手法	coverage	precision	accuracy
(a)	86.0%	82.4%	70.9%
(b)	85.8%	82.4%	70.7%

表 2: 解析実験

4.2　考察

LR 表の大きさ　右再帰規則を制限することで状態数が増える文法・減る文法・変わらない文法、いずれも作ることが可能である。増える原因としては、制限しなければループすることによって状態数が節約できたが、アイテムを削ったためにループできなくなり、別の状態が必要になることがある。また、減る原因としては、アイテムを削ったために shift または goto できる記号の種類が減り、その記号を読んで到達する状態を作る必要がなくなることがある。

しかし、実際には状態数は微増傾向のようだと実験で判った。(b) では 24 個の状態においてアイテムの生成が抑制されたが、結果としては状態数は約 1% 増加している。他の文法（規則数 392 で英語用）で 9 個の右再帰規則を制限した時も状態数は約 2% 増加している。また、アクションの数もそれに伴って増えている。

しかし、(c)(d) と比較すると、(b) の方が状態数が少なくてすむ。もっとも、(b) より (c) の方がアクション数では約 1% 少ないので、LR 表全体としては (b)(c) では大小関係はつけられないが、(d) と比べれば 状態数で約 4.5 倍・アクション数で約 6.2 倍の差があり、LR 表生成時間は 80 倍以上の差があった。これらの結果から、「CFG の変換（前処理）→ 構文解析 → 解析木の逆変換（後処理）」という形で制約を表現するよりも、(b) のように LR 表にプリコンパイルする方が好ましいと言える。

[3] スムージングは行わなかった。

解析結果 (b) の方が被覆率 (coverage) がやや落ちているが、ほとんど差はない。そもそも連続で適用することがなさそうな規則を、連続して用いたような文は、コーパス中にはほとんどないと考えられるから、(a)(b) の解析結果には差が出ないのだろう。(a) で正解を出せたのに (b) で正解にならなかった入力文は、以下のような理由によるものだった。

1. 正解木では、制限された再帰規則はまったく無関係であるもの。これは、(b) の方がわずかながらも LR 表が大きくなるため、学習に際してデータスパースネスの問題が出てきた結果ではないかと思われる。
2. 正解木は、制限した規則を連続で使用した文だったもの。具体的には、“節 → 接続詞 節” と “節 → 連用接続節 節” の 2 つの例が見られた。今回は直観的に制限する規則を決めたため、このような結果となった。

5 おわりに

本稿では、再帰的に記述されていても実際には連続して適用することがない再帰規則に着目し、その適用回数を 1 回に制限するという制約を LR 表に組み込む手法を示した。この手法は CFG を書き換えて別の CFG を作って制約を表すよりも優れていることが実験から判った。

今後の課題として、以下のものがある。

1. 今回は再帰規則の適用を 1 回に制限することを考えたが、前節で述べたように 1 回に制限すると解析できなくなることがあるので、「n 回以下しか連続で適用してはいけない」という形の制約も考えたい（例えば接頭辞はせいぜい 3 つしか続かないので、“NP → prefix NP” という規則について「3 回まで」と指定したい）。適用回数の制限を与えない場合には GOTO グラフでループができるが、ここでループさせるのでなく、図 3 に示すように、アイテム集合のコピーを $n-2$ 回作るようにすれば、実現可能だと考えられる。しかし、3 節で述べた制約が過度にきくという副作用が悪影響を及ぼすおそれもあるので、今後の検討が必要である。
2. GLR 法の応用として、異音列入力による音声認識が考えられている [6] が、音声認識では「えーと」のような冗長語が大きな問題となる。そこで、冗長語まで考慮した文法を書くのではなく、冗長語の直前または直後になりうる記号を宣言することで冗長語に対応する LR 表を作り、かつ、解析木には冗長語を残さないようにすることが考えられる。これについても検討する予定である。

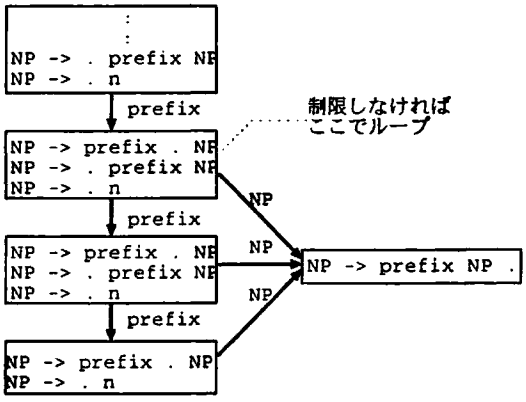

図 3: 再帰を 3 回までに制限

参考文献

[1] M. Tomita. *Efficient Parsing for Natural Language*. Kluwer Academic Publishers, 1986.

[2] H. Tanaka, T. Tokunaga, and M. Aizawa. Integration of morphological and synntactic analysis based on lr parsing. *Journal of Natural Language Processing*, Vol. 2, No. 2, 1995.

[3] 田中穂積. GLR 法に基づく統語解析過程の制御法 — LR 表工学の提案 —. 言語処理学会第 4 回年次大会発表論文集, 1998.

[4] H. Tanaka, T. Takezawa, and S. Etoh. MSLR 法を考慮した音声認識用日本語文法 — LR 表工学 (3). 音声言語情報研究会資料, 情報処理学会, pp. 145–150, 1997.

[5] Kentaro Inui, Virach Sornlertlamvanich, Hozumi Tanaka, and Takenobu Tokunaga. Probabilistic GLR parsing: A new formalization and its impact on parsing performance. 自然言語処理, Vol. 5, No. 3, pp. 33—52, 1998.

[6] H. Li. Integrating connection constraints into a GLR parser and its applications in a continuous speech recognition system. Technical report, Dept. of Computer Science, Tokyo Institute of Technology, 1996.