University of Copenhagen
Computer Science Department

Parallel Parsing using Futhark

Subtitle

Author: William Henrich Due
Advisor: Troels Henriksen
Submitted: June 12, 2023

Contents

1 Introduction

2 Theory

2.1 $LL(k)$ Parser Generator

For the construnction of a $LLP(q, k)$ parser generator the construction of first and follow-set [2, p. 5] and a $LL(k)$ parser generator is needed. A short explanation of the construction of a $LL(k)$ parser generator will be given since in the research of this project $k = 1$ was quite often explained but never $k > 1$ in a manner the author found understable.

The first and follow-set algorithms described comes from modifying Torben Mogensen description in his introductory compiler design book [1, p. 55-65]. The modification are using the truncated product instead of unions as described in the $LL(k)$ parser Wikipedia article[1] [3].

[1]At the time of writing the Wikipedia article does have a description of constructing first and follow-sets for $k > 1$. The problem is the algorithm described does not fullfill the definition of first and follow-sets that is being used in the LLP paper [2, p. 5].

Definition 2.1 (Truncated product)**.** Let $G = (N, T, P, S)$ be a context-free grammar and $A, B \subseteq P((N \cup T)^*)$ be sets of symbol strings. The truncated product is defined in the following way.

$$A \odot_k B \stackrel{\text{def}}{=} \underset{\gamma \in C}{\arg\max}\, |\gamma| \text{ where } C = \{\omega : \alpha \in A, \beta \in B, \omega\delta = \alpha\beta, |\omega| \leq k\}$$

Definition 2.2 (Nonempty substring pairs)**.** Let $G = (N, T, P, S)$ be a context-free grammar and $\omega \in (N \cup T)^*$ be a symbol string. The set of every nonempty way to split ω into two substrings is defined to be.

$$\varphi(\omega) \stackrel{\text{def}}{=} \{(\alpha, \beta) : \alpha \in (T \cup N)^+, \beta \in (T \cup N)^+, \alpha\beta = \omega\}$$

Algorithm 2.1 (Solving $FIRST_k$ set)**.** Let $G = (N, T, P, S)$ be a context-free grammar, the first-sets can be solved as followed.

$$\begin{aligned}
FIRST_k(\epsilon) &= \{\epsilon\} \\
FIRST_k(t) &= \{t\} \\
FIRST_k(A) &= \bigcup_{\delta\,:\,A \Rightarrow \delta \in P} FIRST_k(\delta) \\
FIRST_k(\omega) &= \bigcup_{(\alpha,\beta)\,\in\,\varphi(\omega)} FIRST_k(\alpha) \odot_k FIRST_k(\beta)
\end{aligned}$$

This may result in an infinite loop if implemented as is so fixed point iteration is used. Let $\mathcal{M} : N \to P(T^*)$ be a surjective function which is used as a dictionary which maps nonterminals to sets of terminal strings. $FIRST_k'$ is then the following modified version of $FIRST_k$.

$$\begin{aligned}
FIRST_k'(\epsilon, \mathcal{M}) &= \{\epsilon\} \\
FIRST_k'(t, \mathcal{M}) &= \{t\} \\
FIRST_k'(A, \mathcal{M}) &= \mathcal{M}(A) \\
FIRST_k'(\omega, \mathcal{M}) &= \bigcup_{(\alpha,\beta)\,\in\,\varphi(\omega)} FIRST_k'(\alpha, \mathcal{M}) \odot_k FIRST_k'(\beta, \mathcal{M})
\end{aligned}$$

This function is then used to solve for a $FIRST_k$ function for a fixed k with fixed point iteration the following way.

1. Initialize a dictionary $\mathcal{M}_0$ such that $\mathcal{M}_0(A) = \emptyset$ for all $A \in N$.

2. A new dictionary $\mathcal{M}_{i+1} : N \to P(T^*)$ is constructed by $\mathcal{M}_{i+1}(A) = \bigcup_{\delta\,:\,A \Rightarrow \delta \in P} FIRST_k'(\delta, \mathcal{M}_i)$ for all $A \in N$ where $\mathcal{M}_i$ is the last dictionary that was constructed.

3. If $\mathcal{M}_{i+1} = \mathcal{M}_i$ then terminate the algorithm terminates else recompute step 2.

Let $\mathcal{M}_f$ be the final dictionary after the algorithm terminates then it holds that $FIRST_k(\omega) = FIRST'_k(\omega, \mathcal{M}_f)$ if k stays fixed.

Algorithm 2.2 (Solving $FOLLOW_k$ set)**.** Let $G = (N, T, P, S)$ be a context-free grammar, the follow-sets can be solved as followed.

$$FOLLOW_k(A) = \bigcup_{B\,:\,B \to \alpha A \beta \in P} FIRST_k(\beta) \odot_k FOLLOW_k(B)$$

Once again this may not terminate so fixed point iteration can be used with following altered $FOLLOW_k$ and letting $\mathcal{M} : N \to P(T^*)$ be a surjective function.

$$FOLLOW_k(A, \mathcal{M}) = \bigcup_{B\,:\,B \to \alpha A \beta \in P} FIRST_k(\beta) \odot_k \mathcal{M}(B)$$

This $FOLLOW_k$ function for a fixed k can then be computed using the following algorithm.

1. Extend the grammar $G = (N, T, P, S)$ using $G' = (N', T', P', S') = (N \cup \{S'\}, T \cup \{\Box\}, P \cup \{P \to S\Box^k\}, S')$.

2. Initialize a dictionary $\mathcal{M}_0$ such that $\mathcal{M}_0(A) = \emptyset$ for all $A \in N \backslash \{S\}$ and $\mathcal{M}_0(S) = \{\Box^k\}$.

3. A new dictionary $\mathcal{M}_{i+1} : N \to P(T^*)$ is constructed by $\mathcal{M}_{i+1}(A) = \bigcup_{B\,:\,B \to \alpha A \beta \in P} FIRST_k(\beta) \odot_k \mathcal{M}_i(B)$ for all $A \in N$ where $\mathcal{M}_i$ is the last dictionary that was constructed.

4. If $\mathcal{M}_{i+1} \neq \mathcal{M}_i$ then recompute step 3.

5. Let $\mathcal{M}_f$ be the final dictionary after step 4. is completed. Let $\mathcal{M}_u$ be another dictionary where $\mathcal{M}_u(A) = \{\alpha : \alpha\Box^* \in \mathcal{M}_f(A)\}$ for all $A \in N \backslash \{S'\}$

It then holds that $FOLLOW_k(A) = \mathcal{M}_u(A)$ if k stays fixed for grammar G.

2.2 $LLP(q, k)$ Parser Generator

3 Implementation

4 Testing

5 Conclusion

References

[1] Torben Ægidius Mogensen. *Introduction to Compiler Design*. 2nd ed. London: Springer Cham. DOI: https://doi.org/10.1007/978-3-319-66966-3.

[2] Ladislav Vagner and Bořivoj Melichar. "Parallel LL parsing". In: *Acta Informatica* 44.1 (Apr. 2007), pp. 1–21. ISSN: 1432-0525. DOI: 10.1007/s00236-006-0031-y. URL: https://doi.org/10.1007/s00236-006-0031-y.

[3] Wikipedia. *LL parser — Wikipedia, The Free Encyclopedia*. http://en.wikipedia.org/w/index.php?title=LL%20parser&oldid=1145098081. [Online; accessed 03-May-2023]. 2023.