University of Copenhagen
Computer Science Department

Data-Parallel Compilation
Lexical analysis & Syntax Tree Construction

William Henrich Due (mcj284)
Submitted: 5th of April 2024

Abstract

Abstract.

1 Introduction

Introduction.

2 Theory

Hills paper "Parallel lexical analysis and parsing on the AMT distributed array processor" [1] describes a method to obtain the path in a deterministic finite automata given a input string. This section will describe the theory of this method and extend the it for tokenization.

2.1 Data-parallel Lexical Analysis

To explain the theory of parallel lexical analysis we first remind the reader of the definition of a deterministic finite automaton.

Definition 2.1 (DFA)**.** A deterministic finite automata [2] [3] is given by a 5-tuple $(Q, \Sigma, \delta, q_0, F)$ where.

1. Q is the set of states where $|Q| < \infty$.
2. Σ is the set of symbols where $|\Sigma| < \infty$.
3. $\delta : \Sigma \times Q \to Q$ is the transition function.
4. $q_0 \in Q$ is the initial state.
5. $F \subseteq Q$ is the set of accepting states.

This definition is fine as is but we will need to reformualte it to develop data-parallel lexical analysis. We would want the definition to use a curried transition function. But for this to hold then the DFA would also have to be total.

Definition 2.2 (Total DFA)**.** A DFA $(Q, \Sigma, \delta, q_0, F)$ is said to be total if and only if

$$\delta(a, q) \in Q : \forall(a, q) \in \Sigma \times Q$$

If a DFA is total we may use a curried transition function $\delta : \Sigma \to Q \to Q$.

This is needed since else the the function would not be fully defined in the domains Σ and Q.

The reason for doing so is because if we have any two functions $g = \delta(a)$ and $f = \delta(a')$ then it follows from composition that.

$$g(f(q)) = (g \circ f)(q)$$

This allows for an alternative way of determining if a string can be produced by an DFA. Instead of first evaluating $f(q)$, then $g(f(q))$ and then checking if this state is a member of F. We could instead partially apply δ to the symbols and then compose them to a single function which could be used to determine if a string is valid. This sets the stage for data-parallel lexing, we want to find a way to make the problem into a `map-reduce`. We want to do this because it can be computed using a data-parallel implementation unlike the normal way of traversing a DFA.

For the ability to use a data-parallel `map-reduce` we must have a monoidal structure. Here Δ is the set of all the composed partially applied δ functions needs to be closed under function composition.

Proposition 2.1 (DFA Composition Closure)**.** Given a total then the set of endofunctions $\Delta : Q^Q$ will be closed under composition. The set Δ is the set Δ_i in the recurrence relation with the smallest i such that $\Delta_i = \Delta_{i+1}$.

$$\Delta_1 = \{\delta(a) : a \in \Sigma\}$$
$$\Delta_{i+1} = \Delta_i \cup \{f \circ g : f, g \in \Delta_i\}$$

Proof. We will circt by showing that a solution Δ exists. First note that the cardinality is monotonically increasing i.e. $\Delta_i \subseteq \Delta_{i+1}$ since Δ_{i+1} is a union of Δ_i and another set. Secondly note that since $|Q| < \infty$ then a finite amount of functions of the form $Q \to Q$ can exists. Since the set is bounded and increasing then at some point $\Delta_i = \Delta_{i+1}$ and the smallest i where it holds is the solution Δ.

For Δ to be closed under composition, then for arbitray $f, g \in \Delta$ it must hold that $f \circ g \in \Delta$. Since Δ_1 is the set of endofunctions that contructs Δ and composition is associative then all elements of Δ can be expressed of the form.

$$\delta(a_1) \circ \cdots \circ \delta(a_n) \in \Delta$$

If all permutations with replacement of Δ_1 of any sequence length are members of Δ then Δ would be closed under composition. Futhermore, it is known that Δ is finite so the sequences at some point $\Delta_i = \Delta_{i+1}$ would only add new sequences but no new endofunctions. Therefore it suffices to show that if all sequences of length k where $1 \leq k \leq i$ is a subset of Δ_i then Δ is closed under composition. This can be shown using a proof by induction.

Base: Δ_1 trivially holds since it only contains sequences of length one and they are the initial endofunctions.

Step: Given Δ_i contains every sequence of length i or less then we to show this implies that Δ_{i+1} will contain every sequence of length $i+1$ or less.

By the induction hypothesis Δ_{i+1} must contain every sequence of length i or less due to $\Delta_i \subseteq \Delta_{i+1}$. It remains to show that every sequence of length $i+1$ is a member of Δ_{i+1}. It is known that a direct product of Δ_i is used in the definition of Δ_{i+1} so $\{f \circ g : f, g \in \Delta_i\} \subseteq \Delta_{i+1}$. A direct product between sequences of length 1 and i will create every sequence of length $i+1$ and therefore every sequence of length $i+1$ is a member of Δ_{i+1}. Thereby Δ is closed under composition. $\square$

Since Δ is closed under an arbitrary binary associative operations then it follows that Δ and function composition induces a monoidal structure.

Corollary 2.1 (DFA Composition Monoid)**.** DFA composition closure induces a semigroup which in turn induces the monoid $(\Delta \cup \{id\}, \circ)$ where $id : Q \to Q$ and $id(q) = q$.

Knowing this we can establish the following algorithm

Algorithm 2.1 (Data-parallel String Match)**.** It can be determined in $O(n)$ work and $O(\log n)$ span if a string can be produced by a DFA. First construct the total DFA $(Q, \Sigma, \delta, q_0, F)$ from the DFA.

1. Partially apply δ to every symbol in the input string such that it becomes a sequence of endofunctions.

$$\textbf{map}\ \delta\ [a_1, a_2, \ldots, a_{n-1}, a_n] = [\delta(a_1), \delta(a_2), \ldots, \delta(a_{n-1}), \delta(a_n)]$$

2. Reduce the endofunction into a single endofunction $\delta' : Q \to Q$.

$$\textbf{reduce}\ (\circ)\ id\ [\delta(a_1), \delta(a_2), \ldots, \delta(a_{n-1}), \delta(a_n)] = \delta'$$

3. Evaluate $\delta'(q_0)$ and determine if $\delta'(q_0) \in F$.

2.2 Data-parallel Tokenization

For data-parallel tokenization we need to extent data-parallel algorithm 2.1 will be needed to be extended. The idea will be to use a data-parallel `map-scan` instead since it will gives all the states. This is also the methods described in Hills [1] paper. The problem is we need to be able to recongnize the longest strech of symbols that results in a token. And we also need to restart the traversal of DFA if a final state is hit while no options to traverse further. To do so we first need af function to define a function to recongnize tokens.

Definition 2.3 (Token Function)**.** Given a DFA and a set of tokens T. The token function $\mathcal{T} : F \to T$ is a function that maps accepting states to some token.

We will also need a single state to point to which is the dead state. This will become useful when the DFA needs to be traversed multiple times. Since we will need to be able to recongnize when the end of a traversal is reached and we have to restart.

Definition 2.4 (Total DFA with a Dead State)**.** Given a DFA it is made total with a dead state by defining a new set of states $Q' = Q \cup \{d\}$ where d is the dead state. Additionally a new transition function δ' is defined.

$$\delta'(a, q) = \begin{cases} d & (a, q) \notin \mathrm{dom}(\delta) \\ \delta(a, q) & \text{otherwise} \end{cases}$$

Now that we have a definition of a DFA where the dead state is known another problem is needed to be solved. The problem is as mentioned before that the traversal of the DFA has to be restarted if a dead state is reached after an accepting state. This is done using the following binary operation.

Definition 2.5 (Safe Composition)**.** Given a DFA with two endofunctions $f = \delta(a)$ and $g = \delta(b)$. The operation $\oplus$ makes f safe to compose such that it will continue traversing the DFA.

$$(f \oplus g)(x) = \begin{cases} q_0 & f(g(x)) = d \land g(x) \in F \\ g(x) & \text{otherwise} \end{cases}$$

This definition will make every possible final state become the initial state. This forgets the final state but it allows for the traversal to continue. The forgetfulness is not a problem this can be solved by looking at the previous endofunction. Using this and previous definitions can be put together to the following algorithm.

Algorithm 2.2 (Data-parallel Tokenization)**.** Given a total DFA with a dead state where $q_0 \notin F$ and a token function $\mathcal{T} : F \to T$. A string can be tokenized in $O(n)$ work and $O(\log n)$ span.

1. Let $s = [a_1, a_2, \ldots, a_{n-1}, a_n]$ be a string that will be tokenized then partially apply δ to every symbol.

$$\mathbf{map}\ (\delta)\ s = x$$

2. Make every endofunction safe for composition beside for the last endofunction.

$$\mathbf{map}\ (\oplus)\ (\mathbf{init}\ x)\ (\mathbf{tail}\ x) \mathbin{+\!\!+} [\delta_n] = [\delta_1 \oplus \delta_2, \ldots, \delta_{n-1} \oplus \delta_n, \delta_n] = y$$

3. Do a scan to get every composition.

$$\mathbf{scan}\ (\circ)\ id\ y = z$$

4. Compute the actual state and determine weather it is an end state[1].

```
$\mathbf{let}\ f = \lambda i \to$
  $\mathbf{let}\ s = \mathbf{if}\ i = 0\ \mathbf{then}\ x[i](q_0)\ \mathbf{else}\ (z[i-1] \circ x[i])(q_0)$
  $\mathbf{in}\ (i = n - 1 \lor (z[i](q_0) = q_0 \land s \in F), s)$
$\mathbf{in}\ \mathbf{map}\ f\ (\mathbf{iota}\ n)$
```

5. Remove every state that is not an ending state.

$$\mathbf{filter}\ (\lambda(b, s) \to b)$$

6. Assert that every ending state is an accepting state.

$$(\mathbf{reduce}\ (\land)\ \mathbf{true}) \circ (\mathbf{map}\ (\lambda(b, s) \to s \in F))$$

7. Produce the token sequence.

$$\mathbf{map}\ (\lambda(b, s) \to \mathcal{T}(s))$$

The correctness of this algorithm can be established from in step 2. the accepting states will be mapped to the initial state. Then in step 3. a scan using composition would allow us to find the paths by evaluating each function at q_0. The path will be by the definition $\oplus$ such that the last state becomes the initial state in each path besides the last path.

[1] If you were to also keep track of the index then the span of each token could also be found.

3 Implementation

3.1 Tokenization

4 Conclusion

Conclusion.

References

[1] Jonathan M.D Hill. “Parallel lexical analysis and parsing on the AMT distributed array processor”. In: *Parallel Computing* 18.6 (1992), pp. 699–714. ISSN: 0167-8191. DOI: https://doi.org/10.1016/0167-8191(92)90008-U. URL: https://www.sciencedirect.com/science/article/pii/016781919290008U.

[2] John E. Hopcroft, Rajeev Motwani, and Jeffrey D. Ullman. *Introduction to Automata Theory, Languages, and Computation (3rd Edition)*. USA: Addison-Wesley Longman Publishing Co., Inc., 2006. ISBN: 0321455363.

[3] Wikipedia contributors. *Deterministic finite automaton — Wikipedia, The Free Encyclopedia*. [Online; accessed 4-February-2024]. 2023. URL: https://en.wikipedia.org/w/index.php?title=Deterministic_finite_automaton&oldid=1192025610.

5 I can’t get this to work.

Definition 5.1 (Safe Composition)**.** Given a DFA with two endofunctions $f = \delta(a)$ and $g = \delta(b)$. The operation $\oplus$ makes f safe to compose such that it will continue traversing the DFA.

$$(f \oplus g)(x) = \begin{cases} q_0 & f(g(x)) = d \\ f(g(x)) & x \in Q \end{cases}$$

Proof. It will be shown that safe composition is associative i.e.

$$((f \oplus g) \oplus h)(x) = (f \oplus (g \oplus h))(x)$$

By the definition of safe composition the left side is.

$$((f \oplus g) \oplus h)(x) = \begin{cases} q_0 & (f \oplus g)(h(x)) = d \\ (f \oplus g)(h(x)) & x \in Q \end{cases}$$

By definition of $\oplus$ then we know $(f \oplus g)(h(x)) \neq d$ meaning we only need to consider the case resulting in $(f \oplus g)(h(x))$. By definition this is.

$$((f \oplus g) \oplus h)(x) = \begin{cases} q_0 & f(g(h(x))) = d \\ f(g(h(x))) & x \in Q \end{cases}$$

Now we wish to show the right hand side can be simplified into the defintion of the left hand side. By definition of $\oplus$ the right hand side is.

$$(f \oplus (g \oplus h))(x) = \begin{cases} q_0 & f((g \oplus h)(x)) = d \\ f((g \oplus h)(x)) & x \in Q \end{cases}$$

We can expand the case resulting in q_0 given $f((g \oplus h)(x)) = d$, by definition there are only two cases of $(g \oplus h)(x)$ either it is equal to d or not.

$$(f \oplus (g \oplus h))(x) = \begin{cases} q_0 & f(g(h(x))) = d \wedge g(h(x)) \neq d \\ q_0 & f(q_0) = d \wedge g(h(x)) = d \\ f((g \oplus h)(x)) & x \in Q \end{cases}$$

The two first cases can be simplified into the single predicate $f(g(h(x))) = d$. Since if $g(h(x)) = d$ then $f(g(h(x))) = d$ since all transitions from a dead state maps to itself by definition.

$$(f \oplus (g \oplus h))(x) = \begin{cases} q_0 & f(g(h(x))) = d \\ f((g \oplus h)(x)) & x \in Q \end{cases}$$

We can now expand the last case.

$$(f \oplus (g \oplus h))(x) = \begin{cases} f(q_0) & g(h(x)) = d \\ q_0 & f(g(h(x))) = d \\ f(g(h(x))) & x \in Q \end{cases}$$

$\square$