Proof-Carrying Model for Parsing Techniques

Mohamed El-Zawawy
College of Computer and Information Sciences
Al Imam Mohammad Ibn Saud Islamic University
Riyadh, KSA

November 2, 2015

Parsing:

The parsing process aims at analyzing and checking the structure of a given input (computer program) with respect to a specific grammar.

Each grammar can typically produce an infinite number of *sentences* (programs). However grammars have finite sizes.

A grammar abstracts briefly an infinite number of program architectures of a specific type (structural, functional, object-oriented, etc).

Importance of parsing:

Parsing process is essential for further processing of parsed objects (programs). For example in natural languages processing, recognizing the verb of a sentence facilitates the sentence translation.

Parsing process is also important to understand grammars which summarize our realization of a class of programs. Error-recovering parsers are important tools for correcting some program faults.

Theoretical foundations of parsing excuse it from being described as esoteric or as a mathematical discipline. Using string cutting and pasting, parsing can be realized and applied.

Parsing techniques:

Parsing techniques are classified into two categories; top-down and bottom-up.

A parsing technique that tries to establish a derivation for the input program starting from the start symbol is classified as a top-down algorithm.

If a parsing technique starts with the input program and rolls back trying to establish the derivation in the reverse order until reaching the start symbol, the technique is then described as bottom-up.

Common examples of top-down and bottom- up parsers are LL(1) and LR(1), respectively.

Proof-Carrying Code:

Some applications like proof-carrying code and mobile computing require the parser to associate each parse tree with a correctness proof.

This proof ensures the validity of the tree. Unfortunately most of existing parsers are algorithmic in their style and hence build the parse tree but not the correctness proof.

Proposing new sittings for common parsing algorithms so that correctness proofs are among outputs of these algorithms (rather than parse trees) is a requirement of many modern computing applications.

Motivation of the paper Systems of inference rules were found very useful for optimization phases of compilers. This is so as they provide compact correctness proofs for optimizations. These proofs are required by applications like proof-carrying code and mobile computing.

The motivation of this paper is the need for parsing techniques that associate each parsing process with a simply-structured correctness proof.

The research behind this paper reveals that parsing techniques having the form of inference rules are the perfect choice to build the required proofs.

Contribution of the paper: This paper introduces new models for common existing parsing techniques such as LL(1), operator-precedence, SLR, LR(1), and LALR.

The proposed models are built mainly of inference rules whose outputs in this case are pairs of parse trees and correctness proofs.

Rather than providing the required correctness proofs, the new models are faster than their corresponding ones as is confirmed by experimental results.

Mathematical proofs of the equivalence between each proposed model and its corresponding original parsing technique are outlined in the paper.

The proposed models are supported with illustrative parsing examples that are detailed.

Inference rules for LL(1) parser.

$$\frac{}{\epsilon : \epsilon \to \$}\ \text{(Base)}$$

$$\frac{a \in \text{first}(T) \qquad (N \to T) \in P \qquad a\alpha : TM \to \$}{a\alpha : NM \to \$}\ (\text{Predict}_1)$$

$$\frac{a \in \text{follow}(N) \qquad N \to \epsilon \qquad a\alpha : M \to \$}{a\alpha : NM \to \$}\ (\text{Predict}_2)$$

$$\frac{\alpha : \beta \to \$}{a\alpha : a\beta \to \$}\ \text{(Match)}$$

Soundness of the inference rules for LL(1) parser.

Let (S, N, T, P) be a $LL(1)$ grammar and $w \in T^+$. Then $w \in LL(1)$ – infer if and only if w is accepted by the $LL(1)$-algorithm. Moreover the left-most derivation obtained by the LL(1)-algorithm is the same as **path(w,ll(1))**.

Thanks!