Safety and Completeness of Disambiguation corresponds to Termination and Confluence of Reordering

Luís Eduardo de Souza Amorim[1] and Eelco Visser[2]

[1] Australian National University, Australia
LuisEduardo.deSouzaAmorim@anu.edu.au
[2] Delft University of Technology, The Netherlands
e.visser@tudelft.nl

Abstract

Associativity and priority are well known techniques to disambiguate expression grammars. In recent work we develop a direct semantics for disambiguation by associativity and priority rules and prove that a safe and complete disambiguation relation produces a safe and complete disambiguation. The proof approach relies on a correspondence between disambiguation and term rewriting such that safety of disambiguation corresponds to termination of the rewrite system and completeness of disambiguation correspond to confluence of the rewrite system. In this extended abstract we illustrate that approach using diagrams.

1 Introduction

Associativity and priority are well known techniques to disambiguate expression grammars. Figure 1 illustrates the approach. An expression grammar defines the infix operators of an expression language using left- and right-recursive productions such as `Exp.Add = Exp "+" Exp`. Such a grammar is ambiguous; an expression such as `a + b + c` can be read as `(a + b) + c` or as `a + (b + c)`. One way to disambiguate an expression grammar is to transform it to a grammar that uses extra non-terminals to represent priority levels. However such grammars are harder to read and the direct correspondence to the underlying abstract syntax trees is lost. An alternative approach is to augment an ambiguous expression grammar with associativity and priority rules. The semantics of such disambiguation rules is typically defined *indirectly* in the implementation of parser generators or by means of grammar transformations. In recent work, we have developed a *direct semantics* for associativity and priority in terms of subtree exclusion that extends to expression grammars with prefix and postfix operators, mixfix operators, indirect recursion, and overlap. We are currently revising a paper about this work for the TOPLAS journal [2]. A previous version of the semantics of disambiguation rules appeared in [1], but did not feature the proof technique based on rewriting. We refer to those papers for a discussion of related work.

```
lexical syntax
  ID = [a-zA-Z][a-zA-Z0-9]*
context-free syntax
  Exp.Var = ID
  Exp.Add = Exp "+" Exp {left}
  Exp.Min = Exp "-" Exp {left}
  Exp.Mul = Exp "*" Exp {left}
  Exp.Div = Exp "/" Exp {left}
  Exp.Pow = Exp "^" Exp {right}
context-free priorities
  Exp.Pow
  > {left: Exp.Mul Exp.Div}
  > {left: Exp.Add Exp.Min}
```

Figure 1: SDF3 definition.

To verify the approach we developed a technique based on term rewriting that shows that soundness and completeness of disambiguation corresponds to termination and confluence reordering parse trees. In this extended abstract we illustrate the proof technique for the case of infix expression grammars. We omit formal definitions and mostly explain the approach using diagrams.

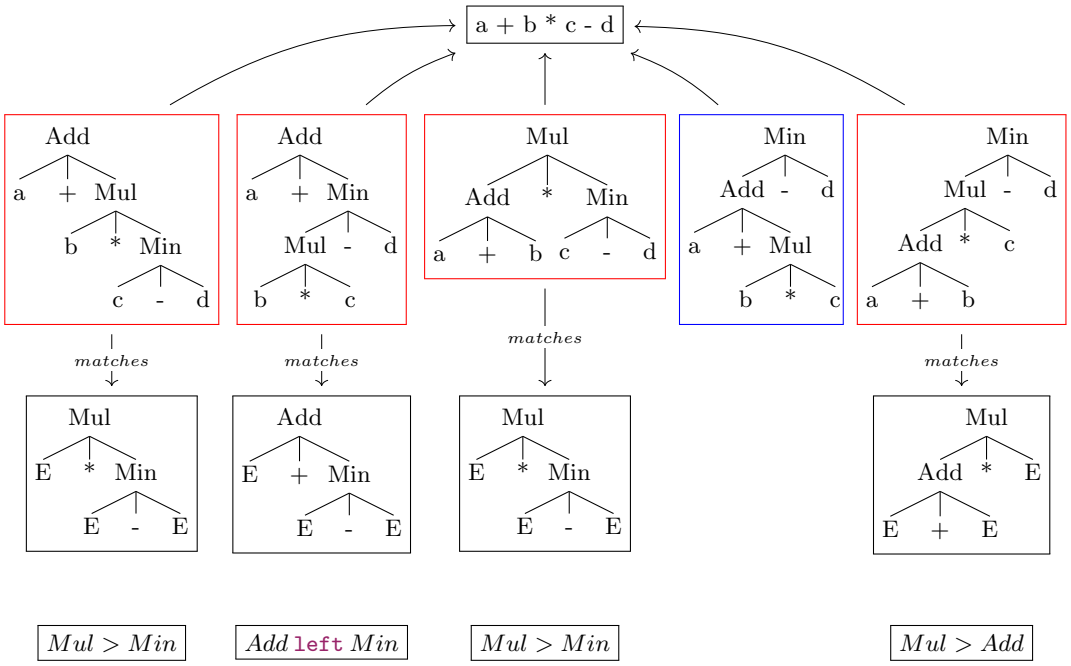
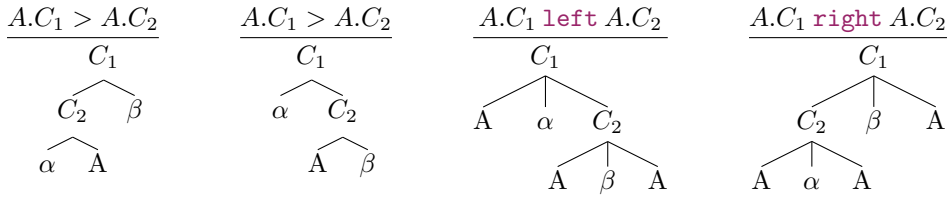


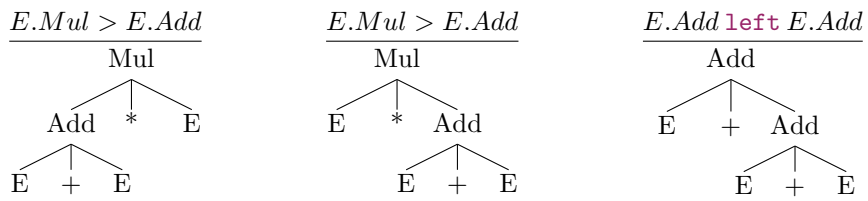
Figure 2: Parse trees for an ambiguous expression, and its disambiguation using subtree exclusion rules

2 Disambiguation by Subtree Exclusion

An ambiguous sentence has multiple parse trees. The diagram in figure 2 shows the parse trees for the expression `a + b * c - d` according the underlying grammar of figure 1. Disambiguation by subtree exclusion defines conflict patterns that should not occur in selected parse trees. The safe subtree exclusion rules (for infix expressions) of SDF3 [2] are defined as follows:



Instantiating these rules to some of the disambiguation rules in figure 1 leads to the following subtree exclusion patterns (aka conflict patterns):

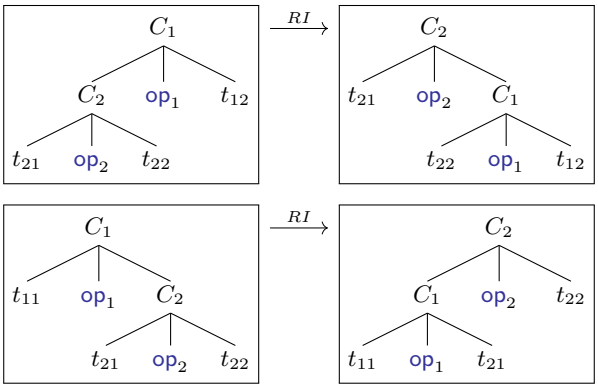


Applying these rules to the expression in figure 2 shows that the sentence is completely disambiguated as all but one parse tree for the expression are rejected.

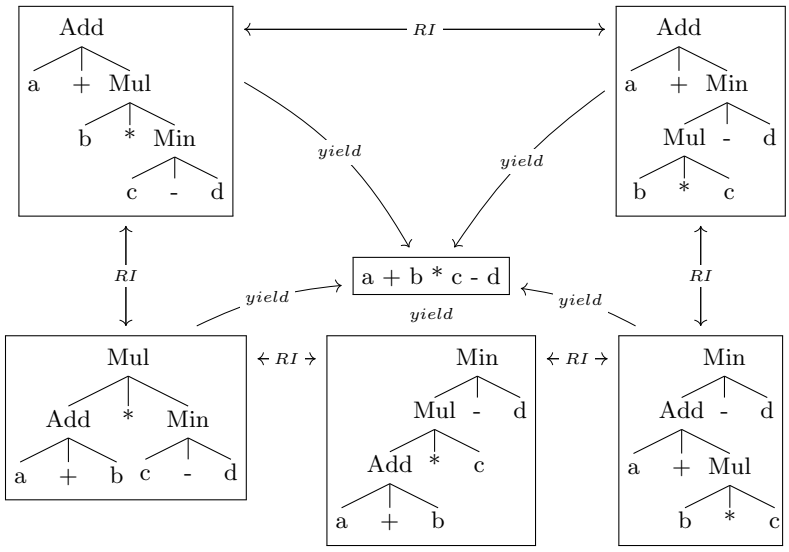
A set of disambiguation rules is *safe* when each sentence in the underlying grammar is also a sentence in the disambiguated grammar. That is, no sentences are excluded. A set of disambiguation rules is *complete* when each sentence in the underlying grammar has at most one parse tree in the disambiguated grammar. That is, each sentence is completely disambiguated. How can we prove that set of disambiguation rules safe and complete? That is a central question in our work on the semantics disambiguation rules [2].

3 Proving Safety and Completeness

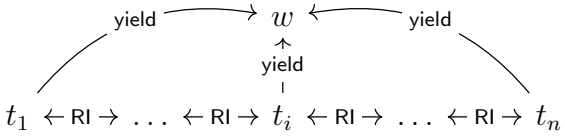
The central insight in our approach to proving safety and completeness of disambiguation is that disambiguation by means of associativity and priority rules corresponds to reordering of parse trees. For infix expression grammars we define a rewrite system generated by instiantiating the following rewrite rule schemas for each pair of productions in a grammar:



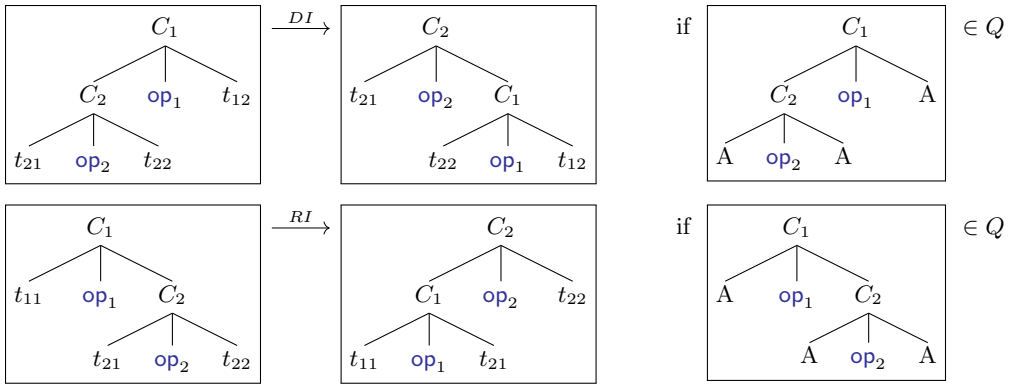
When we apply these rules to the expression in figure 2, we see that all trees can be converted into each other using these rewrite rules:



In general, we have a theorem that states that all ambiguities in an infix expression grammar are related by reordering, expressed diagrammatically as follows:



The reordering rewrite system is non-terminating. Each tree can be converted in each other tree. Using the subtree exclusion patterns generated from the associativity and priority rules of a grammar, we direct the rules of the rewrite system, leading to the following rule schemas:



In our paper [2] we show that safety of disambiguation corresponds to termatination of this rewrite system and that completeness of disambiguation corresponds to confluence of the rewrite system. We illustrate that here using the diagrams in figures 3, 4, and 5.

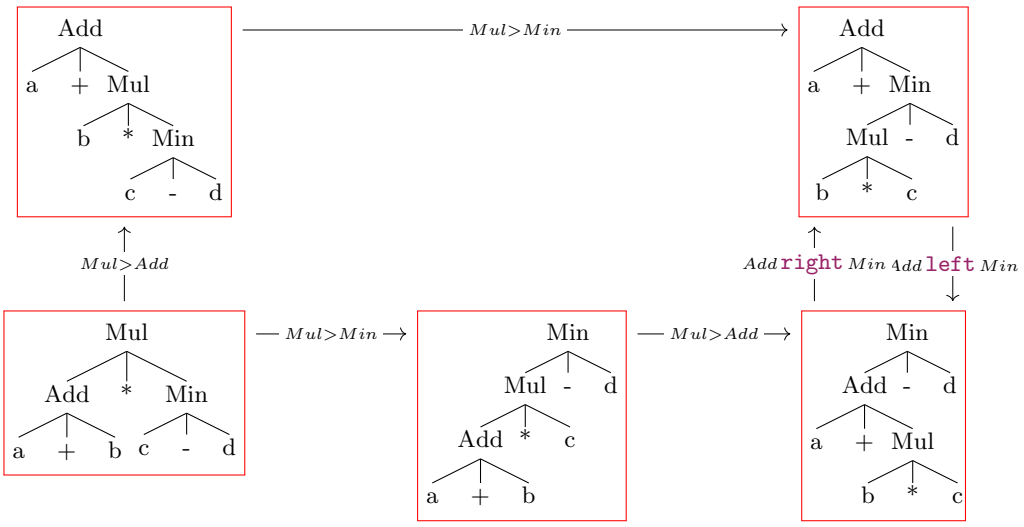


Figure 3: Unsafety corresponds to non-termination

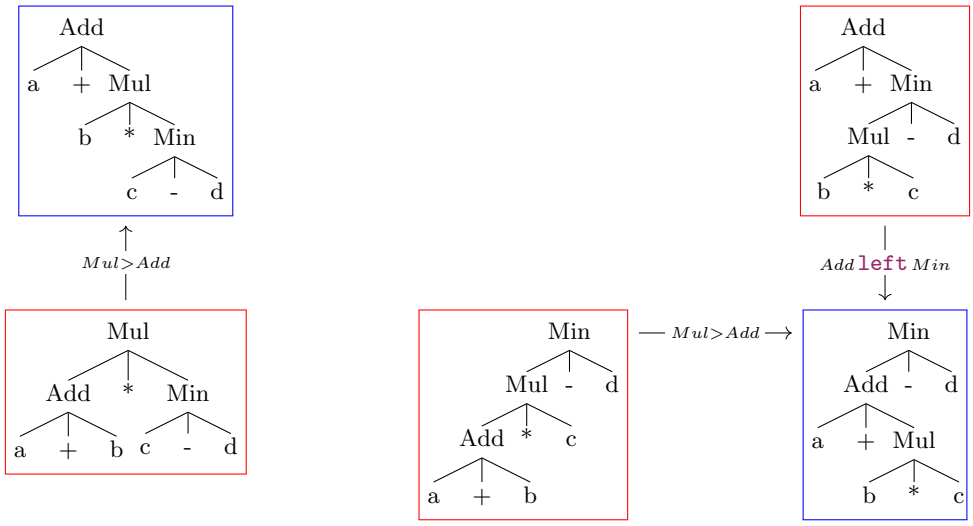


Figure 4: Incompleteness corresponds to non-confluence (non-Church Rosser)

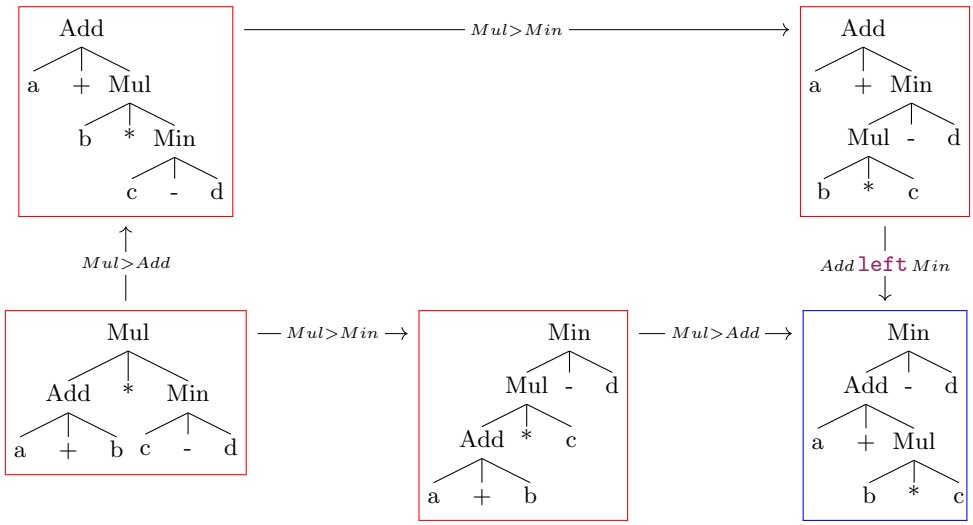


Figure 5: Safety and completeness correspond to termination and confluence.

References

[1] Luís Eduardo de Souza Amorim. *Declarative Syntax Definition for Modern Language Workbenches*. PhD thesis, Delft University of Technology, 2019.

[2] Luís Eduardo de Souza Amorim and Eelco Visser. A direct semantics for declarative disambiguation of expression grammars. *ACM Transactions on Programming Languages*, 2020. under revision.