A Direct Semantics for Declarative Disambiguation of Expression Grammars

LUIS EDUARDO S. AMORIM (Delft University of Technology)
EELCO VISSER (Delft University of Technology)

TU Delft
March 2019
What is the meaning of associativity and priority declarations?

context-free syntax

```
Exp.Var = ID
Exp.Int = INT
Expr = "(" Expr ")" {bracket}
Exp.Add = Exp "+" Exp {left}
Exp.Sub = Exp "-" Exp {left}
Exp.Mul = Exp "*" Exp {left}
Exp.Minus = "-" Exp
Exp.Lambda = "\" ID "." Exp
Exp.Inc = Exp "++"
Exp.If = "if" Exp "then" Exp
Exp.IfElse = "if" Exp "then" Exp "else" Exp
Exp.Subscript = Exp "[" Exp "]"
Exp.While = "while" Exp "do" Exp "done"
Exp.App = Exp Exp {left}
Exp.Function = "function" PMatch+ {longest-match}
PMatch.Clause = ID "-" Exp
```

context-free priorities

```
{Exp.Subscript Exp.Inc} > Exp.App > Exp.Minus >
Exp.Mul > {left: Exp.Add Exp.Sub} > Exp.IfElse >
{Exp.If Exp.Lambda Exp.Function}
```
Research Questions

What is the meaning of a set of disambiguation rules for a grammar?
- What are the parse trees associated with sentences in the language of the disambiguated grammar?
- independent of particular implementation strategy?

Is a set of disambiguation rules safe?
- Do the disambiguation rules preserve the language of the grammar they disambiguate?
- Is it necessary for disambiguation rules to be safe, or can rules exclude sentences?

Is a set of disambiguation rules complete?
- Do the rules identify at most one parse tree for each sentence in the language?
- Not obvious: ambiguity of CFGs is undecidable

What is the coverage of disambiguation rules?
- What classes of ambiguity do the rules solve?

What is an effective implementation strategy for disambiguation rules?

What is the notational overhead of disambiguation rules?
- More effective than an encoding in the grammar?
Contributions

Expression grammars
- Sub-classes of CFGs with decidable ambiguity
- Extraction of embedded expression grammars

Harmless overlap
- Avoid inherent ambiguities

Subtree exclusion patterns
- Deep priority conflict patterns

Safe and complete
- Preserve language and solve all ambiguities
- Proof: induction on trees under subtree exclusion

Implementation in SDF3
- Transformation to contextual grammars
- Data-dependent parsing

Evaluation on 5 programming languages
This Talk

context-free grammars

indirectly recursive distfix (Section 7)

overlapping distfix (Section 6.1)

distfix (Section 6)

basic (Section 5)

prefix (Section 4)

infix (Section 3)
Subtree Exclusion is Complete

(Inductive case) Assume that \( t_1, t_2 \in T^Q_A(G) \) and that their yields are unique.

(2) If \( A.C = A \oplus A \) is an infix production in \( G \), since each constructor uniquely identifies a production, that is the only way we can construct the tree \( t = [A.C = t_1 \oplus t_2] \). Now we need to demonstrate that if \( t \in T^Q(G) \) then there is no tree \( t' \in T^Q(G) \) such that \( t' \neq t \) and \( yield(t) = yield(t') \). We consider the following cases:

- If \( t_1 = [A.C_1 = t_{11} \otimes t_{12}] \) with yield \( u \otimes v \) and \( t_2 = [A.C_2 = \langle t_{21} \rangle] \) with yields \( \langle w \rangle \) then \( t = [A.C = [A.C_1 = t_{11} \otimes t_{12}] \oplus [A.C_2 = \langle t_{21} \rangle]] \) with yield \( u \otimes v \oplus \langle w \rangle \).

By totality of disambiguation rules, we have that there is a disambiguation relation between \( A.C \) and \( A.C_1 \). If \( A.C > A.C_1 \) then \( t \) matches a conflict pattern and therefore \( t \notin T^Q(G) \). If \( A.C_1 > A.C \) then \( t \) does not match a conflict pattern (since there are no other disambiguation relations between the productions). The only other tree with the same yield is \( t' = [A.C_1 = t_{11} \otimes [A.C = t_{12} \oplus [A.C_2 = \langle t_{21} \rangle]]] \in T^Q(G) \). However, \( t' \) does have a priority conflict and therefore \( t' \notin T^Q(G) \). If the disambiguation relation is left, right, or non-assoc, the proof works analogously.
Grammars and Ambiguity
context-free syntax

\[
\begin{align*}
\text{Exp.Add} &= \text{Exp} \; "+" \; \text{Exp} \\
\text{Exp.Sub} &= \text{Exp} \; "-" \; \text{Exp} \\
\text{Exp.Mul} &= \text{Exp} \; "*" \; \text{Exp} \\
\text{Exp.Var} &= \text{ID}
\end{align*}
\]

\[
\begin{align*}
a \in \Sigma \\
a \in T^a(G)
\end{align*}
\]

\[
\begin{align*}
A.C &= X_1\ldots X_n \in P(G) \\
t_i &\in T^{X_i}(G) \\
1 \leq i \leq n \\
A.C = t_1\ldots t_n &\in T^A(G)
\end{align*}
\]

\[
L(G) = \{L^X(G) \mid \text{yield}(T^X(G)), X \in V\}
\]

\[\text{Exp.Add} = [\text{Exp.Var} = \text{ID}] + [\text{Exp.Var} = \text{ID}]\]
Parsing

\[ \Pi(G)(w) = \{ t \in T^X(G) \mid \text{yield}(t) = w, X \in V \} \]
Lemma 2.5. A parse tree directly corresponds to a derivation, modulo the order in which productions are applied.
Parse Tree to Abstract Syntax Tree

class context-free syntax

- Exp.Add = Exp "+" Exp
- Exp.Sub = Exp "-" Exp
- Exp.Mul = Exp "*" Exp
- Exp.Var = ID

signature

<table>
<thead>
<tr>
<th>constructors</th>
</tr>
</thead>
<tbody>
<tr>
<td>Add : Exp * Exp -&gt; Exp</td>
</tr>
<tr>
<td>Sub : Exp * Exp -&gt; Exp</td>
</tr>
<tr>
<td>Mul : Exp * Exp -&gt; Exp</td>
</tr>
<tr>
<td>Var : ID -&gt; Exp</td>
</tr>
</tbody>
</table>

```
[Exp.Add = [Exp.Add = [Exp.Var = a] * [Exp.Var = b] + [Exp.Var = c]]
```

Add(Mul(Var("a"), Var("b")), Var("c"))
Tree Patterns and Pattern Matching

\[
X \in V \\
\overrightarrow{X} \in TP^X(G) \\
A.C = X_1...X_n \in P(G) \quad t_i \in TP^{X_i}(G) \quad 1 \leq i \leq n \\
[A.C = t_1...t_n] \in TP^A(G)
\]

\[
a \in \Sigma \\
\overrightarrow{M}(a, a) \\
[A.C = t_1...t_n] \in T^A(G) \\
\overrightarrow{M}([A.C = t_1...t_n], A) \\
[A.C = t_1...t_n] \in T^A(G) \quad [A.C = q_1...q_n] \in TP^A(G) \\
\overrightarrow{M}(t_i, q_i) \quad 1 \leq i \leq n \\
\overrightarrow{M}([A.C = t_1...t_n], [A.C = q_1...q_n])
\]
Tree Patterns and Pattern Matching: Example

\[
\text{[Exp.Add} = \text{[Exp.Add} = \text{[Exp.Var} = \text{ID}] + \text{[Exp.Var} = \text{ID}] + \text{[Exp.Var} = \text{ID}]\]
\]

\[
\text{[Exp.Add} = \text{[Exp.Add} = \text{Exp + Exp} + \text{Exp}]\]
Ambiguity

\begin{align*}
(i) \quad & \text{Exp} \Rightarrow_G \text{Exp} + \text{Exp} \Rightarrow_G \text{Exp} \Rightarrow_G \text{Exp} \Rightarrow_G a + \text{Exp} \Rightarrow_G a + \text{Exp} + \text{Exp} \Rightarrow * \quad a + b + c \\
(ii) \quad & \text{Exp} \Rightarrow_G \text{Exp} + \text{Exp} \Rightarrow_G \text{Exp} \Rightarrow_G \text{Exp} + \text{Exp} \Rightarrow * \quad a + b + c
\end{align*}

context-free syntax

\begin{align*}
\text{Exp.Add} &= \text{Exp} "+" \text{Exp} \\
\text{Exp.Sub} &= \text{Exp} "-" \text{Exp} \\
\text{Exp.Mul} &= \text{Exp} "*" \text{Exp} \\
\text{Exp.Var} &= \text{ID}
\end{align*}

\begin{align*}
[\text{Exp.Add} &= a + [\text{Exp.Add} = b + c]] \\
[\text{Exp.Add} &= [\text{Exp.Add} = a + b] + c]
\end{align*}
Explicit Disambiguation (Brackets)

context-free syntax

\[\begin{align*}
\text{Exp.Add} &= \text{Exp} \ "+" \ \text{Exp} \\
\text{Exp.Sub} &= \text{Exp} \ "-" \ \text{Exp} \\
\text{Exp.Mul} &= \text{Exp} \ "*" \ \text{Exp} \\
\text{Exp.Var} &= \text{ID} \\
\text{Exp} &= "(" \ \text{Exp} \ ")" \ \{\text{bracket}\}
\end{align*}\]

\[a \times (b + c)\]

\[[\text{Exp.Mul} = a \times [\text{Exp} = ([\text{Exp.Add} = b + c])]]\]

\[\text{Mul}(a, \text{Add}(b, c))\]
Disambiguation Filter

\[ F(\Phi) \subseteq \Phi \text{ for any } \Phi \subseteq T(G) \]

\[ L(G/F) = \{ w \in \Sigma^* \mid \exists \Phi \subseteq T(G), \text{ yield}(\Phi) = \{ w \}, F(\Phi) = \Phi \} \]
Subtree Exclusion Filter

\[ F^Q(\Phi) = \{ t \in \Phi \mid \nexists t' \in \text{sub}(t) : M(t', Q) \} \]
Trees under Subtree Exclusion

\[
\begin{align*}
    a & \in \Sigma \quad \neg \mathcal{M}(a, Q) \\
    a & \in T_a^Q(G) \\
    A.C = X_1 \ldots X_n & \in P(G) \quad t_i \in T_{X_i}^Q(G) \text{ for } 1 \leq i \leq n \quad t = [A.C = t_1 \ldots t_n] \quad \neg \mathcal{M}(t, Q) \\
    t & \in T_A^Q(G)
\end{align*}
\]

\[
    t \in T_X^Q(G) \iff t \in T_X(G) \land t \in F^Q(\{t\})
\]

\[
    L(G/F^Q) = L(G^Q)
\]
Corollary 2.17. A subtree exclusion filter for a set of patterns $Q$ for a grammar $G$ is safe if for each $w \in L(G)$ there is at least one $t \in T^Q(G)$ with $\text{yield}(t) = w$.

Corollary 2.18. A subtree exclusion filter for a set of patterns $Q$ for a grammar $G$ is completely disambiguating if $t_1, t_2 \in T^Q(G) \implies \text{yield}(t_1) \neq \text{yield}(t_2) \lor t_1 = t_2$
Expression Grammars
Embedded Expression Grammars

```plaintext
lexical syntax
ID = [a-zA-Z][a-zA-Z0-9]*
INT = [0-9]+
ID = "if" {reject}
ID = "class" {reject}

lexical restrictions
ID =/[a-zA-Z0-9]
INT =/[0-9]

context-free syntax
Class.Class = "class" ID "{" Mem* "}
Mem.Method = Type ID "(" Arg* ")" "{" Stmt* "}
Stmt.If = "if" "(" Expr ")" Stmt
Stmt.Expr = Expr ";"
Expr.Int = INT
Expr.Var = ID
Expr = "(" Expr ")" {bracket}
Expr.Add = Expr "+" Expr {left}
Expr.Eq = Expr "==" Expr {non-associ}
Expr.Call = Expr "." ID "(" {Expr ","}* ")"

context-free priorities
Expr.Call > Expr.Add > Expr.Eq
```
Classes of Expression Grammars

Basic

\[ A.C = LEX \]
\[ A.C = \triangleright A \triangleleft \]
\[ A.C = A \oplus A \]
\[ A.C = \triangleright A \triangleleft \]

Distfix

\[ A.C = \triangleright A \oplus_1 \ldots \oplus_k A \]
\[ A.C = A \oplus_1 \ldots \oplus_k A \uparrow \]
\[ A.C = A \oplus_1 \ldots A \oplus_k A \]
\[ A.C = \triangleright A \oplus_1 \ldots \oplus_k A \triangleleft \]

Indirectly recursive

\[ A.C = \triangleright B_0 \oplus_1 \ldots \oplus_k B_k \]
\[ A.C = B_0 \oplus_1 \ldots \oplus_k B_k \uparrow \]
\[ A.C = B_0 \oplus_1 \ldots \oplus_k B_k \]
\[ A.C = \triangleright B_0 \oplus_1 \ldots \oplus_k B_k \triangleleft \]
Expression Grammar Hierarchy

context-free grammars

indirectly recursive distfix (Section 7)

overlapping distfix (Section 6.1)

distfix (Section 6)

basic (Section 5)

prefix (Section 4)

infix (Section 3)
Infix Expression Grammars
Infix Expression Grammars

context-free syntax

Exp.Add = Exp "+" Exp {left}
Exp.Sub = Exp "-" Exp {left}
Exp.Mul = Exp "*" Exp {left}
Exp.Pow = Exp "^" Exp {right}
Exp.Eq = Exp "==" Exp {non-assoc}
Exp.Var = ID
Exp = "(" Exp ")" {bracket}

context-free priorities

Exp.Pow > Exp.Mul >
{left: Exp.Add Exp.Sub} > Exp.Eq

\[\text{Exp.Add} = a + [\text{Exp.Sub} = b - c]\]
\[\text{Exp.Sub} = [\text{Exp.Add} = a + b] - c]\]

\[\text{Exp.Add} = [\text{Exp.Add} = a + b] + c]\n\[\text{Exp.Add} = a + [\text{Exp.Add} = b + c]\]

\[\text{Exp.Add} = a + [\text{Exp.Mul} = b * c]\]
\[\text{Exp.Mul} = [\text{Exp.Add} = a + b] * c]\]
context-free syntax

\[
\begin{align*}
\text{Exp.Add} & = \text{Exp} \ "+" \ \text{Term} \\
\text{Exp.Sub} & = \text{Exp} \ "-" \ \text{Term} \\
\text{Exp.Term} & = \text{Term} \\
\text{Term.Mul} & = \text{Term} \ "*" \ \text{Factor} \\
\text{Term.Fact} & = \text{Factor} \\
\text{Factor.Var} & = \text{ID} \\
\text{Factor} & = "(" \ \text{Exp} \ "\)" \ \{\text{bracket}\}
\end{align*}
\]
SDF2 Semantics

\[
\frac{A.C_1 > A.C_2 \in PR}{[A.C_1 = \alpha[A.C_2 = \beta] \gamma] \in Q_G}
\]

\[
\frac{A.C_1 \text{ right } A.C_2 \in PR}{[A.C_1 = [A.C_2 = \beta] \gamma] \in Q_G}
\]

\[
\frac{A.C_1 \text{ left } A.C_2 \in PR}{[A.C_1 = \alpha[A.C_2 = \beta]] \in Q_G}
\]

Exp.\,Mul > Exp.\,Add \in PR

\[
\frac{\text{Exp.\,Mul} = [\text{Exp.\,Add} = \text{Exp} + \text{Exp}] \ast \text{Exp}] \in Q_G}{[\text{Exp.\,Add} = a + [\text{Exp.\,Mul} = b \ast c]]}
\]

\[
[\text{Exp.\,Mul} = [\text{Exp.\,Add} = a + b] \ast c]
\]

Exp.\,Add left Exp.\,Add \in PR

\[
\frac{\text{Exp.\,Add} = \text{Exp} + [\text{Exp.\,Add} = \text{Exp} + \text{Exp}]] \in Q_G}{[\text{Exp.\,Add} = [\text{Exp.\,Add} = a + b] + c]}
\]

\[
[\text{Exp.\,Add} = a + [\text{Exp.\,Add} = b + c]]
\]
Lemma 3.3 (Subtree Exclusion is Safe). Given an infix expression grammar $G$ and a set $Q$ of priority conflict patterns generated by disambiguation rules (not including non-assoc) for $G$, if $w \in L(G)$ then there is a $t \in T^Q(G)$ such that $\text{yield}(t) = w$.

Proof. By induction on the length of sentences in $L(G)$.

(Base case) If $a$ is a lexeme then $a \in T^Q_a(G)$ since disambiguation rules do not exclude lexemes.

(Inductive case) Assume that $u, v \in L(G)$ and that there are $t_1, t_2 \in T^Q_a(G)$ such that $\text{yield}(t_1) = u$, $\text{yield}(t_2) = v$, then there are two cases:

1. If $A.C = \langle A \rangle$ is a closed production in $G$, then $\langle u \rangle \in L(G)$ and $[A.C = \langle t_1 \rangle] \in T^Q_a(G)$, since there is no priority conflict pattern that matches this tree. (Note that the original definition of Visser (1997a) does not restrict priority relations to infix productions. Via Equation 4.2 a priority relation $A.C > A.C'$ for some production $A.C' = \alpha$ in the grammar would lead to rejecting a tree $[A.C = \langle [A.C' = \ldots] \rangle$, and hence the corresponding sentence.)
Subtree Exclusion is Safe

(Inductive case) Assume that \( u, v \in L(G) \) and that there are \( t_1, t_2 \in T^Q_A(G) \) such that \( \text{yield}(t_1) = u \), \( \text{yield}(t_2) = v \), then there are two cases:

(2) If \( A.C = A \oplus A \) is an infix production in \( G \), then \( u \oplus v = w \in L(G) \). Now we need to demonstrate that there is a \( t \in T^Q(G) \) such that \( \text{yield}(t) = w \). By induction \( v = \text{yield}(t_1) \) and \( v = \text{yield}(t_2) \) such that \( t_1, t_2 \in T^Q(G) \). We consider the following cases:

- If \( t_1 \) and \( t_2 \) are lexemes or closed expressions then \( t = [A.C = t_1 \oplus t_2] \in T^Q(G) \) since there are no disambiguation rules that apply.
Subtree Exclusion is Safe

(Inductive case) Assume that \( u, v \in L(G) \) and that there are \( t_1, t_2 \in T_A^Q(G) \) such that \( \text{yield}(t_1) = u \), 
\\( \text{yield}(t_2) = v \), then there are two cases:

(2) If \( A.C = A \oplus A \) is an infix production in \( G \), then \( u \oplus v = w \in L(G) \). Now we need to demonstrate that there is a \( t \in T^Q(G) \) such that \( \text{yield}(t) = w \). By induction \( v = \text{yield}(t_1) \) and \( v = \text{yield}(t_2) \) such that \( t_1, t_2 \in T^Q(G) \). We consider the following cases:

- If \( t_1 = [A.C_1 = t_{11} \otimes t_{12}] \) with yield \( u_{11} \otimes v_{12} \) and \( t_2 = [A.C_2 = \langle t_{21} \rangle] \) with yield \( \langle w_{21} \rangle \). Take \( t = [A.C = [A.C_1 = t_{11} \otimes t_{12}] \oplus [A.C_2 = \langle t_{21} \rangle]] \) as the obvious candidate as tree for \( w \). If \( A.C_1 > A.C \) then \( t \in T^Q(G) \) since it does not match a conflict pattern (since there are no other disambiguation relations between the productions). On the other hand, if \( A.C > A.C_1 \) then \( t \) matches a conflict pattern and therefore \( t \notin T^Q(G) \). However, the reordering \( t' = [A.C_1 = t_{11} \otimes [A.C = t_{12} \oplus [A.C_2 = \langle t_{21} \rangle]]] \) has the same yield and does not have a priority conflict, therefore \( t' \in T^Q(G) \). If \( t_2 \) is a lexeme, or the disambiguation relation is \text{left}, \text{right}, the proof works analogously.
Subtree Exclusion is Safe

(Inductive case) Assume that \( u, v \in L(G) \) and that there are \( t_1, t_2 \in T_A^Q(G) \) such that \( \text{yield}(t_1) = u, \text{yield}(t_2) = v \), then there are two cases:

(2) If \( A.C = A \oplus A \) is an infix production in \( G \), then \( u \oplus v = w \in L(G) \). Now we need to demonstrate that there is a \( t \in T^Q(G) \) such that \( \text{yield}(t) = w \). By induction \( \nu = \text{yield}(t_1) \) and \( \nu = \text{yield}(t_2) \) such that \( t_1, t_2 \in T^Q(G) \). We consider the following cases:

- The proof works analogously when \( t_1 \) is a lexeme or closed expression and \( t_2 \) is an infix expression.
- When both \( t_1 \) and \( t_2 \) are infix expressions we have to consider more cases, but the reasoning is analogous: by the fact that there is at most one disambiguation relation between each pair of operators, we can always construct a non-conflicted tree for the sentence by re-ordering the sub-expressions of \( t_1 \) and \( t_2 \).
Definition 3.4 (Total Set of Disambiguation Rules for Infix Expression Grammars). A set of disambiguation rules $PR$ for an infix expression grammar $G$ is total for a non-terminal $A$:

- If for any pair of productions $A.C_1 = A \ op_1 A \in P(G)$, and $A.C_2 = A \ op_2 A \in P(G)$, such that $A.C_1 \neq A.C_2$, either $A.C_1 \ R \ A.C_2 \in PR$ or $A.C_2 \ R \ A.C_1 \in PR$ where $R \in \{>, \text{right, left}\}$.
- If $A.C = A \ op A \in P(G)$ then $A.C \ R' \ A.C \in PR$ where $R' \in \{\text{right, left, non-assoc}\}$. 

Total Set of Disambiguation Rules
**Lemma 3.5 (Subtree Exclusion is Completely Disambiguating).** Given an infix expression grammar \( G \) and a set \( Q \) of priority conflict patterns generated by a total set of disambiguation rules for \( G \), then all trees in \( T^Q(G) \) have unique yields. That is, if \( t_1, t_2 \in T^Q(G) \) and \( \text{yield}(t_1) = \text{yield}(t_2) \) then \( t_1 = t_2 \).

**Proof.** By induction on \( T^Q(G) \).

*(Base case)* If \( a \) is a lexeme, then \( a \in T^Q_a(G) \) and has a unique yield.

*(Inductive case)* Assume that \( t_1, t_2 \in T^Q_A(G) \) and that their yields are unique.

1. If \( A.C = \langle A \rangle \) is a closed production in \( G \), then \( t = [A.C = \langle t_1 \rangle] \in T^Q_A(G) \), since there is no priority conflict pattern that matches this tree, and the fact that each constructor uniquely identifies a production, by uniqueness of \( t_1 \), \( t \) is also unique.
**Lemma 3.5 (Subtree Exclusion is Completely Disambiguating).** Given an infix expression grammar $G$ and a set $Q$ of priority conflict patterns generated by a total set of disambiguation rules for $G$, then all trees in $T^Q(G)$ have unique yields. That is, if $t_1, t_2 \in T^Q(G)$ and $\text{yield}(t_1) = \text{yield}(t_2)$ then $t_1 = t_2$.

(Inductive case) Assume that $t_1, t_2 \in T^Q_A(G)$ and that their yields are unique.

(2) If $A.C = A \oplus A$ is an infix production in $G$, since each constructor uniquely identifies a production, that is the only way we can construct the tree $t = [A.C = t_1 \oplus t_2]$. Now we need to demonstrate that if $t \in T^Q(G)$ then there is no tree $t' \in T^Q(G)$ such that $t' \neq t$ and $\text{yield}(t) = \text{yield}(t')$. We consider the following cases:
Subtree Exclusion is Complete

(Inductive case) Assume that \( t_1, t_2 \in T_A^Q(G) \) and that their yields are unique.

(2) If \( A.C = A \oplus A \) is an infix production in \( G \), since each constructor uniquely identifies a production, that is the only way we can construct the tree \( t = [A.C = t_1 \oplus t_2] \). Now we need to demonstrate that if \( t \in T_A^Q(G) \) then there is no tree \( t' \in T_A^Q(G) \) such that \( t' \neq t \) and \( \text{yield}(t) = \text{yield}(t') \). We consider the following cases:

- If \( t_1 \) and \( t_2 \) are lexemes or closed expressions then \( t \in T_A^Q(G) \) since there are no disambiguation rules that apply. By uniqueness of \( t_1 \) and \( t_2 \) and non-overlap of productions, there are no other ways to construct a tree with the same yield as \( t \).
(Inductive case) Assume that \( t_1, t_2 \in T^Q_A(G) \) and that their yields are unique.

(2) If \( A.C = A \oplus A \) is an infix production in \( G \), since each constructor uniquely identifies a production, that is the only way we can construct the tree \( t = [A.C = t_1 \oplus t_2] \). Now we need to demonstrate that if \( t \in T^Q(G) \) then there is no tree \( t' \in T^Q(G) \) such that \( t' \neq t \) and \( \text{yield}(t) = \text{yield}(t') \). We consider the following cases:

- If \( t_1 = [A.C_1 = t_{11} \otimes t_{12}] \) with yield \( u \otimes v \) and \( t_2 = [A.C_2 = \langle t_{21} \rangle] \) with yields \( \langle w \rangle \) then \( t = [A.C = [A.C_1 = t_{11} \otimes t_{12}] \oplus [A.C_2 = \langle t_{21} \rangle]] \) with yield \( u \otimes v \oplus \langle w \rangle \). By totality of disambiguation rules, we have that there is a disambiguation relation between \( A.C \) and \( A.C_1 \). If \( A.C > A.C_1 \) then \( t \) matches a conflict pattern and therefore \( t \notin T^Q(G) \). If \( A.C_1 > A.C \) then \( t \) does not match a conflict pattern (since there are no other disambiguation relations between the productions). The only other tree with the same yield is \( t' = [A.C_1 = t_{11} \otimes [A.C = t_{12} \oplus [A.C_2 = \langle t_{21} \rangle]]] \in T^Q(G) \). However, \( t' \) does have a priority conflict and therefore \( t' \notin T^Q(G) \). If the disambiguation relation is \text{left}, \text{right}, \text{or} \text{non-assoc}, the proof works analogously.
Subtree Exclusion is Complete

(Inductive case) Assume that $t_1, t_2 \in T_A^Q(G)$ and that their yields are unique.

(2) If $A.C = A \oplus A$ is an infix production in $G$, since each constructor uniquely identifies a production, that is the only way we can construct the tree $t = [A.C = t_1 \oplus t_2]$. Now we need to demonstrate that if $t \in T_A^Q(G)$ then there is no tree $t' \in T_A^Q(G)$ such that $t' \neq t$ and $\text{yield}(t) = \text{yield}(t')$. We consider the following cases:

- The proof works analogously when $t_1$ is a lexeme or a closed expression and $t_2$ is an infix expression.
- When both $t_1$ and $t_2$ are infix expressions, we have to consider more cases since all combinations of disambiguation relations between the three productions need to be considered, but the reasoning is the same; by totality there are relations between all three productions, and therefore at most one tree is selected.
Theorem 3.6. Disambiguation of an infix expression grammar using a total set of disambiguation rules (not including non-assoc) is safe and completely disambiguating.

Proof. Assume that $G$ is an infix expression grammar and $R$ a total set of disambiguation rules for $G$. Let $Q$ be the set of priority conflict patterns for $R$ according to Definition 3.2. By Lemma 3.3 we have that if $w \in L(G)$ then there is a $t \in T^Q(G)$ such that $\text{yield}(t) = w$. By Corollary 2.17 we have that $F^Q$ is a safe disambiguation filter. By Lemma 3.5 we have that if $t_1, t_2 \in T^Q(G)$ then $\text{yield}(t_1) \neq \text{yield}(t_2) \lor t_1 = t_2$. By Corollary 2.18 we have that $F^Q$ is completely disambiguating. \qed
Deep Priority Conflicts
Prefix Expression Grammars

<table>
<thead>
<tr>
<th>Context-free syntax</th>
<th>(1) [\text{Exp.Minus} = - \ [\text{Exp.Add} = a + b]]</th>
</tr>
</thead>
<tbody>
<tr>
<td>(\text{Exp.Add} \ = \ \text{Exp} \ &quot;+&quot; \ \text{Exp} \ {\text{left}} )</td>
<td>(2) [\text{Exp.Add} = [\text{Exp.Minus} = - a] + b]</td>
</tr>
<tr>
<td>(\text{Exp.Lambda} \ = \ &quot;&quot; \ \text{ID} \ &quot;.&quot; \ \text{Exp} )</td>
<td>(3) [\text{Exp.Add} = a + [\text{Exp.Minus} = - b]]</td>
</tr>
<tr>
<td>(\text{Exp.Minus} \ = \ &quot;-&quot; \ \text{Exp} )</td>
<td></td>
</tr>
<tr>
<td>(\text{Exp.Var} \ = \ \text{ID} )</td>
<td></td>
</tr>
<tr>
<td>(\text{Exp} \ = \ &quot;(&quot; \ \text{Exp} \ &quot;)&quot; \ {\text{bracket}} )</td>
<td></td>
</tr>
</tbody>
</table>

**Context-free priorities**

| \(\text{Exp.Minus} > \text{Exp.Add} > \text{Exp.Lambda}\) |

\[
\text{Exp.Minus} > \text{Exp.Add} \in PR
\]

\[
[\text{Exp.Minus} = - [\text{Exp.Add} = \text{Exp} + \text{Exp}]] \in Q_G
\]
SDF2 Semantics is Unsafe for Prefix Expression Grammars

context-free syntax
Exp.Add = Exp "+" Exp {left}
Exp.Lambda = "\\" ID "." Exp
Exp.Minus = "-" Exp
Exp.Var = ID
Exp = "(" Exp ")" {bracket}

context-free priorities
Exp.Minus > Exp.Add > Exp.Lambda

(4) \[\text{Exp.Add} = [\text{Exp.Lambda} = \backslash x \ . \ a] + b] \]

(5) \[\text{Exp.Lambda} = \backslash x \ . \ [\text{Exp.Add} = a + b]] \]

(6) \[\text{Exp.Add} = a + [\text{Exp.Lambda} = \backslash x \ . \ b]] \]

\[
\begin{align*}
\text{Exp.Add} & > \text{Exp.Lambda} \in PR \\
\text{[Exp.Add} & = [\text{Exp.Lambda} = \backslash \text{ID} \ . \ \text{Exp}] + \text{Exp]} \in Q_G \\
\text{Exp.Add} & > \text{Exp.Lambda} \in PR \\
\text{[Exp.Add} & = \text{Exp} + [\text{Exp.Lambda} = \backslash \text{ID} \ . \ \text{Exp}]] \in Q_G
\end{align*}
\]
Safe Semantics

**context-free syntax**

\[
\begin{align*}
\text{Exp.Add} & = \text{Exp } "\text{+}" \text{ Exp} \{\text{left}\} \\
\text{Exp.Lambda} & = "\text{\textbackslash}" \text{ ID } "." \text{ Exp} \\
\text{Exp.Minus} & = "\text{-}" \text{ Exp} \\
\text{Exp.Var} & = \text{ ID} \\
\text{Exp} & = "(" \text{ Exp } ")" \{\text{bracket}\}
\end{align*}
\]

**context-free priorities**

\[
\text{Exp.Minus} > \text{Exp.Add} > \text{Exp.Lambda}
\]

(4) \[\text{Exp.Add} = [\text{Exp.Lambda} = \text{\textbackslash} \text{ x } . \text{ a}] + \text{ b}]

(5) \[\text{Exp.Lambda} = \text{\textbackslash} \text{ x } . \text{ [Exp.Add = a + b]}\]

(6) \[\text{Exp.Add} = \text{ a} + [\text{Exp.Lambda} = \text{\textbackslash} \text{ x } . \text{ b}]\]

\[
\begin{align*}
A.C_1 > A.C_2 & \in PR \\
[\alpha \in Q_G^\text{safe}] \\
A.C_1 > A.C_2 & \in PR \\
[\alpha \in Q_G^\text{safe}]
\end{align*}
\]

\[
\begin{align*}
\text{Exp.Add} > \text{Exp.Lambda} & \in PR \\
[\text{Exp.Add} = [\text{Exp.Lambda} = \text{\textbackslash} \text{ ID } . \text{ Exp}]+\text{ Exp}] & \in Q_G^\text{safe}
\end{align*}
\]
Safe Semantics for Shallow Conflicts

\[
\frac{A.C_1 > A.C_2 \in PR}{[A.C_1 = [A.C_2 = \alpha A]_\gamma] \in Q^\text{safe}_G}
\]

\[
A.C_1 > A.C_2 \in PR
\]

\[
[A.C_1 = \alpha[A.C_2 = A\gamma]] \in Q^\text{safe}_G
\]

\[
A.C_1 \text{ right } A.C_2 \in PR
\]

\[
[A.C_1 = [A.C_2 = A\beta_2 A]_\beta_1 A] \in Q^\text{safe}_G
\]

\[
A.C_1 \text{ left } A.C_2 \in PR
\]

\[
[A.C_1 = A\beta_1 [A.C_2 = A\beta_2 A]] \in Q^\text{safe}_G
\]

\[
A.C_1 \text{ non-assoc } A.C_2 \in PR
\]

\[
[A.C_1 = A\beta_1 [A.C_2 = A\beta_2 A]] \in Q^\text{safe}_G
\]

\[
A.C_1 \text{ non-assoc } A.C_2 \in PR
\]

\[
[A.C_1 = [A.C_2 = A\beta_2 A]_\beta_1 A] \in Q^W_G
\]

\[
A.C_1 \text{ non-nested } A.C_2 \in PR \quad \neg(\alpha_i \Rightarrow^* Ay)
\]

\[
[A.C_1 = \alpha_1 [A.C_2 = \alpha_2 A]] \in Q^W_G
\]
Deep Priority Conflicts

class context-free syntax
Exp.Add = Exp "+" Exp {left}
Exp.Lambda = "\" ID "." Exp
Exp.Minus = "-" Exp
Exp.Var = ID
Exp = "(" Exp ")" {bracket}

class context-free priorities
Exp.Minus > Exp.Add > Exp.Lambda

\[
\begin{align*}
a + \ \lambda x. \ b + c
\end{align*}
\]
Rightmost Deep Matching

\[ \forall 0 \leq i \leq n : M^{rm}(t_i, q_i) \]
\[ D^{rm}([A.C = t_1...t_n], [A.C = q_1...q_n]) \]

\[ \frac{M(t, q)}{M^{rm}(t, q)} \]

\[ M^{rm}(t_n, q) \]
\[ \frac{M^{rm}(t_n, q)}{M^{rm}([A.C = t_1...t_n], q)} \]

\[ t : [\text{Exp.Add} = [\text{Exp.Add} = a + [\text{Exp.Lambda} = \\backslash x . b]] + c] \]
\[ q : [\text{Exp.Add} = [\text{Exp.Lambda} = \\backslash \text{ID} . \text{Exp}] + \text{Exp}] \]
Rightmost Deep Matching

\[ t: \text{[Exp.Add} = \text{Exp.Add} = a + \text{[Exp.Lambda} = \lambda x . b]\text{]} + c\]  
\[ q: \text{[Exp.Add} = \text{Exp.Lambda} = \lambda \text{ID . Exp}] + \text{Exp}\]  

\[ M^{rm}(\text{[Exp.Lambda} = \lambda x . b], \text{[Exp.Lambda} = \lambda \text{ID . Exp}]) \]
\[ M^{rm}(\text{[Exp.Add} = a + \text{[Exp.Lambda} = \lambda x . b]\text{]}, \text{[Exp.Lambda} = \lambda \text{ID . Exp}]) \]
\[ D^{rm}(t, q) \]
Rightmost Deep Priority Conflict Pattern

\[
\begin{align*}
A.C_1 &> A.C_2 \in PR \\
\Rightarrow [A.C_1 = [A.C_2 = \alpha A]\gamma] \in Q_{G}^{safe} \\
A.C_1 &> A.C_2 \in PR \\
\Rightarrow [A.C_1 = \alpha[A.C_2 = A\gamma]] \in Q_{G}^{safe}
\end{align*}
\]
Etc.