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

TU Delft

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

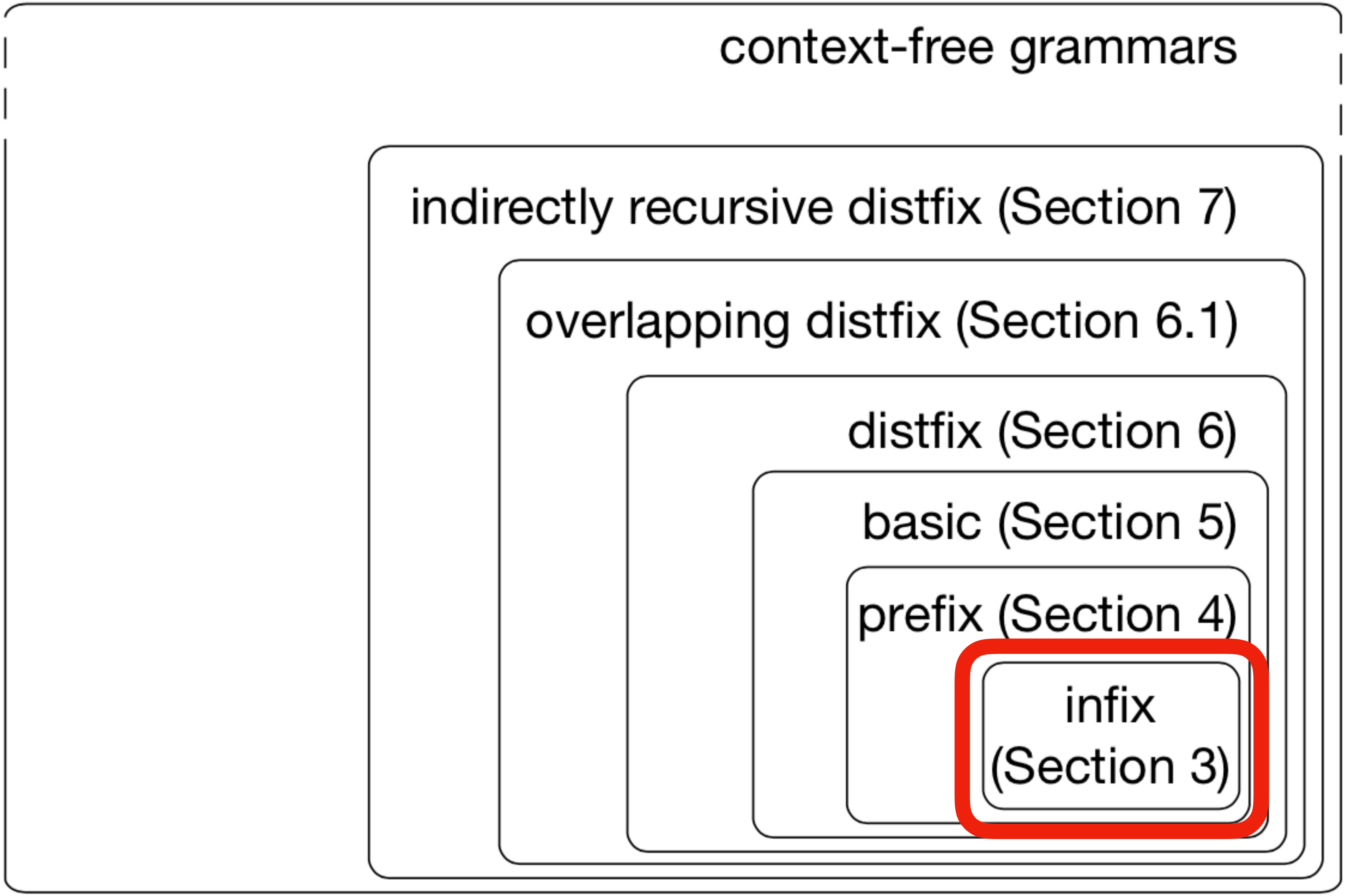
- Preserve language and solve all ambiguities
- Proof: induction on trees under subtree exclusion

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 $t_1, t_2 \in T_A^Q(G)$ 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 $t \in T^Q(G)$ 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 $t_2 = [A.C_2 = \triangleleft t_{21} \triangleright]$ with yields $\triangleleft w \triangleright$ then $t = [A.C = [A.C_1 = t_{11} \otimes t_{12}] \oplus [A.C_2 = \triangleleft t_{21} \triangleright]]$ with yield $u \otimes v \oplus \triangleleft w \triangleright$. 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 $t' = [A.C_1 = t_{11} \otimes [A.C = t_{12} \oplus [A.C_2 = \triangleleft t_{21} \triangleright]]] \in T^Q(G)$. 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

TU Delft

Grammars, Well-Formed Trees, Languages

```
context-free syntax
  Exp.Add = Exp "+" Exp
  Exp.Sub = Exp "-" Exp
  Exp.Mul = Exp "*" Exp
  Exp.Var = ID
```

$$\frac{a \in \Sigma}{a \in T^a(G)}$$

$$\frac{A.C = X_1 \ldots X_n \in P(G) \quad t_i \in T^{X_i}(G) \quad 1 \leq i \leq n}{[A.C = t_1 \ldots t_n] \in T^A(G)}$$

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

```
[Exp.Add = [Exp.Var = ID] + [Exp.Var = ID]]
```

Parsing

$$\boxed{\Pi(G)(w) = \{t \in T^X(G) \mid yield(t) = w, X \in V\}}$$

Derivations

$$\frac{\alpha = \lambda A \rho \qquad \beta = \lambda \gamma \rho \qquad A.C = \gamma \in P(G)}{\alpha \Rightarrow_G \beta}$$

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

```
context-free syntax
  Exp.Add = Exp "+" Exp
  Exp.Sub = Exp "-" Exp
  Exp.Mul = Exp "*" Exp
  Exp.Var = ID
```

```
signature
  constructors
    Add : Exp * Exp -> Exp
    Sub : Exp * Exp -> Exp
    Mul : Exp * Exp -> Exp
    Var : ID -> Exp
```

```
[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

$$\frac{X \in V}{X \in TP^{X}(G)}$$

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

$$\frac{a \in \Sigma}{\mathcal{M}(a, a)}$$

$$\frac{[A.C = t_1 \ldots t_n] \in T^{A}(G)}{\mathcal{M}([A.C = t_1 \ldots t_n], A)}$$

$$\frac{[A.C = t_1 \ldots t_n] \in T^{A}(G) \quad [A.C = q_1 \ldots q_n] \in TP^{A}(G) \quad \mathcal{M}(t_i, q_i) \quad 1 \leq i \leq n}{\mathcal{M}([A.C = t_1 \ldots t_n], [A.C = q_1 \ldots q_n])}$$

Tree Patterns and Pattern Matching: Example

```
[Exp.Add = [Exp.Add = [Exp.Var = ID] + [Exp.Var = ID]] + [Exp.Var = ID]]
```

```
[Exp.Add = [Exp.Add = Exp + Exp] + Exp]
```

Ambiguity

```
context-free syntax
  Exp.Add = Exp "+" Exp
  Exp.Sub = Exp "-" Exp
  Exp.Mul = Exp "*" Exp
  Exp.Var = ID
```

```
a + b + c
```

$$\text{(i)}\quad \underline{Exp} \Rightarrow_G \underline{Exp} + Exp \Rightarrow_G a + \underline{Exp} \Rightarrow_G a + Exp + Exp \overset{*}{\underset{lm}{\Longrightarrow}}_G a + b + c$$

$$\text{(ii)}\quad \underline{Exp} \Rightarrow_G \underline{Exp} + Exp \Rightarrow_G Exp + Exp + Exp \overset{*}{\underset{lm}{\Longrightarrow}}_G a + b + c$$

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

Explicit Disambiguation (Brackets)

```
context-free syntax
  Exp.Add = Exp "+" Exp
  Exp.Sub = Exp "-" Exp
  Exp.Mul = Exp "*" Exp
  Exp.Var = ID
  Exp     = "(" Exp ")" {bracket}
```

```
a * (b + c)
```

```
[Exp.Mul = a * [Exp = ([Exp.Add = b + c])]]
```

```
Mul(a, Add(b, c))
```

Disambiguation Filter

$$\boxed{F(\Phi) \subseteq \Phi \text{ for any } \Phi \subseteq T(G)}$$

$$\boxed{L(G/F) = \{w \in \Sigma^* \mid \exists \Phi \subseteq T(G),\ \mathit{yield}(\Phi) = \{w\},\ F(\Phi) = \Phi\}}$$

Subtree Exclusion Filter

$$\boxed{F^{Q}(\Phi) = \{t \in \Phi \;\mid \nexists t' \in sub(t) : \mathcal{M}(t', Q)\}}$$

Trees under Subtree Exclusion

$$\frac{a \in \Sigma \quad \neg\mathcal{M}(a, Q)}{a \in T_a^Q(G)}$$

$$\frac{A.C = X_1 \ldots X_n \in P(G) \quad t_i \in T_{X_i}^Q(G) \text{ for } 1 \leq i \leq n \quad t = [A.C = t_1 \ldots t_n] \quad \neg\mathcal{M}(t, Q)}{t \in T_A^Q(G)}$$

$$\boxed{t \in T_X^Q(G) \iff t \in T_X(G) \land t \in F^Q(\{t\})}$$

$$\boxed{L(G/F^Q) = L(G^Q)}$$

Safety and Completeness

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 $yield(t) = w$.*

COROLLARY 2.18. *A subtree exclusion filter for a set of patterns Q for a grammar G is completely disambiguating if $t_1, t_2 \in T^Q(G) \implies yield(t_1) \neq yield(t_2) \vee t_1 = t_2$*

Expression Grammars

Embedded Expression Grammars

```
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-assoc}
  Expr.Call   = Expr "." ID "(" {Expr ","}* ")"
context-free priorities
  Expr.Call > Expr.Add > Expr.Eq
```

Classes of Expression Grammars

$$A.C = LEX$$
$$A.C = \triangleright\ A \triangleleft$$
$$A.C = A \ \oplus \ A$$
$$A.C = \blacktriangleright\ A$$
$$A.C = A \ \blacktriangleleft$$

Basic

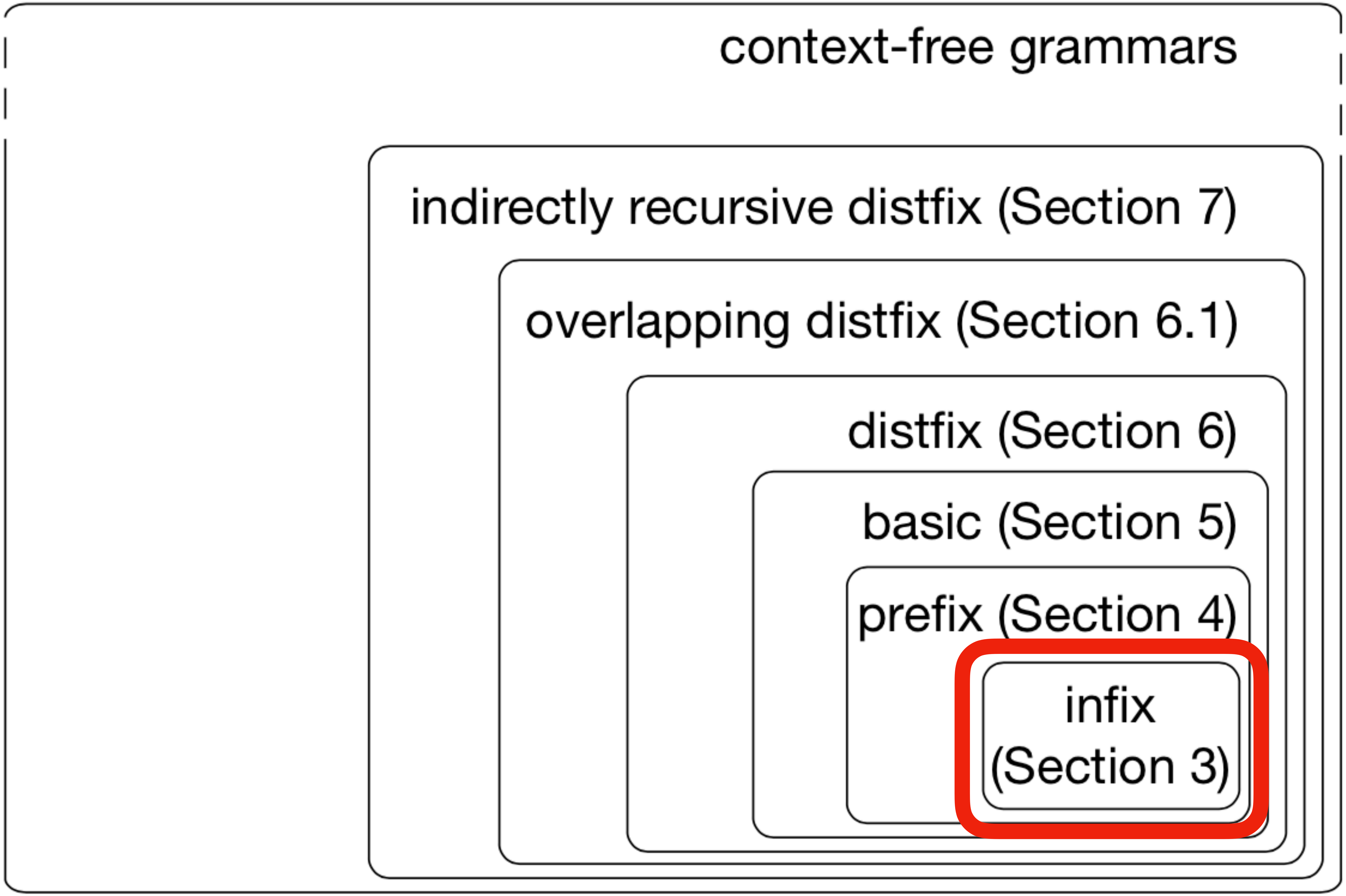
$$A.C = \blacktriangleright \ \ A \ \oplus_1 \ \ \ldots \ \ \oplus_k \ \ A$$
$$A.C = A \ \oplus_1 \ \ \ldots \ \ \oplus_k \ \ A \ \ \blacktriangleleft$$
$$A.C = A \ \oplus_1 \ \ \ldots A \ \oplus_k \ \ A$$
$$A.C = \triangleright\, A \ \oplus_1 \ \ \ldots \ \ \oplus_k \ \ A \triangleleft$$

Distfix

$$A.C = \blacktriangleright \ \ B_0 \ \oplus_1 \ \ \ldots \ \ \oplus_k \ \ B_k$$
$$A.C = B_0 \ \oplus_1 \ \ \ldots \ \ \oplus_k \ \ B_k \ \ \blacktriangleleft$$
$$A.C = B_0 \ \oplus_1 \ \ \ldots \ \ \oplus_k \ \ B_k$$
$$A.C = \triangleright\, B_0 \ \oplus_1 \ \ \ldots \ \ \oplus_k \ \ B_k \triangleleft$$

Indirectly recursive

Expression Grammar Hierarchy



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
```

```
[Exp.Add = a + [Exp.Sub = b - c]]
[Exp.Sub = [Exp.Add = a + b] - c]
```

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

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

Grammar Rewriting

```
context-free syntax
  Exp.Add     = Exp "+" Term
  Exp.Sub     = Exp "-" Term
  Exp.Term    = Term
  Term.Mul    = Term "*" Factor
  Term.Fact   = Factor
  Factor.Var  = ID
  Factor      = "(" Exp ")" {bracket}
```

SDF2 Semantics

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

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

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

$$\frac{A.C_1 \ \texttt{non-assoc} \ A.C_2 \in PR}{[A.C_1 = [A.C_2 = \beta]\gamma] \in Q_G}$$

$$\frac{A.C_1 \ \texttt{non-assoc} \ A.C_2 \in PR}{[A.C_1 = \alpha[A.C_2 = \beta]] \in Q_G}$$

$$\frac{\texttt{Exp.Mul} > \texttt{Exp.Add} \in PR}{\texttt{[Exp.Mul = [Exp.Add = Exp + Exp] * Exp]} \in Q_G}$$

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

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

$$\frac{\texttt{Exp.Add left Exp.Add} \in PR}{\texttt{[Exp.Add = Exp + [Exp.Add = Exp + Exp]]} \in Q_G}$$

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

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

Subtree Exclusion is Safe

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 $yield(t) = w$.*

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 $yield(t_1) = u$, $yield(t_2) = v$, then there are two cases:

(1) If $A.C = \triangleleft\, A\, \triangleright$ is a closed production in G, then $\triangleleft\, u\, \triangleright \in L(G)$ and $[A.C = \triangleleft\, t_1\, \triangleright] \in T^Q_A(G)$, 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 $[A.C = \triangleleft\, [A.C' = \ldots]\, \triangleright]$, and hence the corresponding sentence.)

Subtree Exclusion is Safe

(Inductive case) Assume that $u, v \in L(G)$ and that there are $t_1, t_2 \in T_A^Q(G)$ such that $yield(t_1) = u$, $yield(t_2) = v$, 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 $yield(t) = w$. By induction $v = yield(t_1)$ and $v = yield(t_2)$ 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 $t_1, t_2 \in T^Q_A(G)$ such that $yield(t_1) = u$, $yield(t_2) = v$, 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 $yield(t) = w$. By induction $v = yield(t_1)$ and $v = yield(t_2)$ 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 $t_2 = [A.C_2 = \triangleleft t_{21} \triangleright]$ with yield $\triangleleft w_{21} \triangleright$. Take $t = [A.C = [A.C_1 = t_{11} \otimes t_{12}] \oplus [A.C_2 = \triangleleft t_{21} \triangleright]]$ 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 $t' = [A.C_1 = t_{11} \otimes [A.C = t_{12} \oplus [A.C_2 = \triangleleft t_{21} \triangleright]]]$ 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 `left`, `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 $yield(t_1) = u$, $yield(t_2) = v$, 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 $yield(t) = w$. By induction $v = yield(t_1)$ and $v = yield(t_2)$ 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. $\square$

Total Set of Disambiguation Rules

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 $R \in \{>, \texttt{right}, \texttt{left}\}$.
- If $A.C = A\ op\ A \in P(G)$ then $A.C\ R'\ A.C \in PR$ where $R' \in \{\texttt{right}, \texttt{left}, \texttt{non-assoc}\}$.

Subtree Exclusion is Complete

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 $yield(t_1) = yield(t_2)$ 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 $A.C = \triangleleft\ A\ \triangleright$ is a closed production in G, then $t = [A.C = \triangleleft\ t_1\ \triangleright] \in T^Q_A(G)$, 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.

Subtree Exclusion is Complete

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 $yield(t_1) = yield(t_2)$ 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 $yield(t) = yield(t')$. 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 $t \in T^Q(G)$ 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 and t_2 are lexemes or closed expressions then $t \in T^Q(G)$ 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.

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 $t_2 = [A.C_2 = \triangleleft t_{21} \triangleright]$ with yields $\triangleleft w \triangleright$ then $t = [A.C = [A.C_1 = t_{11} \otimes t_{12}] \oplus [A.C_2 = \triangleleft t_{21} \triangleright]]$ with yield $u \otimes v \oplus \triangleleft w \triangleright$. 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 $t' = [A.C_1 = t_{11} \otimes [A.C = t_{12} \oplus [A.C_2 = \triangleleft t_{21} \triangleright]]] \in T^Q(G)$. 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.

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 $t \in T^Q(G)$ 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:

- 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. $\square$

Disambiguation for Infix Expression is Safe and Complete

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 $yield(t) = w$. 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 $yield(t_1) \neq yield(t_2) \vee t_1 = t_2$. By Corollary 2.18 we have that F^Q is completely disambiguating. □

Deep Priority Conflicts

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
```

```
(1) [Exp.Minus = - [Exp.Add = a + b]]
(2) [Exp.Add = [Exp.Minus = - a] + b]
```

```
(3) [Exp.Add = a + [Exp.Minus = - b]]
```

$$\frac{\texttt{Exp.Minus} > \texttt{Exp.Add} \in PR}{\texttt{[Exp.Minus = - [Exp.Add = Exp + Exp]]} \in Q_G}$$

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) [Exp.Add = [Exp.Lambda = \ x . a] + b]
(5) [Exp.Lambda = \ x .  [Exp.Add = a + b]]
```

```
(6) [Exp.Add = a + [Exp.Lambda = \ x . b]]
```

$$\frac{\texttt{Exp.Add} > \texttt{Exp.Lambda} \in PR}{\texttt{[Exp.Add = [Exp.Lambda = \textbackslash\ ID . Exp] + Exp]} \in Q_G}$$

$$\frac{\texttt{Exp.Add} > \texttt{Exp.Lambda} \in PR}{\texttt{[Exp.Add = Exp + [Exp.Lambda = \textbackslash\ ID . Exp]]} \in Q_G}$$

Safe Semantics

```
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
```

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

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

```
(4) [Exp.Add = [Exp.Lambda = \ x . a] + b]
(5) [Exp.Lambda = \ x .  [Exp.Add = a + b]]
```

```
(6) [Exp.Add = a + [Exp.Lambda = \ x . b]]
```

$$\frac{\texttt{Exp.Add} > \texttt{Exp.Lambda} \in PR}{[\texttt{Exp.Add = [Exp.Lambda = \textbackslash{} ID . Exp] + Exp}] \in Q_G^{safe}}$$

Safe Semantics for Shallow Conflicts

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

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

$$\frac{A.C_1 \texttt{ right } A.C_2 \in PR}{[A.C_1 = [A.C_2 = A\beta_2 A]\beta_1 A] \in Q_G^{safe}}$$

$$\frac{A.C_1 \texttt{ left } A.C_2 \in PR}{[A.C_1 = A\beta_1[A.C_2 = A\beta_2 A]] \in Q_G^{safe}}$$

$$\frac{A.C_1 \texttt{ non-assoc } A.C_2 \in PR}{[A.C_1 = A\beta_1[A.C_2 = A\beta_2 A]] \in Q_G^{safe}}$$

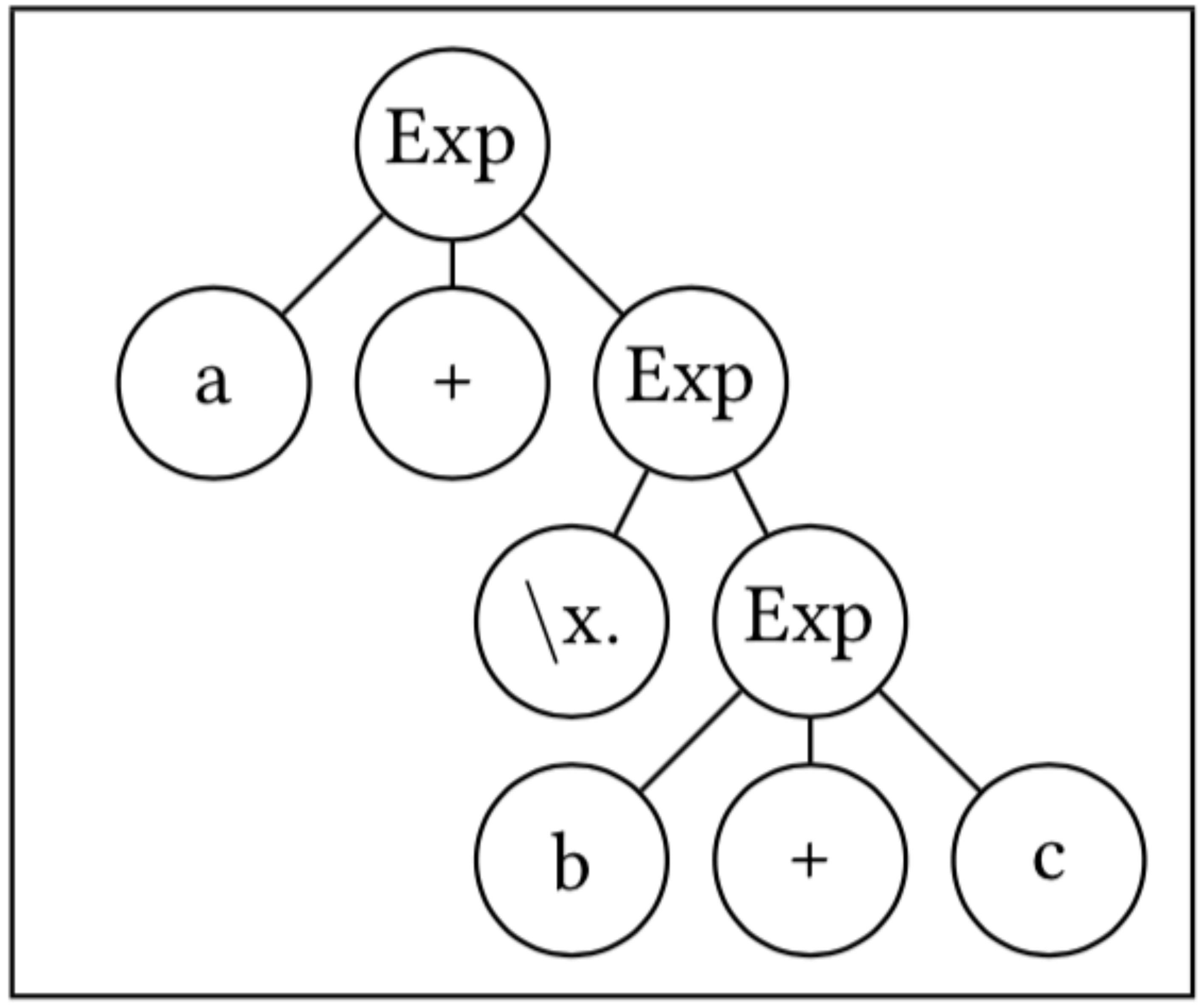
$$\frac{A.C_1 \texttt{ non-assoc } A.C_2 \in PR}{[A.C_1 = [A.C_2 = A\beta_2 A]\beta_1 A] \in Q_G^{W}}$$

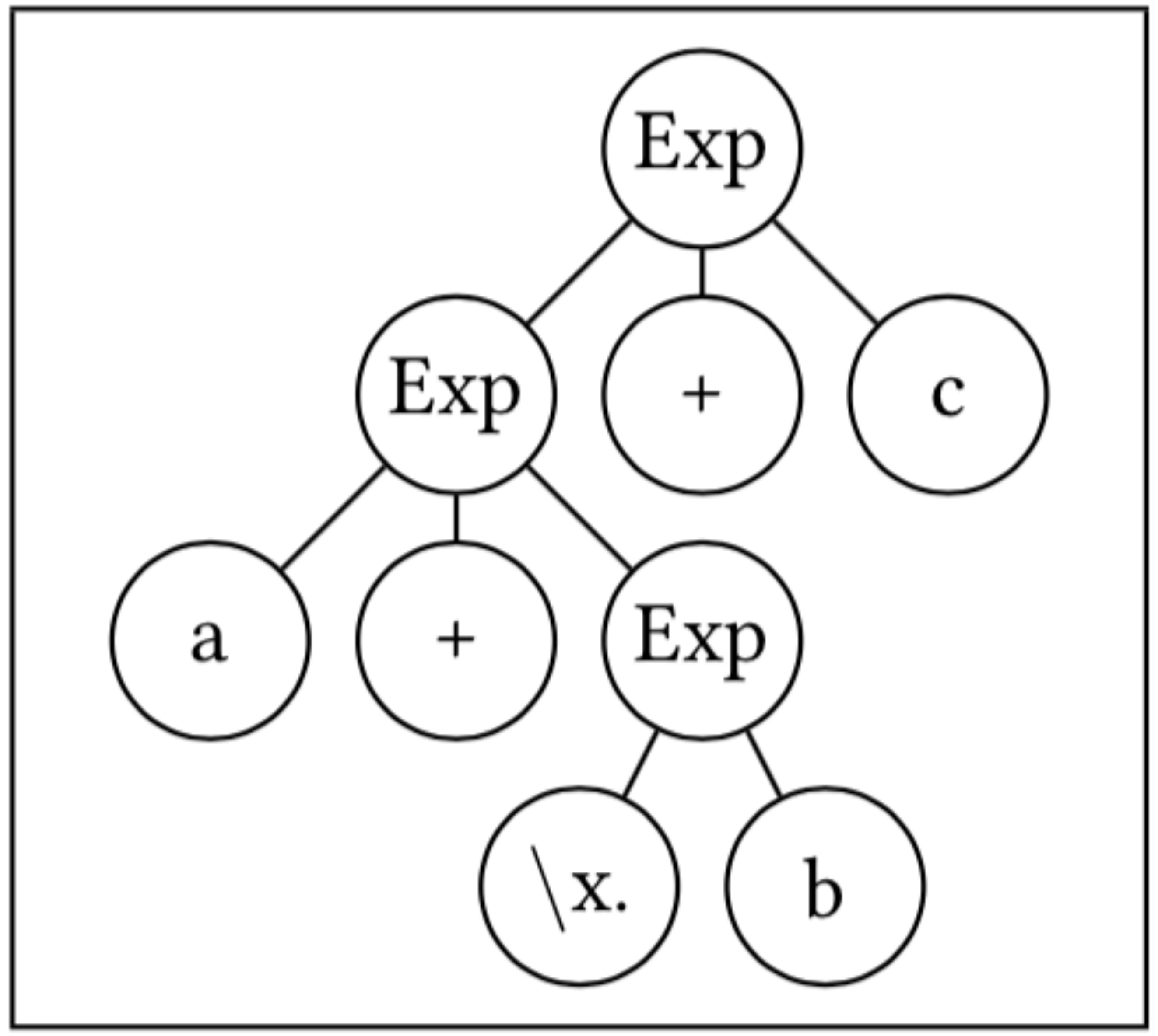
$$\frac{A.C_1 \texttt{ non-nested } A.C_2 \in PR \quad \neg(\alpha_i \Rightarrow^* A\gamma)}{[A.C_1 = \alpha_1[A.C_2 = \alpha_2 A]] \in Q_G^{W}}$$

Deep Priority Conflicts

```
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
```

```
a + \x. b + c
```





Rightmost Deep Matching

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

$$\frac{\mathcal{M}(t, q)}{\mathcal{M}^{rm}(t, q)}$$

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

t : `[Exp.Add = [Exp.Add = a + [Exp.Lambda = \ x . b]] + c]`

q : `[Exp.Add = [Exp.Lambda = \ ID . Exp] + Exp]`

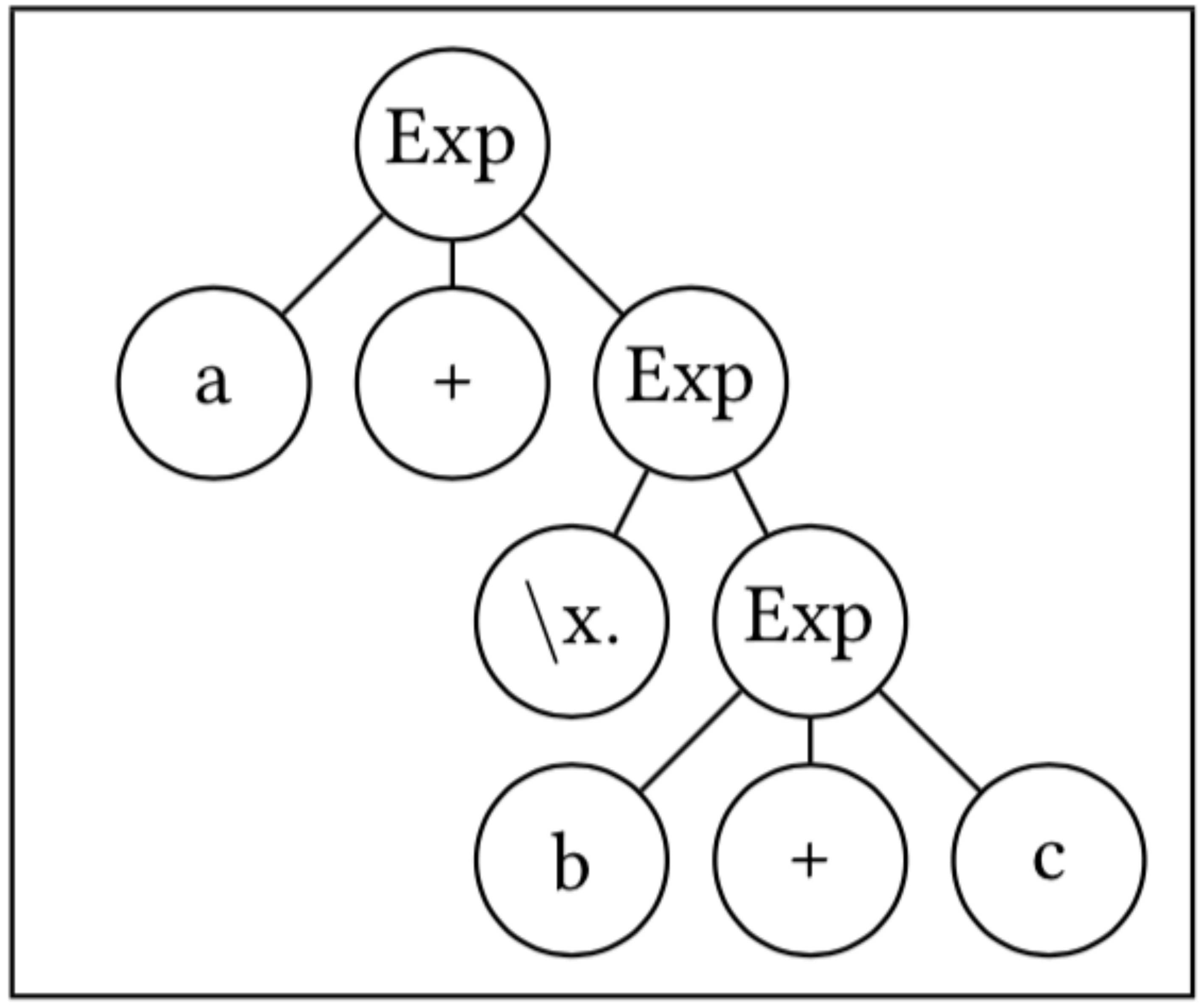
Rightmost Deep Matching

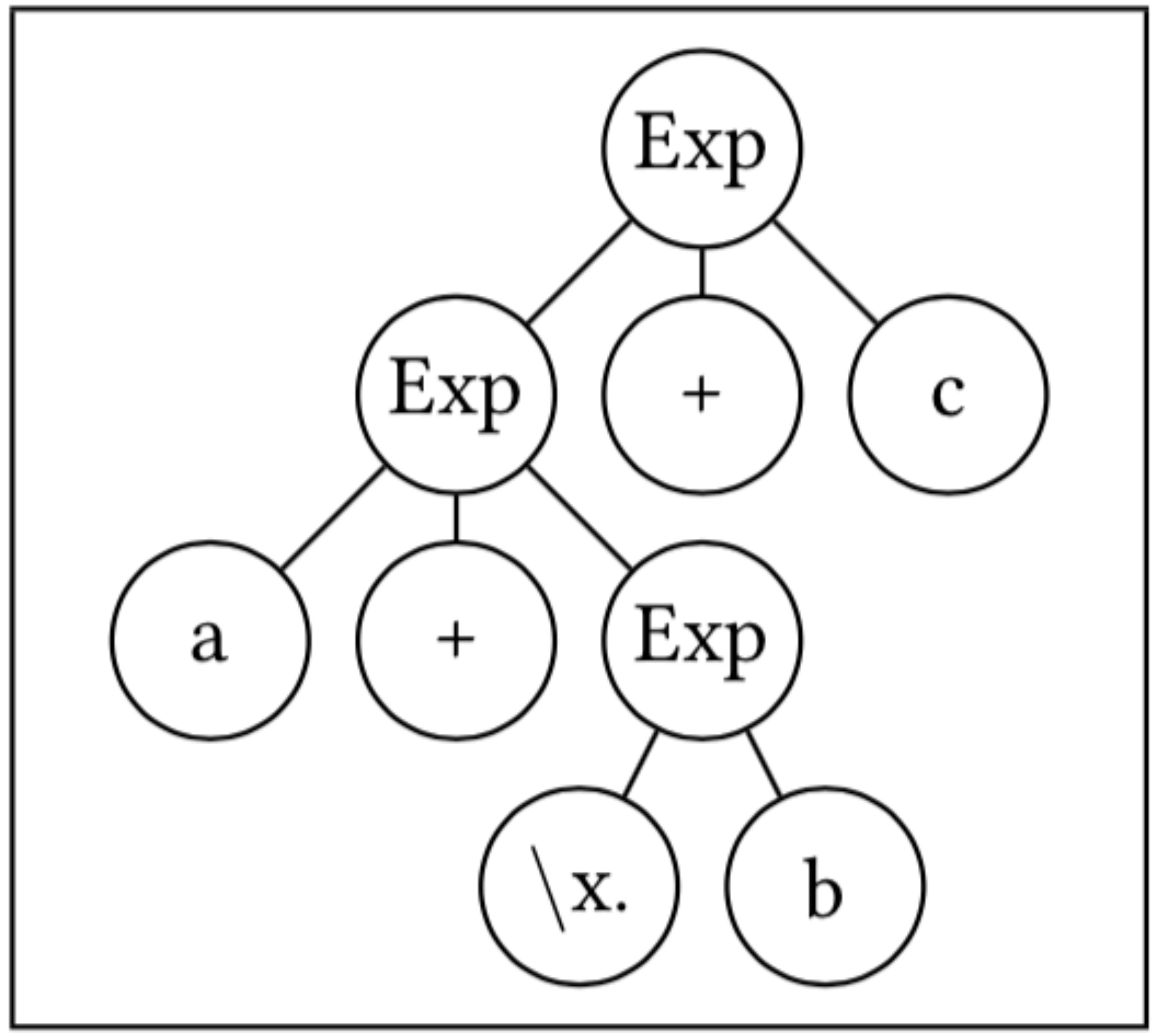
t : `[Exp.Add = [Exp.Add = a + [Exp.Lambda = \ x . b]] + c]`

q : `[Exp.Add = [Exp.Lambda = \ ID . Exp] + Exp]`

$$\frac{\dfrac{\mathcal{M}^{rm}(\texttt{[Exp.Lambda = \textbackslash\ x . b]}, \texttt{[Exp.Lambda = \textbackslash\ ID . Exp]})}{\mathcal{M}^{rm}(\texttt{[Exp.Add = a + [Exp.Lambda = \textbackslash\ x . b]]}, \texttt{[Exp.Lambda = \textbackslash\ ID . Exp]})}}{\mathcal{D}^{rm}(t, q)}$$

Rightmost Deep Priority Conflict Pattern





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

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

$$\frac{A.C_2 > A.C_1 \in PR \qquad \alpha \overset{*}{\not\Rightarrow}_G A\beta}{[A.C_2 = [A.C_1 = \alpha A]\gamma] \in Q_G^{rm}}$$

Etc.