Declarative Disambiguation of Deep Priority Conflicts

Eduardo Souza, Timothee Haudebourg, Michael Steindorfer, Eelco Visser

WG2.11 Kyoto
June 4, 2018

Python Expression Syntax

```
star_expr: '*' expr
expr: xor_expr ('|' xor_expr)*
xor_expr: and_expr ('^' and_expr)*
and_expr: shift_expr ('&' shift_expr)*
shift_expr: arith_expr (('<<'|'>>') arith_expr)*
arith_expr: term (('+'|'-') term)*
term: factor (('*'|'@'|'/'|'%'|'//') factor)*
factor: ('+'|'-'|'~') factor | power
power: atom_expr ['**' factor]
atom_expr: [AWAIT] atom trailer*
atom: ('(' [yield_expr|testlist_comp] ')' |
       '[' [testlist_comp] ']' |
       '{' [dictorsetmaker] '}' |
       NAME | NUMBER | STRING+ | '...' | 'None' | 'True' | 'False')
```

https://docs.python.org/3/reference/grammar.html

Associativity and Precedence in YACC

```
%right  '='
%left  '+'  '-'
%left  '*'  '/'

%%

expr    :       expr  '='  expr
        |       expr  '+'  expr
        |       expr  '-'  expr
        |       expr  '*'  expr
        |       expr  '/'  expr
        |       NAME
        ;
```

Semantics of Disambiguation in YACC

The precedences and associativities are used by Yacc to resolve parsing conflicts; they give rise to disambiguating rules. Formally, the rules work as follows:

1. The precedences and associativities are recorded for those tokens and literals that have them.

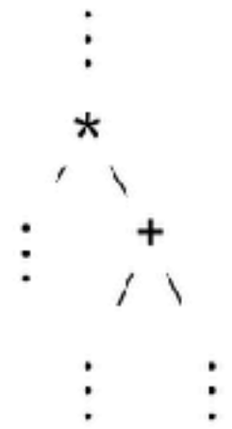
2. A precedence and associativity is associated with each grammar rule; it is the precedence and associativity of the last token or literal in the body of the rule. If the %prec construction is used, it overrides this default. Some grammar rules may have no precedence and associativity associated with them.

3. When there is a reduce/reduce conflict, or there is a shift/reduce conflict and either the input symbol or the grammar rule has no precedence and associativity, then the two disambiguating rules given at the beginning of the section are used, and the conflicts are reported.

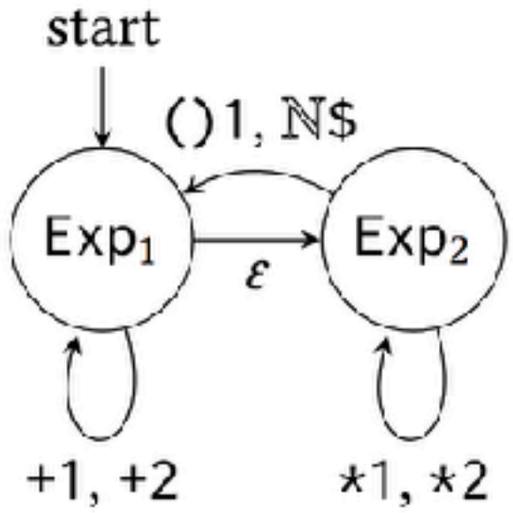
4. If there is a shift/reduce conflict, and both the grammar rule and the input character have precedence and associativity associated with them, then the conflict is resolved in favor of the action (shift or reduce) associated with the higher precedence. If the precedences are the same, then the associativity is used; left associative implies reduce, right associative implies shift, and nonassociating implies error.

Disambiguation with Tree Automata

(a) Violating pattern



(b) Violating pattern



(c) Automaton

```
((+(□, □))* .
 (*(□, □))* .
 (ℕ() | ()(□)))*
```

(d) Regular expression

$$\begin{aligned}
\mathsf{Exp}_1 &\rightarrow_{+} \quad \mathsf{Exp}_1 \ \mathsf{Exp}_1 \\
\mathsf{Exp}_1 &\rightarrow_{\epsilon} \quad \mathsf{Exp}_2 \\
\\
\mathsf{Exp}_2 &\rightarrow_{\mathbb{N}} \quad \epsilon \\
\mathsf{Exp}_2 &\rightarrow_{*} \quad \mathsf{Exp}_2 \ \mathsf{Exp}_2 \\
\mathsf{Exp}_2 &\rightarrow_{()} \quad \mathsf{Exp}_1
\end{aligned}$$

(e) Tree rewrites

[Adams & Might, OOPSLA 2017]

What is the *direct* semantics of associativity and priority?

SDF2 Semantics

[Visser 1997]

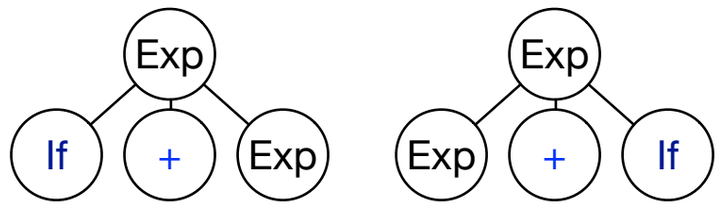
Declarative Disambiguation in SDF3

```
context-free syntax

  Exp.Add = Exp "+" Exp {left}
  Exp.Mul = Exp "*" Exp {left}
  Exp.Int = INT

context-free priorities

  Exp.Mul > Exp.Add
```

SDF2 Semantics

$$\frac{A.C_1 = \alpha B \gamma > B.C_2 = \beta \in \mathsf{Pr}(G)}{[A.C_1 = \alpha[C_2]\gamma] \in Q_G}$$

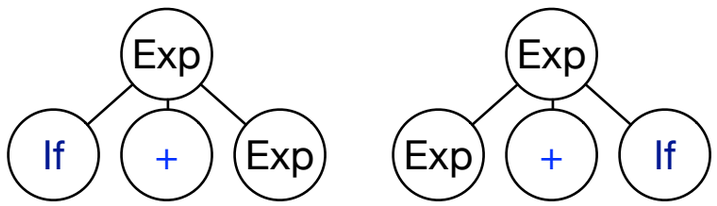
```
context-free syntax

  Exp.Add = Exp "+" Exp
  Exp.If  = "if" "(" Exp ")" Exp

context-free priorities

  Exp.Add > Exp.If
```

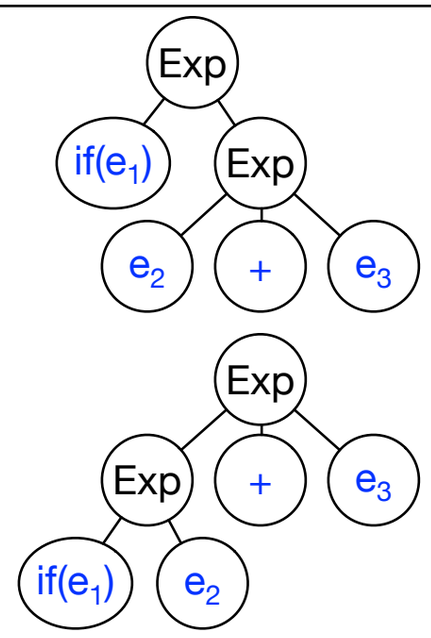


SDF2 Semantics

$$\frac{A.C_1 = \alpha B \gamma > B.C_2 = \beta \in \mathsf{Pr}(G)}{[A.C_1 = \alpha[C_2]\gamma] \in Q_G}$$

```
context-free syntax

  Exp.Add = Exp "+" Exp
  Exp.If  = "if" "(" Exp ")" Exp

context-free priorities

  Exp.Add > Exp.If
```

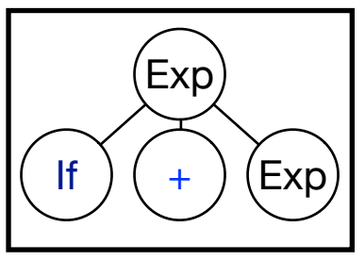


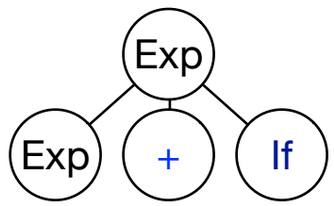
	Exp.Add
	if(e_1) e_2 + e_3
Exp.If	

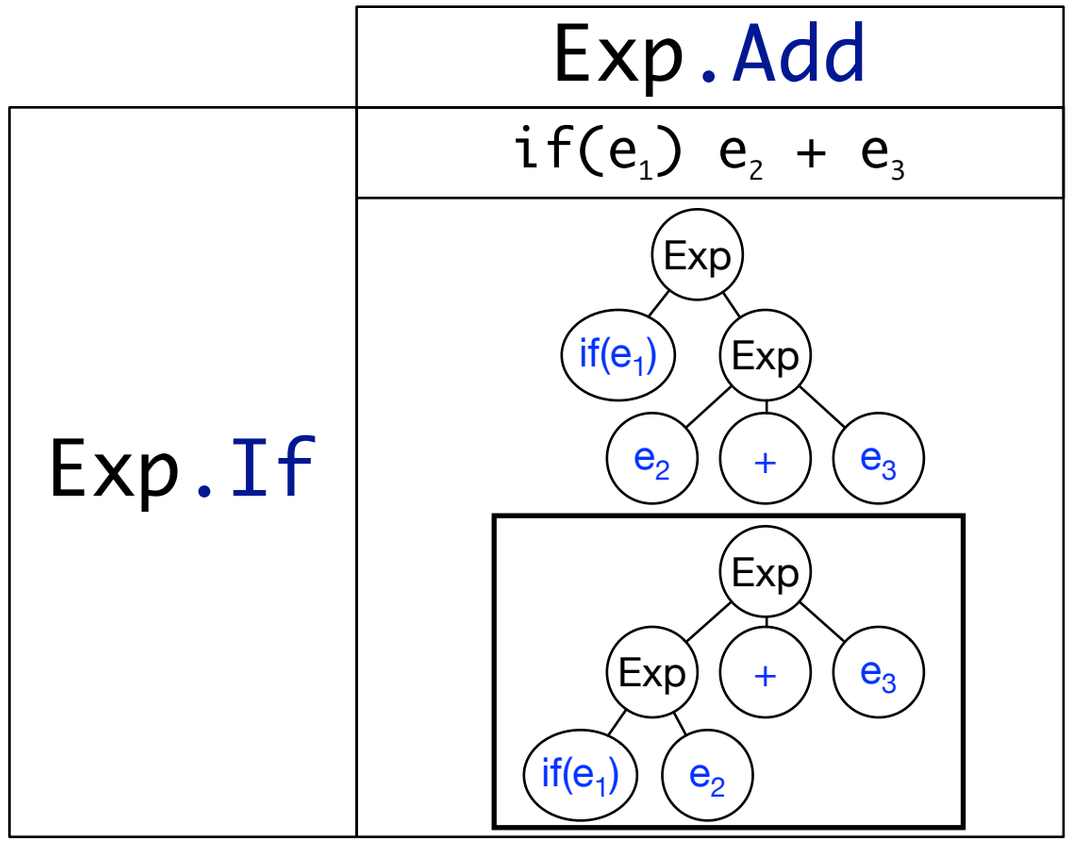
SDF2 Semantics

$$\frac{A.C_1 = \alpha B \gamma > B.C_2 = \beta \in \mathsf{Pr}(G)}{[A.C_1 = \alpha[C_2]\gamma] \in Q_G}$$

```
context-free syntax

  Exp.Add = Exp "+" Exp
  Exp.If  = "if" "(" Exp ")" Exp

context-free priorities

  Exp.Add > Exp.If
```



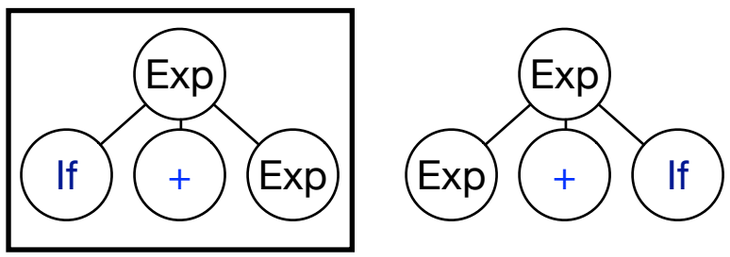


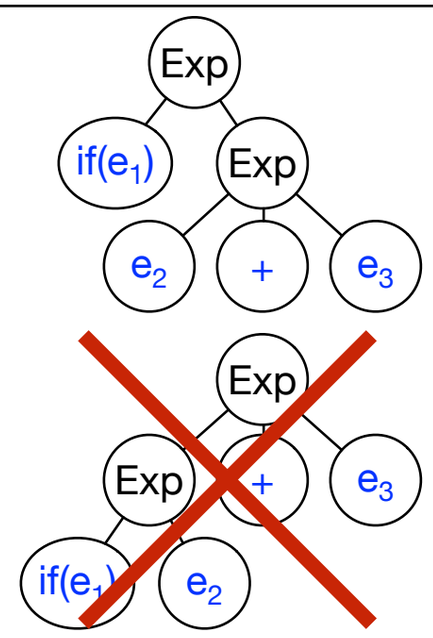


SDF2 Semantics

$$\frac{A.C_1 = \alpha B \gamma > B.C_2 = \beta \in \mathsf{Pr}(G)}{[A.C_1 = \alpha[C_2]\gamma] \in Q_G}$$

```
context-free syntax

  Exp.Add = Exp "+" Exp
  Exp.If  = "if" "(" Exp ")" Exp

context-free priorities

  Exp.Add > Exp.If
```

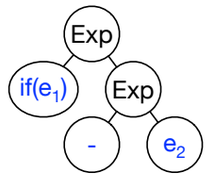
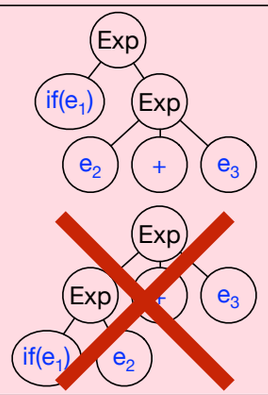
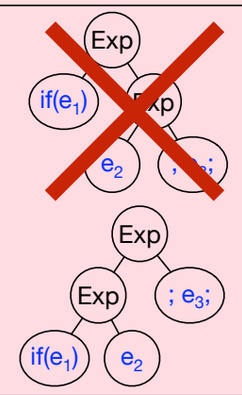
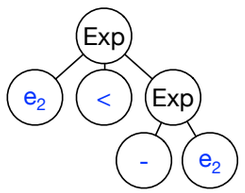
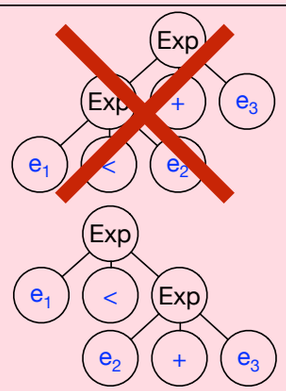
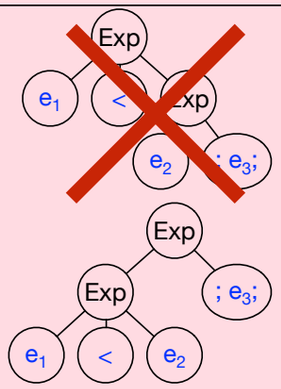
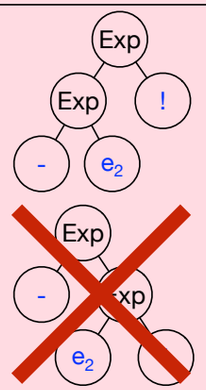
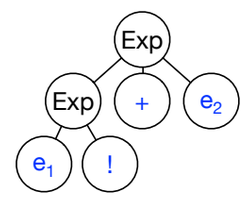
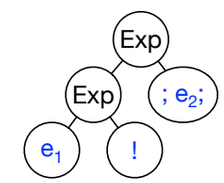


	Exp.Add
	if(e_1) e_2 + e_3
Exp.If	

```
context-free syntax

  Exp.Min = "-" Exp
  Exp.If  = "if" "(" Exp ")" Exp
  Exp.Lt  = Exp "<" Exp
  Exp.Add = Exp "+" Exp
  Exp.Seq = Exp ";" Exp ";"
  Exp.Fac = Exp "!"
  Exp     = "(" Exp ")"
  Exp.Int = INT
```

```
context-free syntax

  Exp.Min = "-" Exp
  Exp.If  = "if" "(" Exp ")" Exp
  Exp.Lt  = Exp "<" Exp {right}
  Exp.Add = Exp "+" Exp {left}
  Exp.Seq = Exp ";" Exp ";"
  Exp.Fac = Exp "!"
  Exp     = "(" Exp ")" {bracket}
  Exp.Int = INT

context-free priorities

 Exp.Min > Exp.Lt > Exp.Add >
 Exp.If > Exp.Seq > Exp.Fac
```

```
Exp.Add > Exp.If
 Exp.If > Exp.Seq
Exp.Add > Exp.Lt
 Exp.Lt > Exp.Seq
Exp.Min > Exp.Fac
```

	Exp.Min	Exp.Add	Exp.Seq
Exp.If	if(e_1) - e_2	if(e_1) e_2 + e_3	if(e_1) e_2; e_3;
			
Exp.Lt	e_1 < - e_2	e_1 < e_2 + e_3	e_1 < e_2; e_3;
			
Exp.Fac	- e_1!	e_1! + e_2	e_1!; e_2;
			

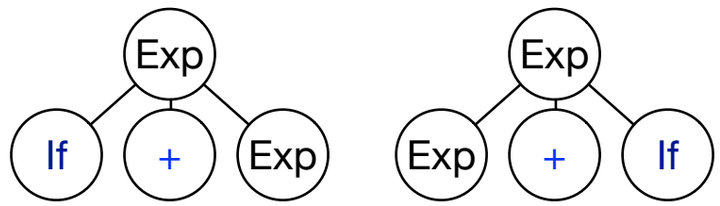
SDF2 Semantics is Unsafe

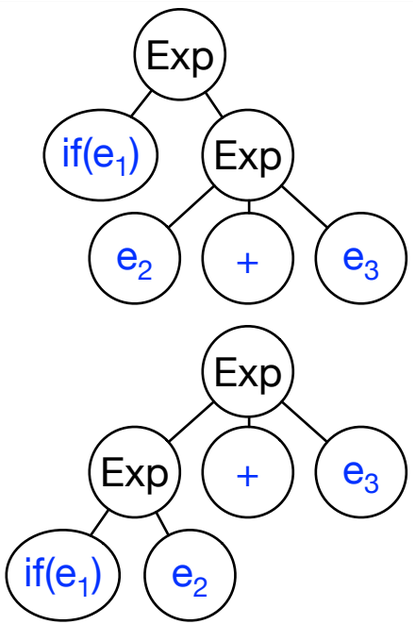
[Afroozeh et al. 2013]

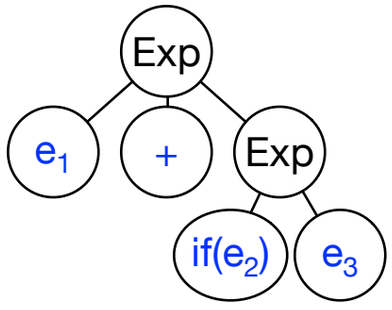
$$\frac{A.C_1 = \alpha B \gamma > B.C_2 = \beta \in \mathsf{Pr}(G)}{[A.C_1 = \alpha[C_2]\gamma] \in Q_G}$$

```
Exp.If  = "if" "(" Exp ")" Exp
Exp.Add = Exp "+" Exp
```

Exp.Add > Exp.If



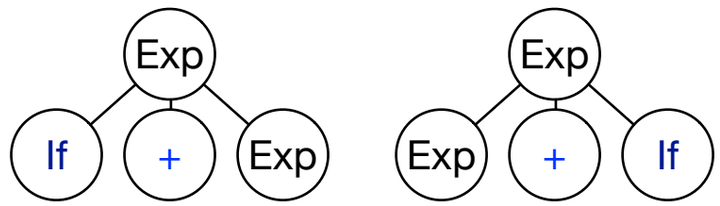
	Exp.Add
	if(e_1) e_2 + e_3
Exp.If	

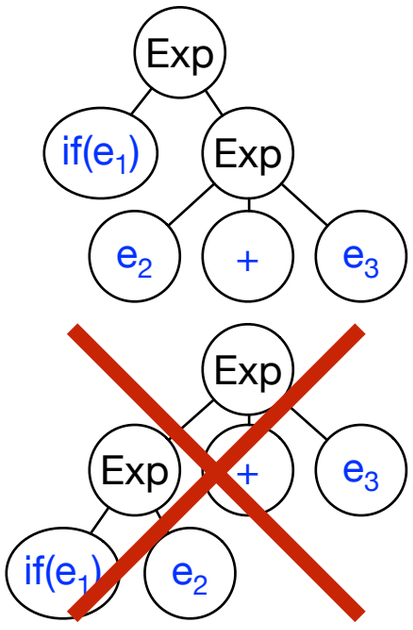
	Exp.If
	e_1 + if(e_2) e_3
Exp.Add	

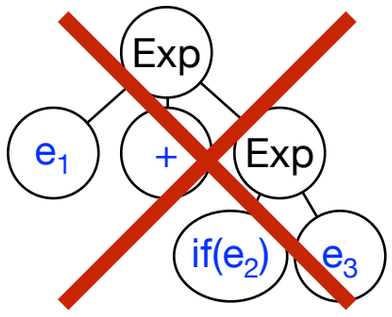
$$\frac{A.C_1 = \alpha B \gamma > B.C_2 = \beta \in \mathsf{Pr}(G)}{[A.C_1 = \alpha[C_2]\gamma] \in Q_G}$$

```
Exp.If  = "if" "(" Exp ")" Exp
Exp.Add = Exp "+" Exp
```

Exp.Add > Exp.If

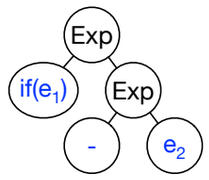
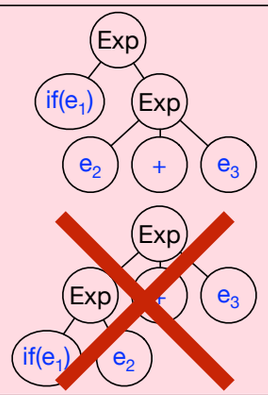
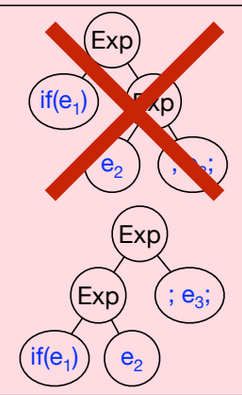
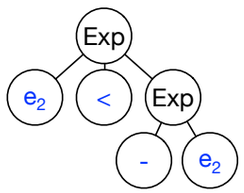
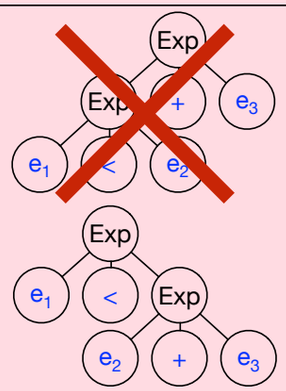
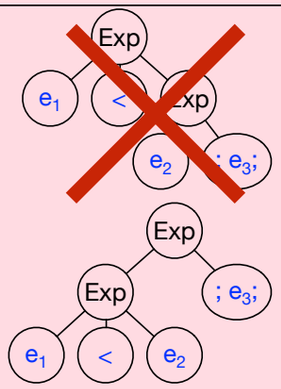
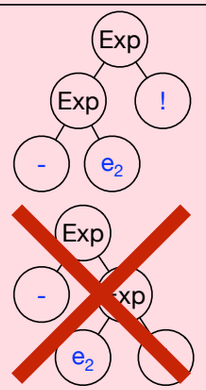
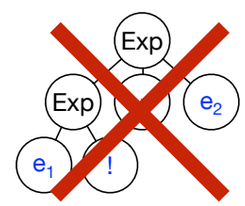
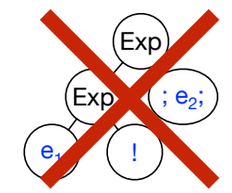


	Exp.Add
	$if(e_1)\ e_2 + e_3$
Exp.If	

	Exp.If
	$e_1 + if(e_2)\ e_3$
Exp.Add	

```
context-free syntax

  Exp.Min = "-" Exp
  Exp.If  = "if" "(" Exp ")" Exp
  Exp.Lt  = Exp "<" Exp {right}
  Exp.Add = Exp "+" Exp {left}
  Exp.Seq = Exp ";" Exp ";"
  Exp.Fac = Exp "!"
  Exp     = "(" Exp ")" {bracket}
  Exp.Int = INT

context-free priorities

 Exp.Min > Exp.Lt > Exp.Add >
 Exp.If > Exp.Seq > Exp.Fac
```

```
Exp.Add > Exp.If
 Exp.If > Exp.Seq
Exp.Add > Exp.Lt
 Exp.Lt > Exp.Seq
Exp.Min > Exp.Fac
Exp.Add > Exp.Fac
Exp.Seq > Exp.Fac
```

	Exp.Min	Exp.Add	Exp.Seq
Exp.If	if(e_1) - e_2	if(e_1) e_2 + e_3	if(e_1) e_2; e_3;
			
Exp.Lt	e_1 < - e_2	e_1 < e_2 + e_3	e_1 < e_2; e_3;
			
Exp.Fac	- e_1!	e_1! + e_2	e_1!; e_2;
			

Disambiguation Safety

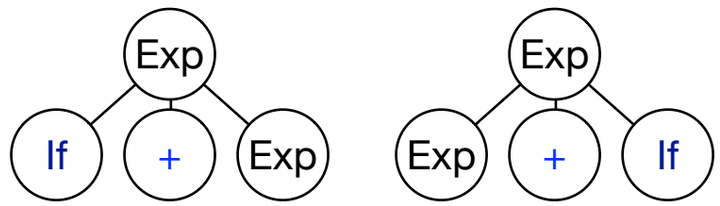
Disambiguation should not reject unambiguous sentences

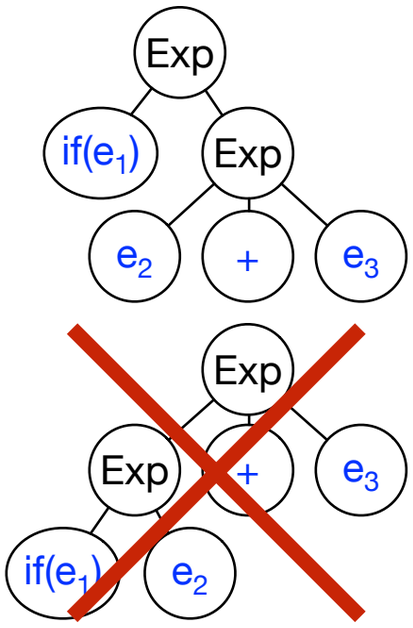
Safe Semantics

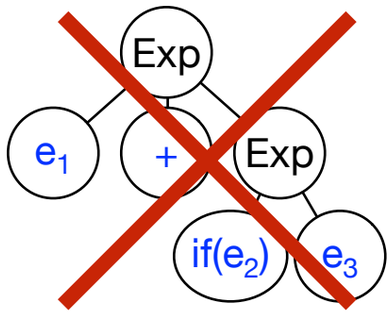
$$\frac{A.C_1 = \alpha B \gamma > B.C_2 = \beta \in \mathsf{Pr}(G)}{[A.C_1 = \alpha[C_2]\gamma] \in Q_G}$$

```
Exp.If  = "if" "(" Exp ")" Exp
Exp.Add = Exp "+" Exp
```

Exp.Add > Exp.If



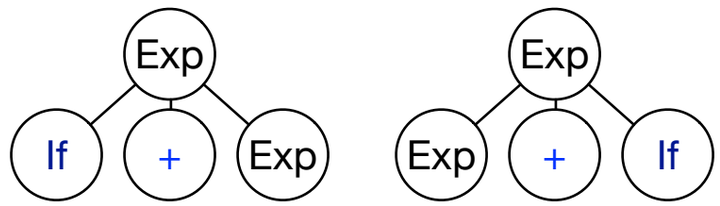
	Exp.Add
	$\texttt{if}(e_1)\ e_2\ \texttt{+}\ e_3$
Exp.If	

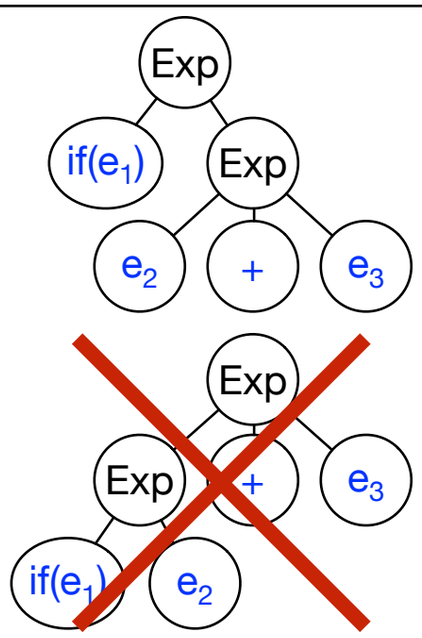
	Exp.If
	$e_1\ \texttt{+}\ \texttt{if}(e_2)\ e_3$
Exp.Add	

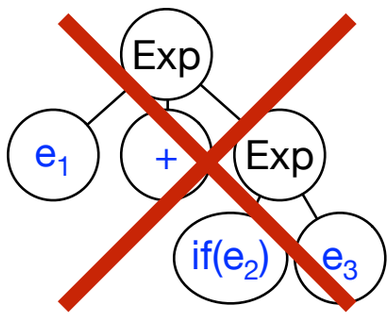
$$\frac{A.C_1 = \alpha B \gamma > B.C_2 = \beta \in \mathsf{Pr}(G)}{[A.C_1 = \alpha[C_2]\gamma] \in Q_G}$$

```
Exp.If  = "if" "(" Exp ")" Exp
Exp.Add = Exp "+" Exp
```

Exp.Add > Exp.If



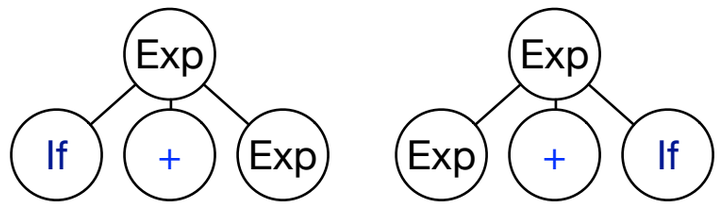
	Exp.Add
	$\texttt{if}(e_1)\ e_2 + e_3$
Exp.If	

	Exp.If
	$e_1 + \texttt{if}(e_2)\ e_3$
Exp.Add	

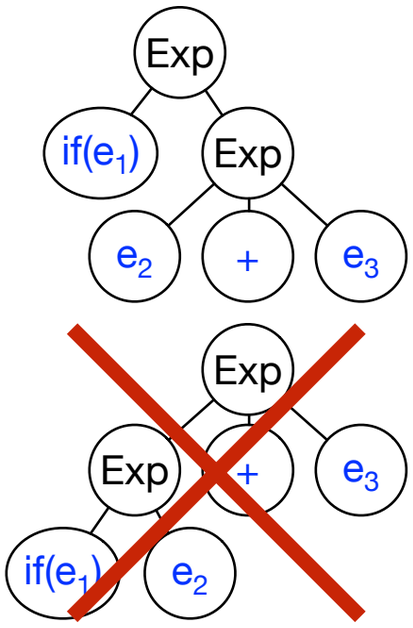
$$\frac{A.C_1 = \alpha B \gamma > B.C_2 = \beta \in \mathrm{Pr}(G)}{[A.C_1 = \alpha[C_2]\gamma] \in Q_G}$$

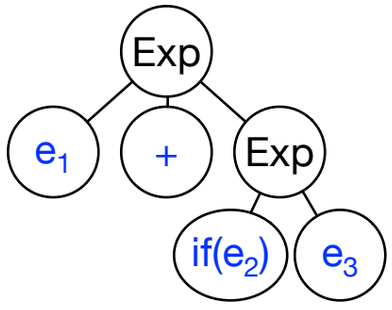
$$\frac{A.C_1 = A\gamma > A.C_2 = \alpha A \in \mathrm{Pr}(G)}{[A.C_1 = [C_2]\gamma] \in Q'_G}$$

```
Exp.If  = "if" "(" Exp ")" Exp
Exp.Add = Exp "+" Exp
```



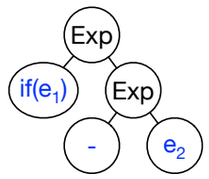
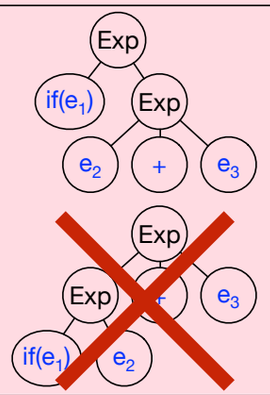
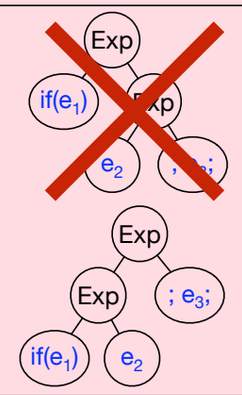
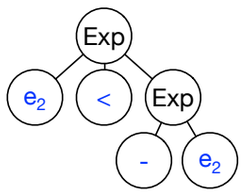
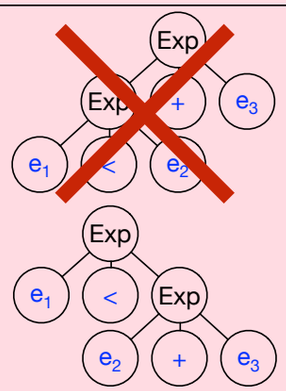
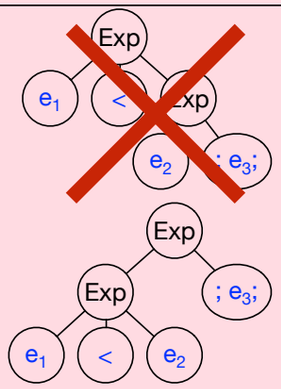
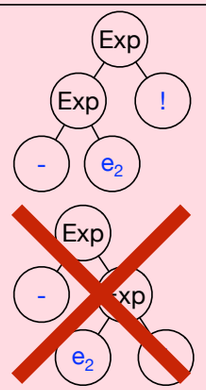
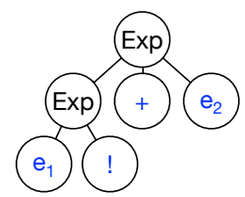
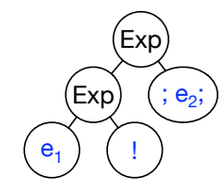
Exp.Add > Exp.If

	Exp.Add
	if(e_1) e_2 + e_3
Exp.If	

	Exp.If
	e_1 + if(e_2) e_3
Exp.Add	

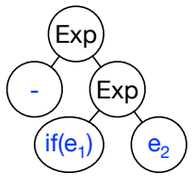
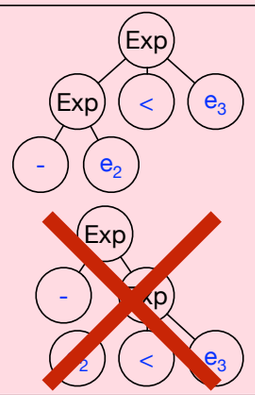
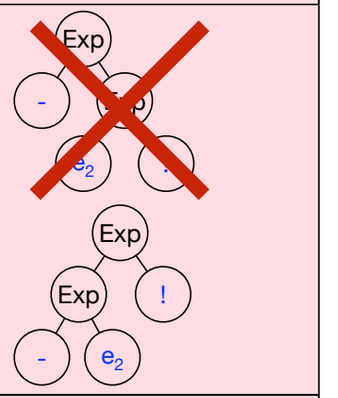
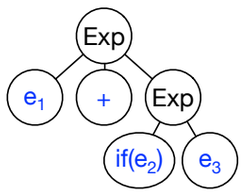
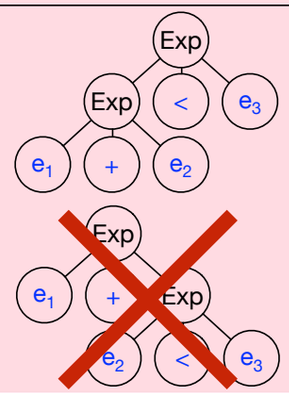
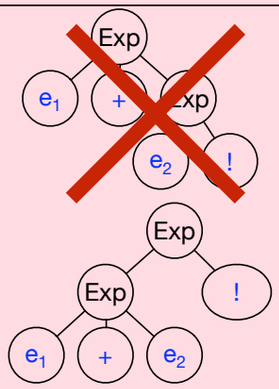
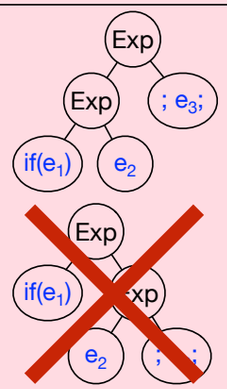
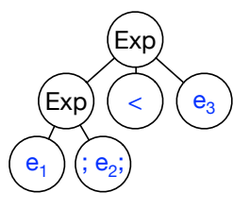
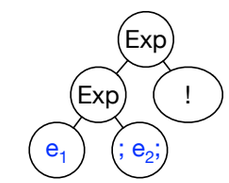
```
context-free syntax

  Exp.Min = "-" Exp
  Exp.If  = "if" "(" Exp ")" Exp
  Exp.Lt  = Exp "<" Exp {right}
  Exp.Add = Exp "+" Exp {left}
  Exp.Seq = Exp ";" Exp ";"
  Exp.Fac = Exp "!"
  Exp     = "(" Exp ")" {bracket}
  Exp.Int = INT

context-free priorities

 Exp.Min > Exp.Lt > Exp.Add >
 Exp.If > Exp.Seq > Exp.Fac
```

```
Exp.Add > Exp.If
 Exp.If > Exp.Seq
Exp.Add > Exp.Lt
 Exp.Lt > Exp.Seq
Exp.Min > Exp.Fac
Exp.Add > Exp.Fac
Exp.Seq > Exp.Fac
```

	Exp.Min	Exp.Add	Exp.Seq
Exp.If	if(e_1) - e_2	if(e_1) e_2 + e_3	if(e_1) e_2; e_3;
			
Exp.Lt	e_1 < - e_2	e_1 < e_2 + e_3	e_1 < e_2; e_3;
			
Exp.Fac	- e_1!	e_1! + e_2	e_1!; e_2;
			

```
context-free syntax

  Exp.Min = "-" Exp
  Exp.If  = "if" "(" Exp ")" Exp
  Exp.Lt  = Exp "<" Exp {right}
  Exp.Add = Exp "+" Exp {left}
  Exp.Seq = Exp ";" Exp ";"
  Exp.Fac = Exp "!"
  Exp     = "(" Exp ")" {bracket}
  Exp.Int = INT

context-free priorities

 Exp.Min > Exp.Lt > Exp.Add >
 Exp.If > Exp.Seq > Exp.Fac
```

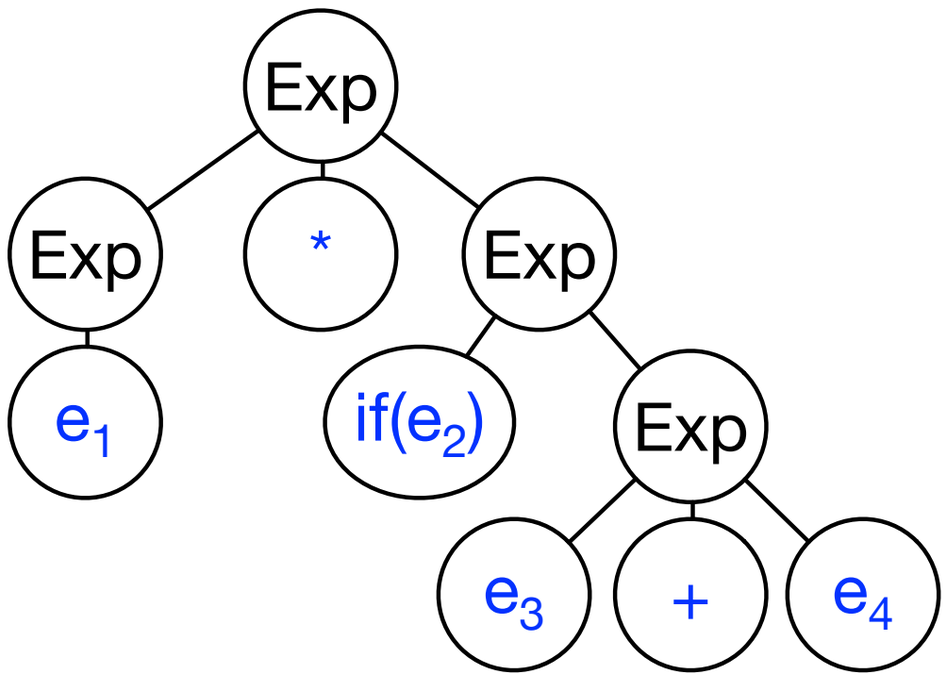
```
Exp.Add > Exp.If
 Exp.If > Exp.Seq
Exp.Add > Exp.Lt
 Exp.Lt > Exp.Seq
Exp.Min > Exp.Fac
Exp.Add > Exp.Fac
Exp.Seq > Exp.Fac
```

	Exp.If	Exp.Lt	Exp.Fac
Exp.Min	- if(e_1) e_2	- e_2 < e_3	- e_1!
			
Exp.Add	e_1 + if(e_2) e_3	e_1 + e_2 < e_3	e_1 + e_2!
			
Exp.Seq	if(e_1) e_2; e_3;	e_1; e_2; < e_3	e_1; e_2;!
			

Deep Conflicts

```
context-free syntax

  Exp.If  = "if" "(" Exp ")" Exp
  Exp.Add = Exp "+" Exp {left}
  Exp.Mul = Exp "*" Exp {left}
  Exp.Int = INT

context-free priorities

  Exp.Mul > Exp.Add > Exp.If
```

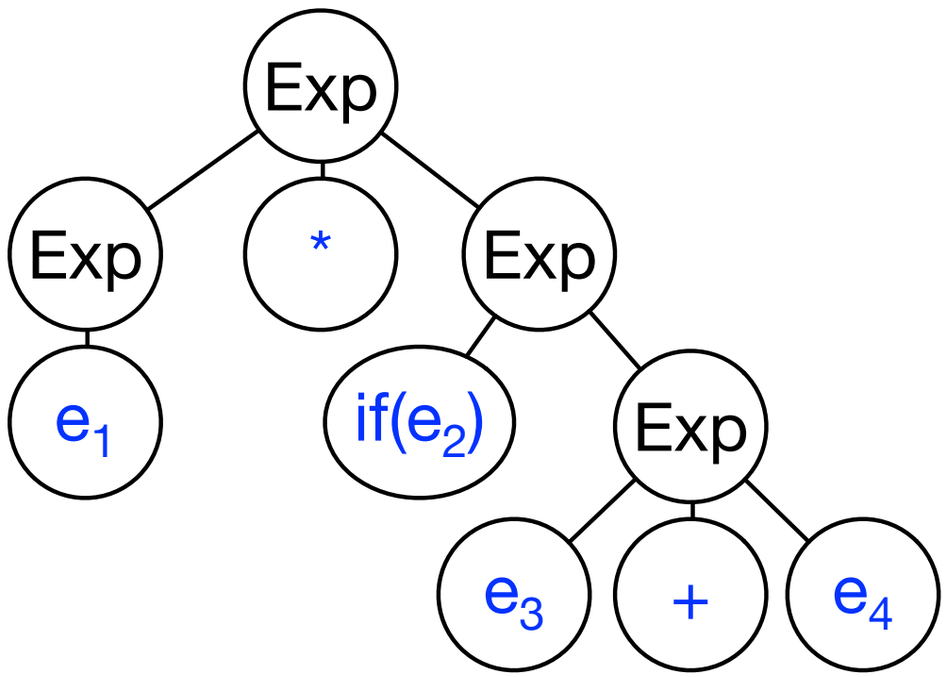
e_1 * if(e_2) e_3 + e_4



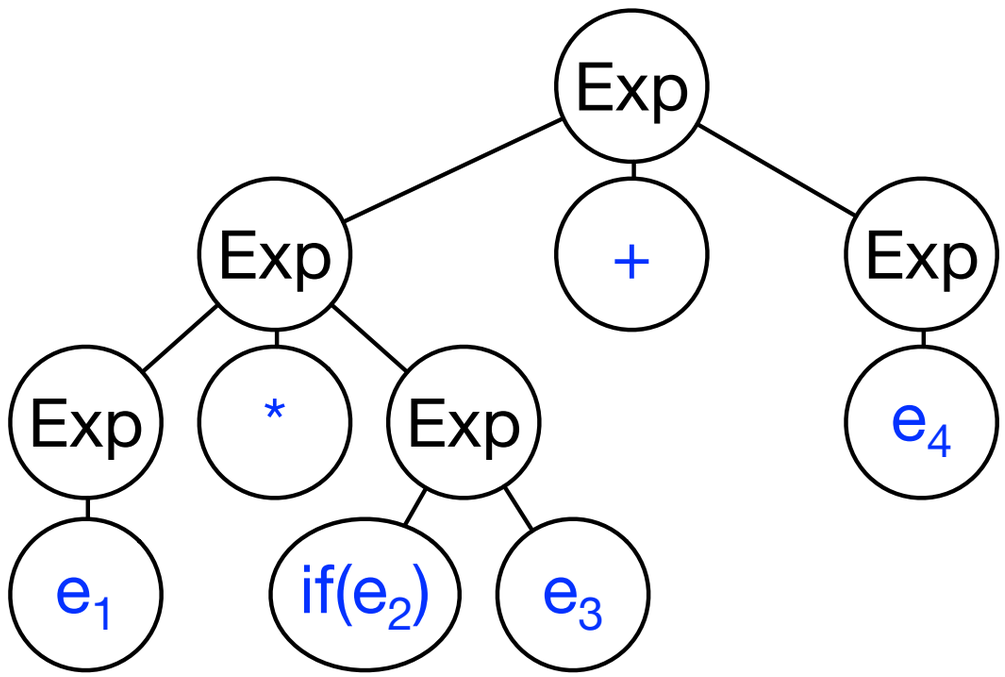
e_1 * [if(e_2) e_3 + e_4]

```
context-free syntax

  Exp.If  = "if" "(" Exp ")" Exp
  Exp.Add = Exp "+" Exp {left}
  Exp.Mul = Exp "*" Exp {left}
  Exp.Int = INT

context-free priorities

  Exp.Mul > Exp.Add > Exp.If
```

e_1 * if(e_2) e_3 + e_4



e_1 * [if(e_2) e_3 + e_4]

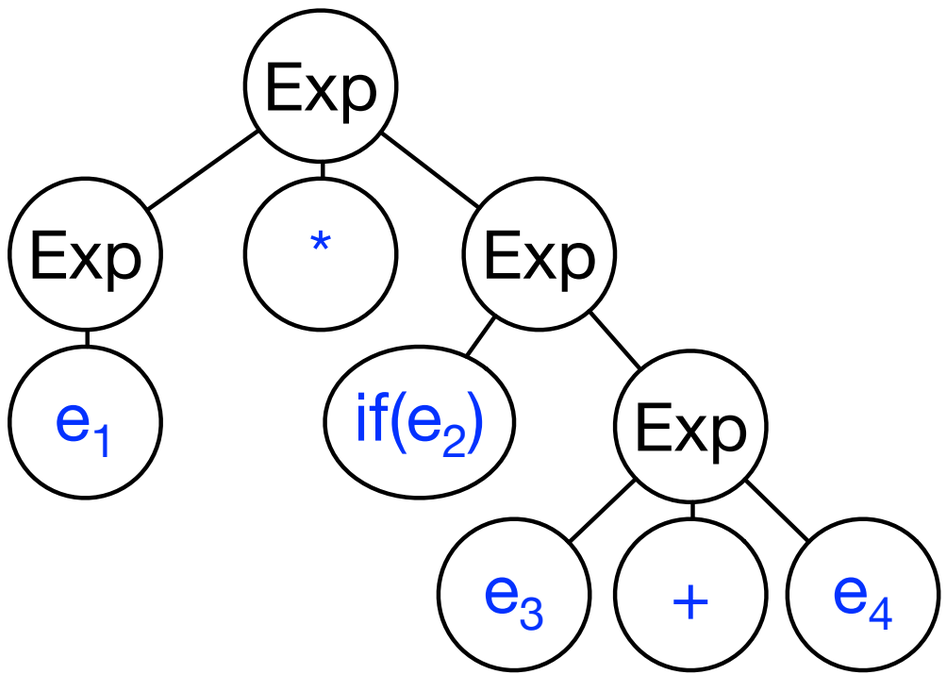


[e_1 * if(e_2) e_3] + e_4

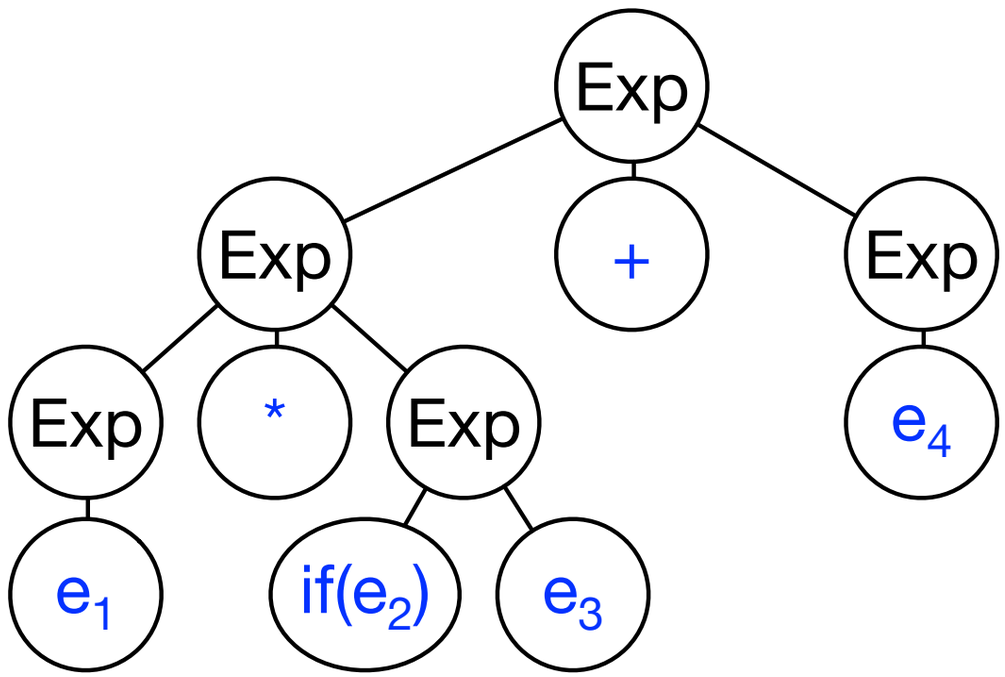
```
context-free syntax

  Exp.If  = "if" "(" Exp ")" Exp
  Exp.Add = Exp "+" Exp {left}
  Exp.Mul = Exp "*" Exp {left}
  Exp.Int = INT

context-free priorities

  Exp.Mul > Exp.Add > Exp.If
```

e_1 * if(e_2) e_3 + e_4

Deep Priority Conflict!



e_1 * [if(e_2) e_3 + e_4]

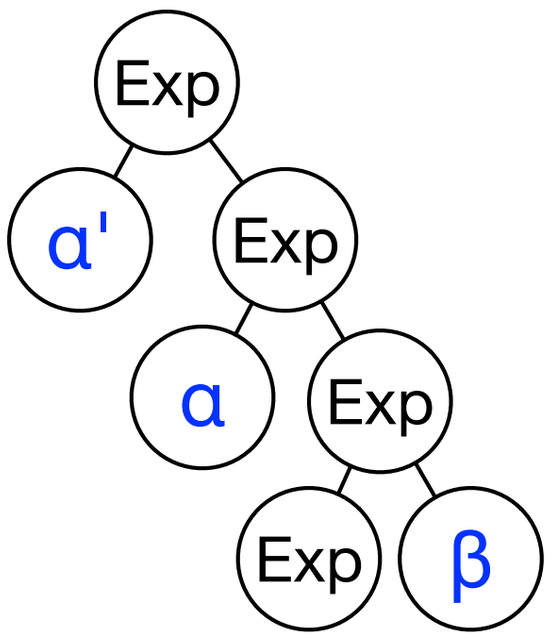


[e_1 * if(e_2) e_3] + e_4

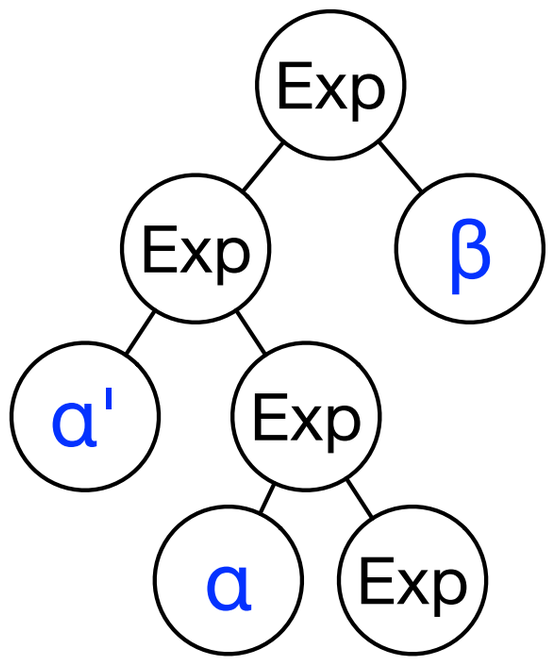
Operator-style Deep Conflict

```
Exp = α Exp
Exp = Exp β
Exp = α' Exp

α' > β > α
```

α' α Exp β



α' α [Exp β]



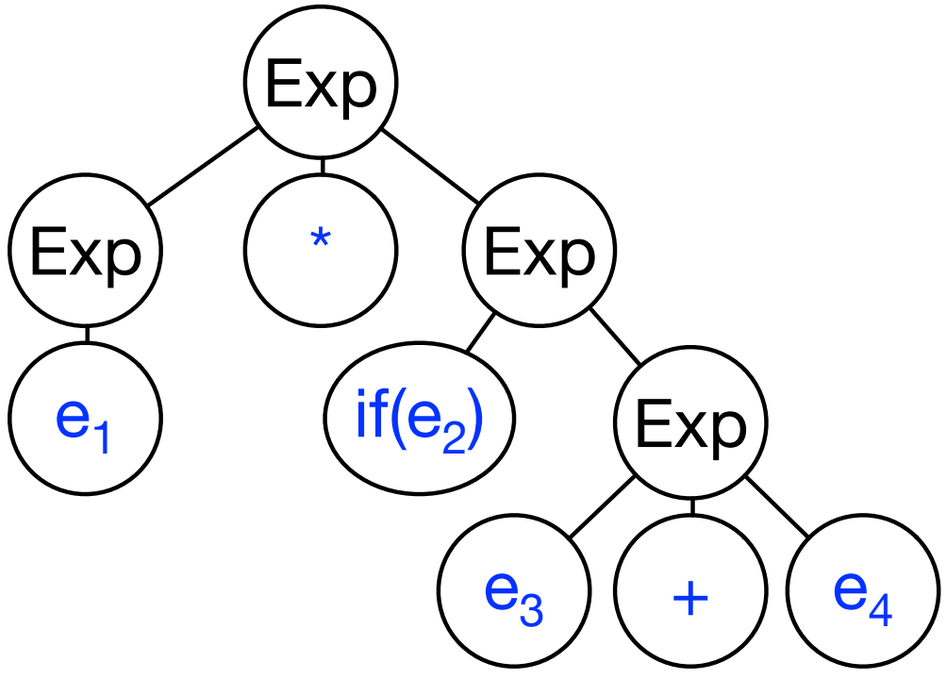
[α' α Exp] β

```
context-free syntax

  Exp.If  = "if" "(" Exp ")" Exp
  Exp.Add = Exp "+" Exp {left}
  Exp.Mul = Exp "*" Exp {left}
  Exp.Int = INT

context-free priorities

  Exp.Mul > Exp.Add > Exp.If
```

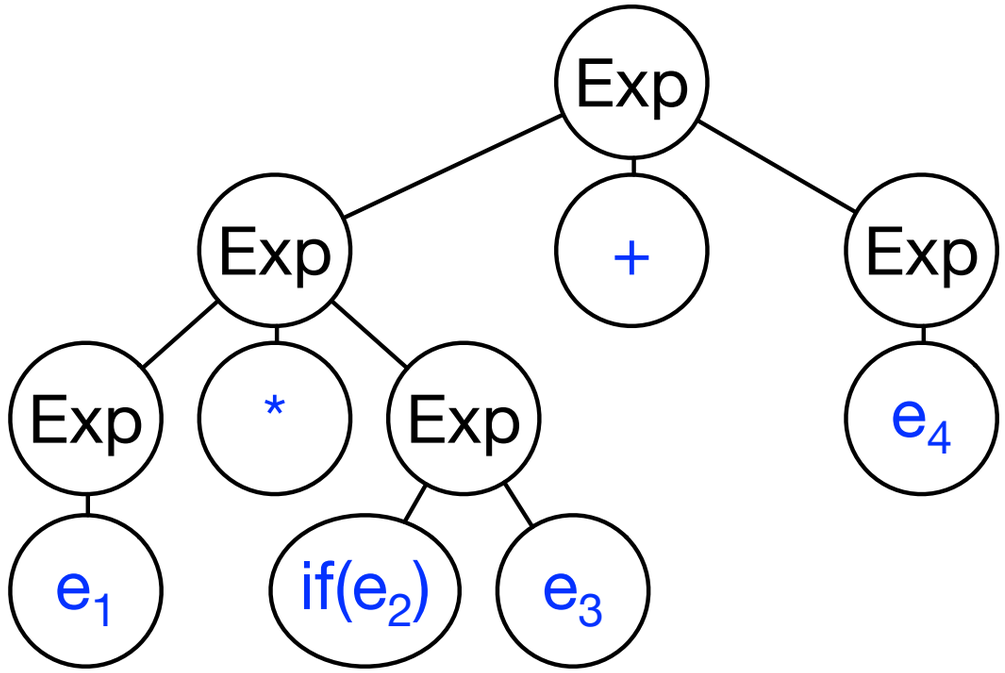
$$\frac{A.C_2 > A.C_1 \in \Pr(G) \quad \alpha \overset{*}{\Rightarrow}_G A\beta}{[A.C_2 = [A.C_1 = \alpha A]\gamma] \in Q_G^{rm}}$$

e_1 * if(e_2) e_3 + e_4

Deep Priority Conflict!



e_1 * [if(e_2) e_3 + e_4]



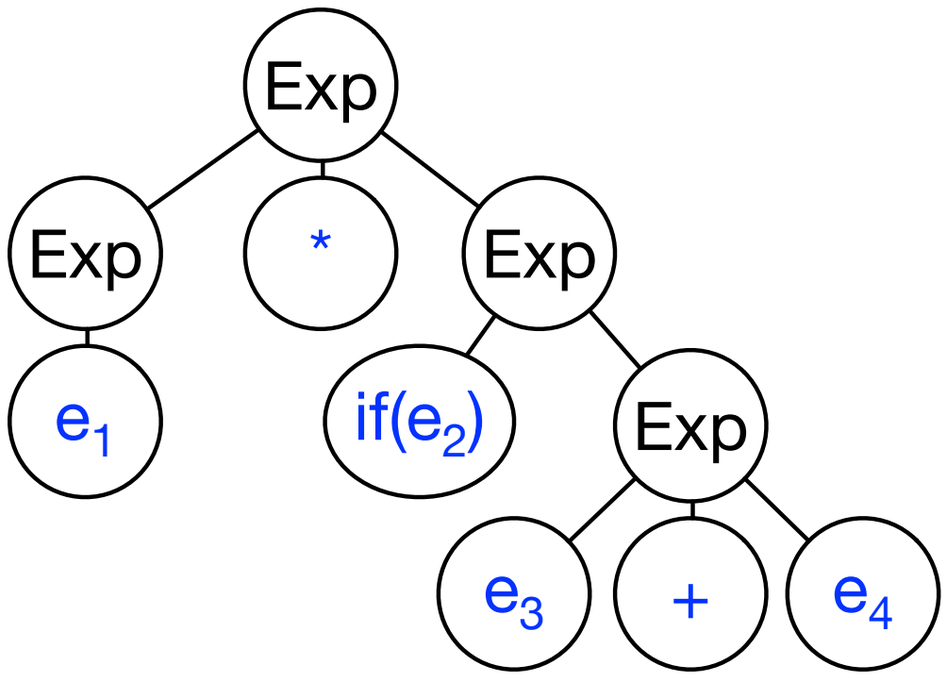
[e_1 * if(e_2) e_3] + e_4

```
context-free syntax

  Exp.If  = "if" "(" Exp ")" Exp
  Exp.Add = Exp "+" Exp {left}
  Exp.Mul = Exp "*" Exp {left}
  Exp.Int = INT

context-free priorities

  Exp.Mul > Exp.Add > Exp.If
```

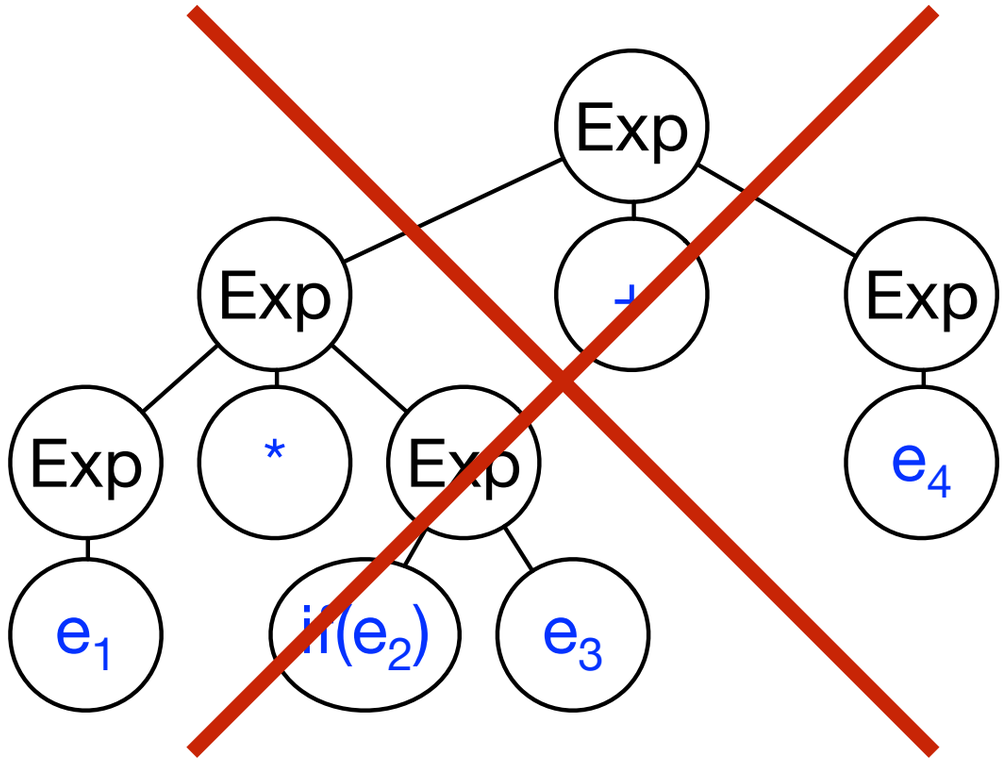
$$\frac{A.C_2 > A.C_1 \in \Pr(G) \quad \alpha \overset{*}{\Rightarrow}_G A\beta}{[A.C_2 = [A.C_1 = \alpha A]\gamma] \in Q_G^{rm}}$$

e_1 * if(e_2) e_3 + e_4

Deep Priority Conflict!



e_1 * [if(e_2) e_3 + e_4]



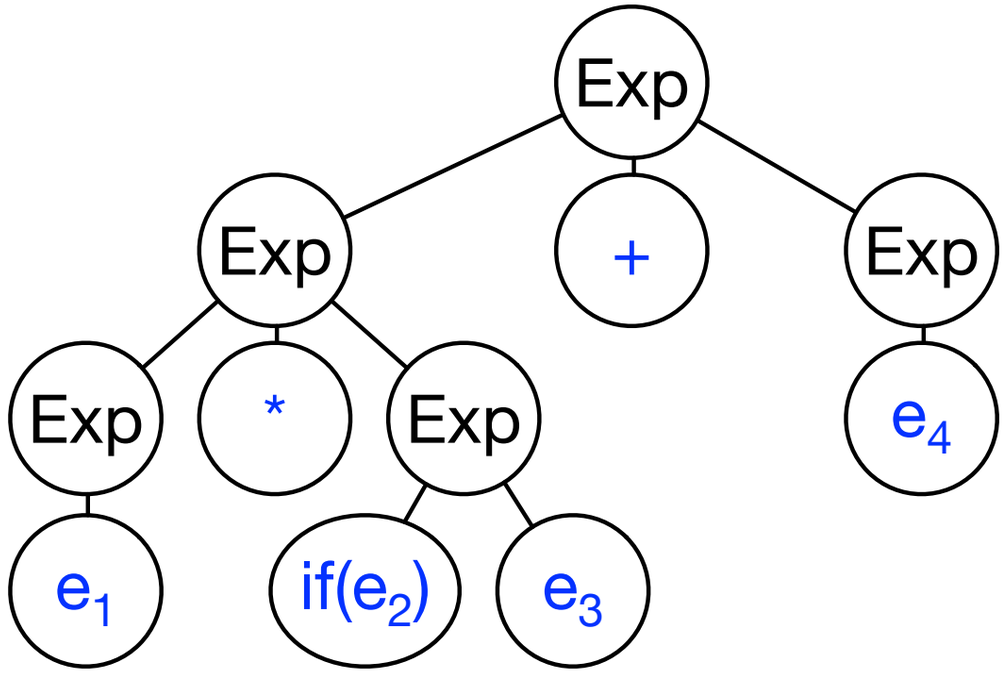
[e_1 * if(e_2) e_3] + e_4

Contextual Grammars

```
context-free syntax

  Exp.If  = "if" "(" Exp ")" Exp
  Exp.Add = Exp "+" Exp {left}
  Exp.Mul = Exp "*" Exp {left}
  Exp.Int = INT

context-free priorities

  Exp.Mul > Exp.Add > Exp.If
```

e_1 * if(e_2) e_3 + e_4

Deep Priority Conflict!

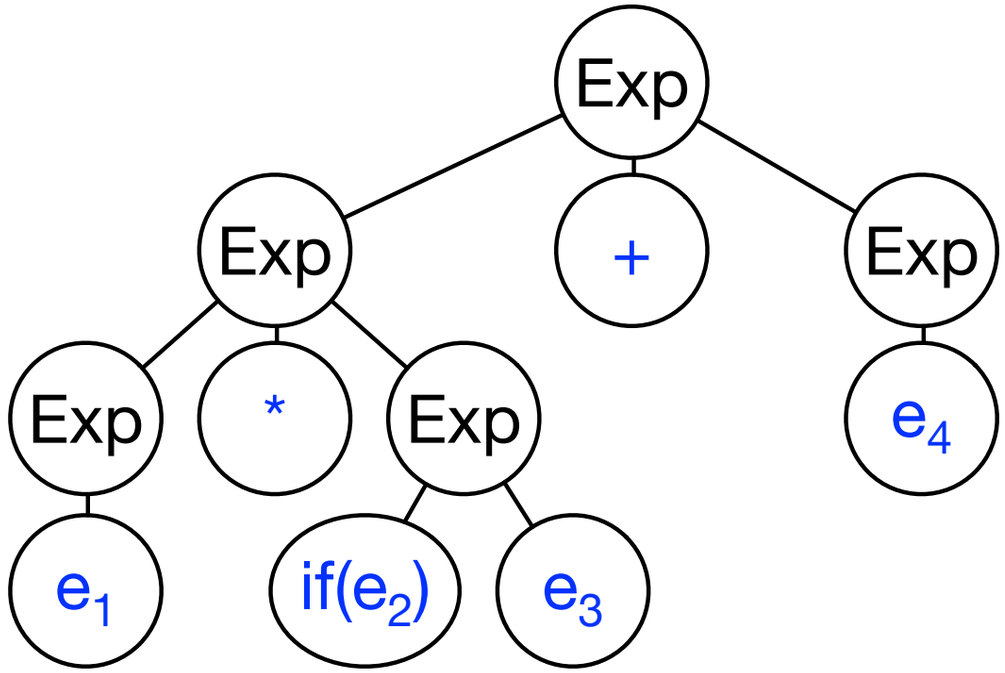


[e_1 * if(e_2) e_3] + e_4

```
context-free syntax

  Exp.If  = "if" "(" Exp ")" Exp
  Exp.Add = Exp{If} "+" Exp {left}
  Exp.Mul = Exp "*" Exp {left}
  Exp.Int = INT

context-free priorities

  Exp.Mul > Exp.Add > Exp.If
```

e_1 * if(e_2) e_3 + e_4

Deep Priority Conflict!

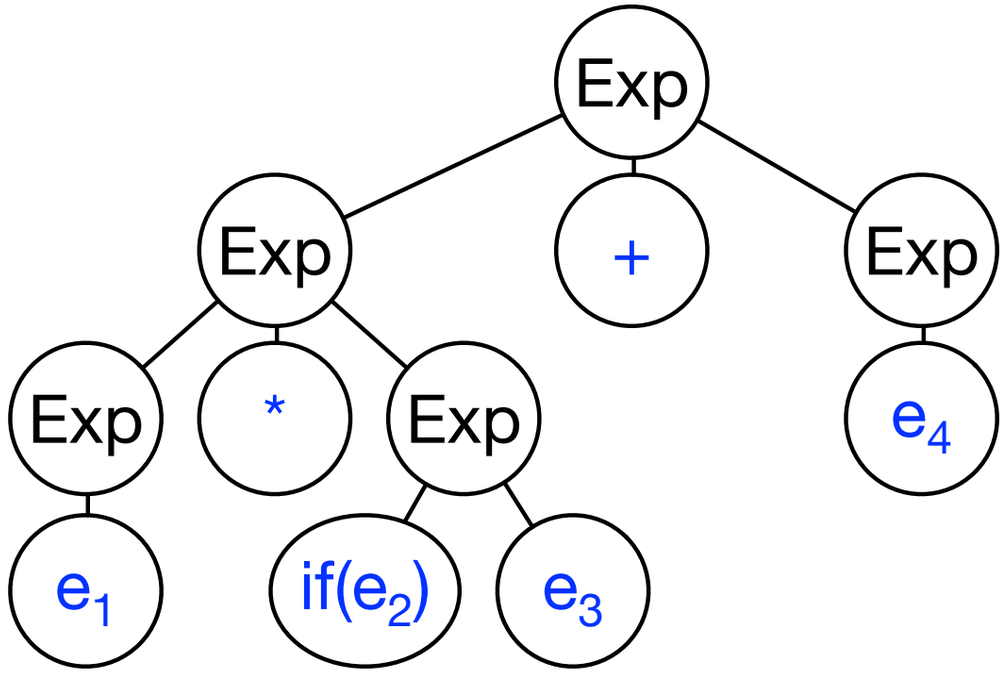


[e_1 * if(e_2) e_3] + e_4

```
context-free syntax

  Exp.If  = "if" "(" Exp ")" Exp
  Exp.Add = Exp{If} "+" Exp {left}
  Exp.Mul = Exp "*" Exp {left}
  Exp.Int = INT

  Exp{If}.Add = Exp{If} "+" Exp{If} {left}
  Exp{If}.Mul = Exp "*" Exp{If} {left}
  Exp{If}.Int = INT

context-free priorities

  Exp.Mul > Exp.Add > Exp.If
```

e_1 * if(e_2) e_3 + e_4

Deep Priority Conflict!

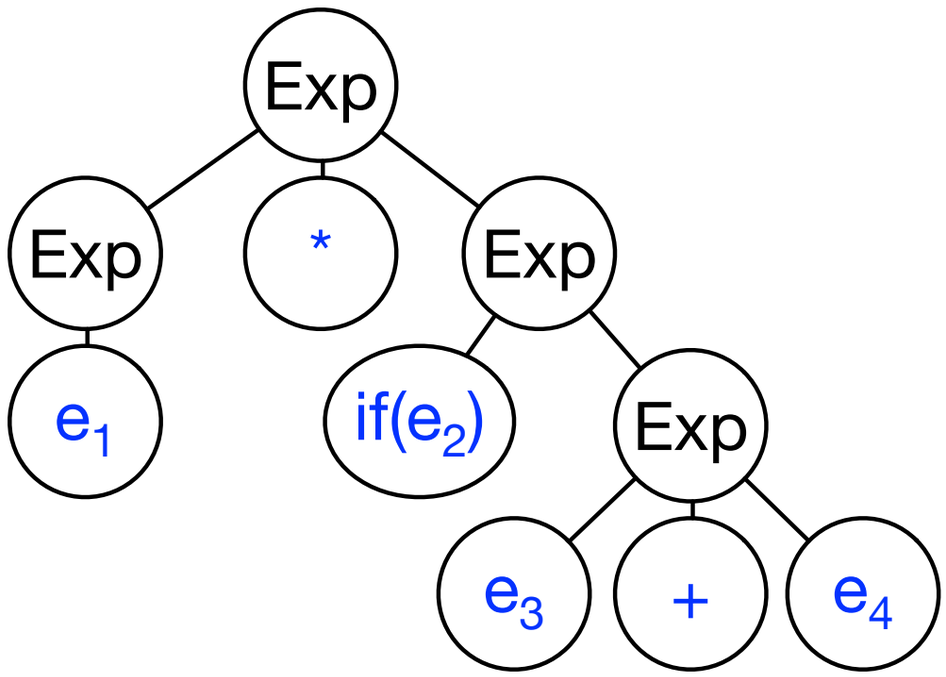


[e_1 * if(e_2) e_3] + e_4

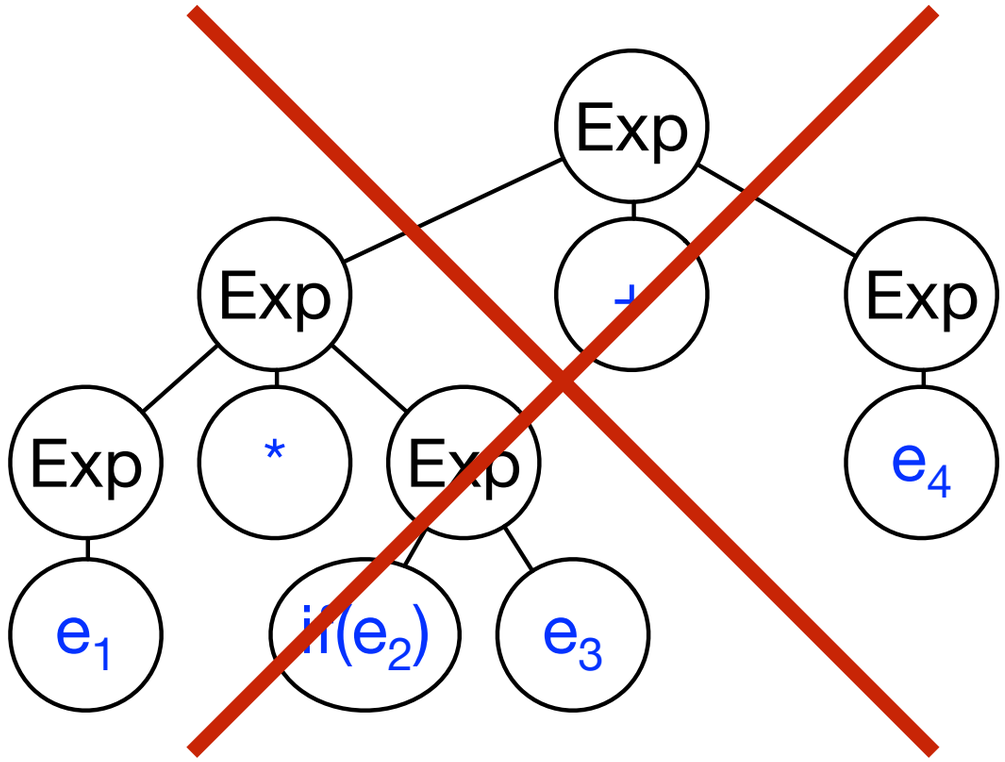
```
context-free syntax

  Exp.If  = "if" "(" Exp ")" Exp
  Exp.Add = Exp{If} "+" Exp {left}
  Exp.Mul = Exp{If} "*" Exp {left}
  Exp.Int = INT

  Exp{If}.Add = Exp{If} "+" Exp{If} {left}
  Exp{If}.Mul = Exp{If} "*" Exp{If} {left}
  Exp{If}.Int = INT

context-free priorities

  Exp.Mul > Exp.Add > Exp.If
```

$e_1 * \texttt{if}(e_2)\ e_3 + e_4$

Deep Priority Conflict!



$e_1 * [\texttt{if}(e_2)\ e_3 + e_4]$



$[e_1 * \texttt{if}(e_2)\ e_3] + e_4$

Spoofax Integration

workspace-tiger - disambdemo/examples/example02.aterm - Eclipse

disamb-demo.sdf3

```
module disamb-demo

imports Common

context-free start-symbols Exp

context-free syntax

  Exp.Int    = INT
  Exp.Var    = ID
  Exp.Add    = <<Exp> + <Exp>> {left}
  Exp.If     = <if(<Exp>) <Exp>>
  Exp.IfElse = <if(<Exp>) <Exp> else <Exp>>
  Exp        = <(<Exp>)> {bracket}

  Exp.Match  = <match <Exp> with <Pat+>> {longest-match}
  Pat.Clause = [[ID] -> [Exp]]

//context-free priorities
//
//  Exp.Add > Exp.IfElse > Exp.If > Exp.Match


```

example03.disamb

```
a - if(1) 2 + 3
```

example02.disamb

```
if(1) if(2) 3 else 4
```

example01.aterm

```
amb(
  [ Add(
      Add(Var("a"), If(Int("1"), Int("2")))
    , Int("3")
    )
  , Add(
      Var("a")
    , If(Int("1"), Add(Int("2"), Int("3")))
    )
  ]
)
```

example03.aterm

example02.aterm

```
amb(
  [ IfElse(
      Int("1")
    , If(Int("2"), Int("3"))
    , Int("4")
    )
  , If(
      Int("1")
    , IfElse(Int("2"), Int("3"), Int("4"))
    )
  ]
)
```

example04.aterm

Writable | Insert | 6 : 6

workspace-tiger - disambdemo/syntax/disamb-demo.sdf3 - Eclipse

disamb-demo.sdf3

```
module disamb-demo

imports Common

context-free start-symbols Exp

context-free syntax

  Exp.Int     = INT
  Exp.Var     = ID
  Exp.Add     = <<Exp> + <Exp>> {left}
  Exp.If      = <if(<Exp>) <Exp>>
  Exp.IfElse  = <if(<Exp>) <Exp> else <Exp>>
  Exp         = <(<Exp>)> {bracket}

  Exp.Match   = <match <Exp> with <Pat+>> {longest-match}
  Pat.Clause  = [[ID] -> [Exp]]

context-free priorities

  Exp.Add > Exp.IfElse > Exp.If > Exp.Match

```

example01.disam

```
if(1 + 2)
  if(3 + 4 + 5)
    6 + 7
  else
    8 + 9
```

example01.aterm | example03.aterm

```
If(
  Add(Int("1"), Int("2"))
, IfElse(
    Add(Add(Int("3"), Int("4")), Int("5"))
  , Add(Int("6"), Int("7"))
  , Add(Int("8"), Int("9"))
  )
)
```

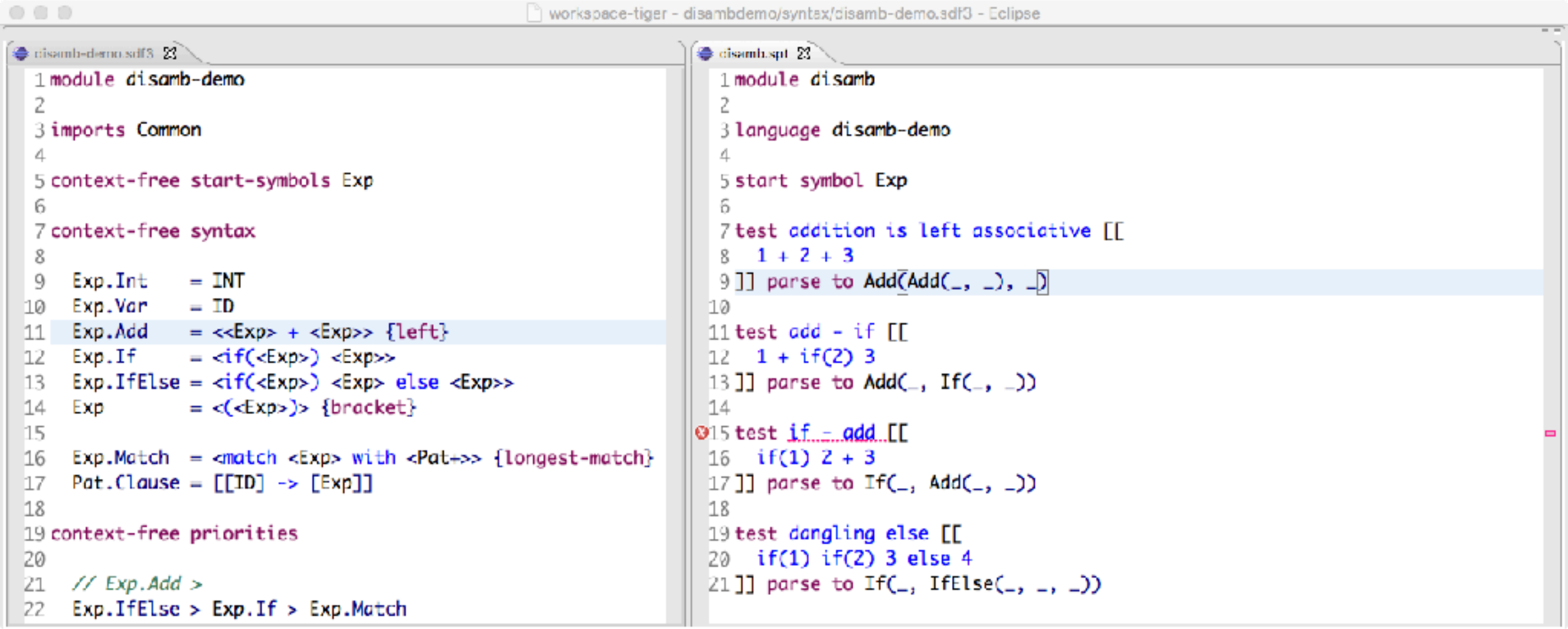
example04.disam

```
match x with
  z -> 1 + match 5 with
          a -> a + x
          y -> 3 + 4
```

example02.aterm | example04.aterm

```
Match(
  Var("x")
, [ Clause(
      "z"
    , Add(
        Int("1")
      , Match(
          Int("5")
        , [ Clause("a", Add(Var("a"), Var("x")))
          , Clause("y", Add(Int("3"), Int("4")))
          ]
        )
      )
    )
  ]
```

Writable | Insert | 23 : 2


workspace-tiger - disambdemo/syntax/disamb-demo.sdf3 - Eclipse
disamb-demo.sdf3
module disamb-demo
imports Common
context-free start-symbols Exp
context-free syntax
Exp.Int = INT
Exp.Var = ID
Exp.Add = <<Exp> + <Exp>> {left}
Exp.If = <if(<Exp>) <Exp>>
Exp.IfElse = <if(<Exp>) <Exp> else <Exp>>
Exp = <(<Exp>)> {bracket}
Exp.Match = <match <Exp> with <Pat+>> {longest-match}
Pat.Clause = [[ID] -> [Exp]]
context-free priorities
// Exp.Add >
Exp.IfElse > Exp.If > Exp.Match
disamb.spt
module disamb
language disamb-demo
start symbol Exp
test addition is left associative [[
1 + 2 + 3
]] parse to Add(Add(_, _), _)
test add - if [[
1 + if(2) 3
]] parse to Add(_, If(_, _))
test if - add [[
if(1) 2 + 3
]] parse to If(_, Add(_, _))
test dangling else [[
if(1) if(2) 3 else 4
]] parse to If(_, IfElse(_, _, _))

Status

- Safe and complete semantics for disambiguation with priority and associativity in SDF3 complete for operator-style ambiguities [revision pending]
- Implementation 1: disambiguation of deep conflicts by transformation to contextual grammar [revision pending]
- Implementation 2: data-dependent extension of SGLR; only requires context annotations in original grammar productions [<Programming> 2018]
- Available in SDF3+SGLR in current Spoofax LWB
- Applied in grammars for OCaml, Java, …
- Study: deep conflicts in the wild (they exist) [SLE 2017]