

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

Eelco Visser

Joint work with Eduardo Amorim

June 30, 2020

International Workshop on Confluence (IWC'20)

Co-located with FSCD and IJCAR

'Paris, France'

Spoofax Language Workbench

```
Calc.sdf3
17 Exp.Num = NUM
18 Exp.Min = <<Exp>
19 Exp.Pow = <<Exp> ^ <Exp> [right]
20 Exp.Pos = <<Exp> + <Exp> [right]
21 Exp.Mul = <<Exp> * <Exp> [left]
22 Exp.Div = <<Exp> / <Exp> [left]
23 Exp.Sub = <<Exp> - <Exp> [left, pref]
24 Exp.Add = <<Exp> + <Exp> [left]
25
26 Exp.Eq = <<Exp> == <Exp> [non-assoc]
27 Exp.Neq = <<Exp> != <Exp> [non-assoc]
28 Exp.Gt = [[Exp] > [Exp]] [non-assoc]
29 Exp.Lt = [[Exp] < [Exp]] [non-assoc]
30
31 context-free syntax // booleans
32
33 Exp.True = <<true>
34 Exp.False = <<false>
35 Exp.Not = <<!Exp>
36 Exp.And = <<Exp> & <Exp> [left]
37 Exp.Or = <<Exp> | <Exp> [left]
38
39 Exp.If = <
40   !f<Exp>
41   <Exp>
42   else
43     <Exp>
44 >
45
46 context-free syntax // variables and fu
47
48 Exp.Var = ID
49
```

```
calc.nabl2
24 [[ e ^ (s) : ty ]].
25
26 Stat[[ Exp(e) ^ (s, s_nxt) ] :=
27   s = s_nxt.
28 [[ e ^ (s) : ty ].
29
30 rules // numbers
31
32 [[ Num(x) ^ (s) : NumTC ]].
33
34 [[ Pow(e1, e2) ^ (s) : NumTC ] :=
35   [[ e1 ^ (s) : NumTC ]],
36   [[ e2 ^ (s) : NumTC ]].
37 [[ Mul(e1, e2) ^ (s) : NumTC ] :=
38   [[ e1 ^ (s) : NumTC ]],
39   [[ e2 ^ (s) : NumTC ]].
40 [[ Add(e1, e2) ^ (s) : NumTC ] :=
41   [[ e1 ^ (s) : NumTC ]],
42   [[ e2 ^ (s) : NumTC ]].
43 [[ Sub(e1, e2) ^ (s) : NumTC ] :=
44   [[ e1 ^ (s) : NumTC ]],
45   [[ e2 ^ (s) : NumTC ]].
46 [[ Div(e1, e2) ^ (s) : NumTC ] :=
47   [[ e1 ^ (s) : NumTC ]],
48   [[ e2 ^ (s) : NumTC ]].
49
50 [[ Eq(e1, e2) ^ (s) : BoolTC ] :=
51   [[ e1 ^ (s) : NumTC ]],
52   [[ e2 ^ (s) : NumTC ]].
53 [[ Lt(e1, e2) ^ (s) : BoolTC ] :=
54   [[ e1 ^ (s) : NumTC ]],
55   [[ e2 ^ (s) : NumTC ]].
56
```

```
transform.str
1 module transform
2
3 imports
4   nabl2shared
5   nabl2runtime
6   transform/desugar
7   statics/calc
8   pp
9
10 rules // Desugaring
11
12 // Desugar
13 desugar-atom: (selected, .., .., path
14   with filename := <guarantee-extens
15   ; result := <desugar-calc> se
16
17 // Desugar and pretty-print
18 desugar-pp: (selected, .., .., path, s
19   with filename := <guarantee-extens
20   ; result := <desugar-calc>; pp
21
```

```
evals
1 Add(Num(x), Num(y)) --> Num(oddB(1, j))
2
3 Lt(Num(x), Num(y)) --> BoolV(LtB(1, j))
4 Eq(Num(x), Num(y)) --> BoolV(EqB(1, j))
5
6 signature
7   arrows
8   ifc(Value, Exp, Exp) --> Value
9   rules // booleans
10
11 False() --> BoolV(false)
12 True() --> BoolV(true)
13
14 If(v1, e2, e3) --> ifc(v1, e2, e3).
15
16 ifc(BoolV(true), e2, ..) --> e2
17 ifc(BoolV(false), .., e3) --> e3
18
19 rules // variables and functions
20
21 E l- Var(x) --> E[x]
22
23 E l- Fun(x, e) --> CloV(x, e, E)
24
25 E l- Let(x, v1, e2) --> v
26 where E [x l--> v1, E] l- e2 --> v
27
28 App(CloV(x, e, E), v_arg) --> v
29 where E [x l--> v_arg, E] l- e --> v
30
31 signature
32   arrows
33   ifc(Stack) --> Value
34
```

```
java.str
1 module codegen/java
2
3 imports
4   signatures/-
5   nabl2/api
6
7 rules // programs
8
9 program-to-java :
10   Program(stats) --> $[import j
11   import j
12
13   public c
14   public
15   [jst
16   retu
17   ]
18
19
20 with
21   <last> stats => last
22 ; <return-type>; type-to-java
23 ; <stats-to-java> stats => j
24 ; <return-exp>; exp-to-java>
25
26 return-type :
27   Bind(_, e) --> ty
28   with
29     <nabl2-get-ast-types> e =>
30
31 return-type :
32   Exp(e) --> ty
33   with
34
```

```
mortgage.calc
1
2
3 rY = 0.017; // yearly interest
4 Y = 30; // number of years
5 P = 379,000; // principal
6
7 N = Y * 12; // number of month
8
9 c = if(rY = 0) // no interest
10   P / N
11   else
12     let r = rY / 12 in
13     let f = (1 + r) ^ N in
14     (P * P * f) / (f - 1);
15
16 c; // payment per month
```

SDF3: Syntax Definition



NaBL2: Static Semantics



DynSem: Dynamic Semantics



Stratego: Program Transformation



Programming Environment

Syntax Definition in Spoofox

Many syntactic (editor) services from single declarative syntax definition

lexical syntax

```
ID = [a-zA-Z][a-zA-Z0-9]*
```

context-free syntax

```
Exp.Var = <<ID>>
```

```
Exp.Add = <<Exp> + <Exp>> {left}
```

```
Exp.Mul = <<Exp> * <Exp>> {left}
```

context-free priorities

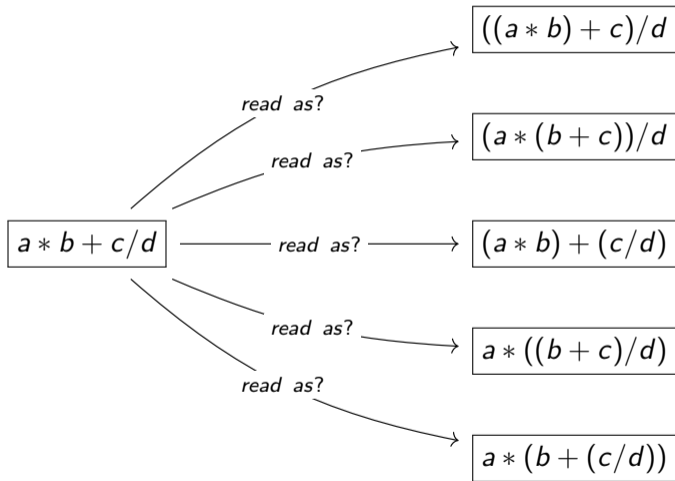
```
Exp.Mul > Exp.Add
```

- ▶ syntax checking
- ▶ error recovery
- ▶ syntax highlighting
- ▶ abstract syntax
- ▶ formatting
- ▶ syntactic completion
- ▶ parenthesis insertion
- ▶ declarative disambiguation

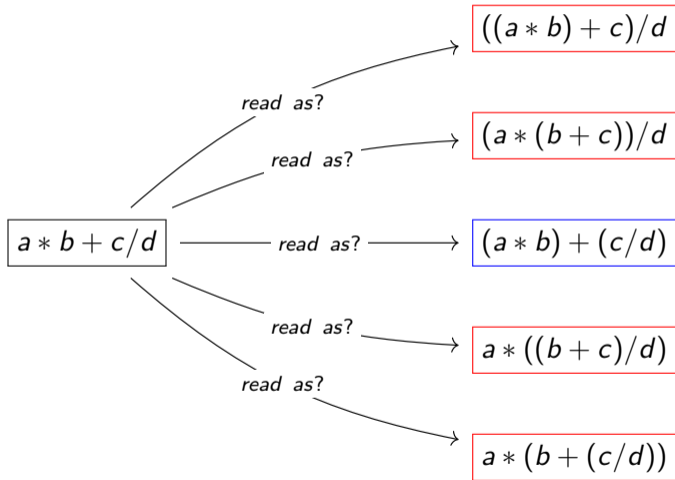
History

- 1997 My PhD thesis with semantics of disambiguation rules for SDF2
- 2011 Peter Mosses observes unsafety of SDF2 rules
- 2013 Afroozeh et al. (SLE'13) define safe disambiguation with grammar transformation; semantics in terms of derivations; no proof of correctness
- 2018 First submission to TOPLAS and implementation of new parser generator for SDF3 integrated in Spoofax
- 2019 Chapter in PhD thesis Eduardo Amorim and major revision for TOPLAS: safe and complete semantics based on subtree exclusion with proof (sketch)
- 2020 Work in progress: TOPLAS 'minor' revision with new approach to proof of safety and completeness

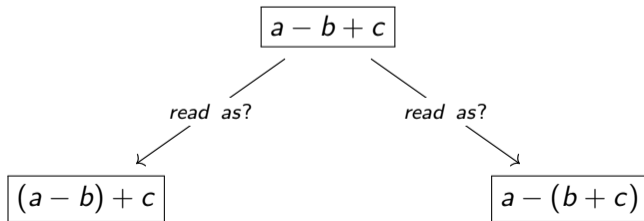
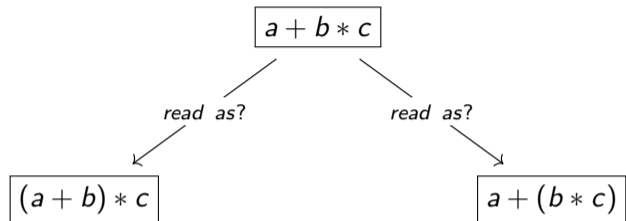
Order of Operations



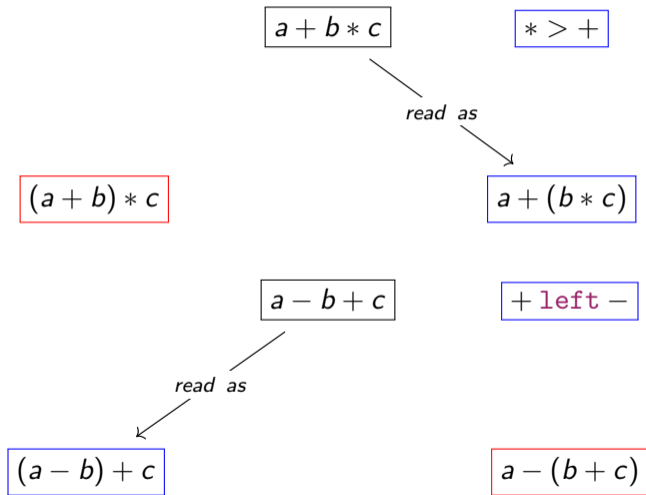
Order of Operations



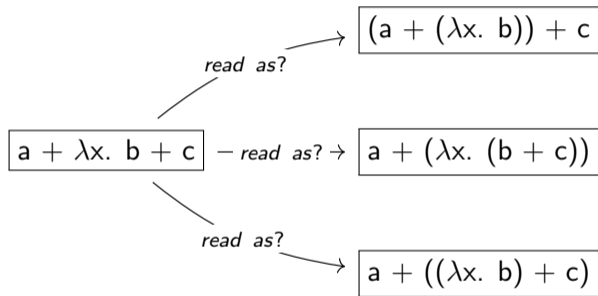
Order of Operations



Associativity and Priority



Order of Operations



$\lambda > +, + \text{left} +$

$\lambda ? +, + ? +$

$\lambda > +, + \text{right} +$

Semantics of Associativity and Priority

In this talk:

- ▶ What is the semantics of associativity and priority rules?
- ▶ Is a set of disambiguation rules safe?
- ▶ Is a set of disambiguation rules complete?
- ▶ How to prove that?

Semantics of Associativity and Priority

In this talk:

- ▶ What is the semantics of associativity and priority rules?
- ▶ Is a set of disambiguation rules safe?
- ▶ Is a set of disambiguation rules complete?
- ▶ How to prove that?

Not in this talk:

- ▶ What classes of ambiguities do associativity and priority rules solve?
- ▶ What is an effective implementation strategy for disambiguation rules?

Semantics of Associativity and Priority

In this talk:

- ▶ What is the semantics of associativity and priority rules?
- ▶ Is a set of disambiguation rules safe?
- ▶ Is a set of disambiguation rules complete?
- ▶ How to prove that?

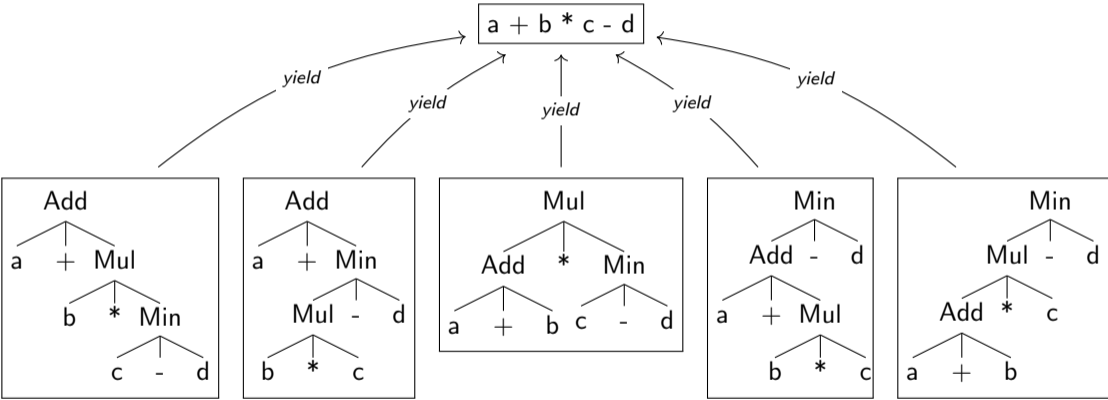
Not in this talk:

- ▶ What classes of ambiguities do associativity and priority rules solve?
- ▶ What is an effective implementation strategy for disambiguation rules?

Why is this not a solved problem?

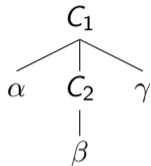
- ▶ Ambiguity of context-free grammars is undecidable; why bother?
- ▶ Existing definitions depend on specific implementations

Ambiguous Sentence has Multiple Parse Trees

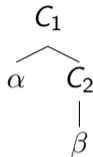


Associativity and Priority as Subtree Exclusion Rules [SDF2 (1997)]

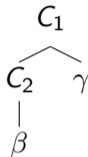
$$\frac{A.C_1 > A.C_2}{C_1}$$



$$\frac{A.C_1 \text{ left } A.C_2}{C_1}$$

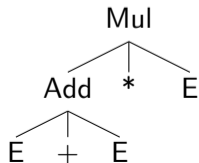


$$\frac{A.C_1 \text{ right } A.C_2}{C_1}$$

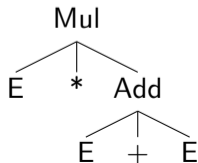


Disambiguation rules generate subtree exclusion patterns (aka conflict patterns)

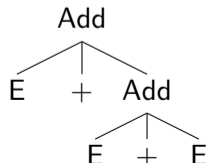
$$\frac{E.Mul > E.Add}{Mul}$$



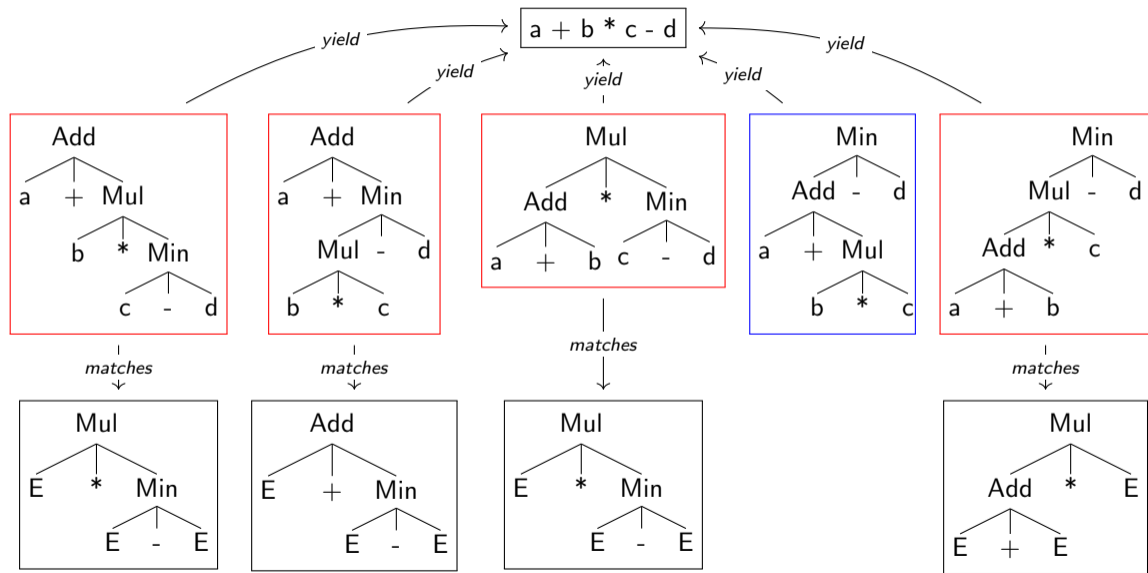
$$\frac{E.Mul > E.Add}{Mul}$$



$$\frac{E.Add \text{ left } E.Add}{Add}$$

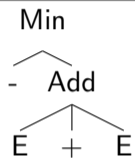


Disambiguation by Subtree Exclusion

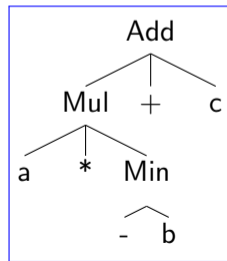
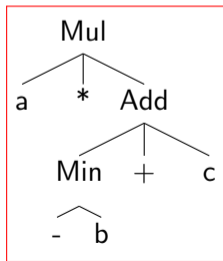
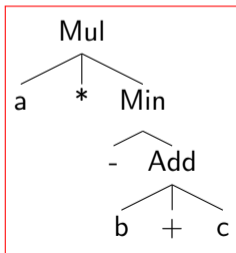
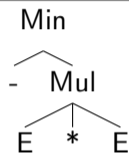


Safe for High Priority Prefix Operators

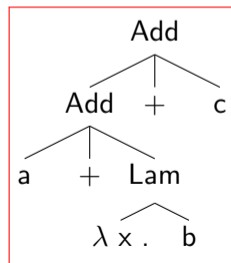
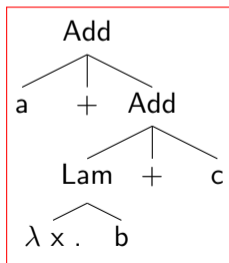
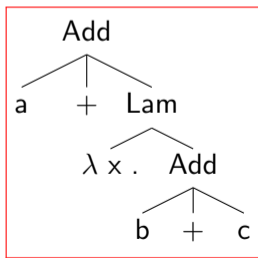
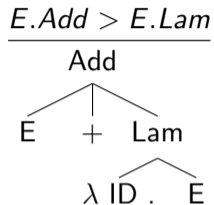
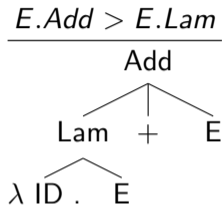
$E.Min > E.Add$



$E.Min > E.Mul$

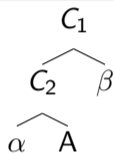


Unsafe for Low Priority Prefix Operators [SDF2]

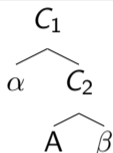


Safe Subtree Exclusion Rules [SDF3 (2019)]

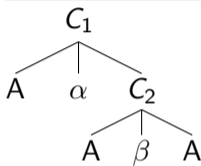
$$\frac{A.C_1 > A.C_2}{C_1}$$



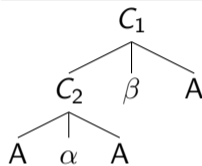
$$\frac{A.C_1 > A.C_2}{C_1}$$



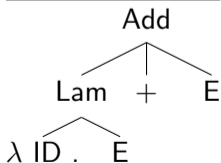
$$\frac{A.C_1 \text{ left } A.C_2}{C_1}$$



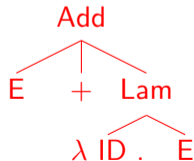
$$\frac{A.C_1 \text{ right } A.C_2}{C_1}$$



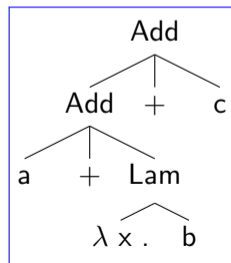
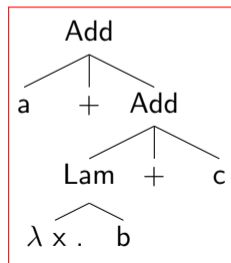
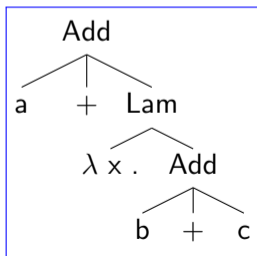
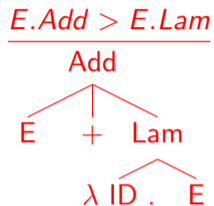
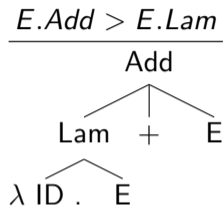
$$\frac{E.Add > E.Lam}{Add}$$



$$\frac{E.Add > E.Lam}{Add} \text{ (not a pattern)}$$

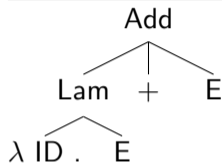


Shallow Interpretation: Safe for Low Priority Prefix Operators

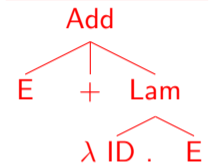


Shallow Interpretation: Incomplete for Low Priority Prefix Operators

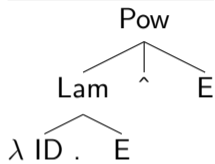
$E.Add > E.Lam$



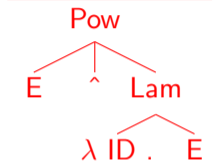
$E.Add > E.Lam$



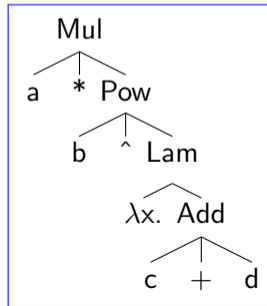
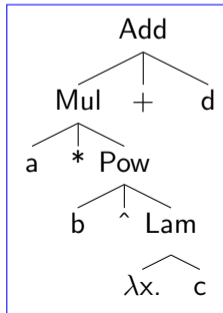
$E.Pow > E.Lam$



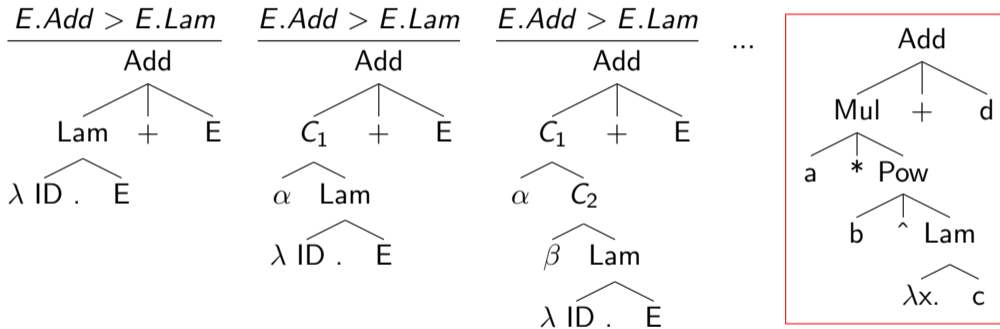
$E.Pow > E.Lam$



$a * b ^ \lambda x. c + d$



Deep Priority Conflicts: Match Subpattern in Right-Most Subtree



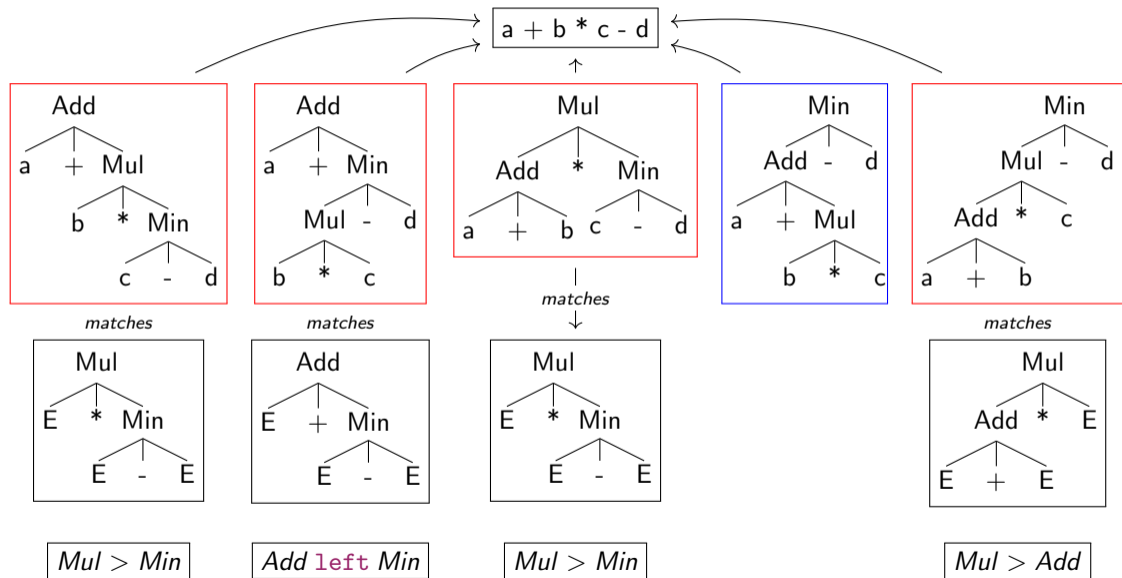
Infinite set of conflict patterns

Semantics of Associativity and Priority

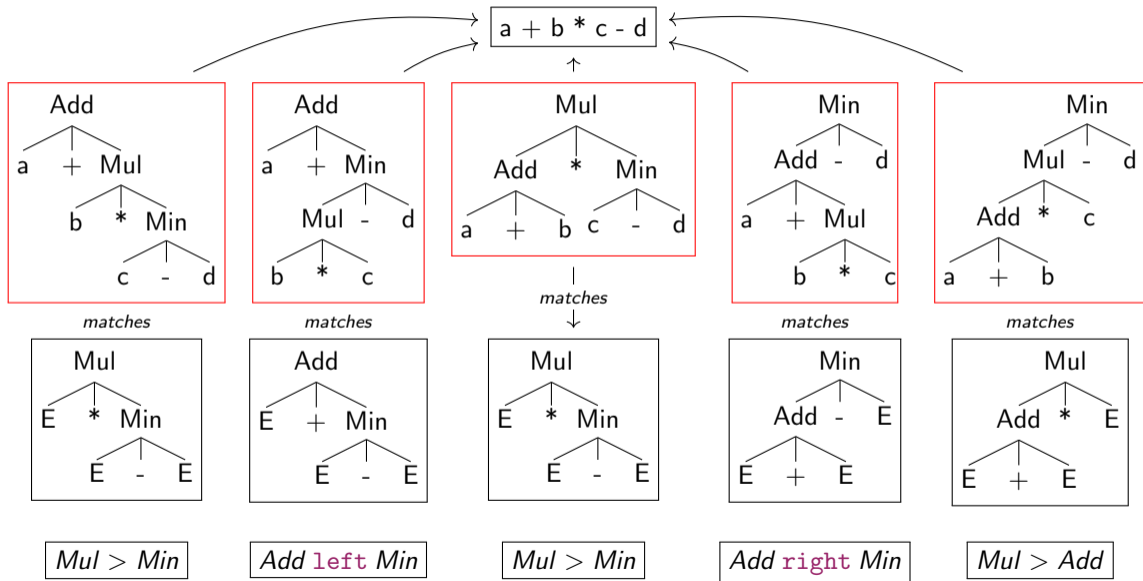
- ✓ What is the semantics of associativity and priority rules?
 - ▶ Integrated in implementation of SDF3 parser generator
 - ▶ Has been available since 2018 in Spoofox
- ▶ Is a set of disambiguation rules safe for a particular grammar?
- ▶ Is a set of disambiguation rules complete for a particular grammar?
- ▶ How to prove that?

Restricting to the case of infix expression grammars for this talk.

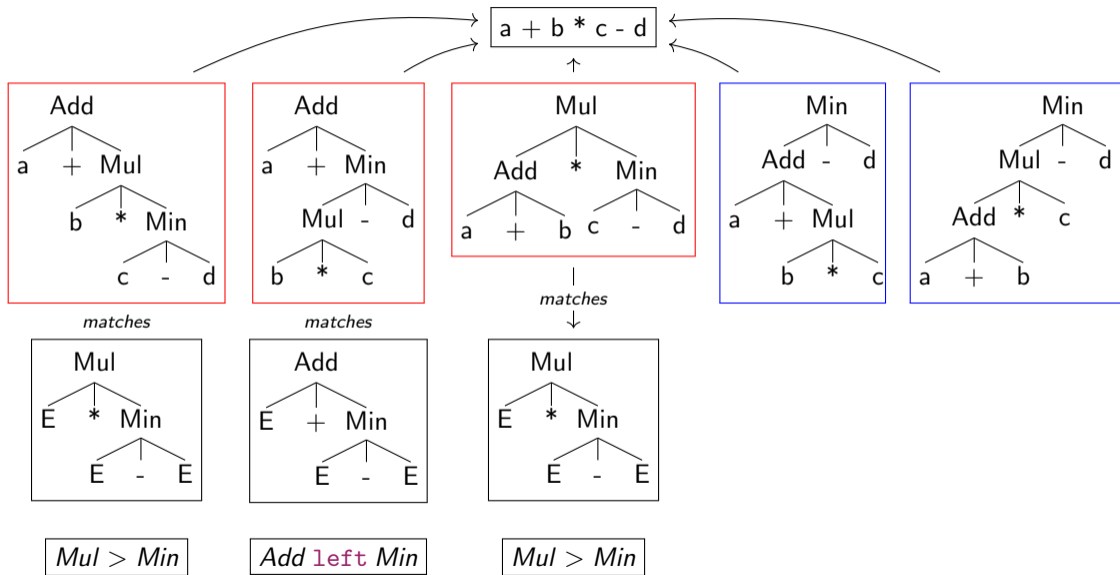
Safe and Complete Disambiguation Rules



Unsafe: Too Many Disambiguation Rules



Incomplete: Too Few Disambiguation Rules



Semantics of Associativity and Priority

- ✓ What is the semantics of associativity and priority rules?
- ✓ Is a set of disambiguation rules safe for a particular grammar?
 - ▶ At most one rule for each pair of productions
 - ▶ + some well-formedness criteria
- ✓ Is a set of disambiguation rules complete for a particular grammar?
 - ▶ At least one rule for each pair of productions
 - ▶ + some well-formedness criteria
- ▶ How to prove that?

Trees under Subtree Exclusion

Definition

A tree $t \in T^Q(G)$ iff $t \in T(G)$ and no subtree of t matches a conflict pattern in Q .

Lemma (Safety)

A disambiguation relation is safe, if for each $w \in L(G)$ there is at least one tree $t \in T^Q(G)$ such that $\text{yield}(t) = w$.

Lemma (Completeness)

A disambiguation relation is complete, if for each $w \in L(G)$ there is at most one tree $t \in T^Q(G)$ such that $\text{yield}(t) = w$.

Essence of the Proof Attempt

To prove safety:

If a tree $t \in T(G)$ has a conflict, then there is another tree for the same sentence that does not have a conflict.

Essence of the Proof Attempt

To prove safety:

If a tree $t \in T(G)$ has a conflict, then there is another tree for the same sentence that does not have a conflict.

To prove completeness:

If a tree $t \in T(G)$ does not have a conflict, then all other trees for the same sentence have conflicts.

Essence of the Proof Attempt

To prove safety:

If a tree $t \in T(G)$ has a conflict, then there is another tree for the same sentence that does not have a conflict.

To prove completeness:

If a tree $t \in T(G)$ does not have a conflict, then all other trees for the same sentence have conflicts.

Subtree exclusion is a statement about a single tree

How do we relate all trees for the same sentence?

Essence of the Proof Attempt

To prove safety:

If a tree $t \in T(G)$ has a conflict, then there is another tree for the same sentence that does not have a conflict.

To prove completeness:

If a tree $t \in T(G)$ does not have a conflict, then all other trees for the same sentence have conflicts.

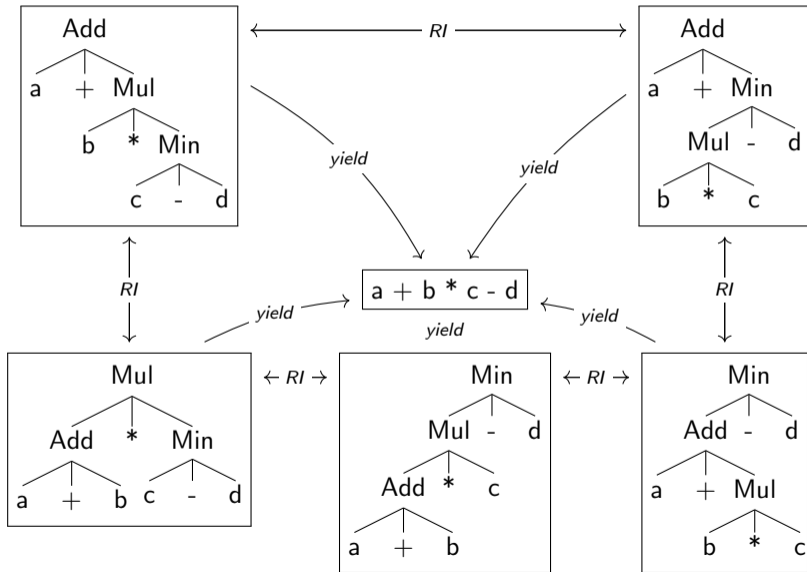
Subtree exclusion is a statement about a single tree

How do we relate all trees for the same sentence?

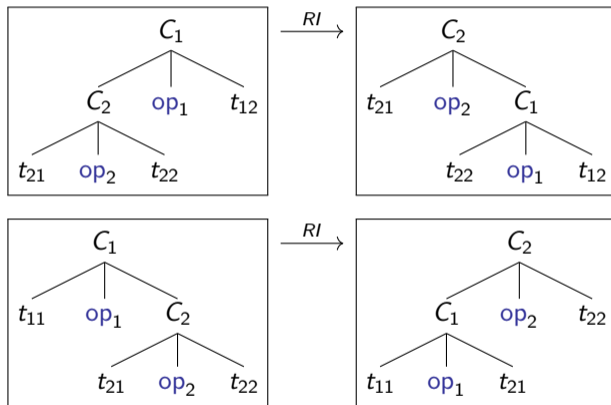
The solution is simple and elegant; once you have seen it you can't unsee it; but for the longest time I didn't see it (nor did co-authors, reviewers, other readers)*.

* But Haskell infix operators are implemented using such reorderings

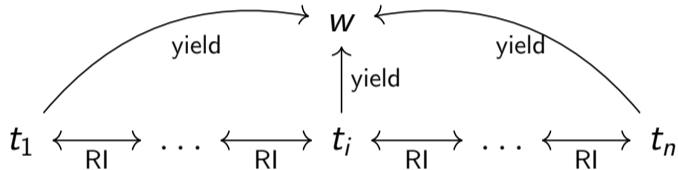
Insight: Trees for Ambiguous Sentence are Rearrangings



Reordering Rewrite System

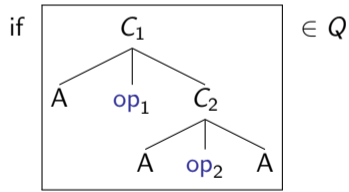
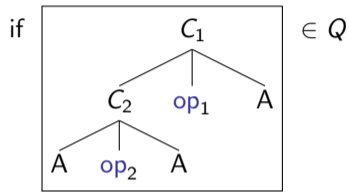
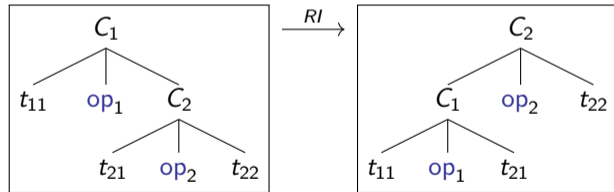
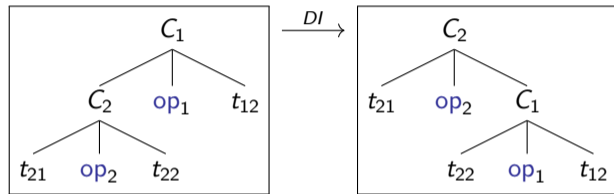


Theorem: Infix Ambiguities are Reorderings

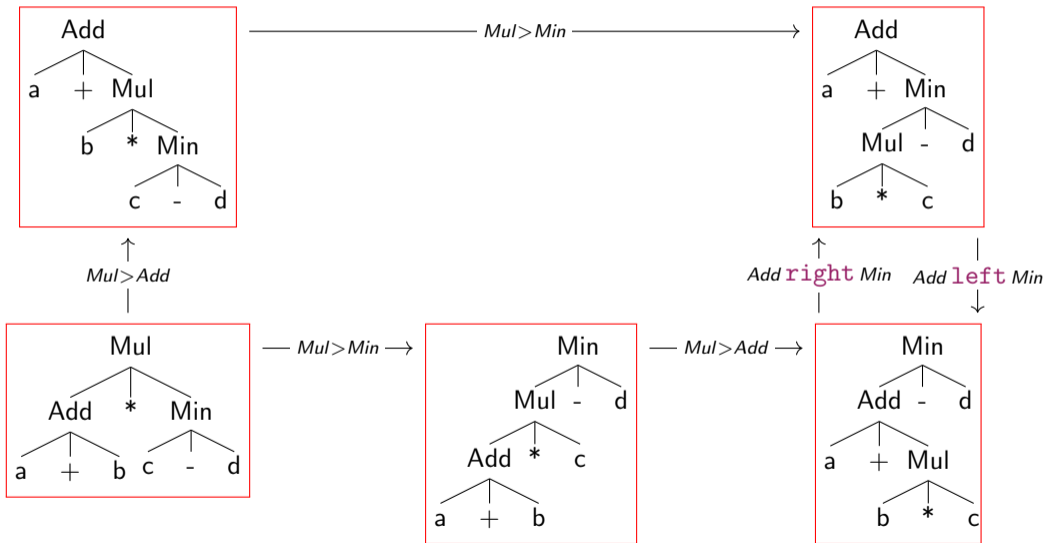


Fine print: for expression grammars beyond infix expression grammars, there are some extra requirements.

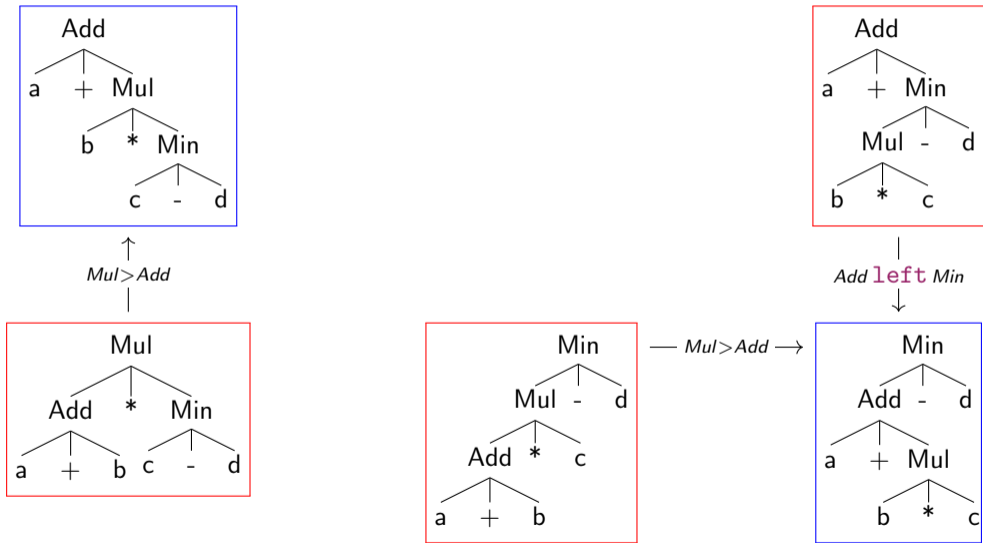
Ordering Reorderings with Conflict Patterns



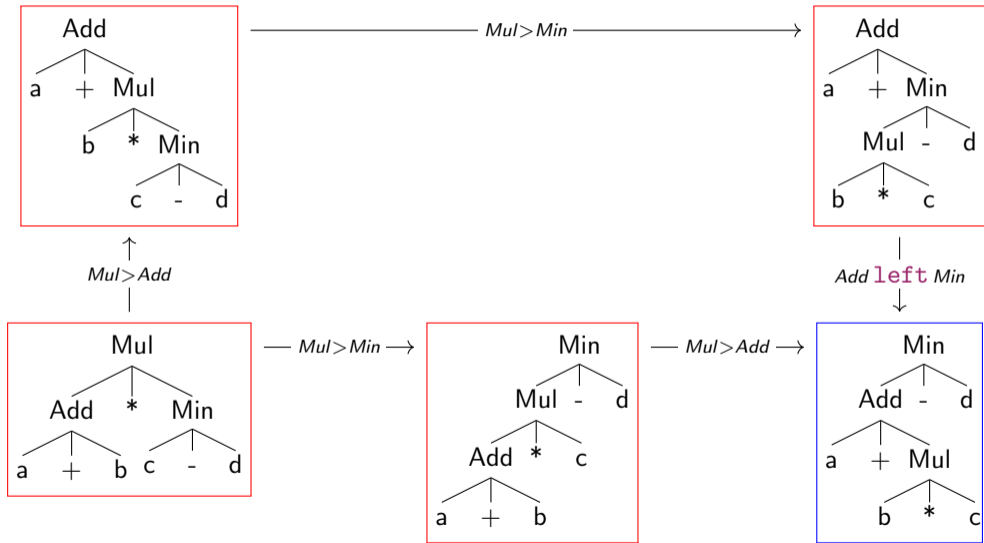
Correspondence: Unsafety is Non-Termination



Correspondence: Incompleteness is Non-Confluence (Non-Church Rosser)



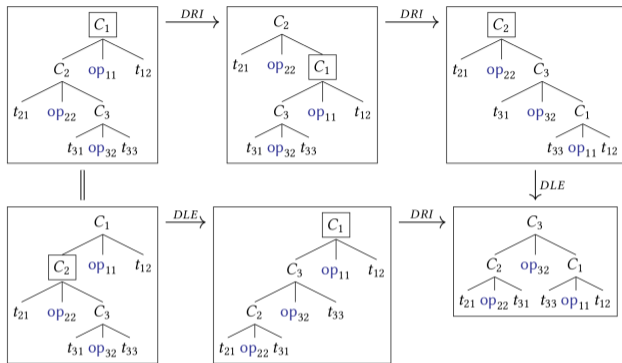
Correspondence: Safety + Completeness is Termination + Confluence



Proving Termination and Completeness

- ▶ Termination = Safety
 - ▶ A safe disambiguation relation induces a reduction order on DI
 - ▶ Roughly: number of conflicts reduces to zero
- ▶ Confluence = Completeness
 - ▶ If disambiguation relation is complete, then DI is locally confluent
 - ▶ DI (infix) has 5 critical pairs
 - ▶ The conditions for the rules in a critical pair + well-formedness criteria on disambiguation relation enable other rules and completion of the diagram
 - ▶ Extension with prefix and postfix operators: 8 rules, 28 critical pairs, 36 cases.
 - ▶ Automated by implementation in Stratego

Proving Local Confluence: Critical Pairs are Joinable



$$(\text{op}_{11} > \text{op}_{22} \wedge \text{op}_{22} > \text{op}_{32} \Rightarrow \text{op}_{11} > \text{op}_{32})$$

$$\vee (\text{op}_{11} > \text{op}_{22} \wedge \text{op}_{22} \text{ left } \text{op}_{32} \Rightarrow \text{op}_{11} > \text{op}_{32})$$

$$\vee (\text{op}_{11} \text{ right } \text{op}_{22} \wedge \text{op}_{22} > \text{op}_{32} \Rightarrow \text{op}_{11} > \text{op}_{32})$$

$$\vee (\text{op}_{11} \text{ right } \text{op}_{22} \wedge \text{op}_{22} \text{ left } \text{op}_{32} \Rightarrow \text{false})$$

Semantics of Associativity and Priority

- ✓ What is the semantics of associativity and priority rules?
- ✓ Is a set of disambiguation rules safe for a particular grammar?
 - ▶ At most one rule for each pair of productions
 - ▶ + some well-formedness criteria
- ✓ Is a set of disambiguation rules complete for a particular grammar?
 - ▶ At least one rule for each pair of productions
 - ▶ + some well-formedness criteria
- ✓ How to prove that?
 - ▶ Disambiguation corresponds to reordering conditional on conflict patterns
 - ▶ Trees under subtree exclusion: normal forms of DI
 - ▶ Safety of disambiguation
 - ▶ DI is terminating iff disambiguation relation is safe
 - ▶ Completeness of disambiguation
 - ▶ DI is confluent iff disambiguation relation is complete

What Else?

- ▶ What class of ambiguities do associativity rules solve?
 - ▶ Short answer: expression grammars for which ambiguities correspond to reorderings
 - ▶ We have investigated several classes of expression grammars: prefix/postfix operators, mixfix grammars, dangling suffix/prefix, indirect recursion, longest match of lists
- ▶ What happened to the undecidability of ambiguity?
 - ▶ Expression grammars without overlap: ambiguities are reorderings
 - ▶ Infix grammars: cannot have overlap
 - ▶ IPP grammars: harmful overlap is decidable (conjecture)
 - ▶ Mixfix grammars: harmful/less overlap undecidable in general
 - ▶ But: need only inspect productions involved in overlap
- ▶ What is an effective implementation strategy for disambiguation rules?
 - ▶ Contextual grammar transformations
 - ▶ Data dependent parsing
- ▶ A full paper is underway
 - ▶ A Direct Semantics for Declarative Disambiguation of Expression Grammars
 - ▶ Under revision for ACM TOPLAS