Parse Forest Disambiguation

Bram van der Sanden

Evaluation committee
Mark van den Brand
Adrian Johnstone
Elizabeth Scott
Tim Willemse
Outline

• Motivation: parsing, ambiguities

• Goal of project and research questions
  • Focus: Ambiguities in expression grammars
  • Questions:
    − Which ambiguities occur in expression grammars?
    − When do they occur?
    − How can we resolve them?

• Resolve ambiguities in parse forest

• Integration of techniques into the parser

• Experimental evaluation

• Conclusions
Motivation: what is parsing?

Grammar in BNF:

E ::= E '//' E
E ::= E '+' E
E ::= E '*' E
E ::= E '//' E
E ::= '(' E ')'
E ::= [0-9]+
Motivation: ambiguity

Grammar in BNF

E ::= E '+' E
E ::= E '*' E
E ::= E '/' E
E ::= '(' E ')'
E ::= [0-9]+  

Input

6/2*(1+2)

Parser Generator

Parser

Output depends on parser:
- Generate all derivations: set of parse trees
- Generate error
- Choose first valid derivation
- Use additional information to select derivation
Motivation: ambiguity
Goal of project

• Disambiguation of expression grammars
• We need: expression grammar with disambiguation rules
• Disambiguation rules specify:
  • Context-sensitive information that guide ambiguity resolving

• Disambiguation rules need:
  • Easy syntax
  • Clear semantics

• Single parse tree as result by discarding incorrect parse trees.
Key questions

• What kind of ambiguities occur in expression grammars and how to resolve them?

• Abstract expression grammars:
  − unary prefix, unary postfix and binary expressions
    -E                        E!                        E+E

• Java expressions

• mCRL2 expressions
  − Expressions like E → E ◊ E (if-then-else construction)

• Resolving by means of **filtering** applied on **parse forests**
generated by generalized parsers
What kind of ambiguities occur?

- **Associativity**
  
  1+2+3; (1+2)+3 or 1+(2+3)?

- **Precedence**
  
  1+2*3;
  
  - (1+2)*3 or
  
  - 1+(2*3)?

- **Dangling else**
  
  E ::= if E then E
  
  E ::= if E then E else E
  
  if a then if b then c else d
  
  if a then
  
  if b then
  
  c
  
  else d
  
  if a then
  
  if b then
  
  c
  
  else d

- **Ambiguities between prefix and postfix:**
  
  - 1+++1 with operators
    
    E+E, E++ and ++E; do we want:
    
    (E++)+E or E+(++E)?
Associativity and Precedence

• When do associativity and precedence conflicts occur?

• Consider the following two grammar rules:
  - \( E ::= E \alpha \) \textit{left-open} \( \alpha, \beta \in (T \cup N)^* \)
  - \( E ::= \beta E \) \textit{right-open}

• Then there are two possible derivations for \( \beta E \alpha \):
  - \( E \Rightarrow E \alpha \Rightarrow (\beta E) \alpha \), and
  - \( E \Rightarrow \beta E \Rightarrow \beta (E \alpha) \)

• For instance: \( E ::= E + E \) and \( E ::= -E \) and input \(-1+1\)
  - \( E \Rightarrow E + E \Rightarrow (-E) + E \Rightarrow (-1) + 1 \)
  - \( E \Rightarrow -E \Rightarrow -(E+E) \Rightarrow -(1+1) \)
Associativity and Precedence

• Specify associativity and precedence

- \( E ::= E + E \)
- \( E ::= E \times E \)
- \( E ::= E \div E \)
- \( E ::= (\ E \ ) \)
- \( E ::= [0-9]+ \)

- \( E ::= E + E \) \{left, 1\}
- \( E ::= E \times E \) \{left, 2\}
- \( E ::= E \div E \) \{left, 2\}
- \( E ::= (\ E \ )\)
- \( E ::= [0-9]+ \)

• Use disambiguation rules to select desired derivation

• By inspecting grammar: look if all ambiguities related to assoc. & prec. are covered.

So, how to implement a filter using these disambiguation rules for disambiguation?
Associativity and Precedence

• Left-open right-open (LORO) filter
  • Looks at precede symbol / follow symbol and corresponding production
  • Node $E ::= E + E$ derives $1 + 2$,
    − Follow symbol: *
    − Follow production: $E ::= E * E$
Associativity and Precedence

- Left-open right-open (LORO) filter
  - Given augmented grammar
    - generate **Precede-restrictions** and **Follow-restrictions**
    - by looking at left-open right-open rules and their assoc./prio.
  - Given \( E + E \ {\text{left, 1}} \) and \( E * E \ {\text{left, 2}} \):
    - Precede restrictions \((E+E, E+E)\) and \((E*E, E*E)\) for associativity
    - Precede restriction \((E+E, E*E)\) and follow restriction \((E+E, E*E)\) for precedence

\[
E ::= E + E \\
E ::= E * E
\]
Associativity and Precedence

- Left-open right-open (LORO) filter
- Given assoc. + prec. rules, look for restriction violations:

  - Input: ! ! n n a | b
  - Precedence: n E > E | E > ! E
  - Follow restriction: (n E, E | E)

Finding follow (or precede) production: path

\[
E ::= n E \quad \{3\} \\
E ::= E | E \quad \{\text{left, 2}\} \\
E ::= ! E \quad \{1\}
\]
Towards forest filtering: What is a parse forest?

- Given some input parse forest contains all corresponding parse trees
  - each tree corresponds to a derivation
  - trees are embedded in the parse forest
- Observations:
  - Parse trees can share subtrees
    - In parse forest: allow a node to have multiple parents
  - Part of the input string can be derived in multiple ways: ambiguities
    - In parse forest: use special type of node, called a packed node, for each derivation
- Cubic size with respect to the input string
- SPPF: Shared Packed Parse Forest
- Generated by generalized algorithms like GLL, GLR, Earley [1]

[1: Elizabeth Scott, SPPF-Style Parsing From Earley Recognisers, (2008)]
SPPF filtering: Associativity and Precedence

• Recall our invalid parse tree
  • $E + E$ nested below $E \times E$
  • Follow restriction: $E + E$ is followed by $E \times E$
SPPF filtering: Associativity and Precedence

- SPPF parse tree instead of parse tree

Packed node: contains production

Intermediate node

Symbol node: stores (non)terminal
SPPF filtering: Associativity and Precedence

- SPPF filtering: remove SPPF parse trees containing specific path

Follow restriction: E+E is followed by E*E
SPPF filtering: removing invalid trees

• Naive implementation:
  • Look at each individual SPPF tree embedded in the SPPF
    - Check whether precede or follow restriction applies
  • Unfeasible when many ambiguities are present
    - For several mCRL2 test files, number of SPPF trees becomes astronomic (quintillions (19 digits or more))

• Smart implementation:
  • Remove all SPPF trees containing certain invalid paths in one go
  • Issue: sharing
SPPF filtering: sharing

- Now consider:
  - $S ::= A \mid B$  
  - $E ::= C \mid D$
  - $A ::= E$
  - $C ::= '1'$
  - $B ::= E$
  - $D ::= '1'$

- Assume production $E ::= D$ below $A ::= E$ not allowed.
SPPF filtering: sharing

Invalid

Valid SPPF trees

(TU/e Technische Universiteit Eindhoven University of Technology)
SPPF filtering

• Issue when filtering SPPF: sharing
  • Need algorithms that remove only invalid parse trees from SPPF
  • In some cases:
    - First split SPPF into multiple copies
    - Remove some edges in each copy
    - Resulting copies do not contain invalid parse tree, but together still contain all valid parse trees
SPPF filtering: sharing

Resulting copies:
Apply filtering at an earlier stage

• Filters are defined at parse forest level
• But why not apply them during parsing if possible?

• Same semantics
• Resolve subset of all ambiguities in parse forest

• Advantage: possibly better performance
  • We avoid the creation of undesired parse trees in the first place

• Case study: GLL parsing algorithm
Ambiguity avoidance during parsing

- Limited form of LORO filter:
  - In GLL during parsing we know:
    - Parent grammar slot & grammar rule
    - Current grammar slot & grammar rule
  - Path: parent rule → current rule
  - Try to avoid creating path if it relates to precede or follow violation

- Filters all ambiguities related to binary operators
- Filters part of ambiguities related to binary and unary operators
Experimental evaluation

• Post-parse filtering:
  • Works well for small files;
  • Large files: lot of SPPF copies, leads to Java runtime exception due to garbage collection
  • Solves all described ambiguities in expression grammars
  • Future work: use efficient sharing mechanism to relate and store copies

• Parsing vs Parsing with parse-time filtering:
  • on average 9.4% higher running time
  • Running time lower for ambiguous files
  • Parse-time filtering solves all ambiguities for binary expressions
  • For mCRL2: in test set only 2 files needed additional post-parse filtering
    - Invalid path of a longer length

• Easy integration of filters into GLL
Conclusions

- Types of ambiguities in expression grammars
- Disambiguation filters defined on parse forest
- SPPF filtering
- Filters can serve as basis for implementation in parser itself
  - Gives significant speedup for files with ambiguities