Deterministic Parsing of Languages With Dynamic Operators

Kjell Post, Allen Van Gelder, James Kerr
UC Santa Cruz
Overview

- Summary of contributions
- Prior and related work
- Operators – definition and properties
- Deferred decision parsing
- Resolution policy
- Definite Clause Grammars
Summary of Contributions

• A parsing methodology to handle dynamic operators.

• Ability to recognize/avoid ambiguous input.

• A parser generator similar to yacc.

• Prototype parsers for Prolog, ML.

• An alternative implementation for Definite Clause Grammars.
Prior and Related Work

- **Earley’s algorithm**: Interpretive and expensive.

- **Tomita’s algorithm**: Language is fixed.

- **Incremental parser generators**: Need to recompute parts of the parse table.

- **Operator precedence parsing**: Cannot handle overloaded operators.

- **EL1, Algol68, Peyton-Jones**: Severe restrictions on operator declarations.

- **read.pl**: Not table driven, difficult to understand, numerous anomalies.
Dynamic Operators:

- Any legal identifier or symbol; even "built-in" operators like "+" and "-" can be redefined.

- The properties of an operator can be changed by the user during parsing of input.

- Operators can act as arguments to other operators.

- Operators can be overloaded, e.g. \(-X-Y\).
Operator Properties

- **Fixity, Scope**

- **Associativity**

\[ \begin{align*}
Left: & \quad X - Y - Z \Rightarrow (X - Y) - Z \\
Right: & \quad X, Y, Z \Rightarrow X, (Y, Z) \\
Non: & \quad X > Y > Z \Rightarrow error
\end{align*} \]
:- op(500, yfx, +). % infix, left-assoc
:- op(400, yfx, *). % infix, left-assoc
:- op(200, yf, *). % postfix, left-assoc

\[
X + Y \ast Z^* : \\
\begin{array}{c}
+ \\
  \ast \\
  X \\
  Y \\
  Z
\end{array}
\]

:- op(700, yf, *).

\[
X + Y \ast Z^* : \\
\begin{array}{c}
+ \\
  \ast \\
  X \\
  Y \\
  Z
\end{array}
\]

\[
X \ast \ast : \\
\begin{array}{c}
\ast \\
  X \\
  \ast \\
  X
\end{array} \text{ or } \\
\begin{array}{c}
\ast \\
  X
\end{array}
\]
\[
\begin{align*}
E & \rightarrow E + T \mid T \\
T & \rightarrow T * F \mid F \\
F & \rightarrow (E) \mid v
\end{align*}
\]
Parser Generator

Parse Table

Semantic Actions

Operator Table

<table>
<thead>
<tr>
<th>Operator</th>
<th>Prefix/Suffix</th>
<th>Precedence</th>
<th>Associativity</th>
</tr>
</thead>
<tbody>
<tr>
<td>+</td>
<td>infix</td>
<td>500</td>
<td>left</td>
</tr>
<tr>
<td>*</td>
<td>infix</td>
<td>400</td>
<td>left</td>
</tr>
</tbody>
</table>

Parser Engine

\[
E \rightarrow E + E \\
E \rightarrow E * E \\
E \rightarrow (E) | v
\]

\( V + V * V \)
\[ E \rightarrow E \text{ op } E \]
\[ E \rightarrow ( E ) \]
\[ E \rightarrow v \]
Resolution of Ambiguities

<table>
<thead>
<tr>
<th>Stack</th>
<th>Input</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>\ldots \alpha ; {op}_A ; \beta</td>
<td>; {op}_B ; \ldots</td>
<td>shift S/reduce \alpha ; {op}_A ; \beta</td>
</tr>
</tbody>
</table>

- If \( {op}_A \) and \( {op}_B \) have equal scope, then
  - If \( {op}_A \) is right-associative, shift.
  - If \( {op}_B \) is left-associative, reduce.

- If \( {op}_A \) is either infix or prefix and has larger scope than \( {op}_B \), shift.

- If \( {op}_B \) is either infix or postfix and has smaller scope than \( {op}_A \), reduce.
Overloading Example

<table>
<thead>
<tr>
<th>Operator Table</th>
</tr>
</thead>
<tbody>
<tr>
<td>+ 500</td>
</tr>
<tr>
<td>+ 500</td>
</tr>
<tr>
<td>* 400</td>
</tr>
<tr>
<td>* 200</td>
</tr>
</tbody>
</table>

**Input:** X * + ...  

**Interpretation 1**  

<table>
<thead>
<tr>
<th>X</th>
<th>*</th>
<th>+</th>
</tr>
</thead>
<tbody>
<tr>
<td>400</td>
<td>infix</td>
<td>500</td>
</tr>
</tbody>
</table>

**Interpretation 2**  

<table>
<thead>
<tr>
<th>X</th>
<th>*</th>
<th>+</th>
</tr>
</thead>
<tbody>
<tr>
<td>400</td>
<td>infix</td>
<td>500</td>
</tr>
</tbody>
</table>

**Interpretation 3**  

<table>
<thead>
<tr>
<th>X</th>
<th>*</th>
<th>+</th>
</tr>
</thead>
<tbody>
<tr>
<td>400</td>
<td>infix</td>
<td>500.1</td>
</tr>
</tbody>
</table>

**Interpretation 4**  

<table>
<thead>
<tr>
<th>X</th>
<th>*</th>
<th>+</th>
</tr>
</thead>
<tbody>
<tr>
<td>200</td>
<td>postfix</td>
<td>500</td>
</tr>
</tbody>
</table>

Only interpretation 4 succeeds.
dynop_token(atom(Name), op(Name)).
dynop_token(‚‚, op(‚‚)).

sentences ::= term(T,_), { action(T) }, ‚‚, sentences.
sentences ::= empty.

term(T,n) ::= name(Name), ‚‚, term(T1,C), ‚‚,
{ comma2list(C,T1,A), T =.. [Name|A] }.
term(T,n) ::= op(Name), term(T1,_), { T =.. [Name,T1] }.
term(T,C) ::= term(T1,C1), op(Name), term(T2,C2),
{ T =.. [Name,T1,T2],
(Name = ‚‚, ‚‚ -> C = c(C1,C2); C = n) }.
term(T,n) ::= term(T1,_), op(Name), { T =.. [Name,T1] }.
term(T,n) ::= op(T).
term(T,n) ::= var(T1), { T = ‚$VAR'(T1) }.
term(T,n) ::= name(T).
term(T,n) ::= number(T).
term(T,n) ::= string(T).
term(T,n) ::= ‚‚, term(T,_), ‚‚.
term(T,n) ::= ‚‚, term(T1,_), ‚‚, { T =.. [‘{}’],T1 }.
term(T,n) ::= ‚‚, term(T1,C), ‚‚, term(T2,_), ‚‚,
{ comma2list(C,T1,A), append(A,T2,T) }.
term(T,n) ::= ‚‚, term(T1,C), ‚‚,
{ comma2list(C,T1,T) }.

name(T) ::= atom(T).
name(T) ::= qatom(T).
name([]) ::= ‚‚, ‚‚.
name({}) ::= ‚‚, ‚‚.
Definite Clause Grammars

Syntactic variant of Prolog, in effect a top down backtracking parser.

- Designed for natural language processing (short, ambiguous input sentences).
- A left-recursive production loops.
- Exponential worst case parsing time.
- Requires whole input stream beforehand.
- Backtracking undesirable for certain semantic actions.
- Error handling difficult.
Inherited Attributes

Example, C-declarations: \texttt{int a}

\[
decl ::= \text{type}(T), \text{list}(T).
\]
\[
type(T) ::= \text{int}, \{ T = \text{integer} \}.
\]
\[
type(T) ::= \text{float}, \{ T = \text{float} \}.
\]
\[
list(T) ::= \text{list}(T), ',', name(T).
\]
\[
list(T) ::= name(T).
\]
\[
name(T) ::= \text{ident}(N), \{ \text{install}(N,T) \}.
\]

A bottom-up parser must delay the execution of \texttt{install}(N,T) until \( T \) has been bound.
Summary

- A parsing technique to handle operators.
- Built into an LR-style parser generator.
- Prototype parsers for Prolog and ML.
- Bottom-up implementation for DCGs.

Future Work

- What languages can/cannot be parsed?
- Error handling and recovery
- Inherited attributes.