Discovery of Visible Semantic Predicates Omitted from *LL(*)*: The Foundation of the ANTLR Parser Generator

Authors omitted for blind review

1. Introduction

The formal semantics of predicated grammars and analysis algorithm in the submitted paper require that disambiguating predicates appear at the left edge of ambiguous productions. This is cumbersome in practice and forces programmers to duplicate predicates. Fortunately, grammar analysis can automatically discover and *hoist* predicates from productions further down the derivation chain into parsing decisions without predicates. For example, it's common to define a $Typename$ production that specifies both semantics and syntax:

$$Typename \rightarrow \{isType(next\ symbol)\}?\,\mathbf{id}$$

References to $Typename$ behave as if inlined, automatically making the predicate visible to parsing decisions. If we restrict analysis to $k = 1$ for demonstration purposes, the DFA for rule

$$Decl \rightarrow Typename\,\mathbf{id}\,|\,\mathbf{id}$$

is $D_0 \xrightarrow{\mathbf{id}} D_1$, $D_1 \xrightarrow{isType} f_1$, $D_1 \xrightarrow{!isType} f_2$. Edge $!isType$ is inferred; see function *resolveWithPreds* in Algorithm 5.

Because our DFA construction algorithm operates on grammars that can have predicates and actions anywhere on the right-hand side, hoisting predicates into parsing decisions introduces a semantic hazard. We cannot hoist predicates over actions because they might be a function of that action. For any derivation sequence $(\mathbb{S}, uA\delta) \overset{\vec{\lambda}}{\Rightarrow}{}^{*}(\mathbb{S}', ua\delta')$, $\lambda_i \in \Pi$ is *visible* if $\nexists\, \lambda_j \in \mathcal{M}$ for $j < i$ and $\vec{\lambda} = \lambda_1 \ldots \lambda_n$.

Definition 1. *Semantic predicate transition $q \xrightarrow{\pi} q'$ is* visible *from state $p_{A,i}$ if the ATN can transition from $p_{A,i}$ to q without consuming input and without encountering an action transition. The set of predicates visible between states p and q is:*

$$\begin{aligned} Visible(p, q) = \{\, \lambda_i \mid (\mathbb{S}, p, w, \gamma) &\overset{\vec{\lambda}}{\mapsto}{}^{*} (\mathbb{S}, q, w, \gamma') \text{ where} \\ \lambda_i &\in \Pi \text{ for } i < j \text{ if } \exists\, \lambda_j \in \mathcal{M} \\ \lambda_i &\in \Pi \text{ for } i \leq |\vec{\lambda}| \text{ if } \nexists\, \lambda_j \in \mathcal{M} \} \end{aligned}$$

For example, if p is from position $A \rightarrow \cdot\{\pi_1\}?\,B$ and q is from position $B \rightarrow \{\pi_2\}? \cdot a$ then $Visible(p, q) = \{\pi_1, \pi_2\}$.

At its most complex, the visible semantic context is a "sum of products." For example, in grammar

$A \rightarrow \{\pi_1\}?\,B\,|\,\{\pi_2\}?\,a$
$B \rightarrow \{\pi_3\}?\,a\,|\,\{\pi_4\}?\,a$

A's DFA is $D_0 \xrightarrow{a} D_1$, $D_1 \xrightarrow{(\pi_1\wedge\pi_3)\vee(\pi_1\wedge\pi_4)} f_1$, $D_1 \xrightarrow{\pi_2} f_2$. Our DFA construction algorithm relies on the following definitions to compute semantic context.

Definition 2. Semantic context π *in ATN configuration (q, i, γ, π) is $\pi = \bigwedge Visible(p, q)$ where p is the ATN state derived from alternative i's left edge.*

Definition 3. *The semantic context for alternative i in DFA state D is $\pi = \bigvee_{(\_,i,\_,\pi_j)\in D} \pi_j$*

Definition 4. *Alternative production i is* sufficiently covered *with predicates if we must evaluate a predicate for every derivation leading to an ambiguous sequence $x \in \mathcal{C}(\alpha_i) \cap \mathcal{C}(\alpha_j)$ for $i \neq j$. $Visible(p_{A,i}, q) \neq \emptyset$ for production left edge $p_{A,i}$ and every transition $q \xrightarrow{1:x} q'$ such that $(p_{A,i}, xw, \gamma) \mapsto^{*} (q, xw, \gamma')$.*

For example, the first alternative of nonterminal A in the following grammar is insufficiently covered because it can match ambiguous sequence b without evaluating a predicate via the second alternative of B.

$A \rightarrow B\,|\,\{\pi_1\}?\,b$
$B \rightarrow \{\pi_2\}?\,b\,|\,b\,|\,c$

Specifically, we have $A \Rightarrow B \overset{\pi_2}{\Rightarrow} b$ but also $A \Rightarrow B \Rightarrow b$.

2. DFA construction algorithm with predicate hoisting

Here we present the same DFA construction algorithm as in the submitted paper but with visible predicate hoisting.

As *closure* passes predicates, it "ands" them into new configuration c's semantic context. We do not hoist semantic predicates derived from syntactic predicates in another nonterminal's submachine.

Function *resolveWithPreds* encodes the definitions above. It first collects configurations by conflicting alternative number and then "ors" together predicates associated with each conflicting alternative. If there exists a conflicting alternative that has fewer predicates than configurations, then at least one configuration isn't covered by a predicate (*resolve* reports this later). If there are $n - 1$ predicate contexts for n alternatives, conjure up the n^{th} context as "not the and" of the other contexts. If there are fewer than $n - 1$ predicate contexts, return and indicate we couldn't resolve D. If we have n contexts, choose a representative configuration, c, and set $c.\pi$ to the combined context "or'd" together for c's alternative held in *preds* array.

Alg. 1: *createDFA*(ATN State p_A) **returns** DFA

$work$:= []; Δ := {}; D_0 := {};
$F := \{f_i \mid f_i :=$ new DFA state$, 1 \ldots numAlts(A)\}$;
$Q := F$;
$DFA.p_0 := p_0$; // save ATN start state in DFA
$D_0 := closure(D_0, A, \{(p_{A,i}, i, [], -) \mid$ edge i is $p_0 \xrightarrow{\epsilon} p\}$, **true**);
$work$ += D_0; Q += D_0;
$DFA := DFA(-, Q, T \cup \Pi, \Delta, D_0, F)$;
foreach $D \in work$ **do**
  for *each input symbol* $a \in T$ **do**
    $D' := closure(D, A, move(D, a),$ **false**);
    if $D' \notin Q$ **then**
      $resolve(D')$;
      switch $findPredictedAlt(D')$ **do**
        case *None: work* += D';
        case *Just j:* $f_j := D'$;
      endsw
      Q += D';
    end
    Δ += $D \xrightarrow{a} D'$;
  end
  if *wasResolved(D)* **then**
    foreach $c \in D$ *such that wasResolved(c)* **do**
      Δ += $D \xrightarrow{c.\pi} f_{c.i}$;
    end
  end
  work -= D;
end
return *DFA*;

Algorithm 2: move(DFA State D, $a \in T$) returns set of configurations

return $\{(q, i, \gamma, \pi) \mid (p, i, \gamma, \pi) \in D,\ p \xrightarrow{a} q\}$;

Alg. 3: resolve(DFA State D)

$conflicts$:= the conflict set of D;
if $|conflicts| = 0$ *and not overflowed(D)* **then return**;
$resolved := resolveWithPreds(D, conflicts)$;
if *resolved and insufficientlyCovered(i)* **then**
  report i insufficiently covered with predicates;
if *not resolved* **then**
  resolve by removing all $c \in D$ such that $c.i \in conflicts$ and $c.i \neq min(conflicts)$;
end
if *overflowed(D)* **then** report recursion overflow;
else report grammar ambiguity;

Alg. 4: *closure*(DFA State D, $c = (p, i, \gamma, \pi)$, boolean collectπ) **returns** set *closure*

if $c \in D.busy$ **then return** {}; **else** $D.busy$ += c;
closure := $\{c\}$;
if $p = p'_A$ *(i.e., p is stop state)* **then**
  if $\gamma = p'\gamma'$ **then**
    closure += $closure(D, (p', i, \gamma', \pi), collect\pi)$;
  else
    closure += $\bigcup_{\forall p_2 \,:\, p_1 \xrightarrow{A} p_2 \in \Delta_M} closure(D, (p_2, i, [], \pi), collect\pi)$;
end
foreach *transition t emanating from ATN state p* **do**
  switch t **do**
    case $p \xrightarrow{\pi_{A'}} q$ *transition and A' is synpred:*
      // make sure pred is not syn pred in another rule
      if *collectπ and* $(\mathbb{S}, DFA.p_0, w, \gamma) \mapsto^* (\mathbb{S}, p, w, \gamma)$ **then** $\pi' := \pi \wedge \pi_{A'}$;
      else $\pi' := \pi$;
      add $closure(D, (q, i, \gamma, \pi'), collect\pi)$ to *closure*;
    case $p \xrightarrow{\pi_p} q$ *transition:*
      if *collectπ* **then** $\pi' := \pi \wedge \pi_p$;
      else $\pi' := \pi$;
      add $closure(D, (q, i, \gamma, \pi'), collect\pi)$ to *closure*;
    case $p \xrightarrow{A} p'$ *transitions to nonterminal A:*
      depth := number of occurrences of A in γ;
      if $depth = 1$ **then**
        add i to $D.recursiveAlts$;
        if $|D.recursiveAlts| > 1$ **then**
          throw *LikelyNonLLRegularException*;
      end
      if $depth \geq m$, *the max recursion depth* **then**
        mark D to have recursion overflow;
        return *closure*;
      closure += $closure(D, A, (p_A, i, p'\gamma, \pi), collect\pi)$;
    case $p \xrightarrow{\mu} q$, $p \xrightarrow{\epsilon} q$:
      closure += $closure(D, A, (q, i, \gamma, \pi), collect\pi)$;
  endsw
end
return *closure*;

Alg. 5: *resolveWithPreds*(DFA State D, set $conflicts$)
returns boolean

$preds := []$; // $preds[i]$ is predicate for alt i
$configs := []$; // configurations for alt i
foreach $i \in conflicts$ **do**
    $configs[i] := \{c \in D \mid c.i = i\}$;
    $preds[i] := \bigvee_{c \in configs[i]} c.\pi$;
    if $0 < |preds[i]| < |configs[i]|$ **then**
        mark alt i as *insufficiently covered*;
end
if $|preds| < |conflicts| - 1$ **then return false**;
if $|preds| = |conflicts| - 1$ **then**
    let j be the alt with missing predicate;
    $preds[j] = \neg(\bigwedge_{i \neq j} preds[i])$; // not the others
end
foreach $i \in conflicts$ **do**
    remove all but one representative $c = (\_, i, \_, \pi) \in D$;
    $c.\pi := preds[i]$; // reset to combined preds
    mark c as *wasResolved*;
end
mark D as *wasResolved*;
return true;