Shift-Reduce Parser

Idea: We delay the decision whether to reduce until we know, whether the input matches the right-hand side of a rule.

Construction: Shift-Reduce parser \( M^R \)

- The input is shifted successively to the pushdown.
- If there is a complete right-hand side (a handle) atop the pushdown, it is replaced (reduced) by the corresponding left-hand side.

Shift-Reduce Parser

Construction:

In general, we create an automaton \( M^R_0 = (Q, T, A, q_0, F) \) with:
- \( Q = \{ q, \epsilon \} \cup \{(q, f) \mid (q, f) \) finish \};
- Transitions:
  \( q = \{ q, f \} \cup \{ (q, f), A \rightarrow \alpha \} \cup \) Shift transitions
  \( \{(q, f), A \rightarrow \alpha \} \cup \) Reduce transitions
  \( \{ (q, f), \alpha \} \) finish

Example-computation:

\[
\begin{align*}
(q_0, A) & \rightarrow (q_0, A, b) \\
(q_0, A, b) & \rightarrow (q_0, A, b) \\
(q_0, A, b) & \rightarrow (q_0, A, b) \\
(q_0, A, b) & \rightarrow (q_0, A, b) \\
(q_0, A, b) & \rightarrow (q_0, A, b) \\
(q_0, A, b) & \rightarrow (q_0, A, b) \\
(q_0, A, b) & \rightarrow (q_0, A, b) \\
(q_0, A, b) & \rightarrow (q_0, A, b) \\
\end{align*}
\]

The Pushdown During an RR-Derivation

Idea: Observe a successful run of \( M^R_0 \)

Input: counter \( = 2 + 40 \)

Pushdown: \( \{ \text{name} \} \)

Result:

Viable Prefixes and Admissible Items

Formalism: use Items as representations of prefixes of righthand-sides

Generic Agreement

In a sequence of configurations of \( M^R_0 \)

\[
(q_0, A, \alpha, v) \rightarrow (q_0, A, \alpha, v) \rightarrow (q_0, S, v)
\]

we call \( \alpha \gamma \) a viable prefix for the complete item \( [B \rightarrow \rightarrow \gamma] \).

Reformulating the Shift-Reduce-Parsers main problem:

Find the items, for which the content of \( M^R_0 \)’s stack is the viable prefix.

\[ \rightarrow \) Admissible Items

Admissible Items

The item \( [B \rightarrow \rightarrow \gamma] \) is called admissible for \( \alpha \gamma \) iff \( S \rightarrow \gamma \) in \( B \rightarrow \rightarrow \gamma \):

Characteristic Automaton

An automaton...

- consuming pushdown symbols, i.e. prefixes of righthand-sides of productions expanding from \( S \)
- tracing admissible items in its states
Characteristic Automaton

Observation:
One can now consume the shift-reduce parser's pushdown with the characteristic automaton. If the input $(A \cup \mathcal{T})^*$ for the characteristic automaton corresponds to a viable prefix, its state contains the admissible items.

States: Items
Start state: $(S' \rightarrow \bullet )$
Final state: $(F \rightarrow \gamma \in P) 
Transitions:
(1) $(A \rightarrow \alpha \gamma \beta, X, \{ A \rightarrow \alpha X \beta \}), X \in (A \cup \mathcal{T}), A \rightarrow \alpha X \beta \in P$
(2) $(A \rightarrow \alpha \beta, B \rightarrow \gamma \beta), A \rightarrow \alpha \beta, B \rightarrow \gamma \in \mathcal{T}$

The automaton $c(G)$ is called characteristic automaton for $G$.

Canonical LR(0)-Automaton

The canonical LR(0)-automaton $LR(0)$ is created from $c(G)$ by:
1. performing arbitrarily many epsilon-transitions after every consuming transition
2. performing the powerset construction
3. Idea: or rather apply characteristic automaton construction to powersets directly?

... for example:

LR(0)-Parser

Idea for a parser:
- The parser manages a viable prefix $\alpha = X_1 \ldots X_n$, on the pushdown and uses $LR(0)$ to identify reduction spots.
- It can reduce with $A \rightarrow \gamma$, if $[A \rightarrow \gamma]$ is admissible for $\alpha$

Optimization:
We push the states instead of the $X_i$ in order not to process the pushdown's content with the automaton anew all the time.
Reduction with $A \rightarrow \alpha$ leads to popping the uppermost $|\gamma|$ states and continue with the state on top of the stack and input $A$.

Attention:
This parser is only deterministic, if each final state of the canonical LR(0)-automaton is conflict free.

LR(0)-Parser

... we observe:
The final states $q_0 \rightarrow 1$, $q_9 \rightarrow 12$, $q_{10} \rightarrow 17$ contain more than one admissible item $\Rightarrow$ non-deterministic!

Correctness:

we show:
The accepting computations of an LR(0)-parser are one-to-one related to those of a shift-reduce parser $LR_{\mathcal{D}}$. 

we conclude:
- The accepted language is exactly $L(G)$
- The sequence of reductions of an accepting computation for a word $w \in \mathcal{T}$ yields a reverse rightmost derivation of $w$ for $G$. 

LR(0)-Parser

The construction of the LR(0)-parser:

States: $Q \cup \{ \text{f} \}$ (f fresh)
Start state: $q_0$
Final state: $q_f$
Transitions:
Reduce: $(q_0, \ldots, q_n, \gamma, q', \text{f})$ if $[A \rightarrow \alpha X \beta] \in q_n, \gamma \notin [A \rightarrow \alpha A \beta]$
Finish: $(q_0, \gamma, \text{f})$ if $[S' \rightarrow \alpha] \in P$

with the canonical automaton $LR(0) = (Q, T, \Delta_q, q_0, F)$.

LR(0)-Parser

Attention:
Unfortunately, the LR(0)-parser is in general non-deterministic.

We identify two reasons for a state $q \in Q$:

Reduce-Reduce-Conflict:

$A \rightarrow \alpha \gamma$

$\alpha \rightarrow \gamma'$

$A \neq A' \neq \gamma \neq \gamma'$

Those states are called LR(0)-unsatisfied.
Revisiting the Conflicts of the LR(0)-Automaton

What differentiates the particular Reductions and Shifts?

Input:

Pushdown: ( w T )

E → E + T | T
T → T * F | F
F → ( E ) | int

LR(k)-Grammars

for example:

(1) S → A | B
   A → a A | 0
   B → a B b | 1

... is not LR(0), but LR(0):

Let S → α X w → α β w. Then α β w is one of these forms:

α | α B + α | α B B
α A | α B B | α B B B
... (n \geq 0)

(2) S → a A c
   A → A b b | b

... is also not LR(k) for any \k \geq 0

Let S → α X w → α β w. Then α β w is one of these forms:

α | α B
α B | α B B
α B B | α B B B
... (n \geq 0)

LR(1)-Parsing

for example:

The LR(1)-Item [B→γ•β, x] is admissable for αγ if:

S→∗
RαBw → αβw with {y}= Firstk(w) ...

... is not LR(k) for any \k \geq 0:

Consider the rightmost derivations:

S→∗
RabnAbnc→abnbbnc

LR(k)-Grammars

for example:

(3) S → a A c
   A → b A b | b

... is not LR(k), but LR(1):

Let S→∗
RαXw → α β w with {y}= Firstk(w) then α β w is one of these forms:

α | α B + α | α B B
α A | α B B | α B B B

(4) S → a A c
   A → a A b b | b

... is not LR(k) for any \k \geq 0:

Consider the rightmost derivations:

S→∗
RabnAbnc→abnbbnc

LR(k)-Grammars

for example:

The reduced contextfree grammar \G is called LR(k)-grammar, if

S→α, \alpha \beta w \in \G \rightarrow α \beta w

S→α, \alpha \beta w \in \G \rightarrow α \beta w

... follows: α = α’ ∧ β = β’ ∧ A = A’

Discussion:

• In the example, the number of states was almost doubled and it can become even worse

• The conflicts in states \{q1, q2, q3\} are now resolved!

• e.g. we have:

\{q1, q2, q3\} ⊆ \{q, e\} ...
The The Action Table:

During practical parsing, we want to represent states just via an integer id. However, when the canonical LR(1)-automaton reaches a final state, we want to know **how to reduce/shift**. Thus we introduce...

The construction of the action table:

* Type: action = Q × T → LR(1)-items ∪ {error}
* Reduce: action[q, w] = [A → β] if $A → β, w ∈ q$
* Shift: action[q, w] = $s$ if $A → β, w ∉ q, w ∈ \text{First}(β) ⊆ q$ (w)
* Error: action[q, w] = error else

**The LR(1)-Parser:**

The construction of the LR(1)-parser:

States: $Q \cup \{f\}$ (f fresh)
Start state: $q_0$
Final state: $\{f\}$
Transitions:
- Shift: action[q, w] = s if $A → β, w ∉ q, w ∈ \text{First}(β) ⊆ q$
- Reduce: action[q, w] = r if $A → β, w ∉ q$ and possible lookahead $w$
- Error: action[q, w] = error else

The Canonical LR(1)-Automaton

In general:

**Reduce-Reduce-Conflict:**

We identify two conflicts for a state $q ∈ Q$:

- $A → γ, A' → γ', q' ∈ Q$

**Shift-Reduce-Conflict:**

- $A → γ, A' → γ, q' ∈ Q$

Such states are called LR(1)-unsuitable.

**Theorem:**

A reduced context-free grammar $G$ is called LR(1) if the canonical LR(1)-automaton $LR(G, 1)$ has no LR(1)-unsuitable states.

What if precedences are not enough?

Many parser generators give the chance to fix Shift-/Reduce-Conflicts by patching the action table either by hand or with token precedences.

... for example:

```
S → E E | E T E | E S E
T → int | + E
E → ( E ) | E int
```

**Shift-Reduce Conflict in state 1:**

```
[E → E → E + int]
[E → E + E E → E + E + int]
```

**Shift-Reduce Conflict in state 2:**

```
[E → E + int]
[E → E + E]
```

What if precedences are not enough?

In practice, LR(k)-parser generators working with the lookahead sets of sizes larger than $k = 1$ are not common, since computing lookahead sets with $k > 1$ blows up exponentially. However:

- there exist several practical LR(k) grammars of $k > 1$,
  e.g. Java 1.6+ (LR(3)), ANSI C, etc.
- often, more lookahead is only exhausted locally
- should we really give up, whenever we are confronted with a Shift-/Reduce-Conflict?

**Theorem:** $LR(k)$-to-LR(1)

Any LR(k) grammar can be directly transformed into an equivalent LR(1) grammar.


---

The LR(1)-Parser:

Possible actions are:

- shift $(A → γ) // Shift-operation$
- reduce $(A → γ) // Reduction with callback/output$
- error $// Error$

The action-table describes for each state $q$ and possible lookahead $w$ the necessary action.
Example cont’d:

$$S \rightarrow A' b 0 \mid B' c 1$$

$$A' \rightarrow a A' 0 \mid a b 1$$

$$B' \rightarrow a B' 0 \mid a b 1$$

$$S$$ rightmost-derives one of these forms:

$$a^n b a^n b a^n b a^n b \Rightarrow LR(1)$$

---

Example 2:

$$S \rightarrow b S S 0 \mid a 1 \mid a a c 2$$

$$S$$ rightmost-derives these forms among others:

$$b S S, b S a, b S a a c, b a, b a a c, ... \Rightarrow min. LR(2)$$

In LR(1), you will have (at least) Shift/Reduce Conflicts between the items [$$a \rightarrow a$$] and [$$a \rightarrow ac$$].

If [$$S \rightarrow a$$]’s right context is a nonterminal ⇒ perform Right-context-extraction.

LR(2) to LR(1)

Example 2 finished:

With fresh nonterminals we get the final grammar:

$$S \rightarrow b S S 0 \mid a 1 \mid a a c 2$$

$$S \rightarrow b C A, 0 \mid b S b B, 1 \mid a 2 \mid a a c 3$$

$$A \rightarrow \epsilon 0 \mid a c 1$$

$$B \rightarrow C A, 0 \mid S b B, 1$$

$$C \rightarrow b C D 0 \mid b S b E, 1 \mid a a 2 \mid a a c 3$$

$$D \rightarrow a 0 \mid a c a 1$$

$$E \rightarrow C D, 0 \mid S b E, 1$$

---

Example 2 cont’d:

If [$$a \rightarrow a$$]’s right context is now terminal a ⇒ perform Right-context-propagation.

LR(2) to LR(1)

Example 2 finished:

With fresh nonterminals we get the final grammar:

$$S \rightarrow b S S 0 \mid a 1 \mid a a c 2$$

$$S \rightarrow b C A, 0 \mid b S b B, 1 \mid a 2 \mid a a c 3$$

$$A \rightarrow \epsilon 0 \mid a c 1$$

$$B \rightarrow C A, 0 \mid S b B, 1$$

$$C \rightarrow b C D 0 \mid b S b E, 1 \mid a a 2 \mid a a c 3$$

$$D \rightarrow a 0 \mid a c a 1$$

$$E \rightarrow C D, 0 \mid S b E, 1$$