

TDDD14 / TDDD85 – Lecture 13

LR(1) parsing

August Ernstsson, 2026 (based on lecture notes by Jonas Wallgren)

Let's start!

Problematic grammar #1: reduce-reduce conflict

- This is a grammar for a simple language:

$S \rightarrow (A) \mid (B)$

$A \rightarrow \text{char} \mid \text{integer} \mid \text{id}$

$B \rightarrow \text{float} \mid \text{double} \mid \text{id}$

- Let's try to construct the LR(0)-item automaton for the grammar. **(On board)**
 - The first state contains the items with the dot first for the two rules for the start symbol. The only thing to do is to read a left parenthesis.
 - That takes us to the next state. The dots are moved over the parentheses. Now since there is a dot in front of A and B, the other six items are added to the state. One possible way to continue to a new state is to read *id* and go to the third state.
 - Now, there is a problem: Which rule to use for reduction? $A \rightarrow \text{id}$ or $B \rightarrow \text{id}$?
- *If an LR(0)-item automaton shows reduce-reduce conflict(s), the grammar isn't LR(0).*

Problematic grammar #2: shift-reduce conflict

- This is a grammar for a language of strings like "a – b – a – b":

$$S \rightarrow L$$
$$L \rightarrow L - E \mid E$$
$$E \rightarrow a \mid b$$

- An initial attempt to construct the LR(0)-item automaton for the grammar (**On board**)
 - The first item for the first rule is found on top in the first state.
 - Since there is a dot in front of **L**, the next two items are added.
 - Since there now is a dot in front of **E** the last two items are added. One of the possible things to do is, as a result of a *reduce* action, to move the dot over **L** and go to the next state.
 - What to do in that state? There is a possibility to read "–" and there is a possibility to reduce.
- *If an LR(0)-item automaton shows shift-reduce conflict(s), the grammar isn't LR(0).*

Conflicts

- **Definition 1.**
*A reduce-reduce conflict is a situation where there are (at least) **two complete items** in a state in the item automaton.*
- **Definition 2.**
*A shift-reduce conflict is a situation where there are (at least) **one complete item** and **one not-complete item** in a state in the item automaton.*

LR(1) items

- The 1 stands for one token lookahead.
- Now, in LR(1) we can take into account what follows A.
- **Definition 4.** *An LR(1) item is an LR(0) item together with lookahead tokens in a set.*
 - \$ is seen as a token here.
 - An example of an LR(1) item is $L \rightarrow L \cdot - E \quad \{\$, -\}$

LR(1) items, cont.

- **Definition 5.** Let $a \in \Sigma$.
 - An item $A \rightarrow \alpha \cdot \beta \{a\}$ is valid for a viable prefix $\delta\alpha$ if $S \stackrel{*}{\Rightarrow}_{\text{rm}} \delta A a w \Rightarrow \delta\alpha\beta a w$.
 - The situation is like in LR(0), with the added constraint that a follows what A is derived to.
- **Definition 6.** An item $A \rightarrow \alpha \cdot \beta \{\$ \}$ is valid for a viable prefix $\delta\alpha$ if $S \stackrel{*}{\Rightarrow}_{\text{rm}} \delta A \Rightarrow \delta\alpha\beta$.
- If a state of items in the automata contains several LR(1) items built from the same LR(0) item, they could be collapsed into one LR(1) item with several elements in its lookahead set.
- E.g. $A \rightarrow b \cdot c \{x\}$ and $A \rightarrow b \cdot c \{y\}$ gives $A \rightarrow b \cdot c \{x, y\}$.
- Elements in the lookahead set are elements in the FOLLOW(A) set for the nonterminal in the left-hand side.

Building the automaton

- **On board**

Using the automaton

- Shift actions work as in the LR(0) case.
- Lookahead sets are involved in reduce actions.
- Look at state 3. Now the shift-reduce conflict can be solved.
 - If we see a "–" it is a token that cannot follow S, so it's OK to shift.
 - If we see \$ it cannot be shifted (\$ never can.) but it can follow S, so it is OK to reduce.
- The automaton usually is described/represented by this table: **(On board)**
- The *empty positions* in the table mean error.

Using the automaton, cont.

- So, parsing using an LR(1) table follows this steps:
 - Start in the start state with the start state on the stack.
 - Look at the next token without really reading it.
 - Look up the next action in the table using the state and the token as indices.
 - Perform the action as for LR(0).
 - Iterate this procedure until an Accept action or an error is found in the table.

A hierarchy of grammars / languages

- We have just seen an example of a grammar that **is not** LR(0), but it **is** LR(1).
- The following relations hold between grammar formalisms:
 - LR(0) languages \subset SLR(1) languages \subset LALR(1) languages \subset LR(1) languages.
 - The two in the middle thus have a power in between LR(0) and LR(1).
 - They will be presented in some compiler course.
- Often LR(1) automata/tables become very large in “real” applications.
- SLR(1) and LALR(1) try to solve that problem to some extent, while still having the capability to express normal programming language constructions.

A hierarchy of grammars / languages, cont.

- A relation between grammars:
- LR(0) grammars \subset LR(1) grammars \subset LR(2) grammars \subset LR(3) grammars $\subset \dots$
 - i.e. you can have a grammar that needs 3 tokens lookahead, it is LR(3), and gets a conflict when handled with LR(2).
- The corresponding relation between languages:
- LR(0) languages \subset LR(1) languages = LR(2) languages = LR(3) languages = \dots
 - i.e. if a language has an LR(3) grammar, that grammar can be rewritten to an LR(2) one and even an LR(1) one.
 - It maybe will be much bigger!
 - It is only the step from LR(0) to LR(1) that makes the set of languages larger.

Coming up soon ...

- Next: Victor is back!
 - **Lecture 14**: Turing machines
 - **Lecture 15**: Undecidability
- Later:
 - August: Chomsky hierarchy

Thanks for today!