Documentation

Raiu Madalin-Augustin

Potirniche Daniel-Mihai

Grammar

- Self.non-terminals: set of non-terminals
- Self.terminals: set of terminals
- Self.starting_nt: starting symbol
- Self.productions: productions
- Check_if_grammar_is_enchanced(): checks if the grammar is enhanced
- augment(): transform the grammar into an enhanced grammar by adding a new starting symbol and a new production with the new starting symbol and old starting symbol
- cgh_check(): checks if the grammar is context-free: it has a production with the starting symbol and the left hand-side of all the productions is a single non-terminal

Item:

-LR(0) item

- Self.lhs: left hand-side of the kernel
- Self.rhs: right hand-side of the kernel
- Self.dot_idx: position of the dot in the right hand-side

Action:

- Denotes the type of action for a state, can be: shift, accept, reduce, reduce reduceconflict or shift reduce conflict

State:

- Self.action: state action for the parsing table
- Self.id: number of the state
- Self.closure_items: the items for which the closure is computed in the state
- Self.closure: the items in the state
- set_action(enrichedSymbol): set the action for the state: accept if the state contains the

enriched grammar production and the dot is at the end, reduce if the dot is at the end

and there is only one item, shift if the dot is not at the end in all the items, reduce

reduce conflict if there are more items with dot at the end and shift reduce conflict

otherwise

- get_all_after_dot (): get the symbols that are after the dot in the state

LR:

- Self.grammar: grammar for the parser
- Self.can_col: set of states
- Self.links: list of Connection objects

- Self.parsing_table: parsing table with action and goto for each state

- closure(items): compute the closure of the given items and return a state which

contains the computed items

- goto(state, symbol): compute the closure of all the items from the state that contain

the dot before the given symbol, with the dot shifted one position; if there is already a

state with the resulting closure, return that state

- canonical_collection(): compute the canonical collection of states for the parser

grammar; go through each state and compute new states for all the symbols that have

dot in front of them

- create_parsing_table(): go through each state from the canonical collection and

depending on the type of action for each state, add the corresponding information: for

accept -> only action, for reduce -> action and the number of the production to be used,

for shift -> action and the gotos from that state; raise an exception if the state has a

reduce reduce conflict

- get_state_links (state): get the resulting states from the connections in which the given state is initial

- get_prod_num_sr(state_id): get the production number

used to reduce in the state with the given id

- parse(sequence): parse the given sequence until the sequence is accepted or

there is an error; accept occurs when the state on top of the work stack is the accept

state; when the action is shift, the top of the input stack is shifted and put at the top of

the work stack, the new state is computed; when the action is reduce, the production

with the corresponding number is used to reduce from the top of the work stack and

prepended to the output band; in case of a shift reduce conflict, shift has priority, but if

shifting is not possible, reduce is applied

ParserOutputEntry:

- Represents a row from the parsing tree (table with father sibling relations)

- Self.index: index of the symbol

- Self.symbol: symbol from the node

- Self.parent: parent of the node

- Self.sibling: right sibling of the node

ParserOutput:

- Class to transform the output band in the chosen representation

- Self.output_band: list of production numbers resulted from parsing a sequence

- Self.parsing_tree: list of entries

- Self.grammar: the grammar for which the parser is intended

- has_children(node): check if the node is father for other nodes

- compute_tree(): transform the output band, starting from the grammar's initial

symbol and going through each production from the output band; for each node

compute the father and right sibling, if they exist; if they don't exist, the entry will have -

1 as father and/or sibling