Joseph Chavez
Duc Minh Ngo
Erick Alvarado

CPSC 323 Project 2 Documentation

This LR parser analyzes input strings based on a defined context-free grammar (CFG). The rules include:

- `E → E + T` (rule 1)
- `E → T` (rule 2)
- `T → T * F` (rule 3)
- `T → F` (rule 4)
- `F → ( E )` (rule 5)
- `F → id` (rule 6)

After tokenizing an input string into recognizable tokens (`id`, `+`, `*`, `(`, `)`, `$`), the parser uses a stack initialized with state `0` to track transitions through states and tokens. Each state has a corresponding function implementing actions defined in the LR parsing table. The state functions handle three primary actions:

- Shift: Adds the current token and next state to the stack.
- Reduce: Replaces a token sequence on the stack with a non-terminal symbol based on grammar rules.
- Accept: Confirms the successful parsing of the input string.

The main loop iterates through the input tokens while invoking state functions based on the top state on the stack. If parsing encounters an error, an error message indicates the string is not parsable. The output shows the parsing stack trace for each input, allowing traceability. It also generates a parse tree representing the grammar rule reductions.

This program implements an LR parser for tracing input strings over `{id, +, *, ), (}` to determine if they conform to a specified grammar. The grammar rules are hardcoded into variables representing non-terminal replacements during reductions. When the user provides an input string, it's split into identifiable tokens like `id`, `+`, `*`, etc., and the special `$` symbol is appended to signify the end of input.

Each state function, such as `state0` and `state1`, corresponds to a particular state in the LR parsing table. The state functions handle the parsing actions, which include shifting tokens to the stack and moving to the next state, reducing tokens to a non-terminal symbol per grammar rules,

or accepting the input string if it is valid. If an unrecognized pattern is encountered, the parser outputs a "NOT PARSABLE" message.

The main parsing loop iterates over the tokens, calling the appropriate state function based on the stack's top state. As states are processed, the stack dynamically shifts or reduces tokens to match the grammar. The program prints each parsing step, showing the stack's progression and the actions taken. It concludes by indicating whether the input string is accepted or not while displaying the stack implementation throughout.

https://github.com/JosephChavxz/CPSC-323-Project2.git