Cheatography

Lark Cheat Sheet

by erezsh via cheatography.com/61630/cs/15903/

Lark Options

`parser="earley"`	**Earley** - default
`parser="lalr"`	**LALR(1)**
`parser="cyk"`	**CYK**
`lexer="standard"`	Revert to simple lexer
`ambiguity='explicit'`	Return all derivations for Earley
`start="foo"`	Set starting rule
`transformer=...`	Apply transformer to tree (for LALR)
`propagate_positions`	Fill tree instances with line number information
`keep_all_tokens`	Don't remove unnamed terminals
`postlex`	Provide a wrapper for the lexer
`tree_class`	Provide an alternative for `Tree`

Tree Reference

`tree.data`	Rule name
`tree.children`	Rule matches
`print(tree.pretty())`	
`tree.iter_subtrees()`	
`tree.find_data("foo")`	Find by rule
`tree.find_pred(...)`	Find by predicate
`tree1 == tree2`	

Token Reference

`token.type`	Terminal name
`token.value`	Matched text
`token.line`	
`token.column`	
`token.end_line`	
`token.end_column`	
`len(token)`	

Tokens inherit from `str`, so all string operations are valid (such as `token.upper()`).

Grammar Definitions

`rule: ...`	Define a rule
`TERM: ...`	Define a terminal
`rule.n: ...`	Rule with priority n
`TERM.n: ...`	Terminal with priority n
`// text`	Comment
`%ignore ...`	Ignore terminal in input
`%import ...`	Import terminal from file
`%declare TERM`	Declare a terminal without a pattern (used for postlex)

Rules consist of values, other rules and terminals.
Terminals only consist of values and other terminals.

Grammar Patterns

`foo bar`	Match sequence
`(foo bar)`	Group together (for operations)
`foo | bar`	Match one or the other

Grammar Patterns (cont)

`foo?`	Match 0 or 1 instances
`[foo bar]`	Match 0 or 1 instances
`foo*`	Match 0 or more instances
`foo+`	Match 1 or more instances
`foo~3`	Match exactly 3 instances
`foo~3..5`	Match between 3 to 5 instances

Terminal Atoms

`"string"`	String to match
`"string"i`	Case-insensitive string
`/regexp/`	Regular Expression
`/re/imslux`	Regular Expression with flags
`"a".."z"`	Literal range

Tree Shaping

`rule: "foo" BAR`	"foo" will be filtered out
`!rule: "foo" BAR`	"foo" will be kept
`rule: /foo/ BAR`	/foo/ will be kept
`_TERM`	Filter out this terminal
`_rule`	Always inline this rule
`?rule: ...`	Inline if matched 1 child
`foo bar -> alias`	Create alias

Rules are a branch (node) in the resulting tree, and its children are its matches, in the order of matching.
Terminals (tokens) are always values in the tree, never branches.
Inlining rules means removing their branch and replacing it with their children.