SPARSER

version 2.3b

USER DOCUMENTATION

Preliminary edition

January 1995

Copyright © 1992-1996, 2010-2014 David D. McDonald

David D. McDonald, Ph.D.
davidmcdonald@alum.mit.edu

1. Introduction 1
1.1 Capabilities 1
1.2 Overview 1
2. Running Sparser 2
2.1 Calling Sparser 2
2.1.1 Managing file input 3
2.1.2 Terminating early 3
2.2 Installation 4
2.2.1 Directory Structure 4
2.2.2 Loading 4
2.2.3 Creating an image 5
2.3 Lisp Packages 5
3. Labels 7
3.1 Words 7
3.1.1 Case-sensitivity 7
3.1.2 Defining words 8
3.1.3 Accessing words 9
3.1.4 The components of the “word” object 9
3.1.5 Predefined words 10
3.1.5.1 Punctuation 10
3.1.5.2 “Meaningless” characters 11
3.1.5.3 Reserved words 11
3.1.5.4 Spaces 12
3.1.6 Whitespace 12
3.1.7 Deleting words 13
3.1.8 Limits on the length of words 14
3.2 Polywords 14
3.2.1 The appearance of polywords in the chart 15
3.2.2 The fields in a polyword 16
3.2.3 Managing polywords 16
3.3 Categories 17
3.3.1 The components of the category object 17
4. Phrase Structure Rules 18
4.1 Context-free phrase structure rules 18
4.1.2 Terminology 18
4.1.3 “CFR” Objects 19
4.1.4 Defining context-free rules 20
4.1.5 More than two terms on a righthand side 22
4.2 “Referents” in phrase structure rules 25
4.2.1 Referent fields 25
4.2.2 Supplying referents directly with the rule 26
4.2.3 Instructions for calculating referents 27

4.2.4 Global variables for referent calculation 29
4.3 Context-sensitive rules 30
4.3.1 Defining context-sensitive rules 31
4.4 Keeping track of rules 32
4.5 Deleting rules 32
4.6 Rule-sets 33
5. The Chart 35
5.1 The data structure for the chart 36
5.1.1 Extracting information from the chart 36
5.2 Positions 37
5.2.1 The fields in a position object 38
5.2.2 Operations over positions 39
5.3 Edge-vectors 40
5.3.1 The fields of an edge-vector 41
5.4 Edges 42
5.4.1 The fields on an edge 43
5.4.2 Constructing an edge 45
5.5 Chart objects as reusable resources 45
6. Analyzing a text 48
6.0.1 The Analysis Core 48
6.0.2 After-Analysis actions 48
6.1 Initializations 49
6.1.1 Initializing sessions 50
6.1.1.1 Switch settings 50
6.1.2 Initializing runs 51
6.2 The Character buffer 52
6.2.1 State information 52
6.2.2 Special control characters 53
6.2.3 References to character buffer positions 54
6.2.4 Modifying characters' interpretations 54
6.3 The Tokenizer 55
6.3.1 What is a token 55
6.3.1.1 'meaningless' characters 55
6.3.1.2 spaces 56
6.3.1.3 punctuation 56
6.3.2 The data structure for 'tokens' 56
6.3.2.1 Known words 57
6.3.2.2 Digit sequences 57
6.3.2.3 Unknown words 57
6.3.3 Calling the Tokenizer 58
6.4 Populating the chart with terminals 58
6.4.1 "Terminals" 59

6.4.2 The standard procedure 60
6.4.3 The Newline hook 62
6.5 Coordinating word-level actions 63
6.6 Introducing edges over words 65
6.6.1 Dependence on how a word appears in a rule 65
6.6.2 Capitalized words 66
6.6.3 Unknown words 67
6.6.4 Digits 69
6.7 Chart-driven processing 69
6.7.1 Just process terminals 70
6.7.2 The 'All-Edges' parsing protocol 70
6.7.3 The 'top-edges only' protocol 72
7. Hooks for User Code 73
7.1 The "Completion" hook 73
7.2 Special-purpose, word-level parsing 75
7.2.1 Initiating a word-level routine 76
7.2.2 Return values 77
7.2.3 Coordinating the scan 77
8. Tracing Sparser's Operations 79
8.1 Run-time traces 79

1. Introduction

1.1 Capabilities

Sparser is a software system written in the language Common Lisp, consisting of a directory of files for loading into the user's Lisp image, a pre-loaded image that may be launched on its own, or some combination of the two.

At its heart, Sparser is a bottom-up, phrase-structure-based chart parser, optimized for semantic grammars and partial parsing. It has a highly efficient implementation based on resources of reused data structures and the option of an indelible, deterministic parsing algorithm.

Included with Sparser is a notation for constructing a grammar of context free and limited context-sensitive rewrite rules (§4). Phrase structure rules may incorporate user-defined objects and operations over them to be used as the referents of the phrases that the rules construct (§4.2).

Since real texts can pose problems for analysis that are not readily or best analyzed using phrase structure techniques, Sparser provides a high-level, task-oriented programming language for defining grammars using a whole range of techniques. A Lisp application programmer's interface (API) for user-defined parsing networks over terminals and initial non-terminals is provided (§7). User-defined code may be executed upon the completion of rules or words (§7.1). The interpretation of input characters is table-driven (§6.2.4). The definition of what constitutes a terminal can be modified (§6.4.1). Special routines may be executed at newlines (§6.4.3).

Sparser takes a string or file of ascii characters as input, and supplies a populated chart as output. An API for examining this chart is supplied (§5), as are trace routines (§8).

1.2 Overview

Before Sparser can be used you must load its files into Lisp or activate a stored Lisp image that already includes them. You must also load the grammar you intend to use, which will typically be a set of files of rule definition forms that you created earlier. At that point you can have Sparser analyze files of text or individual strings by calling the appropriate function (§2.5). You can also embed calls to run Sparser in your own functions, along with code for examining the results of the its analysis.

To write a phrase structure grammar for Sparser you have to understand the conventions of the rule notation (§4); you will also want to look at the trace routines that have been provided (§8). To write routines of your own for examining Sparser's analysis you have to understand the data structures that make up Sparser's representations (§3, §5). To write more advanced routines to take advantage of the user-code hooks within Sparser you also need to understand something of Sparser’s algorithms so that you will know when the hooks are called and what is available for your routines to examine at those points (§6).