Languide

Language-guided editing

J. C. G. Sturdy

Languide provides high-level editing commands, such as turning an in-line block of code into a procedure, and substituting a call to the procedure for the original code.

Languide is one of a pair of companion packages. The other one, Versor, provides flexible keybindings which can be used to navigate code in terms of its structure. Part of Versor's functionality is provided by the lower-level parts of Languide. See section "Versor and Languide" in *Versor: versatile cursors for GNUemacs*, for more information.

1 Introduction

Text editors have traditionally operated in terms of lines and characters.

Languide takes a step out of that, providing operations in terms of the syntactic units (statements, expressions) that make up source files in typical programming languages.

It aims to provide as single commands many of the routine high-level editing actions that normally take many manual editing operations, for example, taking a block of code and turning it into a function, leaving a call to that function where the code originally was. Although complex, these actions are completely stereotyped, and therefore ripe for automation, as they do not really require human intelligence to perform them. It is hoped that automating them will not only reduce the workload for the user's fingers, but may also be more reliable, and provide a more productive and relaxing working environment.

In an attempt to break the tie with keyboard and mouse, on the whole the commands are designed to work well with voice input. It should be possible to work with very little need to type or pronounce syntactic punctuation characters; these are detail that the computer should be able to deal with, leaving the programmer to get on with the abstract thinking. The need to issue a sequence of commands with no punctuation leads naturally to an RPN-style semantics (which should be already comfortable to users of HP-style calculators, Forth, and PostScript). There is, for example, provision for selecting a statement type and then searching for it, or inserting a template for it.

2 Concepts

This package is built around the ideas of statements, compound statements, and expressions.

We have a general model of statements, in which a statement can have head, body, and optionally tail parts – for example, the three parts of an if-then-else statement.

We also use the idea that each statement (except for a top-level definition) is in a container, which is another statement.

Languide provides language guided editing for a variety of languages. To provide equivalent functionality across the range of supported language modes, we use a modal functions package which lets us give a function separate definitions in each major mode.

Languide has two groups of commands:

high-level editing

movement by statements (which you can use directly, or through Versor (see ⟨undefined⟩ [Versor and Languide], page ⟨undefined⟩), and which are used by the high-level editing commands)

3 Commands

You can access Languide's commands directly with M-x, or through a keymap, or through a menu. This manual will describe the commands by their names, as used with M-x.

Languide commands come in three groups, those substituting the contents of the current selection with something of the same nature, those acting on variables, expressions, and functions; and those acting on statements.

3.1 Alterations

The "alteration" facility is accessed through the Versor package, although Languide provides the underlying data.

3.2 Commands acting on variables, expressions, and functions

This group of commands manipulates value handling constructs in source code. For example, you can select an expression (using the Versor selection), and turn it into a variable, so that you can re-use the same value. Likewise, you can convert an expression into a function, so that you can call it elsewhere.

3.2.1 Employ variable

The command `languide-employ-variable`, given an existing variable definition, finds any existing uses of that value, and substitutes the variable for them.

3.2.2 Convert region to variable

The command `versor:convert-selection-to-variable` takes the current Versor selection as an expression, sets up a variable (at the nearest scoping point, Section 3.3.4 [Scoping point], page 5) initialised to that expression, and replaces the original expression with that variable.

In the normal Versor key bindings, this command is bound to *`C-insert v`*.

The command `languide-convert-region-to-variable`, given a region containing an expression, sets up a variable initialised to that expression, and replaces the original expression with that variable.

3.2.3 Convert region to function

The command `versor:convert-selection-to-function` takes the current Versor selection, defines a new function just before the one containing the selection, and replaces the original selection with a call to the new function. It examines the selection and the surrounding code, to find any variables referred to in the selection but defined outside it, and makes those into arguments to the new function.

In the normal Versor key bindings, this command is bound to *`C-insert f`*.

The command `languide-convert-region-to-function` takes the selected region, defines a new function just before the one containing the region, and replaces the original region with a call to the new function. It examines the region and the surrounding code, to find any variables referred to in the region but defined outside it, and makes those into arguments to the new function.

3.2.4 Surround region with call

The command `versor:surround-selection-with-call` wraps the current Versor selection with a call to a specified function, such that the region becomes the argument list of the function.

In the normal Versor key bindings, this command is bound to *`C-insert (`*.

The command `surround-region-with-call` wraps the current region with a call to a specified function, such that the region becomes the argument list of the function.

3.2.5 Remove surrounding call

The command `versor:remove-function-call` removes the function call surrounding the current Versor selection. This leaves the function arguments in place of the call.

In the normal Versor key bindings, this command is bound to *`C-insert )`*.

The command `remove-surrounding-call` removes the function call surrounding point. This leaves the function arguments in place of the call.

3.3 Commands acting on statements

This group of commands acts mostly on imperative statements.

3.3.1 Unify Statements

The command `versor:unify-statements` turns the current Versor selection into a compound statement. In the normal Versor key bindings, this command is bound to *`C-insert {`*.

The command `languide-unify-statements` makes the statement that point, and the following N statements (where N is the prefix argument) into a compound statement.

3.3.2 Make conditional

The command `versor:make-conditional` makes the current Versor selection conditional, and positions point ready for filling in the condition. If the selection is already the body of a conditional construct, an `and` construct is wrapped around the existing condition (unless it already has one) and point is positioned for adding a further condition.

3.3.3 Make repeating

The command `versor:make-repeating` makes the current Versor selection be the body of a repeating construct, and positions point ready for filling in the repeat condition.

3.3.4 Enclosing scoping point

The command `languide-enclosing-scoping-point` moves point to the most nearly enclosing scoping point, that is, a suitable place for putting new variables. This is largely used as an internal function by Languide, but is also exposed for direct use in case it is useful.

In the normal Versor key bindings, this command is bound to *`C-insert =`*.

3.3.5 Enclosing decision point

The command `languide-enclosing-decision-point` moves point to the most nearly enclosing decision point, that is, a suitable place for putting a new condition. This is largely used as an internal function by Languide, but is also exposed for direct use in case it is useful.

In the normal Versor key bindings, this command is bound to *`C-insert ?`*.

4 Setup

To install Languide, unpack the tarball into a suitable directory, and put that directory on your load-path.

5 Extension

Languide comes with definitions of common statement types for several programming languages. You can add more, and of course contribute them for public use should you so wish.

The file statement-nav-directions.el implements the basic movements needed to navigate around statements. Directions for specific statement parts in various programming languages are defined in such files as languide-lisp-like.el, languide-c-like.el and so forth; languide-c-like is a fairly rich source of examples.

Normally, a sequence of directions is followed, and the last one is taken as the result, that is, the thing to leave selected. To allow multipart selections, as versor does, you can indicate a step of the directions as selecting what it moves over, by wrapping it in a call to "`remember`".

Strings in the directions are searched for (as regexps) and moved over.

Any elisp function calls may be used in the directions, and their effect on point will be, but those listed on `statement-navigate-list-selector-functions` are treated specially: they are expected to return a cons of the start and end positions of the piece of text they describe. Thus, only such functions should be used as the last step of a set of directions, and only these should be given as the argument to "remember". You can write your own functions of this nature, but you must add them to `statement-navigate-list-selector-functions` for them to work properly.

The selection functions are as follows:

`expression`
: Selects the following s-exp.

`expressions`
: Selects as many following s-exps as possible at the current level of nesting.

`expression-contents`
: Selects the contents of the following s-exp (but not its brackets).

`preceding-expression`
: Selects the preceding s-exp.

`statement`
: Selects the following statement. This is defined by "`defmodal`" definitions for the major mode concerned, for the functions `move-into-previous-statement`, `move-into-next-statement`, `beginning-of-statement-internal`, and `end-of-statement-internal`. (If you define these, you should also define `identify-statement`, `compound-statement-open`, `compound-statement-close`, `insert-compound-statement-open`, and `insert-compound-statement-close`.)

`statements`
: Selects as many following statements as possible, at the current level of nesting.

`statement-contents`
: If the following statement is a simple statement, select it. If it is a compound statement, select the statements that it is made of, but not the bracketing that groups them together.

`from-start-of-statement`
`upto`

`start-of-match`

6 Languide and Versor

Command Index

(Index is nonexistent)

Concept Index

(Index is nonexistent)

Table of Contents