SPECWARE Quick Reference

Table 1-1 Slang Constructs

Syntax in Infix Grammar	Purpose
spec *name* is [import *spec*] ... end-spec	Specification
translate *spec* by {*name* -> *name*, ...}	Translate
sort[s] *name*, ...	Sort or Sorts
sort-axiom *s1* = *s2*, *s3* project(1) : *s1* -> *s2*	Product Sort
sort-axiom *s1* = *s2* + *s3* embed(1) : *s2* -> *s1*	Sum Sort
sort-axiom *s1* = *s2* | *predicate?* relax(*predicate?*) : *s1* -> *s2*	Subsort
sort-axiom *s1* = *s2* / *relation?* quotient(*relation?*) : *s2* -> *s1*	Quotient Sort
sort-axiom *s1* = *s2* -> *s3* lambda(*var* : *s2*) *<expression>* *f*(*x*)	Function Sort
op *name* : *sort* -> *sort*	Operation
const *name* : *sort*	Constant
axiom [*name* is] *<boolean expression>*	Axiom
theorem [*name* is] *<boolean expression>*	Theorem
definition of *op* is *axioms* end-definition	Definition
constructors {*op*, ...} construct *sort*	Constructor Set
fa(*var*: *s1*, ...) *<boolean expression>* ex(*var*: *s1*, ...) *<boolean expression>*	Quantification
&, or, ~ (not), => (implies), <=> (iff), =	Boolean Operators
morphism *name* : *spec* -> *spec* is *rules*	Morphism
{*name* -> *name*, ...}	Morphism Rules
identity-morphism	Morphism Keyword
import-morphism	Morphism Keyword
translation-morphism	Morphism Keyword
cocone-morphism	Morphism Keyword
diagram *name* is nodes *spec*, ... [arcs *spec* -> *spec* : *morphism*, ...] end-diagram	Spec Diagram (morphism expressed as rules, name, or keyword)
colimit of *diagram*	Colimit

Table 1-1 Slang Constructs

Syntax in Infix Grammar	Purpose
`interpretation` *name* `:` *spec* `=>` *spec* `is` `mediator` *spec* `domain-to-mediator` *morphism* `codomain-to-mediator` *morphism*	Interpretation (morphism expressed as rules, name, or keyword)
`ip-scheme` *name* `:` *spec* `=>` *spec* `is` `mediator` *spec* `domain-to-mediator` *morphism* `codomain-to-mediator` *morphism*	Interpretation Scheme (morphism expressed as rules, name, or keyword)
`ip-scheme-morphism` *name* `:` *ip* `->` *ip* `is` `domain-sm` *morphism* `mediator-sm` *morphism* `codomain-sm` *morphism*	Interpretation Morphism (morphism expressed as rules, name, or keyword)

Table 1-2 Methods for Creating Interpretations

Method (Menu Item)	Description
Declaration	Declare interpretation in Slang code. A=>B through med B+, a definitional extension of B.
Selection	Select existing interpretation by name from menu.
Identity (identity)	A=>A
Convert Spec Morphism (by lifting a spec morphism)	A => B where morphism exists from A to B
Translation (via translation)	B => A where B is translation of A
Definitional Extension (via definitional extension)	A+ => A where A+ is definitional extension of A
Sequential Composition (via sequential composition)	Build A=>C from A=>B and B=>C
Parallel Composition (via colimit)	Build A+C => B+D from A=>B and C=>D

Table 1-3 REFINE commands

Command	Shortcut	Action
`M-x refine-parse-buffer`	`M-x r p b`	Parse entire buffer
`M-x refine-parse-region`	`C-c M-p` `C-c CM-p`	Parse marked region
`M-x refine-parse-definition`	`C-c p` `C-c C-p`	Parse definition containing cursor
`M-x refine-parse-file`		Parse `.re` file
`M-x refine-load-file`		Load `.re` file
`M-x refine-mode`		Enter REFINE mode
`M-x kill-refine-process`		Exit SPECWARE, REFINE, and Lisp
`M-x run-a-refine`		Start REFINE