| bar — core expressions

- `|$` produce a mold
- `|_` produce a door (core with sample)
- `|%` produce a core (battery and payload)
- `|:` produce a gate with a custom sample
- `|.` produce a trap (core with one arm)[
- `|-` produce a trap (a core with one arm) and evaluates it
- `|^` produce a core whose battery includes a $ arm and compute the latter
- `|@` produce a wet core (battery and payload)
- `|~` produce an iron gate
- `|*` produce a wet gate (one-armed core w/ sample)
- `|=` produce a dry gate (a one-armed core with sample)
- `|?` produce a lead trap

$ buc -- structures

- `$|` structure with verification
- `$$` structure from recursion
- `$_` structure that normalizes to an example
- `$%` structure that recognizes a union tagged by head atom
- `$:` form a cell type (tuple)
- `$.` structure as read–write core
- `$/` structure as write-only core
- `$<` structure from filter (excluding)
- `$>` structure from filter (requiring)
- `$-` structure that normalizes to an example gate
- `$^` structure that normalizes union tagged by head depth (cell)
- `$+` standard structure
- `$;` manual structure
- `$&` repaired structure
- `$@` structure that normalizes a union tagged by head atom
- `$~` define a custom type default value
- `$`` structure as read-only core
- `$=` structure that wraps a face around another structure
- `$?` form a type from a union of other types
- `$!` structure as opaque core

digraphs (not runes)

- `--` terminate core expression
- `::` mark a comment
- `==` terminate a series of Hoon expressions

% cen -- calls & samples

- `%_` resolve a wing with changes, preserving type
- `%:` call a gate with many arguments
- `%.` call a gate, inverted
- `%-` call a gate
- `%^` call a gate with triple sample
- `%+` call a gate with a cell sample
- `%~` evaluate an arm in a door
- `%*` evaluate expression; resolve wing w/ changes
- `%=` resolve a wing with changes

: col -- cells

- `:_` construct a cell, inverted
- `:-` construct a cell, 2-tuple
- `:^` construct a cell, 4-tuple
- `:+` construct a cell, 3-tuple
- `:~` construct a null-terminated list
- `:*` construct an n-tuple

. dot -- nock evaluations

- `.^` load from namespace using Nock 12
- `.+` increment an atom using Nock 4
- `.*` evaluate using Nock 2
- `.=` test equality using Nock 5
- `.?` test cell or atom using Nock 3

/ fas -- build operations

- `/-` import file from sur dir. (* pinned with no face, = with specified face)
- `/+` import file from lib dir. (* pinned with no face, = with specified face)
- `/*` import contents of a file in desk converted to a mark (build-time static data)
- `/=` import result of specified path wrapped in face
- `/?` pin a version number

^ ket -- typecasting

- `^|` convert a gold core to iron (invariant)
- `^:` produce a ‘factory’ gate for a type (switch from regular spec/type parsing)
- `^.` typecast on value
- `^-` typecast by explicit type label
- `^+` typecast by inferred type (a fence)
- `^&` convert core to zinc (covariant)
- `^~` fold constant at compile time
- `^*` bunt, produce default mold value
- `^=` bind name to a value
- `^?` convert core to lead (bivariant)

+ lus -- arm definitions

+|	label a chapter (produce no arm)
+$	produce a structure arm (type definition)
++	produce a (normal) arm
+*	produce a type constructor arm

; mic -- macros

;:	call a binary function as an n-ary function
;/	(Sail) yield tape as XML element
;<	glue a pipeline together (monadic bind)
;+	(Sail) make a single XML node
;;	normalize with a mold, asserting fixpoint
;~	glue pipeline together w/ product-sample adapter (monadic bind)
;*	(Sail) make a list of XML nodes from Hoon expression
;=	(Sail) make a list of XML nodes

~ sig -- interperlator hints

~|	print in stack trace if failure
~$	profiler hit counter
~_	print in stack trace, user-formatted
~%	register jet
~/	register jet w/ registered context
~<	raw hint, applied to product (“backward”)
~>	raw hint, applied to computation (“forward”)
~+	cache a computation
~&	print (used for debugging)
~=	detect duplicate
~?	print conditionally (debugging)
~!	print type if fails compilation

= tis -- subject modifications

=:	change multiple legs in subject
=,	combine default type value with subject
=,	expose namespace (define a bridge)
=.	change one leg in subject
=/	combine named noun with subject
=<	compose two expressions, inverted
=>	compose two expressions
=-	combine new noun with subject
=^	pin head of pair; change a leg with tail
=+	combine new noun with subject
=;	combine named noun with subject, inverted
=~	compose many expressions
=*	define an alias
=?	change subject leg conditionally

? wut -- conditionals

?|	logical OR (loobean)
?:	branch on a boolean test
?.	branch on a boolean test, inverted
?<	negative assertion
?>	positive assertion
?#	test pattern match
?-	switch against a union, no default
?^	branch on wing if subject is cell
?+	switch against a union, w/ default
?&	logical AND (loobean)
?@	branch on whether a wing of the subject is an atom
?~	branch on whether a wing of the subject is null
?=	test pattern match
?!	logical NOT (loobean)

! zap -- wildcards

!:	turn on stack trace
!,	emit AST of expression (use as !, *hoon expression)
!.	turn off stack trace
!<	lift dynamic value into static context
!>	wrap a noun in its type
!;	emit the type for an expression using the type of type
!=	make the Nock formula for a Hoon expression
!?	restrict Hoon version
!!	crash