| galois |

Background: Cryptol commands and built-in functions

Commands

This assumes all the cryptol and supporting binaries are in accessible locations. Assume cryptol has started and the prompt `Cryptol>` is displayed on the screen. Commands are commands for controlling the Cryptol session. To see all commands type the single character `:` and then hit the tab key. The result is displayed as follows :

```
:!                  :dumptests          :module             :safe
:?                  :e                  :prove              :sat
:help               :edit               :q                  :t
:ast                :eval               :quit               :type
:b                  :exhaust            :r                  :version
:browse             :extract-coq        :reload             :w
:cd                 :l                  :readByteArray      :writeByteArray
:check              :load               :s
:debug_specialize   :m                  :set
```

To get help on any command type `:help :<command>` and hit return. For example:

```
Cryptol> :help :check

    :check [ EXPR ]

Use random testing to check that the argument always returns true.
(If no argument, check all properties.)
```

Some commands take options and values. To see those options type :help :<command> then hit the tab key. Here is an example:

```
Cryptol> :help :set
ascii           fpBase          monoBinds       satNum          tests
base            fpFormat        path            showExamples    warnDefaulting
coreLint        hashConsing     prover          smtFile         warnShadowing
debug           ignoreSafety    proverStats     tcDebug         warnUninterp
fieldOrder      infLength       proverValidate  tcSolver
```

Values allowed for those options may be displayed by hitting the return key after typing the option followed by =. For example:

```
Cryptol> :set prover=
Prover must be cvc4, yices, z3, boolector, mathsat, abc, offline, any, sbv-cvc4,
sbv-yices, sbv-z3, sbv-boolector, sbv-mathsat, sbv-abc, sbv-offline, sbv-any,
w4-cvc4, w4-yices, w4-z3, w4-boolector, w4-abc, w4-offline, or w4-any
```

One can use shortened versions of a command if it uniquely associates with a command. For example, `:s` can be used for `:set`.

Type `:help :<command> <option>` and hit return to get a description of a command’s option. For example:

```
Cryptol> :help :set satNum

    satNum = 1

Default value: 1

The maximum number of :sat solutions to display ("all" for no limit).
```

Some Frequently Used Commands

Command	Description	Examples
`:l <file-name>`	Load a file into cryptol	`:l mid.cry`
`:s base=X`	Set numbers to base `X`	`:s base=10  :s base=2  :s base=16`
`:s prover=X`	Set the prover to `X`	`:s prover=cvc4 :s prover=yices :s prover=boolector`
`:s satNum=X`	Set max # models to show	`:s satNum=all  :s satNum=2`
`:sat <property>`	Find models for `<property>`	`:sat weakKeys (preceded by :s satNum=all)`
`:prove <property>`	Prove `<property>` correct	`:prove mergeSortIsCorrect`

Operations

Make the same assumptions as above. Operations are used to build Cryptol programs and specifications: nearly all take arguments and return values. To see all operations hit the tab key at the `Cryptol>` prompt. The result is displayed as follows:

```
Cryptol>
Display all 130 possibilities? (y or n)y
(!!)                     (\/)                     fromToDownByGreaterThan  roundToEven
(!)                      (^)                      fromToLessThan           sborrow
(!=)                     (^^)                     fromZ                    scanl
(!==)                    (||)                     generate                 scanr
(#)                      False                    groupBy                  scarry
(%$)                     True                     head                     sext
(%)                      abs                      if                       sort
(&&)                     all                      infFrom                  sortBy
(*)                      and                      infFromThen              split
(+)                      any                      iterate                  splitAt
(-)                      assert                   join                     sum
(/$)                     carry                    last                     tail
(/)                      ceiling                  length                   take
(/.)                     complement               let                      then
(/\)                     curry                    lg2                      toInteger
(<$)                     deepseq                  map                      toSignedInteger
(<)                      demote                   max                      trace
(<<)                     drop                     min                      traceVal
(<<<)                    elem                     negate                   transpose
(<=$)                    else                     number                   trunc
(<=)                     error                    or                       uncurry
(==)                     floor                    parmap                   undefined
(===)                    foldl                    pdiv                     update
(==>)                    foldl'                   pmod                     updateEnd
(>$)                     foldr                    pmult                    updates
(>)                      foldr'                   product                  updatesEnd
(>=$)                    fraction                 random                   where
(>=)                     fromInteger              ratio                    zero
(>>$)                    fromThenTo               recip                    zext
(>>)                     fromTo                   repeat                   zip
(>>>)                    fromToBy                 reverse                  zipWith
(@)                      fromToByLessThan         rnf
(@@)                     fromToDownBy             roundAway
```

To see a description of an operation type `:help <operation>` and hit return. For example:

```
Cryptol> :help ratio

    ratio : Integer -> Integer -> Rational

Compute the ratio of two integers as a rational.
Ratio is undefined if the denominator is 0.

'ratio x y = (fromInteger x /. fromInteger y) : Rational'
```

The first line above is the signature of the operation: the rightmost label is the type of data that is output, the other labels are the argument types. The connector `->` is used to indicate that this binary operator can be curried – the meaning of this is discussed in the lab on typing. The `fromInteger` operator is very important to this very strongly typed language and its description is as follows:

```
Cryptol> :help fromInteger

    fromInteger : {a} (Ring a) => Integer -> a

Converts an unbounded integer to a value in a Ring. When converting
to the bitvector type [n], the value is reduced modulo 2^^n. Likewise,
when converting to Z n, the value is reduced modulo n.  When converting
to a floating-point value, the value is rounded to the nearest
representable value.
```

The following present some of the most frequently used operations:

Some Frequently Used Operations, Arithmetic

Op	Description	Examples
`+`	Addition	`567 + 111 = 678`
`-`	Subtraction	`567 – 111 = 456   111 – 567 = -456`
`*`	Multiplication	`567 * 111 = 62937`
`/`	Integer division	`567 / 111 = 5`
`%`	Remainder	`567 % 111 = 12`
`^^`	Exponentiation	`567 ^^ 11 = 1947213840615891587090802236583`
`/.`	Field division	`(ratio 7 2) /. (ratio 9 2) = (ratio 7 9)`
`ratio`	Rational number	`(ratio 7 2)*4 = (ratio 14 1)`
`floor`	Round down toward -∞	`floor(ratio 7 2) = 3   floor(ratio (-7) 2) = -4`
`ceiling`	Round up toward ∞	`ceiling(ratio 7 2) = 4  ceiling(ratio (-7) 2) = -3`
`trunc`	Round up/down toward 0	`trunc(ratio (-7) 2) = -3  trunc(ratio 7 2) = 3`
`(/$) (%$)`	2's complement div, rem	

Observe the language does not admit decimals. All numbers are rational. A close approximation for the number π is `355/113` or `(ratio 355 113)` in cryptol.

Some Frequently Used Operations, Comparison

Op	Description	Examples
`(==) (!=)`	Equal, not equal	`(ratio 8 4) == 2 is True  (ratio 8 4) != 2 is False`
`(===) (!==)`	Function equal, not equal	
`(>) (<)`	Greater than, less than	`2 > (ratio 7 2) is False  2 < (ratio 7 2) is True`
`(>$) (<$)`	2's complement signed	

Some Frequently Used Operations, Logic

Op	Description	Examples
`\/`	Logical ‘or’	`(2 > 4) \/ (4 == 3) is False`
`/\`	Logical ‘and’	`(2 < 4) \/ (4 != 3) is True`
`or`	Logical ‘or’ over sequences	`(or [True, False, True]) is True`
`and`	Logical ‘and’ over sequences	`(and [True, False, True]) is False`

Some Frequently Used Operations, Bit Vector

Op	Description	Examples
`||`	Bitwise ‘or’	`17 || 29 is 0b11101 (17 is 10001, 29 is 11101)`
`&&`	Bitwise ‘and’	`17 && 29 is 0b10001`
`complement`	Bitwise complement	`complement 0b110110001000111 is 0b001001110111000`
`<< >>`	Shift left, right	`0b110101 << 2 is 0b010100  0b110101 >> 2 is 0b001101`
`<<< >>>`	Rotate left, right	`0b110101 <<< 2 is 0b010111  0b110101 >>> 2 is 0b011101`

Some Frequently Used Operations, Sequences

Op	Description	Examples
`head`	Get 1st element of sequence	`head [23,36,12] is 23    head “peanut” is 0x70`
`tail`	Strip 1st element from sequence	`tail [23,36,12] is [36,12]   tail “pea” is [0x65, 0x61]`
`@`	Get an element of sequence	`[23,36,12,62,11]@3 is 62  “peanut”@3 is 0x6e`
`@@`	Get sub-sequence of sequence	`[23,36,12,62,11]@@[3,1] is [62, 36]`
`!`	Get element, reverse idx	`[23,36,12,62,11]!3 is 36  “peanut”!3 is 0x61`
`!!`	Get sub-sequence, reverse idx	`[23,36,12,62,11]!![3,1] is [36, 62]`
`foldl`	Fold left	`foldl (/) 1000 [1,2,3,4] is 41`
`foldr`	Fold right	`foldr (+) 0 [1,2,3,4] is 10`
`reverse`	Reverse sequence elements	`reverse [23,36,12,62,11] is [11,62,12,36,23]`
`sum`	Sum elements of sequence	`sum [23,36,12,62,11] is 144`
`product`	Multiply elements of sequence	`product [23,36,12,62,11] is 6776352`
`sort`	Sort with <=	`sort [23,36,12,62,11] is [11,12,23,36,62]`
`sortBy`	Sort with comparator	`sortBy (<) [23,36,12,62,11] is [11,12,23,36,62]`