genfunlib Developer Documentation

Ideas and notes

blah

GFeq2asymptoticCoef(gdev)
rec2GFeq
"override" **`GeneratingFunction`**
GFeq2GF(KernelMethod)
GFeq2rec
GFeq2coefs
differentiate eqn, set var to 0, solve

- **GF Frameworks**

  `{DFA, Regex, RRGrammar}2Spec`?
  (not necessary to obtain GFs)

- **Species**
- **Symbolic Method**

  `Spec2GFeq`
  implicit specs
  pointing, substitution
  restrictions, additional params

- **Regular Languages**

  Public (Exported) Rules
  `{NFA,DFA,Regex,RRGrammar,Digraph}2{NFA,DFA,Regex,RRGrammar,Digraph}`
  `{NFA,DFA,Regex,RRGrammar,Digraph}{Union, Intersection, Complement, Concat, Sta`

  `{NFA,DFA,Regex,RRGrammar,Digraph}2GF`
  allow the user to provide a function mapping each letter to a symbol

  `Disambiguate{Regex,RRGrammar,Digraph}`
  Digraph disambiguation is converting to a DFA and back
  `Test{Regex,RRGrammar?,NFA?,Digraph}Ambiguity`

  Representation Descriptions
  `NFA`
  `{numStates_Integer, alphabetSize_Integer, transitionMatrix_,`
  `acceptStates_?VectorQ, initialState_}`
  number of states: integer >=0, where 0 states means null language
  alphabet size: integer >=1
  transition matrix: numStates by alphabetSize+1 matrix where entry i,j is a list of (valid) states accessible from state i and letter j. Letter alphabetSize+1 is ϵ

accept states: list of integers between 1 and number of states
initial state: integer between 1 and number of states

DFA

{numStates_Integer, alphabetSize_Integer, transitionMatrix_, acceptStates_?VectorQ, initialState_}

number of states: integer >=0, where 0 states means null language
alphabet size: integer >=1
transition matrix: numStates by alphabetSize matrix where entry i,j is the (valid) state accessible from state i and letter j. Letter alphabetSize+1 is ϵ
accept states: list of integers between 1 and number of states
initial state: integer between 1 and number of states

StringRegex

string, with or without wrapping head **RegularExpression**, containing [a-z,A-Z,0-9,*,(,),|,] and is a valid *Mathematica* regular expression (POSIX ERE I think)

SymbolicRegex

expression built up from **EmptyString**, **letter[n_]**, where **n** is numeric, and **star,concat,or**

RRGrammar

Digraph

Digraphs have their vertices labeled with letters, have a set of start states and end states, and store whether ϵ is accepted

In rules that take a regex, either string or symbolic regexes can be used, and the output format matches the input format

ambiguity test via NFA test (see Book and Even papers -- is Book necessary, would ordinary construction work?) or recursive test (see Brabrand and Thomsen)
“a**” is not considered ambiguous in Book, niether is “a* | b*”. *our* definition of ambiguity must include *e*.
semantic validity test for grammars via symbolicmethod

Bonus: words with occurrences of patterns
Bonus: accept more regex syntax