adampalay / **circuit-finder**

use tla+ to find programs, from circuits, quantum, to Lisp

35 commits | **5** branches | **0** releases | **2** contributors

Branch: **master** | **New pull request** | **Create new file** | **Upload files** | **Find file** | **Clone or download**

adampalay Merge branch 'master' of github.com:adampalay/circuit-finder — Latest commit `ec6a125` on Jul 24

.gitignore	remove .DS_Store	2 months ago
DeutschAlgorithm.tla	clean up Deutsch's Algorithm	2 months ago
LispFinder.tla	compute -> evaluate	a month ago
README.md	x^2 + 1	a month ago
XfromNAND.tla	rename "QuantumAlgorithm" to "XfromNAND"	2 months ago
XfromNAND2.tla	andfromnand part 2! this time generating the "AST" of the circuit	2 months ago

README.md

circuit-finder

use tla+ to synthesize lisp expressions and circuits (classical or quantum)

TLA+ helps you check for bugs in the designs of your systems. The idea is you use TLA+ to specify your program as a state machine, then check to verify that certain undesireable states are unreachable. For example, you might want to check that your algorithm for facilitating intra-bank transfers doesn't change the total amount of money the bank holds.

Circuit-finder is based on an idea that you can think of activity of programming itself as a kind of state machine. As you construct a program, line by line, you alter the "state" of the program you're writing. If we work with a restricted programming language and have clear specifications, then presumably we can use TLA+ to construct programs from their specifications.

This repo contains some examples of these generated programs.

LispFinder searches a simplified space of Lisp programs of the type `Integer -> Integer` to synthesize programs from specifications. The current version of LispFinder discovers `x^2 + 1` from the specification that the inputs of `0`, `1`, and `2` should return `1`, `2`, and `5`, respectively. LispFinder is a proof of concept that at least simple programs can be generated from specification.

XfromNAND simulates classical chip construction. Starting with two input wires and only using NAND gates, XfromNAND can then be induced to construct any number of other logic gates. In this case, it implements an adder with a carry. But you can imagine changing `Goal` to the truth table of an `AND` gate, a `NOT` gate, or any other number of logical gates.

DeutschAlgorithm "discovers" a simple quantum computing algorithm, Deutsch's Algorithm, through specifying available quantum gates and the condition we want the algorithm to discover.

This approach to program generation generally isn't scalable, since TLA+ exhaustively searches the space of possible programs. But it works very well for these three examples!