Quantomatic – current state and case study

Vladimir Zamdzhiev

Department of Computer Science,
University of Oxford

October 17, 2014

Overview

In this talk we will:

- explain what the Quantomatic project and its subprojects are about
- provide an easy-to-follow tutorial on how to use QuantoDerive
- showcase most features of QuantoDerive
- do a live case study and see how well QuantoDerive can be used for studying some Quantum Secret Sharing protocols which I covered in my master thesis

Quantomatic Project

The Quantomatic project is an initiative to develop software tools which help users work with string diagrams (e.g. ZX).
Currently, the main subprojects are:

Quantomatic Project

The Quantomatic project is an initiative to develop software tools which help users work with string diagrams (e.g. ZX).
Currently, the main subprojects are:

- QuantoDerive (new Quantomatic GUI)
  - serves the same role as the old GUI, but improves it in many aspects
  - allows for diagram creation and manipulation, creating rewriting rules, performing diagrammatic reasoning using a system of rewriting rules, exporting all that in LaTeX
  - written in Scala and actively developed (mostly by Aleks and me)
  - easy to use and install (no compiling, just download zip/tarball and click on executable) on all platforms (GNU/Linux, OSX, Windows)
  - project is hosted on github and we make use of most of their fancy features – it's super easy to report bugs, complain, make suggestions, track how the project is going, etc.

Quantomatic Project

The Quantomatic project is an initiative to develop software tools which help users work with string diagrams (e.g. ZX).
Currently, the main subprojects are:

- QuantoDerive (new Quantomatic GUI)
  - serves the same role as the old GUI, but improves it in many aspects
  - allows for diagram creation and manipulation, creating rewriting rules, performing diagrammatic reasoning using a system of rewriting rules, exporting all that in LaTeX
  - written in Scala and actively developed (mostly by Aleks and me)
  - easy to use and install (no compiling, just download zip/tarball and click on executable) on all platforms (GNU/Linux, OSX, Windows)
  - project is hosted on github and we make use of most of their fancy features – it's super easy to report bugs, complain, make suggestions, track how the project is going, etc.
- QuantoCore
  - the rewriting engine, does all of the complicated graph matching and rewriting stuf
  - bus factor of 1
  - written in ML over many years
  - QuantoDerive communicates with the core which does all of the rewriting and the GUI just displays the results

Quantomatic Project

The Quantomatic project is an initiative to develop software tools which help users work with string diagrams (e.g. ZX).
Currently, the main subprojects are:

- QuantoDerive (new Quantomatic GUI)
  - serves the same role as the old GUI, but improves it in many aspects
  - allows for diagram creation and manipulation, creating rewriting rules, performing diagrammatic reasoning using a system of rewriting rules, exporting all that in LaTeX
  - written in Scala and actively developed (mostly by Aleks and me)
  - easy to use and install (no compiling, just download zip/tarball and click on executable) on all platforms (GNU/Linux, OSX, Windows)
  - project is hosted on github and we make use of most of their fancy features – it's super easy to report bugs, complain, make suggestions, track how the project is going, etc.
- QuantoCore
  - the rewriting engine, does all of the complicated graph matching and rewriting stuf
  - bus factor of 1
  - written in ML over many years
  - QuantoDerive communicates with the core which does all of the rewriting and the GUI just displays the results
- QuantoTactic
  - main goal is to provide theorem prover integration for Quanto
  - this will guarantee proof of corectness for derivations (nice to have as Quanto is very complicated and large which means bugs are inevitable)
  - also will allow using theorem provers and their powerful tactics in order to search for proofs
  - limited progress on project, basically on hold for the time being

Quantomatic Project

The Quantomatic project is an initiative to develop software tools which help users work with string diagrams (e.g. ZX).
Currently, the main subprojects are:

- QuantoDerive (new Quantomatic GUI)
  - serves the same role as the old GUI, but improves it in many aspects
  - allows for diagram creation and manipulation, creating rewriting rules, performing diagrammatic reasoning using a system of rewriting rules, exporting all that in LaTeX
  - written in Scala and actively developed (mostly by Aleks and me)
  - easy to use and install (no compiling, just download zip/tarball and click on executable) on all platforms (GNU/Linux, OSX, Windows)
  - project is hosted on github and we make use of most of their fancy features – it's super easy to report bugs, complain, make suggestions, track how the project is going, etc.
- QuantoCore
  - the rewriting engine, does all of the complicated graph matching and rewriting stuf
  - bus factor of 1
  - written in ML over many years
  - QuantoDerive communicates with the core which does all of the rewriting and the GUI just displays the results
- QuantoTactic
  - main goal is to provide theorem prover integration for Quanto
  - this will guarantee proof of corectness for derivations (nice to have as Quanto is very complicated and large which means bugs are inevitable)
  - also will allow using theorem provers and their powerful tactics in order to search for proofs
  - limited progress on project, basically on hold for the time being
- QuantoCoSy
  - CoSy stands for Conjecture Synthesis
  - small tool which generates rulesets from concrete models

Quantomatic Project (contd.)

There's also the Old Quantomatic GUI (java based):

- legacy software (no longer developed)
- this is the version of Quanto which was used in all CQM practicals and almost all Quanto demos
- because of several design limitations it was dropped in favor of QuantoDerive

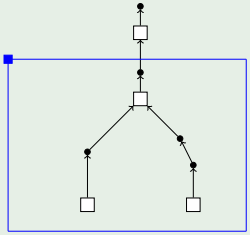
Prerequisites

In order to effectively use QuantoDerive, you need to be:

- somewhat familiar with String Diagrams and String Graphs
- depending on your usage, good-weak understanding of !-boxes
- well familiar with the particular theory you will be working with (in our case ZX)
- aware of some of the more high-profile bugs (check the issue tracker)
- aware of the software's limitations (e.g. you can't describe all kinds of diagrammatic patterns as !-boxes)
- understand that undiscovered bugs do exist, so there is no guarantee of correctness, you should still try to verify derivations yourself

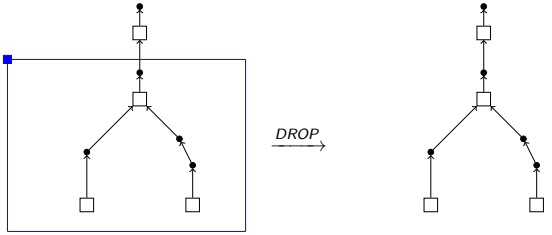
A !-graph is a generalised string graph which allows us to represent infinite families of string graphs in a formal way.
The motivation behind !-graphs is to remove some of the informalities in expressing infinite families of rewrite rules and infinite families of diagrams using the "dot dot dot" $(\cdots)$ notation in order to ease the development of software proof-assistants.

Example

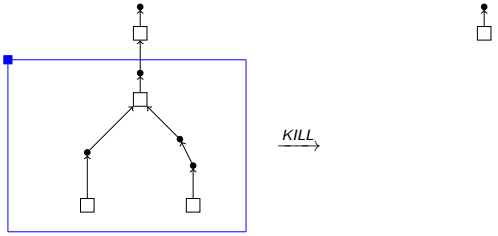


This is how !-graphs are represented syntactically. However, semantically, a !-graph should be thought of as an infinite set of concrete string graphs, each of which is obtained after a finite application of three different operations on the !-boxes of the graph. The three operations are called COPY, DROP and KILL and are presented below.

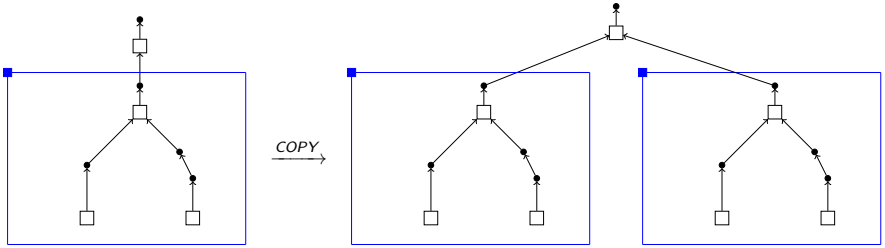
Applying a DROP operation removes a !-box, but not its contents:



Applying a KILL operation removes a !-box and its contents:



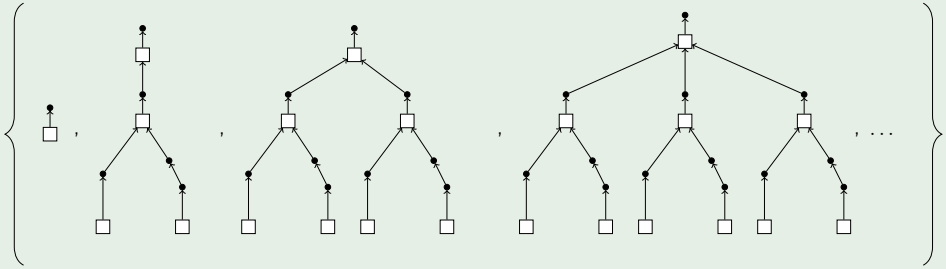
Applying a COPY operation creates another copy of the !-box and its contents which is connected in the same way to the rest of the graph:



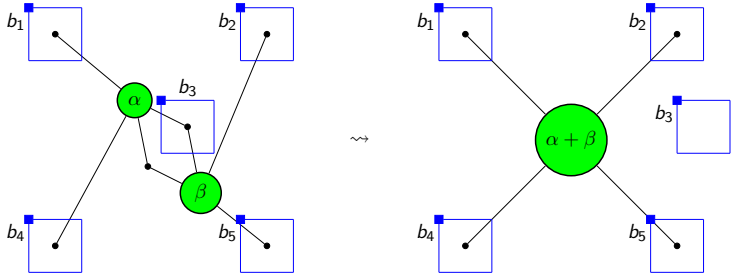
Semantically, a !-graph represents the infinite set of concrete string graphs obtained after applying all possible sequences of !-box operations.

Example

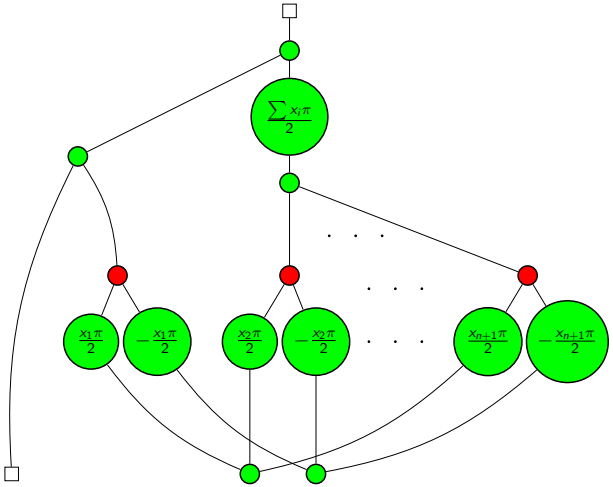
The above !-graph represents the following family of graphs:



Here's how the spider rule looks like as a !-graph rewriting rule

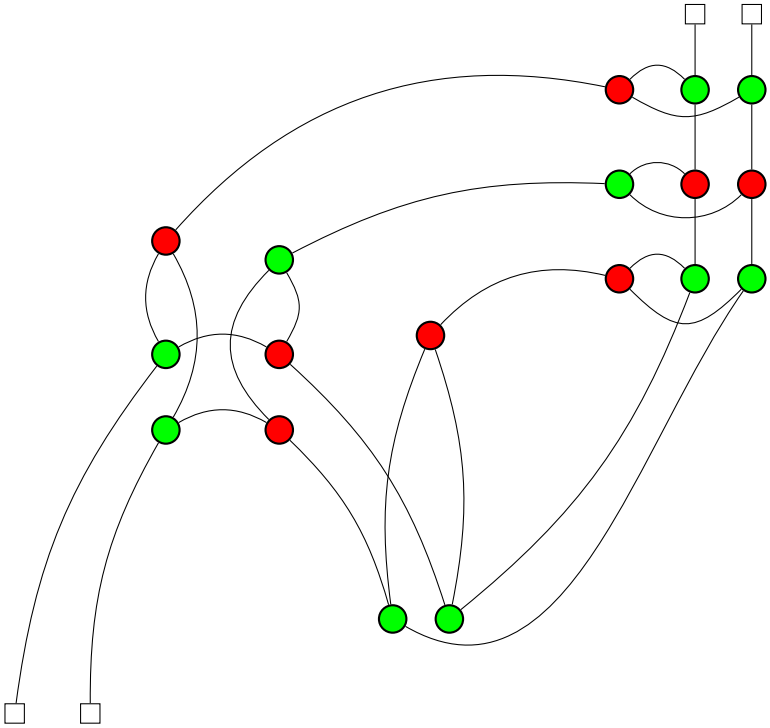


This diagram represents the secret reconstruction phase of HBB CQ(n,n). It cannot be directly represented in quantomatic because it depends on n binary variables.

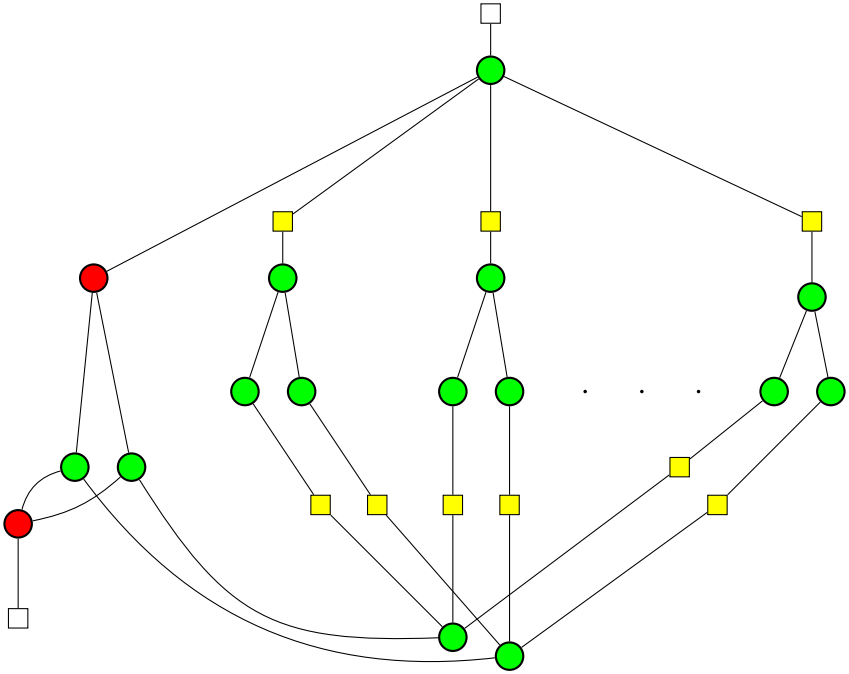


HBB QQ(2,2)

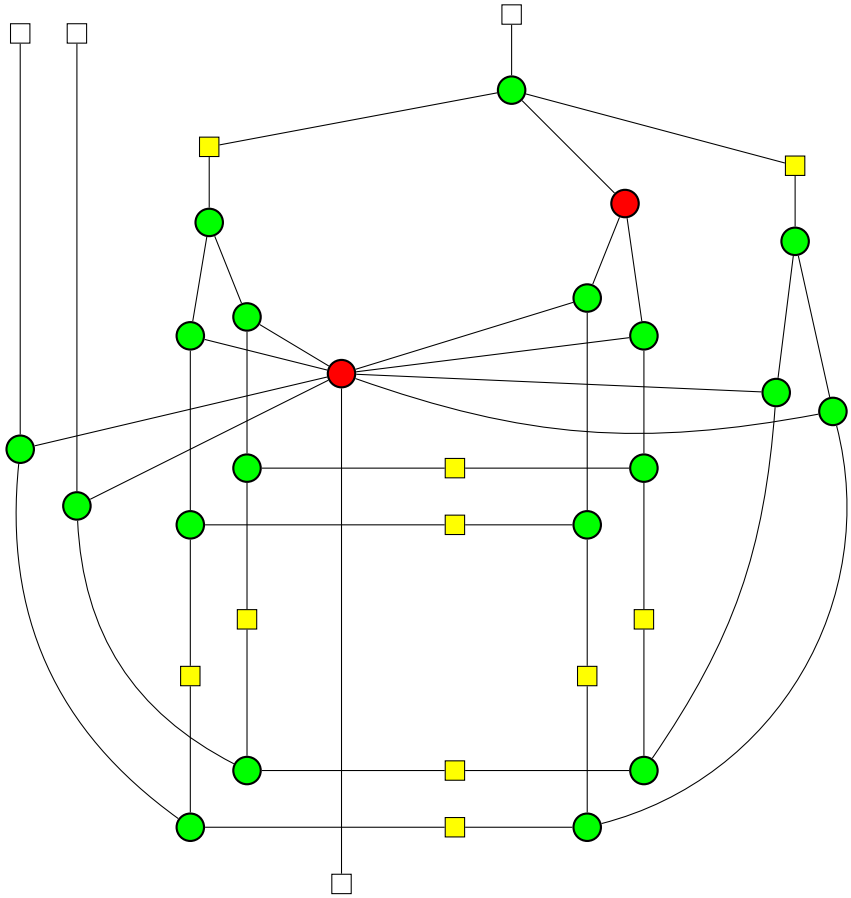
The proof is fully automated. (See Demo)



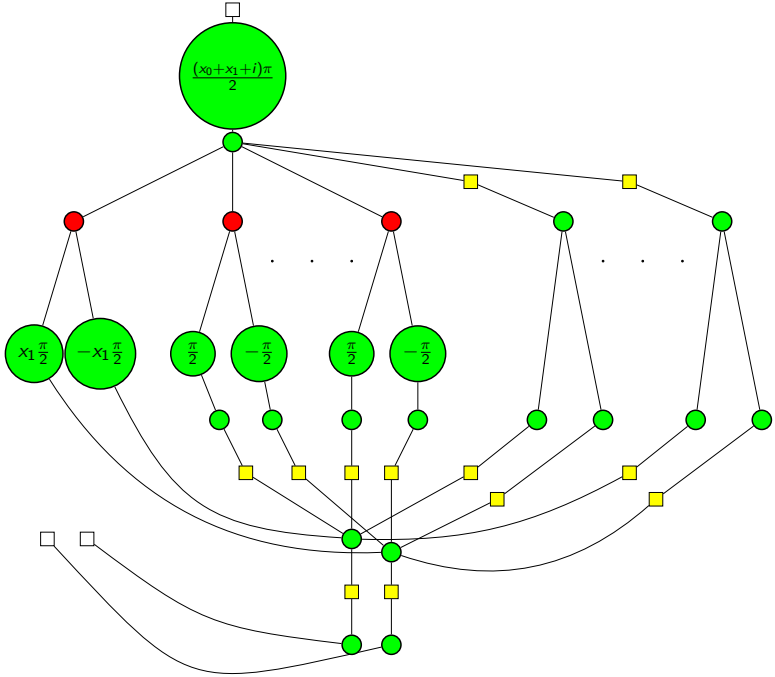
Proof can easily be done in Quanto, but with some minor annoyances. (See Demo)



Proof almost fully automated, just need to manually apply generalised bialgebra law. (See Demo)

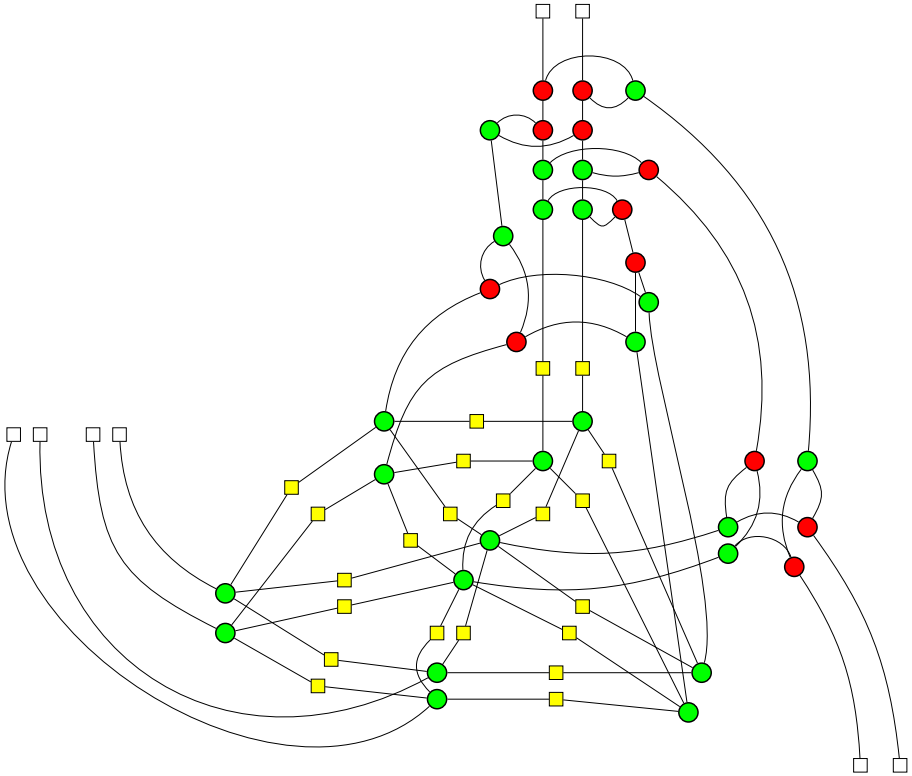


Can't be done directly done again, because the classical correction which needs to be performed depends on the number of Y measurements (i).

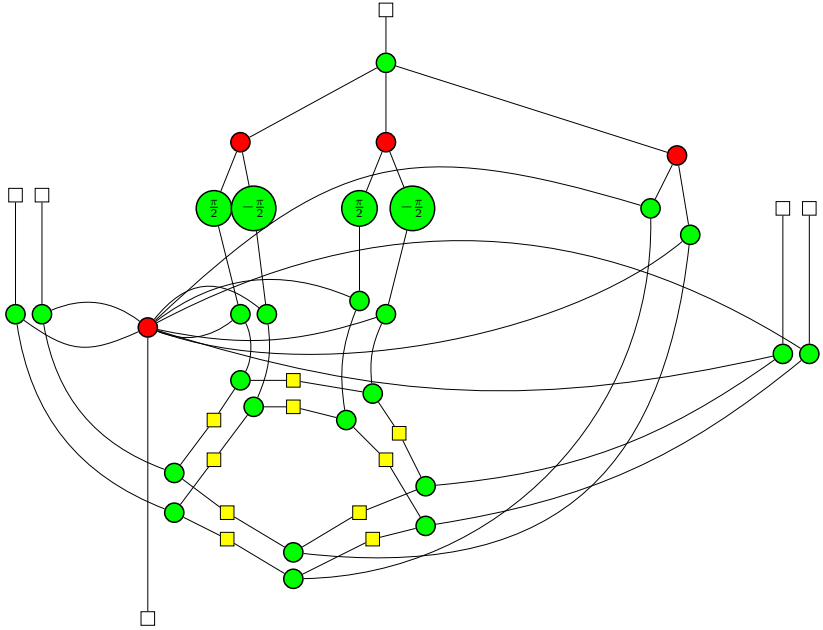


GSS QQ(3,5)

Very large graph, couldn't do it in thesis, couldn't do it in Quantomatic – some (previously) undiscovered bugs when working with this graph and there is no obvious rewriting strategy which can be used. However, provable in principle.



Doable in principle, but annoying to do in Quanto. Requires to apply some rules in reverse (notably spider law) and this is currently clunky in Quanto (we should improve this in the future).



Conclusion

- can be used to study real-world stuff
- bugs do exist, so not ideal
- there are theoretical limitations on what can be represented
- definitely worth using, works well in many cases and in worst case everything can be exported to latex/tikz/tikzit and continue from there
- it would be nice if we can get more people to contribute (code, documentation, bug reports, suggestions, etc.)
- go and check out project and run Quanto yourself : `http://quantomatic.github.io`