yquant.sty package documentation

Typesetting quantum circuits in a human-readable language

Benjamin Desef

June 7, 2020

This manual introduces yquant, a LaTeX-only package that outputs quantum circuits. They are entered using a human-readable language that, even from the source code, allows for a fluent understanding of the logic that underlies the circuit. yquant internally builds on TikZ and can be easily combined with arbitrary LaTeX code. More than forty pages of examples complement the formal manual.

Contents

3

1 Introduction

This document outlines the scope and usage of the yquant package. It contains both a reference and a huge number of examples. yquant is a package that makes typesetting quantum circuits easy; the package is available on CTAN. This alpha version 0.2.1 *should* be stable and interfaces are not very likely to change in an incompatible way in the future. Please do report all issues and desirable additions.

1.1 How to read the manual

The probably fastest way to start using yquant is by just scanning through the examples in section 6. A more formal description of the yquant grammar and its fundamental concepts can be found in section 2. If your desire is to change the appearance of yquant elements, use the configuration reference in section 3. The full list of all available gates is provided in section 5. Finally, you may find that yquant *almost* does what you want, but there is some final tweak that you cannot achieve…. Then, have a look at section 4 (or section 1.3).

1.2 Installation

The recommended way of installation is through CTAN. A direct installation from this Git repository to obtain the latest additions and features is be possible by just cloning it to a path visible to your TeX compiler. While the repository may contain new additions, they are not thoroughly tested until they end up on CTAN; features that are not documented in this manual are entirely unreliable. For example, you may put them in the same directory as your document (if you just want to give a try), or you may extract them to `tex/latex/yquant` in your local *texmf* (followed by an update of the file name database).

1.3 Purpose of yquant, alternatives

yquant is the acronym for "yet another quantum circuit package." This highlights the fact that nothing that this package provides cannot be achieved by other means. In particular, there are at least the following methods to typeset quantum circuits in TeX.

- Use some external program to draw them and include the output via `\includegraphics`.

- Use either TeX's own drawing capabilities (the `picture` environment) or other drawing packages such as TikZ or `pstricks`.

- Use a package specifically designed to draw quantum circuits (if you feel some other package should be mentioned here, please file an issue):

  – qasm is probably the first of them (in terms of age). It was developed to typeset the circuits found in Nielsen and Chuang's famous *Quantum Computation and Quantum Information* book. qasm consists of a Python 2 script (qasm2circ) that reads a quantum circuit written in a very intuitive language: declare names for your qubits, perform gates on them in each line. qasm2circ converts those circuits into TEX files that internally make use of the xy package to display the output. Consequently, the user is restricted to the set of features that qasm directly offers (which is small). Changes to the output, while possible, will be overwritten if qasm2circ is run again. qasm output often looks sub-optimal do to the fact that, e.g., rectangles are made up of four lines that do not properly connect and give a crumbly general feeling.

  Maintenance status: last update of qasm in 2005. Also, xy was last updated in 2013, and the script is not compatible out-of-the-box with Python 3, though an automatic conversion should work.

  – qcircuit is probably the most-widely used package. It provides commands that make it much easier to create quantum circuits using the xy package. Its syntax therefore is grid-oriented; inferring what a circuit does or locating a gate in the code can be tough. This is particularly true for multi-qubit gates. Additionally, the **\xymatrix** syntax is also somewhat cryptic. qcircuit provides some flexibility within the limits of xy as to configuring the output.

  Maintenance status: last update in 2018; and remember this is xy based, with last update in 2013.

  – quantikz is a relatively recent package that, following the same grid-based approach as qcircuit, instead builds on Ti*k*Z as a backend. As a consequence, it provides the full flexibility of customization that Ti*k*Z offers, where hardly anything cannot be done. It also reduces burdens of the xy syntax. However, the disadvantages of the grid-based syntax still remain.

  Maintenance status: last update in 2020; the underlying Ti*k*Z is actively maintained again by now.

  – qpic follows the approach of qasm: It makes use of an external Python program that reads the quantum circuits in an own language and converts them into Ti*k*Z commands. The language qpic follows is much more powerful than qasm's. The disadvantage that modifications in the

output code will not remain after running the Python script again is mitigated by the possibility to define own TeX macros. Being an external program, `qpic`'s intrinsic set of features (including, e.g., vertically set circuits) are huge. However, the language `qpic` uses cannot be understood without a detailed study of the manual, it appears to have been designed with the aim to minimize the length of command names. A disadvantage of external programs is that the amount of space gates need is not accessible by the script; hence, manual intervention may be required.

Maintenance status: last update in 2016; the underlying Ti*k*Z is actively maintained, and the script is compatible with Python 3.

1.4 License

This work may be distributed and/or modified under the conditions of the LaTeX Project Public License, either version 1.3c of this license or (at your option) any later version. The latest version of this license is in

<p align="center">http://www.latex-project.org/lppl.txt</p>

and version 1.3c or later is part of all distributions of LaTeX version 2005/12/01 or later.

2 Basic elements of `yquant`

`yquant`, as some of the aforementioned packages, builds on Ti*k*Z. Its basic syntax is similar to `pgfplots`: Start a `tikzpicture` environment (perhaps passing some options); inside, start a `yquant` environment.

Inside the `yquant` environment, TeX will now understand the `yquant` language—so `yquant` falls into the same category as `qasm` and `qpic`, providing a human-readable language for the specification of the circuit that is not fixed to the actual layout.

However, `yquant` is a TeX-only package (actually, LaTeX2_ε, but not LaTeX3) that requires no external script to run—so it also falls into the same category as `qcircuit` and `quantikz`.

Since it runs entirely within TeX, you can at any time interject `yquant` code with arbitrary TeX or Ti*k*Z code (though if it is "too arbitrary," you may need to restart the `yquant` interpreter).

2.1 General usage

```
% preamble: \usepackage{yquant}
\begin{tikzpicture}% tikz options possible
   % tikz commands go here
   \begin{yquant}% yquant options possible. Watch the newlines!
      % yquant and tikz commands go here
   \end{yquant}
   % tikz commands go here
\end{tikzpicture}
```

Note that `yquant` depends on `etoolbox`, Ti*k*Z, and `trimspaces`. Additionally, it requires a moderately recent version of LaTeX2_ε, using either LuaLaTeX, or (untested), pdfLaTeX or XeLaTeX.

Optional arguments

The optional arguments for the `yquant` environment have to appear *on the same line* as the environment itself. If you want to put the arguments into a new line, it is crucial to mask the line break by putting a comment symbol after the environment: `\begin{yquant}%`. Without this comment, `yquant` will detect your line break (this is one of the few places in TeX where line breaks and spaces are different) and assume that the expression in square brackets instead provides arguments for the following operation!

Finally note that in (non-fragile) `beamer` frames, this discrimination between

spaces and new lines does not work; the optional arguments will always be counted for the environment, not for the gate. In this case, you can either declare the frame as fragile or (recommended) introduce a blank line between the environment and the options for the first gate.

2.2 Starred vs. unstarred environment

You may choose to use either the `yquant` or the `yquant*` environment. The former one requires you to define all your registers before you use them (though you may decide to define a register after some operations on *different* registers, but before its first usage).

The starred form additionally supports the use of undeclared registers: it basically declares a registers upon its first usage. This will always be a `qubit` register; but if you use the corresponding option and the first usage is an `init` command, you may overwrite this. Subcircuits always use the unstarred form.

Additionally, if you refer to the index i of a vector register of length $L < i$, this register will automatically be enlarged to $i := L$. It is also possible to convert a scalar register into a vector register in this manner. To enlarge a register in the unstarred environment, you must precede the number of registers to be added in the second declaration by a plus sign. Note that in this manner, you may even create discontiguous vectors.

This might be a good point to proceed to the examples section 6.

2.3 Formal syntax

Every yquant command has the same structure (described here in EBNF syntax):

```
Command = { Arguments }, ?command?, [ Value ], [ RegisterList ], Controls,
↪   ";";
Arguments = "[", ?pgfkeys?, "]";
Value = "{", ?TeX code?, "}";
Controls = [ "|", [ RegisterSingleList ] ], [ "~", [ RegisterSingleList ] ];

RegisterList = (RegisterSingle | RegisterMulti), [ ",", RegisterList ];
RegisterSingleList = RegisterSingle, [ ",", RegisterSingleList ];

RegisterSingle = RegisterSingleNoRange | RegisterRange;
RegisterSingleNoRange = ?name?, [ "[", IndexMultiList, "]" ];
RegisterMulti = "(", ( RegisterMultiNoRange | [ "*" ], RegisterRange ), ")";
RegisterMultiNoRange = [ "*" ], ?name?, [ "[", IndexSingleMainList, "]" ];
RegisterRange = [ RegisterUnique ], "-", [ RegisterUnique ];
RegisterUnique = ?name?, [ "[", ?number?, "]" ];
```

```
IndexMultiList = IndexMulti, [ ",", IndexMultiList ];
IndexSingleList = IndexSingle, [ ",", IndexSingleList ];
IndexSingleMainList = [ "*" ], IndexSingle, [ ",", IndexSingleMainList ];
IndexMulti = IndexSingle | ( "(", IndexSingle, ")" );
IndexSingle = ?number? | ( [ ?number? ], "-", [ ?number? ] );
```

Note that yquant is quite tolerant with respect to whitespaces. Virtually every comma in the EBNF notation may consist of an arbitrary (including zero) number of whitespaces. Not all combinations that can be constructed by this grammar are actually allowed semantically; but it would make the grammar too verbose to spell this out in detail. Deviations are noted in this manual.

Valid values for ?command? (case-insensitive) are documented in a section 5. We use ?pgfkeys? to describe any valid content passed to the \pgfkeys macro (rather, \yquantset is invoked with some subtleties); and by ?name? we denote any valid register name. Register names must not contain any of the control literals used before (semicolon, comma, parentheses, square brackets, dash, pipe, tilde, beginning star); and you should avoid using special TeX characters. Note that for performance reasons, yquant does not check whether a register name is valid or not, but expect to either see unintended output or not-so-helpful error messages if you choose an invalid name. ?number? is a decimal integer larger or equal to zero (in the context of register creation, strictly larger; in this context, it may also contain a leading "+").

2.4 Registers

Every quantum circuit is structured by means of *registers*. A register has a *type* that specifies how its wire is drawn, and that may even change during its lifetime. At the moment, yquant supports four types:

1. qubit is the most common type, used for a quantum register. It corresponds to a single line.

2. cbit is a classical register, which can be either declared from the beginning or arises by using measurements. It corresponds to a double line.

3. qubits is a "quantum bundle," i.e., a bunch of quantum registers that are always addressed in a group as a single register. Operations between bundles of the same length should be interpreted as transversal. It corresponds to a triple line. An alternative (and more common) representation is to use the qubit type and a slash gate at its very beginning.

4. `nobit` is the most obscure type, corresponding to a non-existing wire. Mostly, this register type arises by using the `discard` command. However, it can also be directly declared, which on rare occasions might be necessary (its type can then be changed by means of an `init` or `setwire` pseudo-gate). If you want to declare a register only at a certain horizontal position in the circuit, consider using the `after` argument instead.

Registers must be declared before they can be used (though in the `yquant*` environment, this declaration may be implicit, creating a `qubit` register).

Registers can have a vector character, i.e., not only a *name*, but also an *index* (or, in the declaration, a *length*). The index (zero-based) or length is specified in square brackets following the name, which closely mimics the OpenQASM language.

Since version 0.1.1, vector registers may be non-contiguous: Whenever you create a bunch of registers, it is put at the bottom of the circuit. If you later on again create registers of the same name—either implicitly in the `yquant*` environment, or explicitly by preceding the length of the vectors entries to be added by a plus, as in `qubit a[+3];`—they will be put to what is *now* the bottom of the circuit, even if some other registers are interspersed.

Registers are referenced—i.e., used in operations—by their name and index. If the latter is omitted, all indices of the register are targeted. Multiple registers can be referenced by joining their names in a comma-separated list, or by means of a range specifier: give the name of the first (topmost), a dash, and the last (bottom-most) register. Both are inclusive. In a range specifier, omitting the start name means that the range begins at the first known register; omitting the end name means that the range ends at the last known (at the moment of its use) register. Omitting both indicates a range over all known registers.

Since version 0.1.1, it is also possible to use comma-separated lists and ranges within the indices themselves, so that, e.g., `a[0, 2, 5-]`, `b[-2]` will target the zeroth and second index of a; the remaining indices of a starting from five; and the first three indices of b. However, if you use an *outer* range (i.e., a range between indices of registers with different names), the initial and final register of the range must be unique, i.e., either you omit the index (targeting the first or last register with the given name) or specify a single one.

Ranges and discontiguous registers

Assume a configuration in which the vector register a begins with one qubit, then the single register b follows, and after that a is continued with another qubit.

The range `a-b` will target `a[0]` and `b[0]`, but not `a[1]`. As a is used as the

initial register in the range without an explicit index specification, `yquant` automatically translates this into `a[0]`, while `b`, being used as the final register, is automatically translated into the last register of name `b` (which here happens to be `b[0]`). Ranges between different register names (outer ranges) are *visual* ranges, i.e., they refer to the top-to-bottom order that is visible. Consequently, the register `a[1]` is left out since it is visually below the others.

Likewise, the range `b-a` will target `b[0]` and `a[1]`.

Ranges within indices are *logical* ranges. Hence, `a`, `a[-]`, `a[0-]`, `a[-1]`, and `a[0-1]` are all equivalent: they all refer to the registers `a[0]` and `a[1]`, but never to `b`, regardless of any visual position.

All that was said so far refers to the operation being carried out on each of the registers *individually*, i.e., producing several copies of the operation. This is different from using the operation multiple times on the individual single registers only with regard to the vertical positioning: if specified as a register list with one operation, all copies of the operation will be aligned at the same vertical position (as if an `align` command had been carried out before).

It is forbidden (in the sense of "not useful," but `yquant` does not check for this) to list the same register multiple times (explicitly or via ranges) in one operation.

Instead of copies of single-register operations, one might want to carry out a multi-register operation. In this case, the desired list of registers (comma separated, range, or both) must be surrounded by parentheses. It is possible to mix single- and multi-register operations arbitrarily. In an index list, you may also choose to surround only certain indices with parenthesis, provided the whole register is not already a multi-register.

Note that some gates, such as the `swap` gate, always require (semantically, not grammatically) multi-register operations. Since version 0.1.2, the number of constituents is no longer fixed; while a `swap` gate with more than two targets is no longer well-defined, other registers such as `zz` may still be useful. `yquant` will prevent you from using a gate in a multi-qubit setting when it may only be used for single registers.

Typically, multi-register operations should only be carried out on adjacent registers—but sometimes, one might want to carry out a multi-qubit operation on a visually discontiguous set of registers (which, due to a particular quantum computer topology, might even be physically feasible). Since version 0.1.2, `yquant` supports these discontiguous operations explicitly (before, a single gate was drawn

with all intermediate non-affected register lines laying on top). It will draw a *main* part of the gate at the first contiguous slice of registers in the target list—you may select another register for this part by preceding the name or index with a star (which, contrary to the simplified grammar, may only occur *once* in a target specification). All other contiguous slices of target registers will be drawn in a *subordinate style* for this gate. Finally, all slices will be connected by a single vertical line with the style `/yquant/every multi line`. Subcircuits, supported as of version 0.2, will always span the full region from the first to the last register specified in a multi-qubit gate. This is due to the fact that they may contain arbitrary ancilla registers which may be positioned somewhere in between the parts that actually constitute the subcircuit—so this whole region must not have wires of other registers crossing.

Discontiguous targets and control lines

A control line extends from the very first to the very last affected register in an operation. A sub-gate line that is used for discontiguous registers will only span the range of a multi-register. This distinction becomes crucial if you want to carry out a *controlled* operation on more than one multi-register, where at least one is discontiguous. Without the controls, the separate multi-registers could be identified, since no connecting vertical line extends between them (unless, which you should strictly avoid, they are intertwined). However, with the controls, the control line will make it hard (for some gates, impossible) to visually distinguish the connected parts. `yquant` will kindly provide a warning in this case. You may choose to suppress this warning using the boolean key `/yquant/operator/multi warning`.

There is no established style for discontiguous gates. Note that at the moment, main and subordinate style coincide for all gates except for the `measure` gate with a value. In order to still make it possible to visually distinguish discontiguous multi-register gates operating on slices of a single register from just a bunch of single-register gates that are executed in a parallel manner, if controls are present, `yquant`'s default vertical line style for the former case is a wavy line instead of a straight one. Still, the meaning of this should probably be explained. Please feel free to submit issues or pull requests with propositions of how default styles or alternative subordinate gate shapes may additionally help to mitigate the problem.

2.5 Arguments

Every command may take one or multiple arguments. Those are specified in square brackets that precede the command itself. The content of those square brackets is essentially fed to a `\pgfkeys`-like macro. The default path is set appropriately such that the arguments of the command can be accessed without and path specifiers. If the key is not a valid argument for the command or a global argument and it is not given by an absolute path, it is searched for in the `/yquant` namespace. If it cannot be found there, it is passed to `/yquant/operator style`.

Note that commands may have required arguments. If a required argument is missing, an error will be issued.

The `value` attribute can alternatively be given inside curly brackets after the command name and before the register specification. This has the advantage that special characters such as a closing square bracket need not be escaped. If both alternatives are present, the value inside curly brackets takes precedence and a warning is issued.

2.6 Controls

Lots of gates may have controls, i.e., they are only to be executed if some other gate is set or unset. The former case is called a *positive control*, the latter one a *negative control*. Those are indicated by filled and empty circles on the control registers and a vertical line that joins the registers that belong together.

The gate specification is followed by the list of target registers. By then writing a pipe ("|"), the list of positive controls is introduced; this mimics the mathematical syntax "conditioned on" for probabilities or "given" for sets. If there are no positive controls, the list may be empty or, together with the pipe, omitted. Preceded by a tilde ("~"), the list of negative controls then follows; this mimics the syntax of many programming languages that denote logical negation by a tilde. If there are no negative controls, the list may be empty or, together with the pipe, omitted.

2.7 Importing circuits from files

Since version 0.2, yquant provides a simple way to import circuits that are stored in external files. The macro **\yquantimport** can be used in three different contexts:

- Outside of a TikZ picture environment.
  In this case, **\yquantimport**[<options>]{<filename>} will be equivalent to

```
\begin{tikzpicture}
   \begin{yquant}[<options>]
      % the content of <filename> goes here
   \end{yquant}
\end{tikzpicture}
```

  The starred form, **\yquantimport***[<options>]{<filename>}, instead inserts the starred yquant environment. Note that the options are always yquant options; if you want to pass TikZ options, you will have to create the picture environment by yourself or change the option path to the correct one (/tikz/.cd).

- Inside a TikZ picture environment, but outside of a yquant environment. This is the same as before, just that no extra picture environment will be added.

- Inside both a TikZ picture environment and a yquant environment. The file will be inserted directly into the environment. yquant's parser is automatically restarted after this. The content will always be put in a TeX group; if additional options are provided, yquant also inserts a TikZ scope and executes **\yquantset**{<options>} directly after the scope. If **\yquantimport** is used, the content will be read as if the containing environment was an unstarred one; if **\yquantimport*** is used, the content will be read as if the containing environment was a starred one.

Note that yquant internally uses plain TeX's **\import** command (i.e., **\@@import** in LaTeX). However, when the import package is loaded, it uses **\subimport**{\yquantimportpath}{<filename>}, where **\yquantimportpath** defaults to ./—so by changing this, files from other folders may be imported which by themselves again include other files, and the relative path resolution will work.

Note that you may in particular import the content of a subcircuit.

2.8 Defining own gates

Scope
All gate declarations are always global.

Since version 0.2.1, if you want to define a gate that corresponds to a single `box` gate with a certain pre-defined content, you may use the macro `\yquantdefinebox{<name>}[<style>]{<content>}`, which is far more efficient than the much more general `\yquantdefinegate` introduced below. It works in the following way:

- It creates a new gate with name `<name>` that can be accessed as all the other build-in gates. Note that `<name>` is case-insensitive and may not contain spaces. Special characters are allowed if TeX can cope with them (i.e., no comment signs, no unbalanced braces, no backslashes...).

- It creates a style /yquant/operators/every `<name>` and assigns the optional `<style>` to it. If no style is provided, the default style will inherit from `/yquant/operators/every box`.

- It defines `<content>` to be the value that is written into the box. This `<content>` is expanded in a protected manner at the time of gate declaration. You may need to prefix fragile macros by `\protect`.

Sometimes, you may wish to define gates that are more than just a single box—perhaps a succession of multiple gates or even multi-register gates with individual operations on the input registers.
Since version 0.2, `yquant` provides a simple macro that allows this. The macro `\yquantdefinegate{<name>}[<style>]{<content>}` works in the following way:

- It creates a new gate with name `<name>` that can be accessed as all the other built-in gates. Note that `<name>` is case-insensitive and may not contain spaces. Special characters are allowed if TeX can cope with them (i.e., no comment signs, no unbalanced braces, no backslashes...).

- It creates a style /yquant/operators/every `<name>` and assigns the optional `<style>` to it. If no style is provided, the default style will inherit from `/yquant/operators/every custom gate`. This will make the gate "seamless," i.e., avoid highlighting the fact that this is a custom gate.

- It defines a macro that contains `<content>` (expanded in a protected manner) and that will be inserted as a subcircuit whenever this gate is invoked. This in particular means that if you use `\yquantimport` within the gate, the file will only be loaded once at the time of declaration.

When the gate is later drawn, the styles are invoked in the following order—remember custom gates are implemented by means of subcircuits—:

1. `/yquant/every operator`

2. `/yquant/operators/every <name>`

3. `/yquant/operators/every subcircuit box`

4. `/yquant/this operator`

5. `/yquant/operators/this subcircuit box`

Gates defined in this way can only make use of the default gates or other custom gates. They do not accept custom arguments, and it is not possible to declare own, custom shapes in this way (though other predefined shapes may be used). If they are used in a multi-qubit manner, they will never be split into contiguous slices (but their content will be, so if you use the default style that turns off the box, the only way to notice this is that intermediate unaffected gates will not be allowed to place gates within the custom gate).

Redefining existing gates

The above macros will issue an error if the gate already exists. You can use `\yquantredefinebox` or `\yquantredefinegate` to overwrite existing gate definitions. Note that this will overwrite *any* gate, even the built-in ones.

Generally, it is discouraged to make use of this possibility. For custom gates, if you redefine a gate as a box which was previously a general subcircuit-based gate, the macro that contains the subcircuit will still be held in memory. Overwriting built-in gates will not clear the attributes associated to this gate (though required attributes will no longer be required afterwards). Again, this is not a problem but prevents yquant from issuing potentially helpful error message if such a—now meaningless—attribute is used.

Finally, once a built-in gate is overwritten, it cannot be restored. In particular, the register creation pseudo-gates `qubit`, `cbit`, `qubits`, and `nobit` perform some magic that cannot be mimicked with custom gates.

More advanced declaration of custom gates requires the use of backend macros. Refer to yquant-lang.tex for this. For the declaration of custom shapes, see yquant-shapes.tex for examples.

16

3 Configuration

`yquant` uses `pgfkeys` to control its options, which are located in the path `/yquant`. The following list contains all options and styles that are recognized, apart from gate arguments. Those are listed together with their operations.

3.1 Circuit layout

`/yquant/register/minimum height`   default: 3mm

> `yquant` automatically determines the total height of a register as the height of the largest operation. This might be too small for two reasons:
>
> - if the register is used only with small gates (e.g., only as a control, or as a swap), and it does not have a label (or one containing only x-height letters).
>
> - if the register is used only with multi-qubit gates. For those, `yquant` cannot decide where to put the height—and it is easy to see that an equal distribution over all affected registers is not necessarily a good solution. Hence, multi-qubit gates are ignored in the height calculation. Usually, this is not a problem since those operations are large enough as they take the height of all involved registers and separations.
>
> This key provides an easy alleviation of the problem by requiring a minimal height for every register.

`/yquant/register/separation`   default: 1mm

> This key controls the amount of vertical space that is inserted between two successive registers. Half of this value is also the length that multi-`init` or multi-`output` braces extend beyond the mid position of the register.

`/yquant/operator/minimum width`   default: 3mm

> `yquant` automatically determines the width of an operator according to its content. However, single-letter boxes are among the most common operators, and giving them slightly different widths would result in a very uneven spacing, as `yquant` does not use a grid layout but stacks the operators horizontally one after each other. Hence, this key provides a minimum width that will be set for every operator. This does not imply that the *visual* appearance (i.e., the `x radius` key) is enlarged, but that operators of a smaller actual width will be centered in a virtual box of the minimum width.

/yquant/operator/separation   default: 1mm
>   This key controls the amount of horizontal space that is inserted between two
>   successive operators.

/yquant/operator/multi warning   default: true
>   If this key is true, a warning is displayed whenever more than a single multi-
>   register gate, where at least one is discontiguous, are employed together with
>   controls. Even if a visual distinction between control and multi-qubit line may be
>   possible (depending on the style in use), they will overlap and produce unaesthetic
>   output. You may disable this warning globally, on a per-circuit, or even on a per-
>   gate basis.

3.2 Register creation

/yquant/register/default name   default: **\regidx**
>   The printed name that is used by default if a new register is created explicitly
>   (qubit, cbit, qubits; not used for nobit or for implicit declarations) and no
>   value is specified. The following macros are available:
>
>   - **\reg** contains the internal name that is used to identify this register.
>
>   - **\idx** contains the index (zero-based) of the current register within a vector
>   register.
>
>   - **\regidx** expands to **\reg** if the register is of length one, and to **\reg**[\idx]
>   else.
>
>   - **\len** contains the length of the current register vector.

/yquant/every label   default: shape=yquant-text, anchor=center,
                                                                        align=right
>   This style is installed for every single register name label (i.e., upon creation and
>   when used with the init gate). The default style allows to use line breaks in the
>   labels.

/yquant/every initial label   default: anchor=east
>   This style is installed for every single register name label at the left border of the
>   circuit. Hence, it is only used for the init gate if in the yquant* environment, the
>   gate occurs for a new register (which allows to override the default register type).

/yquant/**every qubit label**   default:
> This style is installed for every single register name label of a register of type `qubit`.

/yquant/**every cbit label**   default:
> This style is installed for every single register name label of a register of type `cbit`.

/yquant/**every qubits label**   default:
> This style is installed for every single register name label of a register of type `qubits`.

/yquant/**every multi label**   default: `shift={(-.075, 0)}, draw,`
>   `decoration={gapped brace, mirror}, decorate, /yquant/gapped`
>   `brace/apply shift, every node/.append style={shape=yquant-text,`
>   `anchor=east, align=right, shift={(-.05, 0)}, pos=-1}`
> This style is installed for every register name label that is attached to a multi-qubit register by means of the `init` gate. yquant additionally inserts a straight line that connects the topmost and the bottom-most register at their left ends. The default style turns this line into a brace and places the description at the appropriate position. The decoration `gapped brace` allows to additionally specify the regions in which a line should be drawn by using the `/tikz/decoration/from to` key, which expects a comma-separated list of dimension ranges, and which is automatically populated by yquant. Since it may happen that the arch of the brace needs to be shifted from the value specified in `/tikz/decoration/aspect` (else, it would be drawn into a gap), the special key `/yquant/gapped brace/apply shift` is installed, which transforms the `pos=-1` specification into a position that corresponds to the actual `aspect` value.

/yquant/**every input label**   default:
> This style is installed for every register name label in a `subcircuit` when the register is an input (or input and output) register.

3.3 Register outputs

/yquant/**every output** default: `shape=yquant-text, anchor=west, align=left`
> This style is installed for every `output` label at the end of the circuit. The default style allows to use line breaks in the labels.

/yquant/**every qubit output**   default:
> This style is installed for every `output` label of a register of type `qubit`.

/yquant/**every cbit output**   default:
    This style is installed for every output label of a register of type cbit.

/yquant/**every qubits output**   default:
    This style is installed for every output label of a register of type qubits.

/yquant/**every multi output**   default: shift={(.075, 0)}, draw,
     decoration={gapped brace}, decorate, /yquant/gapped brace/apply
     shift, every node/.append style={shape=yquant-text, anchor=west,
                        align=left, shift={(.05, 0)}, pos=-1}
    This style is installed for every output label that is attached to a multi-qubit
    register. yquant additionally inserts a straight line that connects the topmost and
    the bottom-most register at their right ends. The default style turns this line into a
    brace and places the description at the appropriate position. See /yquant/every
    multi label for a more detailed explanation.

3.4 General styling

/yquant/**every circuit**   default: every node/.prefix style={transform
            shape}, every label/.prefix style={transform shape=false}
    Style that is installed for every yquant and yquant* environment, as if it had
    been given as an option. Since version 0.1.2, the style's default path is, as with
    all other styles, /tikz. The default style will make all nodes (which in particular
    means, all gates) respect outer canvas transformations. Due to TikZ bug #843,
    this must be undone for labels.

/yquant/**every wire**   default: draw
    This style is installed whenever a wire is drawn.

/yquant/**every qubit wire**   default:
    This style is installed whenever a wire for a register of type qubit is drawn.

/yquant/**every cbit wire**   default:
    This style is installed whenever a wire for a register of type cbit is drawn.

/yquant/**every qubits wire**   default:
    This style is installed whenever a wire for a register of type qubits is drawn.

/yquant/**every control line**   default: draw
    This style is used to draw the vertical control line that connects controlled opera-
    tions and their controls.

/yquant/every control   default: shape=yquant-circle, anchor=center,
                                                                   radius=.5mm
    This style is used to draw the node for a control, both positive and negative.

/yquant/every positive control   default: fill=black
    This style is installed for every positive control (i.e., one that conditions on the
    register being in state $|1\rangle$ or 1).

/yquant/every negative control   default: draw
    This style is installed for every negative control (i.e., one that conditions on the
    register being in state $|0\rangle$ or 0).

/yquant/every operator   default: anchor=center
    This style is installed for every gate (and also pseudo-gates such as the slash
    operator) that acts on one or multiple registers.

/yquant/every multi line   default: draw, decoration={snake,
                           amplitude=.25mm, segment length=5pt}, decorate
    This style is used to draw the vertical line that connects discontiguous slices of
    sub-gates.

/yquant/this operator   default:
    This style is appended to the current style installed for an operator; it should be
    used only locally to overwrite any global configuration effect.

/yquant/this control   default:
    This style is appended to the current style installed for a control; it should be used
    only locally to overwrite any global configuration effect.

/yquant/operator style default: /yquant/this operator/.append style={#1}
    This is a shorthand that can be used to modify the appearance of the current
    operator.

/yquant/control style   default: /yquant/every control line/.append
                 style={#1}, /yquant/this control/.append style={#1}
    This is a shorthand that can be used to modify the appearance of the current
    control and its associated line.

/yquant/style   default: /yquant/operator style={#1}, /yquant/control
style={#1}

This is a shorthand that modifies the appearance of both the current operator and
any controls or control lines.

/yquant/operator/multi as single   default: /yquant/every multi
line/.style=/yquant/every control line

This option is automatically set for certain gates such as the swap or the zz gate.
For those gates, neighboring registers will be treated as discontiguous; and this
style will enforce their connecting line to have the style used by control lines.
The default /yquant/every multi line is a wavy line; this allows to distinguish
discontiguous multi-qubit gates from multiple single-qubit gates when using con-
trols. Still, some gates have such an established appearance that—despite being
logically misleading—we rather use the same style as for a control line.

3.5 Styles for operators

/yquant/operators/every barrier default: shape=yquant-line, dashed, draw

This style is installed for every barrier pseudo-gate, i.e., the one that is used to
explicitly denote a separation between "before" and "after" within the circuit.

/yquant/operators/every box   default: shape=yquant-rectangle, draw,
align=center, inner xsep=1mm, x radius=2mm, y radius=2.47mm

This style is installed for every box operator.

/yquant/operators/every custom gate   default: /yquant/operators/this
subcircuit box/.append style={draw=none, inner sep=0pt},
/yquant/register/default name=

This style is by default installed for every user-defined gate (since version 0.2).
User-defined gates are implemented via subcircuits; this style suppresses the box
that surrounds the subcircuit and by default suppresses all register names. This
allows a seamless integration of the gate/subcircuit into the main circuit, without
putting particular emphasis to the fact that what was defined as the custom gate
indeed belongs together.

/yquant/operators/every dmeter   default: shape=yquant-dmeter, x
radius=2mm, y radius=2mm, fill=white, draw

This style is installed for every dmeter gate. The yquant-dmeter shape consists
of a rectangle whose right side is replaced by a circle, resembling the letter "D."

/yquant/operators/every h   default: /yquant/operators/every box
> This style is installed for every `h` (Hadamard) operator.

/yquant/operators/every measure   default: `shape=yquant-measure, x`
`radius=4mm, y radius=2.5mm, draw`
> This style is installed for every `measure` gate. The `yquant-measure` shape is a rectangle that contains a "meter" symbol. It allows for a text to be put inside (e.g., a basis), which then shifts the meter symbol accordingly.

/yquant/operators/every measure meter   default: `draw,`
`-{Latex[length=2.5pt]}`
> This style is applied to the path that resembles the "meter" symbol that is drawn by the `yquant-measure` shape. Due to the default style, the TikZ library `arrows.meta` is automatically loaded with `yquant`.

/yquant/operators/every not   default: `shape=yquant-oplus, radius=1.3mm,`
`draw`
> This style is installed for every `not` or `cnot` gate (which are synonyms, and actually do the same as the Pauli σ_x gate). The `yquant-oplus` shape resembles the addition-modulo-two symbol $\oplus$.

/yquant/operators/every pauli   default: /yquant/operators/every box
> This style is installed for every Pauli operator, i.e., `x`, `y`, and `z`.

/yquant/operators/every phase   default: `shape=yquant-circle, radius=.5mm,`
`fill`
> This style is installed for every `phase` gate $|0\rangle\langle 0| + \mathrm{e}^{\mathrm{i}\phi} |1\rangle\langle 1|$.

/yquant/operators/every slash   default: `shape=yquant-slash, x radius=.5mm,`
`y radius=.7mm, draw`
> This style is installed for every `slash` pseudo-gate, i.e., the one that is used to indicate that a single register line actually denotes multiple registers.

/yquant/operators/every subcircuit   default:
> This style is installed for every `subcircuit`.

/yquant/operators/every subcircuit box   default: /yquant/operators/every
box

> This style is installed for every `subcircuit`. Note that in contrast to all other styles such as `/yquant/operators/every subcircuit` or `/yquant/this operator`, this style is only applied to the "container" node of the subcircuit, but not to the elements in the subcircuit themselves.

/yquant/subcircuit box style default: /yquant/operators/every subcircuit
box/.append style={#1}

> This is a shorthand to apply styles to the subcircuit box only.

/yquant/operators/this subcircuit box   default:

> This style is appended to the current style installed for the `subcircuit`, but will not apply to its contents. Additionally, this style will be reset to an empty style at the beginning of each subcircuit, so that it really only applies to exactly the subcircuit box it is explicitly specified on, not to nested subcircuit boxes.

/yquant/this subcircuit box style   default: /yquant/operators/this
subcircuit box/.append style={#1}

> This is a shorthand to apply styles to the current subcircuit box only.

/yquant/operators/every swap   default: shape=yquant-swap, radius=.75mm,
draw

> This style is installed for every `swap` gate that interchanges two qubits. The yquant-swap shape consists of a single cross.

/yquant/operators/every wave   default: shape=yquant-circle, radius=.5mm,
fill

> This style is installed for every `correlate` gate.

/yquant/operators/every x   default: /yquant/operators/every pauli

> This style is installed for every Pauli operator σ_x, i.e., `x`.

/yquant/operators/every xx default: shape=yquant-rectangle, radius=.75mm,
draw

> This style is installed for every `xx` gate in symmetrized notation ($|++\rangle\langle++| + |+-\rangle\langle+-| + |-+\rangle\langle-+| - |--\rangle\langle--|$).

/yquant/operators/every y   default: /yquant/operators/every pauli

> This style is installed for every Pauli operator σ_y, i.e., `y`.

`/yquant/`**`operators/every z`**   default: `/yquant/operators/every pauli`
   This style is installed for every Pauli operator σ_z, i.e., **`z`**.

`/yquant/`**`operators/every zz`**   default: `shape=yquant-circle, radius=.5mm,`
                                                                     `fill`
   This style is installed for every **`zz`** gate (aka CPHASE) in symmetrized notation
   $(|00\rangle\langle00| + |01\rangle\langle01| + |10\rangle\langle10| - |11\rangle\langle11|)$.

4 Doing the impossible

yquant will almost certainly never be able to do everything an author has in mind. Sometimes, there is the need to draw something non-standard, and this cannot be implemented in the yquant language. However, since yquant is a layer on top of TikZ, it should be very hard to find something (meaningful) that cannot be done by combining the power of both packages.

Before or after any gate, you may interrupt the yquant instructions to perform arbitrary TikZ path operations. After every such operation, yquant will automatically restart its parser so that you can fluently jump between yquant and TikZ code. You can even interject arbitrary TeX code (or, say, low-level pgf commands); however, then, yquant is not able to restart its parser. For this reason, after the last command in a block of TeX commands, you must issue `\yquant`, which then re-enables the yquant language.

The feature to perform arbitrary TikZ operations is powerful in itself, but would be of limited use were there no way to access the elements in the quantum circuit. yquant provides a global attribute name that can be assigned to every gate. All quantum operations are in fact TikZ nodes, and the name you give to them then becomes a TikZ name, which you can easily reference to get the coordinates of a particular operator. Note that the name you specify is only available if a single register is targeted. The name is suffixed by `-\idx`, where `\idx` refers to the (zero-based) index of the operation ordered from top to bottom (i.e., if an operator acts on two qubits and should be named op, the topmost operator will be available as op-0 and the second as op-1). Multiple slices in a discontiguous multi-register are additionally suffixed by `-s<slice index>`. All controls are also named, suffixed by `-p\idx` or `-n\idx` for positive and negative controls (i.e., the topmost positive control of the previous operator will be available as op-p0). Counters for target registers, positive, and negative controls are all independent. Finally, you can even access names within a subcircuit, provided you give a name to the subcircuit. All nodes in the subcircuit will then have the name `<subcircuit name>-<name specified in the subcircuits>`. For nested subcircuits, you will get multiple prefixes.

All yquant shapes have the anchors available you would typically expect from a TikZ shape of the given outline. Before version 0.1.2, yquant shapes needed to provide a circuit anchor and projection anchors. These were removed in version 0.1.2 in favor of the common center anchor and clipping paths. The latter is a yquant extension to TikZ shapes: it is an additional path that must be declared for a given shape. This path has to provide the "clipping outline," i.e., anything that should not contain register or control lines. There may be a difference between horizontal and vertical clipping outlines. To understand clipping paths, yquant's

drawing pipeline needs to be explained.

- In a first run—this is what happens directly at the position where you type the gate command—yquant will "virtually" draw the gates in order to determine their dimensions and calculate register heights. The actual drawing commands are written to a macro (this is the cause that some macros must be preceded by `\protect` if used in a gate value).

- When `\end{yquant}` is encountered, the actual drawing commands are executed.

- Unless the operation changes the wire type or style, do the following (first two items for every register at which an operator node has to be created).
  - Create the operator node at the appropriate position.
  - Call `\pgfshapeclippath` on the newly created node. This will first determine whether the node was stroked; if not, `\pgflinewidth` is set to zero. Then, it will call the horizontal clipping path, which is supposed to create some soft path commands. Those soft path commands are collected in a macro on a per-register basis and the soft path is cleared. The same happens for the vertical clipping path, which is collected in a macro on a per-operation basis.
  - If control lines or multi lines are to be drawn, the vertical clipping path commands are now executed and installed as an inverted[1] clipping path.
  - Control lines and multi lines are drawn (in this order) from one to the next `center` anchor. Due to the clipping commands, this will create a perfect connection with the shape of the gate, but even transparent gates are possible without the lines being visible.

- If the operation changes the wire type or style, or if there is no operation left on this register, the following is done.
  - Load the clipping paths accumulated for all the gates acting on this register and install the inverted clipping.

---

[1]Inverting the clipping has two consequences: First, the region of inversion in vertical direction is given by the width of the operators. The line cannot be wider than the operator. Second, if you specify a register multiple times, whether as target, control, or mixed, funny effects can be expected, as the clipping region is inverted multiple times. Note that using a register more than once is always an error, but yquant does not check for it due to the high overhead.

- Draw the wire as one continuous line from where the last wire ended (or the beginning of the circuit) to the center of the last gate, or to the common end position for all wires of the circuit.

- Remove the clipping paths stored so far on this register, apart from the clipping on the last gate (which will be needed again if this was not the end of the circuit).

5 Reference: Gates and operations

This section lists all operations yquant currently understands. It also details all arguments that can be given to customize the operation, apart from `name`, which is always available. Note that the [value=<value>] attribute can (and should) alternatively be given as a braced expression that follows the name of the register.

5.1 `addstyle`

Syntax: setstyle <target>;
This is an invisible pseudo-gate that immediately changes the TikZ style with which the register lines of all target registers are drawn. It adds to the styles that are already installed. Use `setstyle` to replace styles. It may not span multiple registers and does not allow for controls.
   Possible attributes:

- [value=<styles>] (required)
  Denotes the new styles; this should be a string that could be passed to **\tikzset**.

5.2 `align`

Syntax: align <target>;
This is an invisible pseudo-gate that enforces all affected registers to share a common horizontal position for their next gate, which is determined by the largest position of all gates involved. It may not span multiple registers and does not allow for controls.
   Possible attributes: none

5.3 `barrier`

Syntax: barrier <target>;
This is a pseudo-gate that denotes some physical barrier that ensures execution with a specific timing; it is basically a visible version of the `align` gate, denoted by a vertical line. It may span multiple registers, but does not allow for controls. The style /yquant/operators/every barrier is installed.
   Possible attributes: none

5.4 `box`

Syntax: `box <target> | <pcontrol> ~ <ncontrol>;`
This is a generic register of a rectangular shape that can be filled with arbitrary content. It may span multiple registers and allows for controls. The style `/yquant/operators/every box` is installed.
   Possible attributes:

- `[value=<value>]`
   Denotes the content of the box.

5.5 `cbit`

Syntax: `cbit <name>[<len>];`
Declares a register of type `cbit`.
   see `qubit`

5.6 `correlate`

Syntax: `correlate <target>;`
This is a pseudo-gate that indicates a correlation (usually a Bell-state) present between the multi-registers involved. This gate should span multiple registers and does not allow for controls.
   Possible attributes: none

5.7 `cnot`

Syntax: `cnot <target> | <pcontrol> ~ <ncontrol>;`
This is a synonym for the `not` gate. Note that despite its name, controls are not mandatory and also here, the style `/yquant/operators/every not` is installed.

5.8 `discard`

Syntax: `discard <target>;`
This is an invisible pseudo-gate that changes the type of all target registers to `nobit`, i.e., no line will be drawn for them. This has effect already for the outgoing line of the last visible gate on the target registers. The gate may not span multiple registers and does not allow for controls. To change into a register type on-the-fly into something different from `nobit`, use the `settype` pseudo-gate.
   Possible attributes: none

5.9 `dmeter`

Syntax: `dmeter <target>;`
This is a measurement gate, denoted by a "D" shape. It changes the type of all targets involved. It may span multiple registers, but does not allow for controls. The style `/yquant/operators/every dmeter` is installed.
   Possible attributes:

- `[value=<value>]`
  Allows to specify a text that will be included inside the gate, possible enlarging its width. For outside texts, use Ti*k*Z `labels` instead.

- `[type=<qubit|cbit|qubits>]`
  Allows to specify the type into which the affected targets are converted. Default is `cbit`.

5.10 `h`

Syntax: `h <target> | <pcontrol> ~ <ncontrol>;`
This is a Hadamard gate, $\frac{1}{\sqrt{2}}(|0\rangle\langle0|+|0\rangle\langle1|+|1\rangle\langle0|-|1\rangle\langle1|)$, denoted by a rectangle that contains the letter H. It may not span multiple registers, but allows for controls. The style `/yquant/operators/every h` is installed.
   Possible attributes: none

5.11 `hspace`

Syntax: `hspace <target>;`
This is an invisible pseudo-gate that inserts a certain amount of white space into all target registers. It may not span multiple registers and does not allow for controls.
   Possible attributes:

- `[value=<dim>]` (required)
  Gives the amount of white space that is to be inserted. Must be a valid TeX dimension.

5.12 `init`

Syntax: `init <target>;`
This is a pseudo-gate that (re)initializes a registers to a given state. It may span multiple registers, but does not allow for controls. The style `/yquant/every label` is installed. Note that this pseudo-gate, unlike all others, behaves differently if it the first operation acting on a register: in this case, it does not increment

the horizontal position, but uses the space available to the left. If it is the first operation, the style `/yquant/every initial label` is installed additionally. For multiple registers, the style `/yquant/every multi label` is installed at the end, and a path is constructed that extends from the left end of the first to the left end of the last register in the multi-register compound.

Possible attributes:

- `[type=<qubit|cbit|qubits>]`
  Allows to specify the type into which the affected target registers are converted. Default is the type of the first target register that is different from `nobit`, or `qubit` if they all are `nobit`. The style `/yquant/every <type> label` is installed additionally.

- `[value=<value>]` (required)
  Denotes the label that is printed to the left of the wire.

  Inside the value, `\idx` expands to the current index within the register list.

5.13 `measure`

Syntax: `measure <target>;`
This is a measurement gate, denoted by a rectangle with a meter symbol. It changes the type of all targets involved. It may span multiple registers, but does not allow for controls. The style `/yquant/operators/every measure` is installed.

Possible attributes:

- `[type=<qubit|cbit|qubits>]`
  Allows to specify the type into which the affected targets are converted. Default is `cbit`.

- `[value=<value>]`
  Allows to specify a text that will be included at the bottom of the rectangle (which will shift the meter symbol upwards accordingly). For outside texts, use TikZ labels instead.

5.14 `nobit`

Syntax: `nobit <name>[<len>];`
Declares a register of type `nobit`. The `<name>` must be a self-chosen name for the register which was not previously used as a register name in this yquant environment. Names are case-insensitive. The register can be made into a vector register by specifying `<len>` (default 1).

Possible attributes:

- [out] or [ancilla] (required in subcircuits)
  see `qubit`

5.15 `not`

Syntax: `not <target> | <pcontrol> ~ <ncontrol>;`
This is a NOT gate, $|0\rangle\langle 1|+|1\rangle\langle 0|$, denoted by the $\oplus$ symbol. It may not span multiple registers, but allows for controls. Due to its common usage, the synonymous gate `cnot` is provided. The style `/yquant/operators/every not` is installed.
   Possible attributes: none

5.16 `output`

Syntax: `output <target>;`
This is a pseudo-gate that allows to write some text at the very end of the register line. It may only be specified once per register. It may span multiple registers, but does not allow for controls. The style `/yquant/every output` is installed, and also the style `/yquant/every <type> output`, where `<type>` is the type of the affected register (at the time of printout). For outputs on multiple registers, the style `/yquant/every multi output` is installed instead of the two previously mentioned ones; and additionally, a path is constructed that extends from the first to the last register in the multi-register compound.
   Possible attributes:

- [value=<value>] (required)
  Denotes the text that is to be printed. Inside the value, `\idx` expands to the current index within the register list.

5.17 `phase`

Syntax: `phase <name> | <pcontrol> ~ <ncontrol>;`
This is a phase gate, $|0\rangle\langle 0| + e^{i\phi}\,|1\rangle\langle 1|$, denoted by a filled circle. It may not span multiple registers, but allows for controls (and should have them, to make any sense). The style `/yquant/operators/every phase` is installed.
   Possible attributes:

- [value=<value>] (required)
  Denotes the angle ϕ that is to be printed together with the gate. Position and appearance can be influenced by setting the position of TikZ labels, as

33

this is internally used. Note that at the moment, it is not possible to change any label options on a gate-type basis, only locally or fully globally (TikZ feature request #811).

5.18 `qubit`

Syntax: `qubit <name>[<len>];`
Declares a register of type `qubit`. The `<name>` must be a self-chosen name for the register which was not previously used as a register name in this yquant environment. Names are case-insensitive. The register can be made into a vector register by specifying `<len>` (default 1).
   Possible attributes:

- `[after=<regname>]`
  If given, the register will start not at the left of the circuit but instead at the position at which the last gate in the register `<regname>` ended.

  This attribute may not be given in combination with `[in]` or `[inout]`.

- `[in]`, `[out]`, `[inout]`, or `[ancilla]`
  Default: `[ancilla]` for top-level circuits (do not change there); `[inout]` for subcircuits.

  Determines how a subcircuit interacts with its parent circuit.

  Registers declared with the `[ancilla]` attribute are available only to the subcircuit; they cannot be connected to an outside wire.

  Registers declared with the `[in]` or `[inout]` attribute will expect an outer wire of the same type to be present and will then be identical with this outer wire. Any changes applied to the wire within the subcircuit automatically also happen on the associated outer wire. If the attribute is `[in]`, the wire will automatically be discarded at the end of the subcircuit (and hence also in the outer circuit, where it may be re-initialized). This is different from applying the `discard` gate in that the wire will still extend until the end of the subcircuit and may thus receive proper `output`s.

  Registers declared with the `[out]` attribute will expect a discarded outer wire to be present, which will be initialized to a `qubit` at the beginning of the subcircuit, and from then on be identical with the outer wire.

- `[value=<value>]`
  Denotes the label that is printed to the left of the wire. If the value is omitted, the default is used (`/yquant/register/default name`, preinitialized to `\regidx`).

34

Inside the value, `\reg` expands to `<name>`, `\len` expands to `<len>`, `\idx` expands to the current index within the vector register ($0 \leq$ `\idx` $<$ `<len>`), and `\regidx` expands to `\reg` if `<len>` is one, or to `\reg[\idx]` else.

5.19 qubits

Syntax: qubits `<name>[<len>]`;
Declares a register of type `qubits`.
  see *qubit*

5.20 setstyle

Syntax: setstyle `<target>`;
This is an invisible pseudo-gate that immediately changes the Ti*k*Z style with which the register lines of all target registers are drawn. It replaces all previous styles. Use `addstyle` to accumulate styles. It may not span multiple registers and does not allow for controls.
  Possible attributes:

- [value=`<styles>`] (required)
  Denotes the new styles; this should be a string that could be passed to `\tikzset`.

5.21 settype

Syntax: settype `<target>`;
This is an invisible pseudo-gate that immediately changes the type of the targets registers, taking effect with the output line extending from the last drawn gate. It may not span multiple registers and does not allow for controls.
  Possible attributes:

- [value=`<qubit|cbit|qubits>`] (required)
  Denotes the new type that is assigned to all registers. To change the type to `nobit`, use the `discard` pseudo-gate instead.

5.22 setwire

Deprecated as of version 0.1.2. Use *settype* *instead.*

5.23 `slash`

Syntax: `slash <target>;`
This is a pseudo-gate used to denote that a single line actually represents multiple registers. It is drawn as a short slash through the line of the register. Note that this gate, in contrast to all others, is positioned on the line extending from the last gate or the initialization line of the registers and does not advance the register's horizontal position. The style `/yquant/operators/every slash` is installed.
    Possible attributes: none

5.24 `subcircuit`

Syntax: `subcircuit <target>;`
This is a subcircuit gate which inserts independent quantum circuits at the current position within the circuit. It may span multiple registers, but is never split into contiguous slices. It allows for controls and may change the type of any target involved, depending on the particular subcircuit. The style `/yquant/operators/every subcircuit` is installed.
    Possible attributes:

- `value=<subcircuit>` (required)
  Denotes the content of the subcircuit. It is specified in the usual syntax of `yquant`. Note that, regardless of the outer environment, a subcircuit always implicitly uses the unstarred form, i.e., you must declare every register explicitly before its first usage. This is to make sure that the interface of the circuit, i.e., which registers are taken as input and/or output parameters and in which order, is not accidentally mistaken.

  The mapping between input and output registers is trivial for single-qubit uses. For multi-qubit uses, it works in the following way—in short, it matches in visual order. You declare input and output registers by using the appropriate attributes on the `qubit`, `cbit`, `qubits` (or even `nobit`) gates. The list of all non-ancillas, from the topmost to the bottom-most, forms the list of parameter registers of the subcircuit. This is exactly the number of registers that must be supplied within one multi-qubit target. Also within the multi-qubit target, we sort all registers from the topmost to the bottom-most (in the order as they visually appear, not the order in which they are entered). Those two lists of equal length are then mapped $1 : 1$ to each other. Intermixing with ancillas is possible at every position and will lead to a vertical shift of the wires, until all registers, inner and outer, can be displayed flawlessly.

As subcircuits follow the same rules as ordinary circuits, it is possible to mix them with arbitrary TeX code, and also to access named gates within the subcircuit—but note that named gates in the outer circuit cannot be accessed (at least unless you play with the `name prefix` key in TikZ). In order to access inner nodes from the outer circuit, the subcircuit itself must be named; the inner nodes are then prefixed by the name of the subcircuit and a dash.

It is possible to nest subcircuits arbitrarily.

5.25 `swap`

Syntax: swap <targets> | <pcontrol> ~ <ncontrol>;
This is the two-qubit SWAP gate $|00\rangle\langle00| + |01\rangle\langle10| + |10\rangle\langle01| + |11\rangle\langle11|$ that exchanges two qubits. It is denoted by crosses at the affected registers which are connected by a control line. It may span multiple registers (in fact, it should always span exactly two registers, though yquant does not enforce this), and it allows for controls. However, refrain from combining *multiple* two-qubit targets *together* with controls. The control line will extend from the first to the last of all registers involved in the operation, so that it is impossible to discern visually which registers should actually be swapped. Using multiple swaps without controls in one operation is fine, as well as a single controlled swap. The style /yquant/operators/every swap is installed.
   Possible attributes: none

5.26 `x`

Syntax: x <target> | <pcontrol> ~ <ncontrol>;
This is a Pauli σ_x gate $|0\rangle\langle1| + |1\rangle\langle0|$, denoted by a rectangle that contains the letter X. It may not span multiple registers, but allows for controls.
The style /yquant/operators/every x is installed.
   Possible attributes: none

5.27 `xx`

Syntax: xx <targets>;
This is a symmetric flip gate, denoted by joined open squares. It should span multiple registers and it allows for controls. The same warnings as for the swap gate apply. The style /yquant/operators/every xx is installed.
   Possible attributes: none

5.28 y

Syntax: `y <target> | <pcontrol> ~ <ncontrol>;`
This is a Pauli σ_y gate $-\mathrm{i}\,|0\rangle\langle 1| + \mathrm{i}\,|1\rangle\langle 0|$, denoted by a rectangle that contains the letter Y. It may not span multiple registers, but allows for controls.
The style `/yquant/operators/every y` is installed.
 Possible attributes: none

5.29 z

Syntax: `z <target> | <pcontrol> ~ <ncontrol>;`
This is a Pauli σ_z gate $|0\rangle\langle 0| - |1\rangle\langle 1|$, denoted by a rectangle that contains the letter Z. It may not span multiple registers, but allows for controls.
The style `/yquant/operators/every z` is installed.
 Possible attributes: none

5.30 zz

Syntax: `zz <targets>;`
This is a symmetric phase gate, denoted by joined filled circles. It should span multiple registers, but does not allow for controls. The same warnings as for the `swap` gate apply. The style `/yquant/operators/every zz` is installed.
 Possible attributes: none

6 Examples

This section will contain lots of examples. On the left-hand side, the output is given, while the code to construct the example is on the right. All examples that are provided originate from the examples supplied with `qasm`, `qcircuit`, and `quantikz`. We will essentially follow their manuals example-by-example, which gives a nice comparison in how to achieve the given feature using these packages and `yquant` instead. All examples of course require inclusion of the `yquant` package in the preamble, and some also require `braket`.

6.1 `qasm` documentation

The `qasm` documentation most often names the registers in the way $|\text{register}_{\text{index}}\rangle$. This can be achieved by writing

```
qubit {$\ket{<name>_{\idx}}$} <name>[<len>];
```
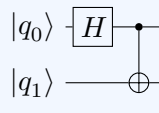
but if you want to realize this naming scheme for all circuits in your document, it is more convenient to say

```
\yquantset{register/default name=$\ket{\reg_{\idx}}$}
```

in the preamble, as is done here.

test1 (create an EPR pair)

$|q_0\rangle$ —[H]——•——
$|q_1\rangle$ ————————⊕——

```
\begin{tikzpicture}
    \begin{yquant}
        qubit q[2];

        h q[0];
        cnot q[1] | q[0];
    \end{yquant}
\end{tikzpicture}
```

test2 (simple teleportation circuit)

$|q_0\rangle$ ——•——$\boxed{H}$——$\boxed{\measuredangle}$——————

$|q_1\rangle$ —$\boxed{H}$—•——⊕——$\boxed{\measuredangle}$————

$|q_2\rangle$ ————⊕————$\boxed{Z}$—$\boxed{X}$—
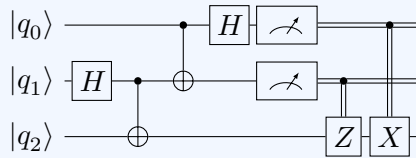
```
\begin{tikzpicture}
    \begin{yquant}
        qubit q[3];

        h q[1];
        cnot q[2] | q[1];
        cnot q[1] | q[0];
        h q[0];
        measure q[0-1];

        z q[2] | q[1];
        x q[2] | q[0];
    \end{yquant}
\end{tikzpicture}
```

test3 (swap circuit)

$|q_0\rangle$ ——•——⊕——•——

$|q_1\rangle$ ——⊕——•——⊕——
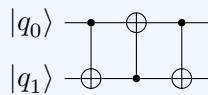
```
\begin{tikzpicture}
    \begin{yquant}
        qubit q[2];

        cnot q[1] | q[0];
        cnot q[0] | q[1];
        cnot q[1] | q[0];
    \end{yquant}
\end{tikzpicture}
```

test4 (quantum fourier transform on three qubits)

$|j_0\rangle$ —[H]—[S]—[T]——————×——

$|j_1\rangle$ ————[H]—[S]————————

$|j_2\rangle$ —————————[H]—×——
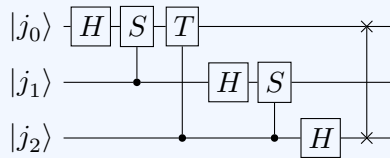
```
\begin{tikzpicture}
    \begin{yquant}
        qubit j[3];

        h j[0];
        box {$S$} j[0] | j[1];
        box {$T$} j[0] | j[2];
        h j[1];
        box {$S$} j[1] | j[2];
        h j[2];
        swap (j[0, 2]);
    \end{yquant}
\end{tikzpicture}
```

test5 (demonstrate arbitrary qubit matrix ops)

$|j_0\rangle$ ————•————$\begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix}$——

$|j_1\rangle$ —$\begin{bmatrix} e^{i\alpha} & 0 \\ 0 & e^{-i\alpha} \end{bmatrix}$————————
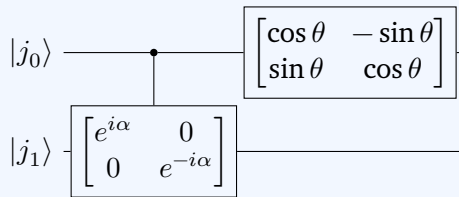
```
% \usepackage{amsmath}
\begin{tikzpicture}
    \begin{yquant}
        qubit j[2];

        box {$\begin{bmatrix}
            e^{i \alpha} & 0 \\
            0 & e^{-i \alpha}
        \end{bmatrix}$} j[1] | j[0];
        box {$\begin{bmatrix}
            \cos\theta & -\sin\theta \\
            \sin\theta & \cos\theta
        \end{bmatrix}$} j[0];
    \end{yquant}
\end{tikzpicture}
```

test6 (demonstrate multiple-qubit controlled single-q-gates)
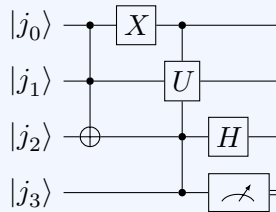

```
\begin{tikzpicture}
    \begin{yquant}
        qubit j[4];

        cnot j[2] | j[0, 1];
        x j[0];
        box {$U$} j[1] | j[0, 2-3];
        h j[2];
        measure j[3];
    \end{yquant}
\end{tikzpicture}
```

test7 (measurement of operator with correction)
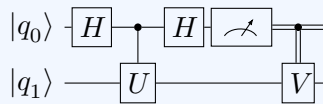

```
\begin{tikzpicture}
    \begin{yquant}
        qubit q[2];

        h q[0];
        box {$U$} q[1] | q[0];
        h q[0];
        measure q[0];
        box {$V$} q[1] | q[0];
    \end{yquant}
\end{tikzpicture}
```

test8 (stage in simplification of quantum teleportation)

$$|q_0\rangle = |\psi\rangle$$
$$|q_1\rangle = |0\rangle \quad \boxed{H}$$
$$|q_2\rangle = |0\rangle$$
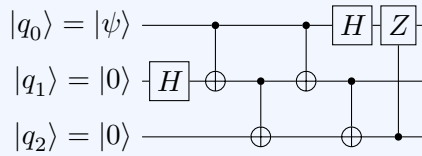
```
\begin{tikzpicture}
  \begin{yquant}
    qubit {$\ket{q_0} = \ket\psi$}
    ↪  q[1];
    qubit {$\ket{q_{\idx}} = \ket0$}
    ↪  q[+2];

    h q[1];
    cnot q[1] | q[0];
    cnot q[2] | q[1];
    cnot q[1] | q[0];
    h q[0];
    cnot q[2] | q[1];
    z q[0] | q[2];
  \end{yquant}
\end{tikzpicture}
```

Note that we left out two Hadamards at the end.

Before version 0.1.1, the recommended approach (which of course still works) to define a vector qubit register with various texts was to use case discrimination on **\idx**, for example in the following manner:

```
qubit {$\ket{q_{\idx}} = \ifcase\idx\relax \ket\psi \else \ket0 \fi$}
↪  q[3];
```

test9 (two-qubit gate circuit implementation of Toffoli)

$$|q_0\rangle$$
$$|q_1\rangle$$
$$|q_2\rangle \quad \boxed{\sqrt{X}} \quad \boxed{\sqrt{X}^\dagger} \quad \boxed{\sqrt{X}}$$
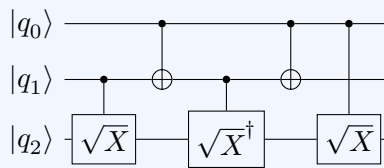
```
\begin{tikzpicture}
  \begin{yquant}
    qubit q[3];

    box {$\sqrt X$} q[2] | q[1];
    cnot q[1] | q[0];
    box {$\sqrt X^\dagger$} q[2] |
    ↪  q[1];
    cnot q[1] | q[0];
    box {$\sqrt X$} q[2] | q[0];
  \end{yquant}
\end{tikzpicture}
```

test10 (multi-qubit gates also demonstrates use of classical bits)
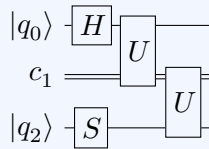

```
\begin{tikzpicture}
  \begin{yquant}
    qubit {$\ket{q_0}$} q;
    cbit {$c_1$} c;
    qubit {$\ket{q_2}$} q[+1];

    h q[0];
    box {$U$} (q[0], c);
    box {$S$} q[1];
    box {$U$} (c, q[1]);
  \end{yquant}
\end{tikzpicture}
```

Instead of a discontigous vector register, we could also have used three scalar registers. The labels chosen for qasm do not fit well to the indices yquant assigns. We might also have used a three-register vector and used the settype pseudo-gate to immediately change the second register into a classical one, which would give indices matching the labels—but still, the registers would have a common name, which would make this a very unnatural approach.

test11 (user-defined multi-qubit ops)
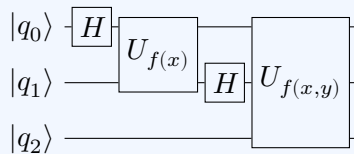

```
\begin{tikzpicture}
  \begin{yquant}
    qubit q[3];

    h q[0];
    box {$U_{f(x)}$} (q[0, 1]);
    h q[1];
    box {$U_{f(x, y)}$} (q);
  \end{yquant}
\end{tikzpicture}
```

Here we used the fact that a vector register can also be addressed as a whole. Instead of (q), we could have also written, e.g., (q[0]-q[2]) or (q[0-2]), or enumerated all sub-registers in a comma-separated list.

test12 (multi-qubit controlled multi-qubit operations)

$|q_0\rangle$ — H — • — H — V

$|q_1\rangle$ — U

$|q_2\rangle$ — •
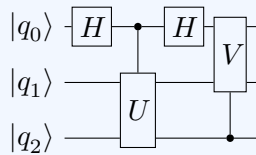
```
\begin{tikzpicture}
    \begin{yquant}
        qubit q[3];

        h q[0];
        box {$U$} (q[1-2]) | q[0];
        h q[0];
        box {$V$} (q[0-1]) | q[2];
    \end{yquant}
\end{tikzpicture}
```

test13 (three-qubit phase estimation circuit with QFT and controlled-U)

$|j_0\rangle = |0\rangle$ — H — • — H — • — • — measure

$|j_1\rangle = |0\rangle$ — H — S — H — • — measure

$|j_2\rangle = |0\rangle$ — H — T — S — H — measure

$|s_0\rangle$ — U^4 — U^2 — U

$|s_1\rangle$ — U^4 — U^2 — U
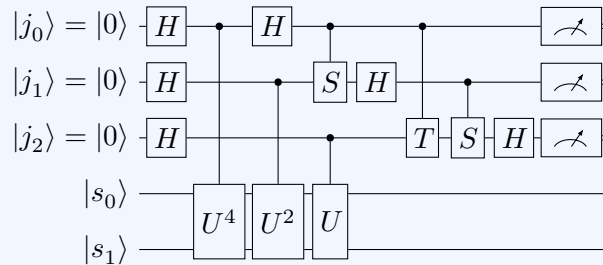
```
\begin{tikzpicture}
    \begin{yquant}
        qubit {$\ket{j_{\idx}} = \ket0$} j[3];
        qubit s[2];

        h j;
        box {$U^4$} (s) | j[0];
        box {$U^2$} (s) | j[1];
        box {$U$} (s) | j[2];
        h j[0];
        box {$S$} j[1] | j[0];
        h j[1];
        box {$T$} j[2] | j[0];
        box {$S$} j[2] | j[1];
        h j[2];
        measure j;
    \end{yquant}
\end{tikzpicture}
```

test14 (three-qubit FT QEC circuit with syndrome measurement)
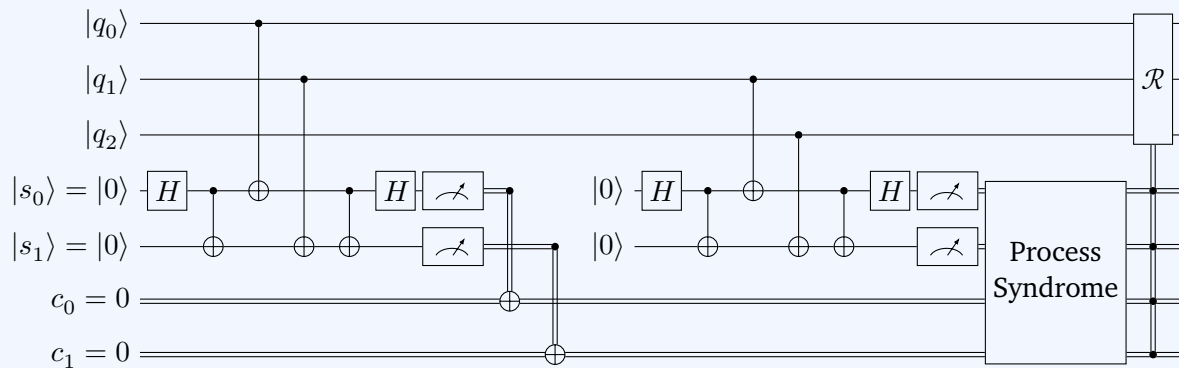

```
\begin{tikzpicture}
   \begin{yquant}
      qubit q[3];
      qubit {$\ket{s_{\idx}} = \ket0$} s[2];
      cbit {$c_{\idx} = 0$} c[2];

      h s[0];
      cnot s[1] | s[0];
      cnot s[0] | q[0];
      cnot s[1] | q[1];
      cnot s[1] | s[0];
      h s[0];
      measure s;
      cnot c[0] | s[0];
      cnot c[1] | s[1];
      discard s; % to suppress wires extending until re-initialization

      init {$\ket0$} s;
      h s[0];
      cnot s[1] | s[0];
      cnot s[0] | q[1];
      cnot s[1] | q[2];
      cnot s[1] | s[0];
      h s[0];
      measure s;

      box {Process\\Syndrome} (s, c);
      box {$\symcal R$} (q) | s, c;
   \end{yquant}
\end{tikzpicture}
```

test15 ("D-type" measurement)

$|q_0\rangle = |\psi\rangle$ —•— $\boxed{HZ_\theta}$

$|q_1\rangle = |+\rangle$ —•—
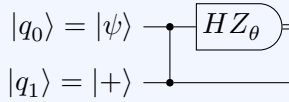
```
\begin{tikzpicture}
  \begin{yquant}
    qubit {$\ket{q_0} = \ket\psi$} q;
    qubit {$\ket{q_1} = \ket+$} q[+1];

    zz (q);
    dmeter {$H Z_\theta$} q[0];
  \end{yquant}
\end{tikzpicture}
```

test16 (example from Nielsen paper on cluster states)

$|q_0\rangle = |\psi\rangle$ ——/——

$|q_1\rangle = |\psi\rangle$ —•—

$|q_2\rangle = |\phi\rangle$ —•—

$|q_3\rangle = |0\rangle$ ——⊕—▷
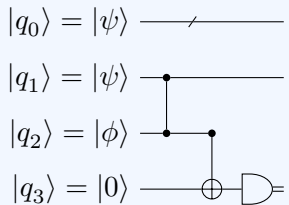
```
\begin{tikzpicture}
  \begin{yquant}
    qubit {$\ket{q_{\idx}} = \ket\psi$} q[2];
    qubit {$\ket{q_2} = \ket\phi$} q[+1];
    qubit {$\ket{q_3} = \ket0$} q[+1];

    zz (q[1], q[2]);
    align q;
    cnot q[3] | q[2];
    slash q[0];
    dmeter q[3];
    discard q[2];
  \end{yquant}
\end{tikzpicture}
```

We needed to include an `align` pseudo-gate to put the slash at the desired position. Usually, this would be sufficient to put the `cnot` and the `slash` gate directly under each other, as it is in the `qasm` example. However, the `slash` gate is special in that it does not need horizontal space and is put with only half of the usual operator separation into the circuit (for this reason, it can be put at the beginning of a wire without creating weird shifts with respect to the "unslashed" registers—it is put in the initial line that every wire even without an operation has). Hence, you should normally only use the `slash` gate as the very first gate in a circuit. It is not possible to construct the exact same appearance as in the `qasm` example. Note that `discard` currently just drops the wire directly after the last operation.

test17 (example from Nielsen paper on cluster states)

$|q_0\rangle = |\psi\rangle$

$|q_1\rangle = |+\rangle$

$|q_2\rangle = |+\rangle$

$|q_3\rangle = |\phi\rangle$
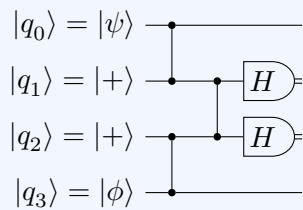
```
\begin{tikzpicture}
  \begin{yquant}
    qubit {$\ket{q_0} = \ket\psi$} q;
    qubit {$\ket{q_{\idx}} = \ket+$} q[+2];
    qubit {$\ket{q_3} = \ket\phi$} q[+1];

    zz q[(0-1), (2-3)];
    zz (q[1-2]);
    dmeter {$H$} q[1-2];
  \end{yquant}
\end{tikzpicture}
```

This example shows how the multi-qubit delimiter (the parenthesis) can even be used within indices.

test18 (multiple-control bullet op)

$|q_0\rangle = |\psi\rangle$

$|q_1\rangle = |+\rangle$

$|q_2\rangle = |+\rangle$

$|q_3\rangle = |\phi\rangle$
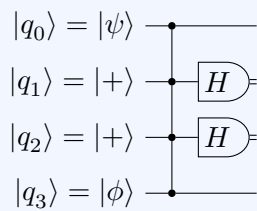
```
\begin{tikzpicture}
  \begin{yquant}
    qubit {$\ket{q_{\idx}} =
    ↪ \ket{\ifcase\idx\relax \psi \or + \or +
    ↪ \or \phi \fi}$} q[4];

    zz (q);
    dmeter {$H$} q[1, 2];
  \end{yquant}
\end{tikzpicture}
```

This gate is probably a generalization of zz, $\mathbb{1} - 2\,|1\cdots1\rangle\langle1\cdots1|$, and indeed since version 0.1.2, we can use zz for this purpose. This time, we also used the common way to initialize a gate with various identifiers as was done before version 0.1.1, using case distinctions.

6.2 `qcircuit` documentation

For a better orientation, we use the same section headings as the `qcircuit` manual. The manual uses unnamed registers a lot; often, we will use the `yquant*` environment to make things more concise. As `qcircuit` uses a much large separation between the operators than `yquant`'s default, we globally say
`\yquantset{operator/separation=3mm}`.

6.2.1 I. Introduction
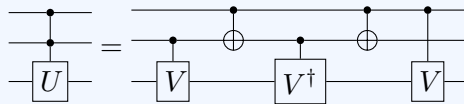

```
\begin{tikzpicture}[baseline=(current bounding box.center)]
   \begin{yquant*}
      box {$U$} q[2] | q[0, 1];
   \end{yquant*}
\end{tikzpicture}
$=$
\begin{tikzpicture}[baseline=(current bounding box.center)]
   \begin{yquant*}
      box {$V$} q[2] | q[1];
      cnot q[1] | q[0];
      box {$V^\dagger$} q[2] | q[1];
      cnot q[1] | q[0];
      box {$V$} q[2] | q[0];
   \end{yquant*}
\end{tikzpicture}
```

Here, we chose to realize the equality using two `tikzpictures` with appropriately set baselines.

If mangling with the baselines becomes problematic, a different approach would be to use an outer `tikzpicture` with three nodes (left circuit, equals, right circuit); but the circuits themselves are `tikzpictures` again, and nesting those is dangerous (but may work). Instead they could have been put into **\savebox**es and just used.

Finally, using nested `tikzpictures` for the outer nodes is not really necessary. Not using nodes but putting the two `yquant*` environments in a TikZ scope with `shift` transformation would have also worked.
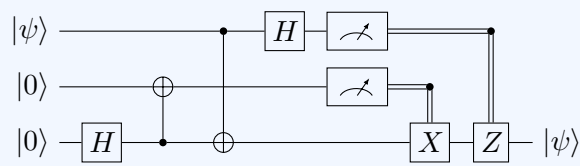
```
\begin{tikzpicture}
   \begin{yquant}
      qubit {$\ket\psi$} a;
      qubit {$\ket0$} b[2];

      h b[1];
      cnot b[0] | b[1];
      cnot b[1] | a;
      h a;
      align a, b;
      measure a;
      measure b[0];

      x b[1] | b[0];
      z b[1] | a;

      discard a;
      discard b[0];
      output {$\ket\psi$} b[1];
   \end{yquant}
\end{tikzpicture}
```
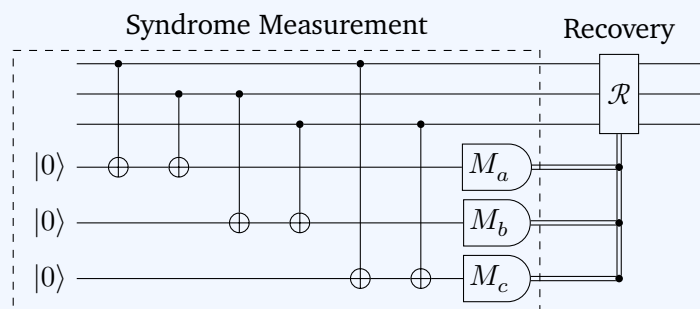
```
% \usetikzlibrary{fit, quotes}
\begin{tikzpicture}
  \begin{yquant}
     qubit {} msg[3];
     [name=inits]
     qubit {$\ket0$} syndrome[3];

     [name=scnot0]
     cnot syndrome[0] | msg[0];
     cnot syndrome[0] | msg[1];
     cnot syndrome[1] | msg[1];
     cnot syndrome[1] | msg[2];
     cnot syndrome[2] | msg[0];
     cnot syndrome[2] | msg[2];
     [name=smeas]
     dmeter {$M_{\symbol{\numexpr`a+\idx}}$} syndrome;
     ["Recovery"]
     box {$\symcal R$} (msg) | syndrome;
     discard syndrome;
  \end{yquant}
  \node[draw, dashed, fit=(inits-2) (scnot0-p0) (smeas-2), "Syndrome
  ↪  Measurement"] {};
\end{tikzpicture}
```

In this case, an implicit register declaration would not have worked: we would
have needed to define the first part of the syndrome register *before* the sec-
ond part of the message register. But this would then have mixed data with
syndrome registers in the vertical ordering.

This also is a first demonstration of how to access yquant objects from within
Ti*k*Z. We name several elements that visually form the enclosing rectangle;
then, we use the Ti*k*Z library `fit` to put a node around them all.

Then we see how to apply an operation to multiple registers in parallel while
using the **\idx** macro to still give them a different text. Since **\idx** gives a nu-
merical index (zero-based), we exploit the ASCII code (actually, this document
is compiled in Unicode mode...) to turn this into a letter.

The example also demonstrates how to put a description next to a gate. In
general, those descriptions should be realized using the Ti*k*Z feature `label`.
Using the Ti*k*Z library `quotes`, the label is most easily specified. Since the label
is not part of the valid arguments and also cannot be found in the /yquant
path, it is automatically passed to `/yquant/operator style`.

6.2.2 IV. Simple Quantum Circuits

```
\begin{tikzpicture}
    \begin{yquant*}
        x q;
    \end{yquant*}
\end{tikzpicture}
```

$-\boxed{X}-$

A. Wires and gates
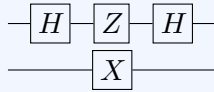
```
\begin{tikzpicture}
    \begin{yquant*}
        h a;
        align a, b;
        z a;
        x b;
        h a;
    \end{yquant*}
\end{tikzpicture}
```

$-\boxed{H}-\boxed{Z}-\boxed{H}-$
$\quad\;\boxed{X}$
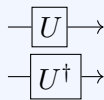
```
\begin{tikzpicture}
    \begin{yquant*}
        box {$U$} a;
        box {$U^\dagger$} b;
        setstyle {->} -;
    \end{yquant*}
\end{tikzpicture}
```

$-\boxed{U}\rightarrow$
$-\boxed{U^\dagger}\rightarrow$

Since version 0.1.2, `yquant` allows to change wire styles by means of the `setstyle` and `addstyle` pseudo-gates. Here, we use the gate on all wires in order to set an arrow style. Note that arrowheads are actually very special in two respects:

- `yquant` draws continuous wires for as long as possible. In this example, the wire path extends from the very left to the end of the circuit; since version 0.1.2, `yquant` does not draw a wire *to* the gate and then a separate one *from* the gate to the next or the end. The only way to force `yquant` to draw multiple wires is to change the wire style or type mid-circuit. For example, by saying `addstyle {} -;`, all wire paths will be separated

52

at the current position, which *in theory* allows to draw arrowsheads on intermediate wires.

- *In practice,* this will not work due to the clipping commands that `yquant` installs. Every wire extends from the center of the left to the center of the right gate, and the gate's shape acts as a clipping path. Consequently, though the arrowhead is drawn, it is actually drawn at the center of the gate instead of the `west` anchor and then clipped away (unless the gate is small, in which case you might still see some fragments of the arrowhead).

Thus, it is currently not possible to use arrowheads on intermediate wires. If you really need to do this (say, for only a single gate), you may experiment with the TikZ `shorten` keys, which allow you to manually reduce the length of the wire, but the amount of reduction must be hand-computed for every gate. If you need this more often, consider filing a feature request.

B. CNOT and other controlled single qubits gates
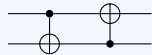
```
\begin{tikzpicture}
  \begin{yquant*}
    cnot a[1] | a[0];
    cnot a[0] | a[1];
  \end{yquant*}
\end{tikzpicture}
```
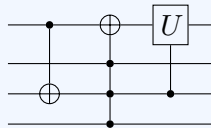
```
\begin{tikzpicture}
  \begin{yquant*}
    cnot q[2] | q[0];
    cnot q[0] | q[1-3];
    box {$U$} q[0] | q[2];
  \end{yquant*}
\end{tikzpicture}
```

C. Vertical wires
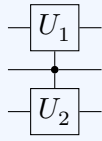
```
\begin{tikzpicture}
    \begin{yquant*}
        box {$U_{\protect\the\numexpr\idx+1}$} q[0, 2] | q[1];
    \end{yquant*}
\end{tikzpicture}
```

There is no direct support for this construction, but as with the initialization of a vector registers, `yquant` allows to access the macro `\idx` within an operator value. This macro follows the same rules as the name suffix, i.e., it assigns indices (zero-based) to the target registers in top-to-bottom order, regardless of which order was specified in the target list. Since we instead want a one-based subscript, we need to add one. Note that if you want to output `\idx` directly or within an unexpandable expression, you don't need to take any action. However, here, `\the` is expandable; and since `yquant` needs to process all its output twice (first in order to determine the vertical spacing, second to actually typeset), you must manually take care that the command is *not* expanded prematurely by inserting `\protect`. Had you not done this, the subscript would have been "1" for both operators. Note this is not the case if this macro is used upon creation of a register (as is evident by the fact that the previous examples that used `\ifcase` within the value did not need to say `\protect\ifcase` ... `\protect\or` ... `\protect\fi`). Probably we can avoid the need for protection in a future release….

D. Labeling input and output states
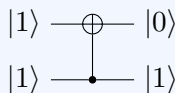
```
\begin{tikzpicture}
    \begin{yquant*}
        qubit {$\ket1$} q[2];
        cnot q[0] | q[1];
        output {$\ket\idx$} q;
    \end{yquant*}
\end{tikzpicture}
```

$|0^k\rangle$ ───

$|\psi\rangle\,\{$ ───

$|\psi\rangle\ \begin{matrix} A \\ B \end{matrix}$ ─── ───

```
\begin{tikzpicture}
  \begin{yquant*}
    {
      \yquantset{every multi label/.append
      ↪  style={decorate=false, draw=none}}
      init {$\ket{0^k}$} (a[-1]);
    }
    init {$\ket\psi$} (b[-1]);
    qubit {\ifcase\idx\relax$A$\or$B$\fi} c[2];
    [every multi label/.append style={decorate=false,
    ↪  draw=none, every node/.append style={shift={(-.3,
    ↪  0)}}}]
    init {$\ket\psi$} (c);
  \end{yquant*}
\end{tikzpicture}
```

Here, three different styles for the initialization of multi-qubit labels are used. The second one (using a curly brace) corresponds to the default. It is overwritten for the first qubit, and to make this modification local, this is done in a group. The third qubit pair uses an overall label and additionally individual labels on the lines. This is achieved by some trickery: the individual labels are given as initialization labels on the register; the global label is given as an `init` multi-qubit gate.

6.2.3 V. More Complicated Circuits: Multiple Qubit gates and Beyond

A. Multiple qubit gates
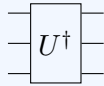
```
\begin{tikzpicture}
  \begin{yquant*}
    box {$U^\dagger$} (a[-2]);
  \end{yquant*}
\end{tikzpicture}
```
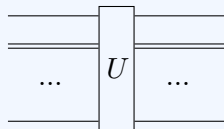
```
\begin{tikzpicture}
  \begin{yquant}
    qubit {} a;
    cbit {} b;
    nobit ellipsis;
    qubit {} c;

    [draw=none]
    box {$\dots$} ellipsis;
    box {$U$} (a, b, ellipsis, c);
    [draw=none]
    box {$\dots$} ellipsis;
  \end{yquant}
\end{tikzpicture}
```

This demonstrates how a register of type `nobit` might even be useful if the register is never used. We use `box` registers with disabled border to put the ellipsis dots in place.
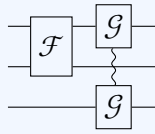
```
\begin{tikzpicture}
    \begin{yquant*}
        box {$\symcal F$} (a[-1]);
        box {$\symcal G$} (a[0, 2]);
    \end{yquant*}
\end{tikzpicture}
```

This demonstrates yquant's capabilities of discontiguous multi-qubit gates that was added in version 0.1.2. yquant automatically splits multi-qubit gates into slices of directly adjacent wires (which, for the $\mathcal{G}$ case, are the single wires a[0] and as second slice a[2]).

B. Measurements and classical bits
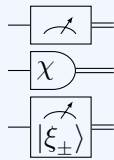

```
\begin{tikzpicture}
    \begin{yquant*}
        measure a;
        dmeter {$\chi$} b;
        measure {$\ket{\xi_\pm}$} c;
    \end{yquant*}
\end{tikzpicture}
```

The "tab" and "measure" type are not supported yet. Extracting a meter symbol on its own will not be supported. If you are interested in the code, have a look at yquant-shapes.tex and search for the yquant-measure shape.
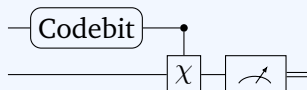

```
\begin{tikzpicture}
    \begin{yquant*}
        [rounded corners]
        box {Codebit} a;
        box {$\chi$} b | a;
        discard a;
        measure b;
    \end{yquant*}
\end{tikzpicture}
```

Rectangles with rounded corners are not a specific style, but since the yquant-rectangle shape internally uses \pgfpathrectangle, the ordinary TikZ option can be used (also, an inset specification can control how much the corner is rounded).

57

$|\psi\rangle - \boxed{|\widehat{\xi_\pm}\rangle} =$
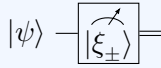
```
\begin{tikzpicture}
    \begin{yquant}
        qubit {$\ket\psi$} q;

        measure {$\ket{\xi_\pm}$} q;
    \end{yquant}
\end{tikzpicture}
```

$\boxed{\text{Bell}}$

```
\begin{tikzpicture}
    \begin{yquant*}
        dmeter {Bell} (a[0, 1]);
        discard a;
    \end{yquant*}
\end{tikzpicture}
```

$\boxed{|\widehat{\xi_\mp}\rangle}$
$\boxed{|\xi_\mp\rangle}$
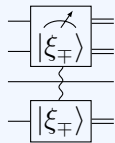
```
\begin{tikzpicture}
    \begin{yquant*}
        measure {$\ket{\xi_\mp}$} (a[-1, 3]);
    \end{yquant*}
\end{tikzpicture}
```

Multi-qubit gates (including measurements) on non-adjacent registers are properly supported since version 0.1.2. As explained in section 2.4, there is one main and multiple subordinate gate in such a discontiguous multi-qubit operation (though at the moment, the `measure` gate with text is the only gate that makes this distinction). In our case, the main part contains the measurement symbol and the text, while the subordinate gates only contain the text. By default, `yquant` uses the first slice as main part, but you may influence this by preceding what you want to be "main" by a star:

$\boxed{|\xi_\mp\rangle}$
$\boxed{|\widehat{\xi_\mp}\rangle}$

```
\begin{tikzpicture}
    \begin{yquant*}
        measure {$\ket{\xi_\mp}$} (a[-1, *3]);
    \end{yquant*}
\end{tikzpicture}
```

58
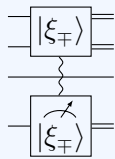
C. Non-gate inserts, forcing space, and swap

Defective Circuit
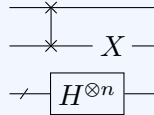

```
\begin{tikzpicture}
  \begin{yquant*}
    [name=sw]
    swap (a[0-1]);
    [draw=none]
    box {$X$} a[1];
    slash b;
    box {$H^{\otimes n}$} b;
    \node[anchor=199] at (sw-0.north) {Defective
    ↪ Circuit};
  \end{yquant*}
\end{tikzpicture}
```

Here, the intermediate text was inserted by using a `box` without drawing.
Another way would be to use an `init` command, although this is semantically
wrong (probably).
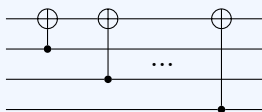

```
\begin{tikzpicture}
  \begin{yquant*}
    qubit {} a;
    [name=ypos]
    qubit {} b[3];

    cnot a | b[0];
    [name=left]
    cnot a | b[1];
    hspace {7mm} -;
    [name=right]
    cnot a | b[2];
  \end{yquant*}
  \path (left |- ypos-0) -- (right |- ypos-1)
  ↪ node[midway] {$\dots$};
\end{tikzpicture}
```

Note how the register range – was used to denote all registers. We positioned
the dots by first naming the relevant registers, so that the vertical position is at
the coordinates ypos-0 and ypos-1; and then, we also named the `cnot` gates,
so that we are able to discern the horizontal position.

D. Barriers
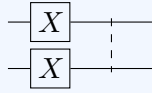
```
                    \begin{tikzpicture}
                      \begin{yquant*}
  ┌─┐                  x a[0, 1];
──┤X├──  ┆  ───         barrier (a);
  └─┘                 \end{yquant*}
  ┌─┐              \end{tikzpicture}
──┤X├──  ┆  ───
  └─┘
```

Now the `qcircuit` manual lists three circuits with barriers at different positions. They cannot be drawn with `yquant`; however, since neither of them is a valid circuit, this is of no concern.

E. How to control anything
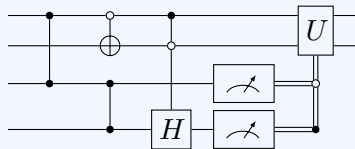
```
                    \begin{tikzpicture}
                      \begin{yquant*}
                        zz (a[0, 2]);
                        cnot a[1] ~ a[0];
                        zz (a[2, 3]);
                        h a[3] | a[0] ~ a[1];
                        measure a[2, 3];
                        box {$U$} (a[0, 1]) | a[3] ~ a[2];
                        discard a[2, 3];
                      \end{yquant*}
                    \end{tikzpicture}
```

Note that it is not possible to draw a control to measurement (the measurement operations are explicitly defined not to accept controls): Either the measurement is performed or not (which transforms the register type), but a measurement conditioned on a quantum state is not possible. In principle, one could think of a measurement conditioned on a classical register (in which case the register type cannot change, as maybe the state stays quantum; the measurement operation then is similar to a complete dephasing). If there is need for this, please file a feature request. But note the relatively common (though unsupported by `yquant` at the moment) use of control lines that directly go from the measurement operator to the controlled operation; in this case, the opposite direction would be meant by using the same notation.

6.2.4 VI. Bells and Whistles: Tweaking Your Diagram to Perfection

For options how to configure the circuits, refer to section 3.

A. Spacing
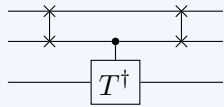
```
\begin{tikzpicture}
    \begin{yquant*}
        swap (a[0, 1]);
        box {$T^\dagger$} a[2] | a[1];
        swap (a[0, 1]);
    \end{yquant*}
\end{tikzpicture}
```

B. Labeling
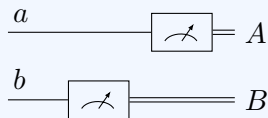
```
\begin{tikzpicture}
    \begin{yquant}[every initial
    ↪ label/.style={anchor=south east}]
        qubit {\rlap{\hskip2mm $a$}} a;
        qubit {\rlap{\hskip2mm $b$}} b;
        hspace {5mm} -;

        measure b;
        align -;
        measure a;
        output {$A$} a;
        output {$B$} b;
    \end{yquant}
\end{tikzpicture}
```

Measurement with vertical output are not supported (yet). Repositioning the initial labels needs some care and manual fine-tuning.

C. Grouping
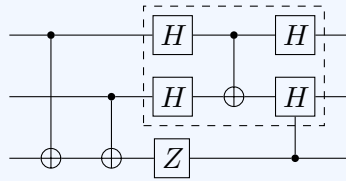

```
% \usetikzlibrary{fit}
\begin{tikzpicture}
    \begin{yquant*}[register/separation=3mm]
        cnot a[2] | a[0];
        cnot a[2] | a[1];
        [name=left]
        h a[0, 1];
        z a[2];
        cnot a[1] | a[0];
        [name=righttop]
        h a[0];
        [name=rightbot]
        h a[1] | a[2];

        hspace {2mm} -;
    \end{yquant*}
    \node[draw, dashed, fit=(left-0) (left-1) (righttop) (rightbot-0)] {};
\end{tikzpicture}
```

Note that **\begin**{yquant*} must not be followed by a line break (unless masked by %) if options follow.

6.3 `quantikz` documentation

Again, our section headings will be the same as in the `quantikz` manual. And since `quantikz` also has even more space between the gates, we globally say `\yquantset{operator/separation=4mm}`.

6.3.1 II. A single wire
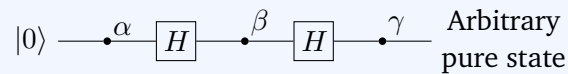

```
\begin{tikzpicture}[label position=north east, every label/.style={inner
↪   sep=1pt}]
   \begin{yquant}
      qubit {$\ket0$} a;

      phase {$\alpha$} a;
      h a;
      phase {$\beta$} a;
      h a;
      phase {$\gamma$} a;

      [every output/.append style={align=center}]
      output {Arbitrary\\pure state} a;
   \end{yquant}
\end{tikzpicture}
```

The captions of `phase` commands are internally implemented using TikZ `labels`. At the moment, it is not possible to change any label options on a gate-type basis, only locally or fully globally (TikZ feature request #811).

63
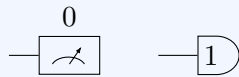
A. Measurements


```
% \usetikzlibrary{quotes}
\begin{tikzpicture}
  \begin{yquant*}
    ["$0$"]
    measure a;
    discard a;

    init {} a;
    dmeter {$1$} a;
    discard a;
  \end{yquant*}
\end{tikzpicture}
```

Other measurement shapes are not supported at the moment.

B. Wires and arrows
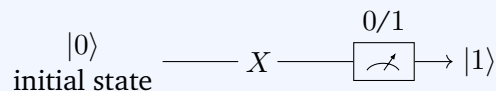

```
% \usetikzlibrary{quotes}
\begin{tikzpicture}
  \begin{yquant}[operator/separation=1cm, every label/.append
  ↪  style={align=center}]
    qubit {$\ket0$\\initial state} a;

    [draw=none]
    box {$X$} a;

    ["$0$/$1$", type=qubit]
    measure a;

    addstyle {->} a;
    output {$\ket1$} a;
    \yquantset{operator/separation=5mm}
  \end{yquant}
\end{tikzpicture}
```

This example demonstrates how to instruct the `measure` gate to use a different output type than the standard `cbit`.

In general, any macros that are used within a TikZ path or a yquant operation must not be fragile, or must be preceded with **\protect**. In this example, **** is a

robust command (at least in newer kernels), so protection is not required. Since it may occur quite frequently that yquant is used within a center environment or in **\centering** mode (in which **** is still fragile), yquant takes care of this (it actually robustifies **\@centercr**, which is the meaning of **** in these surroundings).

In order to change the style of an individual wire, we use addstyle as of version 0.1.2. To make the final line smaller, we change the operator separation by issuing **\yquantset** at the end.

$|0\rangle^{\otimes n} \ \overset{3}{\diagup} \ \boxed{H}$
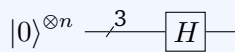
```
% \usetikzlibrary{quotes}
\begin{tikzpicture}
    \begin{yquant*}
        qubit {$\ket0^{\otimes n}$} a;
        ["north east:3" {font=\protect\footnotesize,
        ↪   inner sep=0pt}]
        slash a;
        hspace {2mm} a;
        h a;
    \end{yquant*}
\end{tikzpicture}
```

Again, you see an example of how some commands need to be **\protect**ed when used in yquant options, and that you can indeed exploit all features of the quotes library.

$|0\rangle^{\otimes n} \ \boxed{H}$
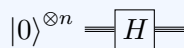
```
\begin{tikzpicture}
    \begin{yquant}
        qubits {$\ket0^{\otimes n}$} a;
        h a;
    \end{yquant}
\end{tikzpicture}
```

6.3.2 III. Multiple Qubits
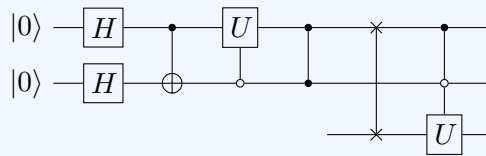

```
\begin{tikzpicture}
   \begin{yquant}
      qubit {$\ket0$} a;
      qubit {$\ket0$} b;

      h a, b;
      cnot b | a;
      box {$U$} a ~ b;
      zz (a, b);

      [after=a]
      qubit {} c;

      swap (a, c);
      box {$U$} c | a ~ b;
   \end{yquant}
\end{tikzpicture}
```

This example demonstrates the use of the `after` argument that instructs the register creation to begin the register only after the current position of another register that already exists.
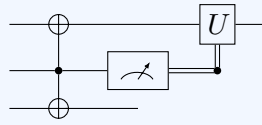
```
\begin{tikzpicture}
   \begin{yquant*}
      [name=c]
      cnot a[0, 2] | a[1];
      [name=m]
      measure a[1];
      discard a[2];
      \path[/yquant/every wire, /yquant/every qubit wire] (c-1) --
      ↪   (m.center |- c-1);
      box {$U$} a[0] | a[1];
      discard a[1];
   \end{yquant*}
\end{tikzpicture}
```

It is not possible for the double control line to directly exit the measurement gate. Also, if you discard a gate, this will prevent it from exiting from its last gate. yquant will not allow you (apart from manual drawing) to extend the wire to some arbitrary position, then drop it. But of course, as done here, you can always resort to the full power of TikZ. Still, this is inferior to a wire drawn by yquant, as it does not use clippings: the connection with the cnot gate may not be accurate; in particular, if the wire is of a different color or if you need to draw classical or bundle wires, the connection will become unpleasant.
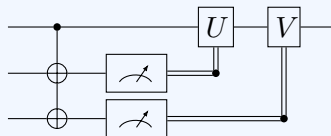

```
\begin{tikzpicture}
   \begin{yquant*}
      cnot a[1, 2] | a[0];
      measure a[1], a[2];
      box {$U$} a[0] | a[1];
      box {$V$} a[0] | a[2];
      discard a[1]-;
   \end{yquant*}
\end{tikzpicture}
```

6.3.3 IV. Operating on many Qubits
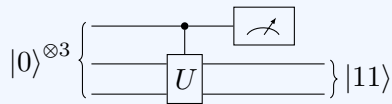

```
\begin{tikzpicture}
   \begin{yquant*}
      init {$\ket0^{\otimes3}$} (a[-2]);

      box {$U$} (a[1-2]) | a[0];
      measure a[0];
      discard a[0];
      output {$\ket{11}$} (a[1-2]);
   \end{yquant*}
\end{tikzpicture}
```

Multi-qubits inputs are possible using the `init` command. The text assigned to a register declaration is always for an individual register.
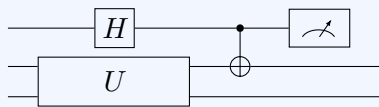

```
\begin{tikzpicture}
   \begin{yquant*}
      hspace {7.5mm} a;
      h a;
      hspace {7.5mm} a;
      [x radius=1cm]
      box {$U$} (b, c);
      cnot b | a;
      measure a;
      discard a;
   \end{yquant*}
\end{tikzpicture}
```

`yquant` does not use a grid layout: operators are stacked next to each other. Therefore, there is no automatic centering of a column, though it could be emulated using hand-crafted `hspace` commands, as was done here (the Hadamard gate uses the `/yquant/operator/minimum width`, which is 5mm, while the large box has a width of 2cm, so that we need two 7.5mm spacings at the end, as the `hspace` pseudo-gate only inserts exactly the space you give, but not additional [twice] `/yquant/operator/separation`, as would be the case for a hypothetical zero-width gate). In fact, we don't even need the second `hspace`, since the two-qubit `cnot` will automatically enforce correct alignment.
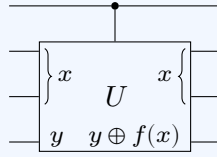
```
\begin{tikzpicture}[braced/.style={decoration={brace, #1, pre=moveto, pre
↪  length=-1pt, post=moveto, post length=-1mm}, decorate},
↪  inner/.style={font=\footnotesize}]
  \begin{yquant}[register/separation=3mm]
    [name=a]
    qubit {} a[4];
    [x radius=1cm, name=u]
    box {$U$} (a[1-3]) | a[0];
  \end{yquant}
  \draw[braced]
    ([xshift=2pt] a-1 -| u.west) -- ([xshift=2pt] a-2 -| u.west)
    node[inner, anchor=west, xshift=1pt, pos=.55] {$x$};
  \draw[braced=mirror]
    ([xshift=-2pt] a-1 -| u.east) -- ([xshift=-2pt] a-2 -| u.east)
    node[inner, anchor=east, xshift=-1pt, pos=.55] {$x$};
  \node[inner, anchor=west] at (a-3 -| u.west) {\smash{$y$}};
  \node[inner, anchor=east] at (a-3 -| u.east) {\smash{$y \oplus f(x)$}};
\end{tikzpicture}
```

There is no simple way to draw *within* a gate, unless this gate is a proper quantum circuit itself. Instead, here the intricate parts were reproduced using TikZ: first, we make sure we assign a name to every relevant coordinate. Then we use some TikZ styles to draw the braces and nodes at the intersection of these coordinates. Here, we also make use of the moveto decoration transformation that comes with yquant and that allows to enlarge the braces slightly for a good overall appearance. Finally, as y has a much smaller height than $y \oplus f(x)$, we make sure this does not affect the vertical positioning; and also, as x has no ascender, we need to slightly position it off-mid for a good look.
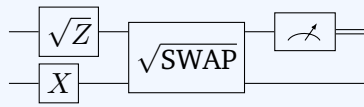
```latex
\begin{tikzpicture}
    \begin{yquant*}
        box {$\sqrt Z$} a;
        box {$X$} b;
        box {$\sqrt{\mathrm{SWAP}}$} (a, b);
        measure a;
    \end{yquant*}
\end{tikzpicture}
```
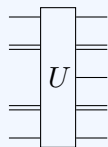
This time, we did not artificially discard the lines.
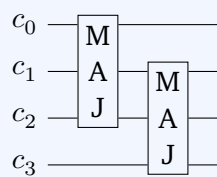
```latex
\begin{tikzpicture}
    \begin{yquant}
        qubit {$c_{\idx}$} c[4];
        box {M\\A\\J} (c[-2]);
        box {M\\A\\J} (c[1-]);
    \end{yquant}
\end{tikzpicture}
```
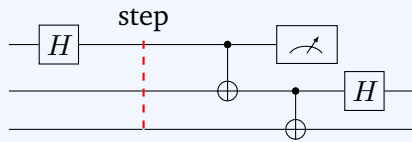
A. Different connections

```latex
\begin{tikzpicture}
    \begin{yquant}[register/default name=]
        qubit a;
        cbit b;
        nobit c;
        cbit d;
        qubit e;
        box {$U$} (-);
        settype {qubit} c;
    \end{yquant}
\end{tikzpicture}
```

This example for the first time demonstrates the declaration of a non-existing register and the `settype` pseudo-gate that acts as a zero-width, no-content `init` gate.

6.3.4 V. Slicing


```
\begin{tikzpicture}
    \begin{yquant}
        qubit {} a[3];
        h a[0];
        [red, thick, label=step]
        barrier (a);
        cnot a[1] | a[0];
        measure a[0];
        discard a[0];
        cnot a[2] | a[1];
        h a[1];
    \end{yquant}
\end{tikzpicture}
```

There is nothing like a `slice all` keyword, as yquant's underlying layout is not grid-based. Changing the style of slice captions simply means providing label options. This time, we used the `label` key instead of the shorter syntax provided by the `quotes` library, which is of course also possible.
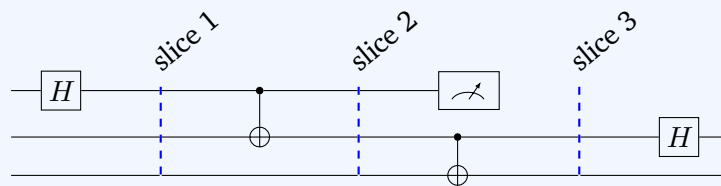
```
% \usetikzlibrary{quotes}
\begin{tikzpicture}[every label/.style={rotate=40, anchor=south west}]
   \begin{yquant}[operators/every barrier/.append style={blue, thick}]
      qubit {} a[3];
      h a[0];
      ["slice 1"]
      barrier (-);
      cnot a[1] | a[0];
      ["slice 2"]
      barrier (-);
      measure a[0];
      discard a[0];
      cnot a[2] | a[1];
      ["slice 3"]
      barrier (-);
      h a[1];
   \end{yquant}
\end{tikzpicture}
```
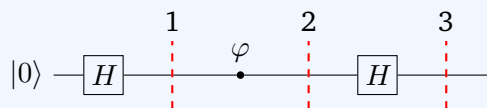


```
% \usetikzlibrary{quotes}
\begin{tikzpicture}[label distance=4mm]
   \begin{yquant}[operators/every barrier/.append style={red, thick,
   ↪  shorten <= -2.5mm, shorten >= -2.5mm}]
      qubit {$\ket0$} a;
      h a;
      ["1"]
      barrier (a);
      phase {[label distance=0pt]$\varphi$} a;
      ["2"]
      barrier (a);
      h a;
      ["3"]
      barrier (a);
      output {$\cos\frac\varphi2 \ket0 - i\sin\frac\varphi2 \ket1$};
   \end{yquant}
\end{tikzpicture}
```

Usually, the shorten keys do not have any effect on yquant operations, since the latter are all made up of nodes. However, the yquant-line shape explicitly

takes care of correctly handling them. It is the only one that does so. Since barriers usually end quite closely to the wires—and the default `dashed` style may make this worse—the `shortening` may often prove useful. Note that if the barriers are enlarged by means of negative `shortenings`, this will not affect the bounding box or internal register height calculations, and you must take care of appropriately shifting labels. Also note that we used much larger magnitudes in order to achieve a similar appearance as in `quantikz`. To avoid that the large distance also affects the `phase` gate badly, we locally reset the distance; for this, there are two ways. The easiest one is to make use of the fact that the value of the `phase` gate is passed directly as `label` argument, so that we can locally reset the distance. The other possibility would be to write

```
{
    \yquantset{/tikz/label distance=0pt}
    phase {$\varphi$} a;
}
```

since due to the aforementioned lack of support for a style that sets the options in TikZ, we must manually use a (grouped) `\yquantset` instruction for this. Note that whenever you change a TikZ style in a `yquant` environment, use the `\yquantset` macro, *not* `\tikzset` or `\pgfkeys`. Not only will the latter two not automatically restart the parser (so that you would have to issue `\yquant` after their use), but `yquant` has to process all its content twice in order to properly determine the register height. Only `\yquantset` will be properly captured and re-issued at the correct position when the content is actually typeset. Had we written `\tikzset{label distance=0pt}` `\yquant`, no effect at all would have been visible, since this command would only have taken effect in the first (invisible) round when `yquant` determines heights.

`yquant` does not provide a mechanism for vertical labels, but you may of course just insert line breaks at appropriate positions (and set the `align` property of the labels).

6.3.5 VI. Spacing

A. Local adjustment
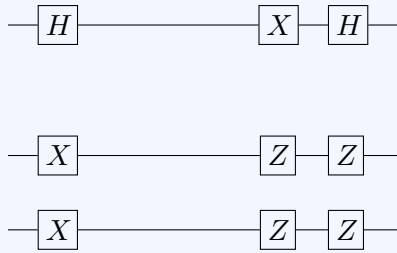

```
\begin{tikzpicture}
  \begin{yquant}[register/default
  ↪  name=]
    [register/minimum height=2cm]
    qubit a;
    qubit {\vbox to 1cm{}} b;
    qubit c;

    h a;
    x b-;
    hspace {2cm} -;
    x a;
    z b-;
    h a;
    z b-;
  \end{yquant}
\end{tikzpicture}
```

At the moment, the distance between registers is calculated by yquant auto-
matically. We show various possibilities to intervene in this example. The first
is to locally, upon creation of the register, reset /yquant/register/minimum
height to a different value. The second is to artificially enlarge the label that
the qubit initializer takes. However, both approaches enlarge the *height* of the
registers, i.e., add half of the specified amount to the top and the bottom.


```
\begin{tikzpicture}
  \begin{yquant*}
    [x radius=1cm]
    x a;
    box {\hbox to 1cm{\hfil$X$\hfil}} a;
    hspace {1cm} a;
    x a;
    discard a;
  \end{yquant*}
\end{tikzpicture}
```

Here, we demonstrate two possibilities to enlarge a box: The first is by specify-
ing its size in terms of the x radius or y radius keys beforehand. Those values

74

serve as minimum sizes and would be extended if the text extended beyond the box. The second option is to just enlarge the text artificially by explicitly putting it into a fixed-width box. Note that in the first case, the *radius* is specified, i.e., the half-width, while in the second case, it is the *total* width (both times modulo the inner separation). Also note that the `/yquant/operator/minimum width` style is unsuitable for the given task: it would not change the visual width, only what `yquant` assumes its width to be.

B. Global Adjustment
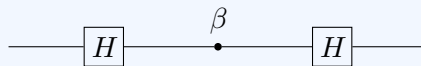

```
\begin{tikzpicture}
   \begin{yquant*}[operator/separation=1cm]
      h a;
      phase {$\beta$} a;
      h a;
   \end{yquant*}
\end{tikzpicture}
```
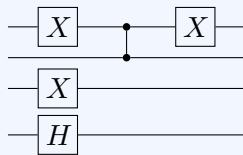


```
\begin{tikzpicture}
   \begin{yquant*}[register/minimum height=0pt]
      x a[0, 2];
      zz (a[0, 1]);
      x a[0];
      h b;
   \end{yquant*}
\end{tikzpicture}
```

By default, `yquant` will use the height that is required by the individual gates, but at least `/yquant/register/minimum height` (which defaults to 3mm). Only manually reducing the default height will produce the cramped spacing displayed here.

C. Alignment
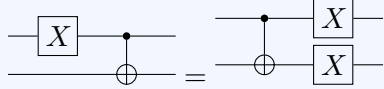
```
\begin{tikzpicture}
    \begin{yquant*}
        x a[0];
        cnot a[1] | a[0];
    \end{yquant*}
\end{tikzpicture}
$=$
\begin{tikzpicture}
    \begin{yquant*}
        cnot a[1] | a[0];
        x a;
    \end{yquant*}
\end{tikzpicture}
```

Not specifying anything for the vertical alignment will lead to the common Ti*k*Z problem: the baseline will be at the bottom, which is particularly bad in this case due to the missing X gate. The `/yquant/register/minimum height` key does not help here, since it only affects yquant's internal handling, but not the bounding box calculated by Ti*k*Z. In the first example of the `qcircuit` documentation, we demonstrated how the desired task can easily be achieved in terms of `baselines`. We will now do the same with scopes instead.
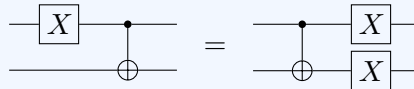
```
\begin{tikzpicture}[/yquant/register/minimum height=5mm]
    \begin{yquant*}
        x a[0];
        cnot a[1] | a[0];
    \end{yquant*}
    \path (current bounding box.east |- 0, 0) ++(1, 0) coordinate (shift);
    \begin{scope}[shift=(shift)]
        \begin{yquant*}
            cnot a[1] | a[0];
            x a;
        \end{yquant*}
    \end{scope}
    \node at (current bounding box) {$=$};
\end{tikzpicture}
```

Here, we increased the minimum height so that in the left circuit despite the absence of the X gate, the second register has the same separation. We used

the meta-node `current bounding box` to avoid the need to manually specify hard-coded positions.

1. Perfecting Vertical Alignment

$$|0\rangle \quad \bullet$$
$$\boxed{H} \quad \oplus \quad \equiv \quad \boxed{H} \quad \boxed{X}$$
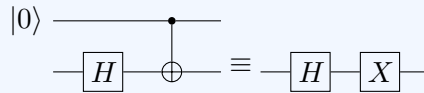
```
\begin{tikzpicture}[baseline=(W)]
    \begin{yquant}
        qubit {$\ket0$} anc;
        [name=W]
        qubit {} x;
        h x;
        cnot x | anc;
    \end{yquant}
\end{tikzpicture} $\equiv$ \begin{tikzpicture}[baseline=(W)]
    \begin{yquant}
        [name=W]
        qubit {} x;
        h x;
        x x;
    \end{yquant}
\end{tikzpicture}
```

While here, we opted for the most logical choice to name the declaration of the register, a name put to any operation on the desired register would also serve the purpose (as long as this operation is vertically symmetrical, which subcircuits may not be).
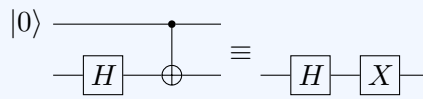
```
\begin{tikzpicture}[baseline={([yshift=.2cm]W)}]
    \begin{yquant}
        qubit {$\ket0$} anc;
        [name=W]
        qubit {} x;
        h x;
        cnot x | anc;
    \end{yquant}
\end{tikzpicture} $\equiv$
\begin{tikzpicture}[baseline={([yshift=.2cm]new)}]
    \begin{yquant}
        [name=new]
        qubit {} x;
        h x;
        x x;
    \end{yquant}
\end{tikzpicture}
```

Of course, you may also use the features of the TikZ library `calc` to achieve the same shift.

D. Scaling
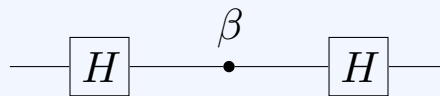

```
\begin{tikzpicture}[scale=1.5, every label/.append style={scale=1.5}]
    \begin{yquant*}
        h a;
        phase {$\beta$} a;
        h a;
    \end{yquant*}
\end{tikzpicture}
```

Here, we first scaled the circuit itself. The default style for `/yquant/every circuit` sets the `transform shape` key for every node (which means any gate), so that those are also scaled. This is explicitly undone for `labels` due to TikZ bug #843, so the scaling must be specified explicitly for the `labels`.

6.3.6 VII. Typesetting

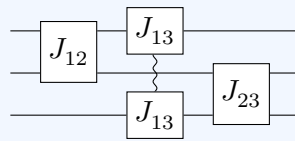**A. Global Styling**
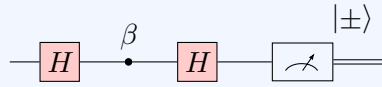

```
% \usetikzlibrary{quotes}
\begin{tikzpicture}
   \begin{yquant*}[operators/every h/.append style={fill=red!20}]
      h a;
      phase {$\beta$} a;
      h a;
      ["$\ket\pm$" above right]
      measure a;
   \end{yquant*}
\end{tikzpicture}
```

Instead of setting `/yquant/operators/every h`, we could also have changed `/yquant/operators/every box`. Had we used `/yquant/every operator`, then the measurement would also have changed. Again, due to a Ti*k*Z limitation, it is not possible to change the position of labels on a per-style basis, only by using `label` options or a global setting.
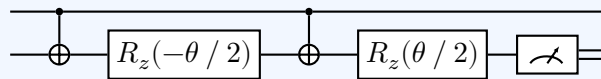

```
\begin{tikzpicture}[thick]
   \begin{yquant*}[every operator/.prefix style={fill=white}]
      cnot a[1] | a[0];
      box {$R_z(-\theta\fracslash2)$} a[1];
      cnot a[1] | a[0];
      box {$R_z(\theta\fracslash2)$} a[1];
      measure a[1];
   \end{yquant*}
\end{tikzpicture}
```

As the "thin" style is the default, we present the opposite. By default, all operators are transparent; we changed this by giving all of them a white background color (but as a style *prefix*, so that, e.g., black fillings overwrite this). Contrary to `quantikz`, this also fills the `cnot`s. If you only want to fill certain operators, you have to selectively target them using their styles.

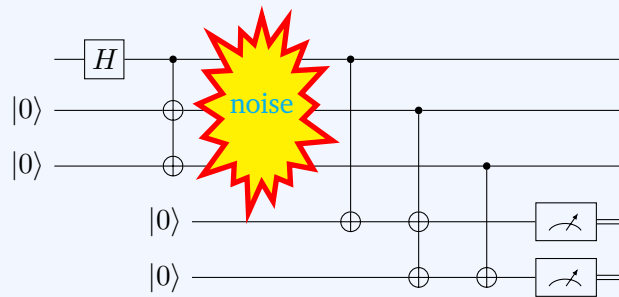79

```
\begin{tikzpicture}
    \begin{yquant}[operators/every box/.append style={fill=white}]
        qubit {} j[3];
        box {$J_{12}$} (-j[1]);
        box {$J_{13}$} (j[0, 2]);
        box {$J_{23}$} (j[1]-);
    \end{yquant}
\end{tikzpicture}
```

yquant properly splits discontiguous multi-qubit operations as of version 0.1.2.

B. Per-Gate Styling
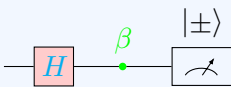

```
% \usetikzlibrary{quotes}
\begin{tikzpicture}
    \begin{yquant*}
        [fill=red!20, font=\color{cyan}]
        h a;
        [green]
        phase {[green]$\beta$} a;
        ["$\ket\pm$"]
        measure a;
        discard a;
    \end{yquant*}
\end{tikzpicture}
```

```
% \usetikzlibrary{shapes.symbols, fit}
\begin{tikzpicture}
   \begin{yquant}
      qubit {} data;
      qubit {$\ket0$} anc1[2];

      h data;
      cnot anc1 | data;
      [after=data]
      qubit {$\ket0$} anc2[2];
      [name=box, draw=none]
      box {\phantom{noise}} (data, anc1);
      cnot anc2[0] | data;
      cnot anc2 | anc1[0];
      cnot anc2[1] | anc1[1];
      measure anc2;
   \end{yquant}
   \node[starburst, cyan, fill=yellow, draw=red,
        line width=2pt, inner xsep=-4pt, inner ysep=-5pt, fit=(box)]
        ↪ {noise};
\end{tikzpicture}
```

TikZ shapes cannot simply be used with yquant. Any yquant shape must be
aware of the keys x radius and y radius that control its width and height.
Additionally, yquant shapes must implement clipping paths. Those objects,
which are a yquant addition to TikZ allow yquant to properly clip wires and
vertical lines to the shape of the gate. yquant draws its elements sequentially;
hence, a wire that comes into an operator will be hidden by anything the
operator draws on top of it; but outgoing wires will in turn draw on the
operator (modulo clipping). To avoid the issues, we construct an invisible box
operator and name it; *outside* of the yquant environment, we fit the special
TikZ shape on top of it.

C. Boxing/Highlighting Parts of a Circuit
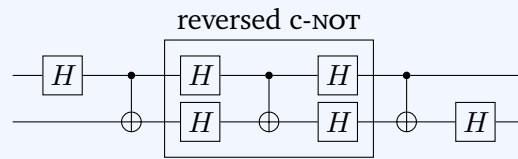
reversed c-NOT


```
% \usetikzlibrary{quotes}
\begin{tikzpicture}
   \begin{yquant*}
      h a;
      cnot b | a;
      [name=left]
      h -;
      cnot b | a;
      [name=right]
      h -;
      cnot b|a;
      h b;
   \end{yquant*}
   \node[fit=(left-0) (left-1) (right-0) (right-1),
         draw, inner sep=6pt, "reversed c-\textsc{not}"] {};
\end{tikzpicture}
```

In this case, since version 0.2, a subcircuit may provide a similar experience:
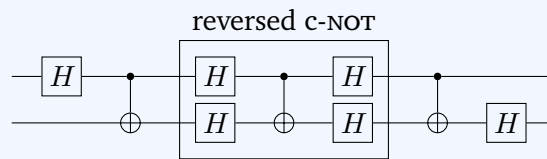
reversed c-NOT

```
% \usetikzlibrary{quotes}
\begin{tikzpicture}
   \begin{yquant*}
      h a;
      cnot b | a;
      [this subcircuit box style={inner sep=6pt, "reversed
      ↪  c-\textsc{not}"}]
      subcircuit {
         qubit {} x;
         qubit {} y;
         h -;
         cnot y | x;
         h -;
      } (-);
      cnot b | a;
      h b;
   \end{yquant*}
\end{tikzpicture}
```

Here, we used the key `/yquant/this subcircuit box style` to influence
only the style of the subcircuit box itself instead of providing global options
that apply to every object in the subcircuit (you wouldn't want the label be
assigned to every single gate).
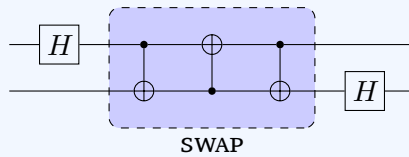
```
% \usetikzlibrary{quotes}
\begin{tikzpicture}
   \begin{yquant*}
      h a;
      [this subcircuit box style={draw, dashed, rounded corners,
       ↪  fill=blue!20, inner xsep=6pt, inner ysep=10pt, "\textsc{swap}"
       ↪  below}, register/default name=]
      subcircuit {
         qubit a;
         qubit b;
         cnot b | a;
         cnot a | b;
         cnot b | a;
      } (a-b);
      h b;
   \end{yquant*}
\end{tikzpicture}
```

Since version 0.2, fully enclosing a bunch of operations (with no controls extending to some inner component) is possible by means of `subcircuit`s. Before, this had to be done using named operations and layers. Note that here we used the style `/yquant/this subcircuit box style` to assign a styling that only applies to the box containing the subcircuit, but not to the inner gates—which would have happened had we just given the arguments to the subcircuit directly.

`yquant` does not support the fancy nearest-neighbor swap gate that `quantikz` has. It would however not be very difficult to implement this particular shape and make it available. Maybe even a multi-swap gate using the `knots` library would be possible.

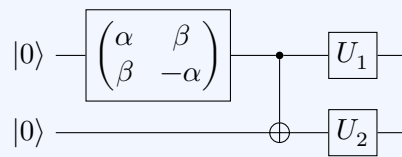### 6.3.7 VIII. Otherwise undocumented features
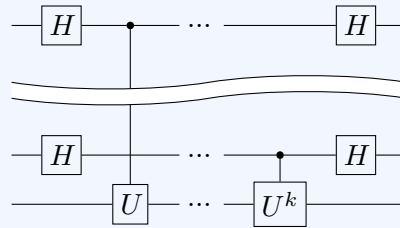

```
% \usetikzlibrary{quantikz}
\begin{tikzpicture}
    \begin{yquant}[register/default name=]
        qubit a;
        [name=wave, register/minimum height=1cm]
        nobit wave;
        qubit b;
        qubit c;

        h a, b;
        box {$U$} c | a;
        [draw=none]
        box {$\dots$} a, b-;
        box {$U^k$} c | b;
        h a, b;
    \end{yquant}
    \node[wave, fit=(wave) (current bounding box.east |- wave), inner
    ↪  ysep=.5pt, inner xsep=0pt] {};
\end{tikzpicture}
```

Here, we included `quantikz`, which provides the `wave` shape, then introduced a register that will contain this wave (and enlarged it sufficiently). After the circuit is drawn, we `fit` the wave along. Since the name assigned to a register without any text actually is of a `coordinate` shape, we need to enlarge the height of the wave by providing a slightly increased `inner ysep`. Additionally, `quantikz` sets a negative `inner xsep`, which is probably required for its grid layout; but `yquant` positions exactly, so we also need to reset this.

85

6.3.8 X. Troubleshooting

$$|0\rangle \;-\; \begin{pmatrix} \alpha & \beta \\ \beta & -\alpha \end{pmatrix} \;-\bullet-\; \boxed{U_1} -$$

$$|0\rangle \;-\!\!-\!\!-\!\!-\!\!\oplus-\; \boxed{U_2} -$$

```
\begin{tikzpicture}
   \begin{yquant}
      qubit {$\ket0$} a[2];
      box {$\begin{pmatrix}
              \alpha & \beta \\
              \beta & -\alpha
            \end{pmatrix}$} a[0];
      cnot a[1] | a[0];
      box {$U_{\protect\the\numexpr\idx+1}$} a;
   \end{yquant}
\end{tikzpicture}
```

7 Wishlist

This section contains some thoughts on future improvements and features.

- Support for other languages.
  It would be particularly nice to introduce a language mode. While the yquant language will always provide the set of everything yquant can do at the moment, it would be nice if yquant can automatically detect OpenQASM and parse its content correctly. OpenQASM is much more limited than yquant and, being a language designed for actual execution of the circuits, does not provide means to change visual appearance. Probably some yquant additions to OpenQASM would be ok, as long as they only complement the original language? Also, OpenQASM support would probably require subcircuits.

  Another nice feature would be to support qasm. Also here, the feature set is much more limited and it would probably be hard to implement an automatic detection, the user would have to specify the language by hand.

- Vertical layout.
  Sometimes, long quantum circuits on a portrait page can be better represented in a vertical layout. Also if lots of explanations are to be added, this becomes problematic in the horizontal version. In principle, yquant's approach could allow for a simple key switch that changes horizontal to vertical. Currently, this is largely unsupported by all quantum circuit packages except for qpic.

8 Changelog

8.1 2020-03-15: Version 0.1

Initial release

8.2 2020-03-22: Version 0.1.1

Complete rewrite of the register name parser. `yquant` now understands comma-separated lists and ranges in indices, and also is far more tolerant with respect to whitespaces.

`yquant` now also supports non-contiguous vector registers and allows to add new registers into an already existing vector that is not the last register, and also in the unstarred mode.

8.3 2020-04-11: Version 0.1.2

Introduce `setstyle` and `addstyle` pseudo-gates that allow to style individual wires; rename `setwire` to `settype` (the old name is still available and shows a deprecation warning).

Complete rewrite of the way `yquant` draws wires; projection anchors are removed in favor of clipping paths. This allows perfect connections between gates and wires, even if the (rather rectangular) wire lines meets with nonplanar shapes, while still preserving the possibility of transparent wires.

`yquant` now also properly draws non-contiguous multi-qubit operations.

New gate: `correlate`. Various bug fixes.

8.4 2020-06-02: Version 0.2

Introduce `subcircuit`; required rewriting how `yquant` internally positions vertically. Provide simple macros to load circuits (or parts) from a file and to declare own custom gates.

8.5 2020-06-07: Version 0.2.1

Introduce a macro to declare a lightweight custom gate, which is only a single box with custom content.