USING XY TO TYPESET AUTOMATA

ARUN DEBRAY
FEBRUARY 1, 2014

Contents

1. Introduction

Hello! If you are reading this guide, you are probably a student in an automata theory class, such as Stanford's CS 103 or CS 154, and you want to learn how to typeset automata in LaTeX. I find that many people use a library called `tikz` for this; however, it's not the easiest to learn, and also makes it hard to typeset automata quickly. I learned to use XY, which to me achieves the right balance between power and ease of use.

...that said, I have yet to see a guide for XY that makes it easy to learn. I have found [2] to be a pretty comprehensive reference, but I spent lots of time being confused by it, especially since it has only one example of an automaton and doesn't really bother to explain it. So I hope this overview can accomplish the following:

(1) teach you enough of the XY library to typeset good-looking automata for course notes and/or homework, and
(2) provide explanations, rather than just examples, to minimize befuddlement and allow you to spend less time on the mechanics of typesetting your automata.

Note however that this is a guide for XY, not for LaTeX, so I will assume that you've used LaTeX before. You don't need to know anything too fancy, but knowing the basics of typesetting mathematics will make life easier. There are many good places to learn this, such as [1].

XY is a package, so to use it one should somewhere in the preamble call `\usepackage[all]{xy}`.

For the forseeable future, the latest version of this guide will be posted at `http://www.stanford.edu/~adebray/using_xy.pdf`, and the TeX source will be posted at `http://www.stanford.edu/~adebray/using_xy.zip`.

2. Simple Diagrams

XY was originally envisioned in order to create commutative diagrams, which mathematicians use to display functions between objects. Thus, it is by default easier to typeset commutative diagrams than finite automata. Though this guide is directed towards automata, I think that illustrating a couple basic examples will illuminate the basic principles of XY.

Here's a simple commutative diagram.[1]

$$\begin{array}{ccc} A & \longrightarrow & B \\ \downarrow & & \downarrow \\ C & \longleftarrow & D \end{array}$$

The code to generate this is as follows:

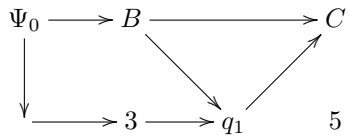
[1] What it means depends on context, but you could think of it as a choice of four functions between the vector spaces A, B, C, and D.

```
\[\xymatrix{
    A \ar[r] \ar[d] & B \ar[d] \\
    C               & D \ar[l]
}\]
```

This should shed some light on the basic mechanics of XY. Specifically:

- Items are typeset on a grid layout. Much like a table, array, or matrix environment, columns are separated with `&` and rows with `\\`.
- `xymatrix` isn't an environment; just a command. Moreover, it's called in math mode.
- The basic diagram command is an arrow, of the form `\ar[`*destination*`]`. The command is called at the origin of the arrow, and the destination is specified with directions: `\ar[d]` points the arrow one row downward, `\ar[r]` one column to the right, `\ar[u]` points it one row upwards, and `\ar[l]` points it one column to the left.

Here's another more interesting commutative diagram.

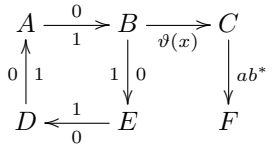


The code for this diagram is as follows:

```
\[\xymatrix{
    \Psi_0 \ar[r] \ar[d] & B \ar[rr] \ar[dr] & & C \\
    \ar[r] & 3 \ar[r] & q_1 \ar[ur] & 5
}\]
```

Notice that directions can stack, so that `\ar[dl]` typesets an arrow that goes diagonally down and to the left, and `\ar[rr]` goes right, but over two grid entries. Notice also that since we're in math mode, any math symbol you like can be a grid entry, or no text at all.

Finally, it'll be important to label the arrows, too. This can be done by attaching superscripts and subscripts to them. Unlike normal math-mode typesetting, though, these are placed relative to the arrow: if one looks along the direction of the arrow, anything within a superscript is placed to its left, and anything within a subscript is placed to its right. In the following diagram, all of the 0s are superscripted, and all of the 1s subscripted.

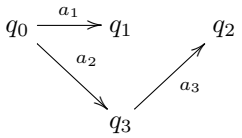


The code for this diagram is produced below.

```
\[\xymatrix{
    A \ar[r]^0_1 & B \ar[d]^0_1 \ar[r]_{\vartheta(x)} & C \ar[d]^{ab^*}\\
    D \ar[u]^0_1 &E \ar[l]^0_1 &F
}\]
```

This should be all that you need to know before introducing automata.

Exercise 2.1. Try typesetting



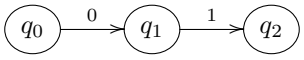
3. Automata

So this is all excellent, you say, but you're here to learn how to typeset automata. How can the above diagrams be turned into automata? There are three things that have to be done, and I'll deal with each in turn.

- The starkest difference between automata and commutative diagrams is that in a finite automaton, each state is traditionally bordered by a circle. In XY, one can wrap a given state in a circle by prefacing it with `++[o][F]`, but since we're going to be doing this for every state anyways, XY allows us to declare it at the top, before calling `\xymatrix`. Thus, one can use the syntax below.

```
\[\entrymodifiers={++[o][F-]}\xymatrix{
    q_0 \ar[r]^0 & q_1 \ar[r]^1 & q_2
}\]
```

This produces the following diagram:

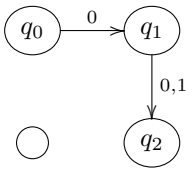


I usually find it easier to wrap that syntax up in a macro, e.g. `\newcommand{\atm}{\entrymodifiers={++[o][F-]}}`, and then it's a lot easier to invoke, as `\atm\xymatrix{...}`. I'm going to follow this convention for the rest of the guide, so that everything is less cluttered.

Notice that `\entrymodifiers` modifies every entry — even empty ones. This means that if you use empty states in automata, something unexpected might happen. Look what happens to the following code.

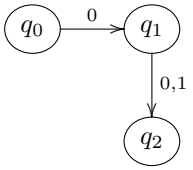
```
\[\atm\xymatrix{
    q_0 \ar[r]^0 & q_1\ar[d]^{0,1}\\
    & q_2
}\]
```

The lower left entry is empty, but it gets surrounded by a circle anyways.



The way to fix this is to "reset" this state, which is done by with `*{}`. Here's what the correct code looks like:

```
\[\atm\xymatrix{
    q_0 \ar[r]^0 & q_1\ar[d]^{0,1}\\
    *{} & q_2
}\]
```

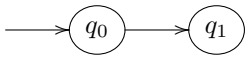


That looks much nicer.

- Automata also have a start state, generally denoted with an arrow pointing to the first state from the left.[2] This isn't hard — you can just add an empty state to the left of the start state with an arrow pointing to it. But as noted above, you have to use `*{}` to make sure it doesn't get a circle drawn around it. Here's an example.

```
\[\atm\xymatrix{
    *{} \ar[r] & q_0 \ar[r] & q_1
}\]
```

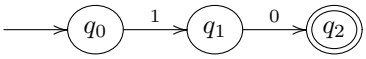
Here's how it looks.



[2] If you have some other way of indicating the start state of an automaton, drop me an email and I'll add it here. But I've only seen this one.

- Finally, automata need accept states, traditionally denoted with a double circle. The incantation is `*++[o][F=]`, which means that first you reset the entry modifiers by adding the `*`, and then indicate a double circle (with the `=`) rather than a single one (which is `[F-]`). I find it convenient to define a macro akin to `\newcommand{\accept}[1]{*++[o][F=]{#1}}`, which makes my code more readable. I will follow this convention for the rest of the guide.

  This is how it looks when you try it:

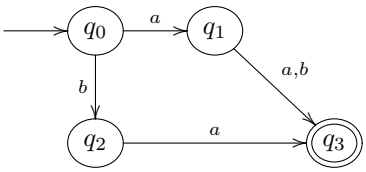


  And the code looks like this.

```
\[\atm\xymatrix{
    *{} \ar[r] & q_0 \ar[r]^1 & q_1 \ar[r]^0 &\accept{q_2}
}\]
```

  At this point, you have the techniques to typeset many simple automata.

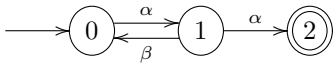
  Exercise 3.1. Try typesetting



4. More Interesting Arrows

One of the reasons XY-pic is so powerful is that there are so many interesting things one can do with the arrows. This is what makes it so useful for automata.

One useful trick is setting arrows going back and forth between two states, like this.



The code to do this looks like this.

```
\[\atm\xymatrix{
    *{} \ar[r] & 0 \ar@<0.1cm>[r]^\alpha & 1 \ar@<0.1cm>[l]^\beta \ar[r]^\alpha
       & \accept{2}
}\]
```

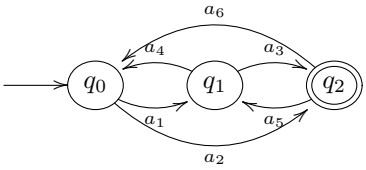
This probably isn't terribly intuitive, so let's understand better what's going on.

- The general form of an arrow is `\ar[`*dir*`]` followed optionally by a superscript or subscript for the label. But this can be extended by `@`-statements, which look like `@<...>`, `@(...)`, `@[...]`, `@/.../`, and so on. Each of these means something different, and more than one can be used on an arrow. But this is just a point about syntax; what each statement actually means will be explained in the relevant sections.
- The syntax `@<...>` can be used to slide an arrow perpendicular to its line of direction. You specify how far to shift it within the delimiters; for example, `@<1cm>` shifts the arrow by one centimeter in the `^` direction (i.e. to the left of the arrow), and `@<-1cm>` shifts it one cm in the `_` direction (i.e. to the right of the arrow). In general, 1 cm is far too much; in the example above, 0.1 cm was about the right amount. It is also possible to specify the distance in units of inches (in), x-height (ex, i.e. the height of the letter 'x' in the current font), or points (pt). Then, the syntax is `\ar@<`*dist*`>[`*dir*`]`.

Sliding arrows are very useful, and in particular when typesetting automata I use them whenever I have transitions $q_0 \to q_1$ and $q_1 \to q_0$.

A variant on this can be done by replacing `@<...>` with `@/.../`, which curves the arrows rather than sliding them. The syntax is slightly different: this time, you need to specify the direction. For example, `@/^0.2cm/` curves 0.2 cm to the left, and `@/_0.2cm/` curves 0.2 cm to the right (where the arrowhead is considered forward). So now you can

produce automata like this one:

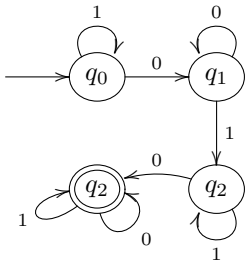


The code to produce this automaton looks like

```
\[\atm\xymatrix{
    *{} \ar[r] & q_0\ar@/_0.3cm/[r]_{a_1} \ar@/_0.8cm/[rr]_{a_2}
               & q_1\ar@/^0.3cm/[r]^{a_3} \ar@/_0.3cm/[l]_{a_4}
               & \accept{q_2}\ar@/^0.3cm/[l]^{a_5} \ar@/_0.8cm/[ll]_{a_6}
}\]
```

A third useful tool to have is the self-loop, for when a state has a transition to itself. This is the use of the `@(...)` syntax. The specific syntax is `\ar@(`*in*`,`*out*`)`, where *in* and *out* are one of the eight directions `u`, `ur`, `r`, `dr`, `d`, `dl`, `l`, and `ul`. These correspond to the reasonable directions (e.g. `ur` means to the upper right), and produces an arrow from the *in* direction to the *out* direction. Labels still work, but since the look is very tight only one of the sides is really useful.

Here's an example of several of the directions.

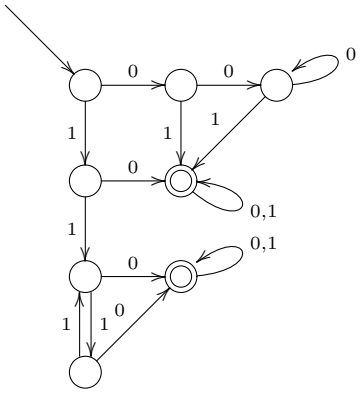


It was generated with the following code:

```
\[\atm\xymatrix{
    *{} \ar[r] & q_0 \ar@(ul,ur)^1 \ar[r]^0
        & q_1\ar[d]^1 \ar@(ur,ul)_0\\
    *{} & \accept{q_2}\ar@(dl,l)^1 \ar@(d,r)_0
        & q_2\ar@/_0.2cm/[l]_0 \ar@(dr,dl)^1
}\]
```

Interestingly, the syntax `\ar@(`*in*`,`*out*`)[]`, with brackets at the end, is the syntax used in [2]. It seems equivalent in these cases, but I don't know why they would default to it; if you find out, let me know!

Exercise 4.1.



There are many, many things that XY can do with arrows, but these are the few that I've found to be useful for working with automata. Consult [2] for some more.

5. More Interesting Labels

This section isn't super important for automata, but in some cases it can make the diagrams look better.

XY-pic typesets on a grid format, as you know by now, but to accomodate the fact that different grid entries may have different sizes, it adjusts the arrows in order to make everything consistent. The idea is that arrows may have different lengths, but the center of each grid element is always consistent. Here's an example, albeit not from the world of automata:

$$\begin{array}{ccc} A_1 \times A_2 \times A_3 \times A_4 & \longrightarrow & B_1 \\ \downarrow & & \uparrow \\ A_1 & \longrightarrow & B_2 \end{array}$$

However, labels aren't by default adjusted with the arrows. That is, the label is placed at the halfway point between the centers of two grid entries, not the halfway point of the arrow. To show you what that means, I'll add some labels to the above diagram:

$$\begin{array}{ccc} A_1 \times A_2 \times A_3 \times A_4 & \xrightarrow{f_1} & B_1 \\ \downarrow{\scriptstyle g_1} & & \uparrow{\scriptstyle g_2} \\ A_1 & \xrightarrow{f_2} & B_2 \end{array}$$

The code for this diagram looks like this:

```
\[\xymatrix{
    A_1 \times A_2\times A_3\times A_4\ar[r]^{f_1}\ar[d]^{g_1} & B_1\\
    A_1\ar[r]^{f_2} & B_2\ar[u]_{g_2}
}\]
```

The labels f_1 and f_2 are aligned, but f_1 looks bad because it runs into the grid object. There are a bunch of tricks one can use to fiddle with the labels' spacing:

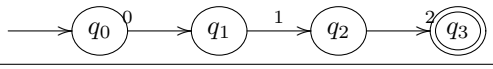
- One can more explicitly position labels by specifying a "place" after the `^` or `_`. The one you probably want is the hyphen, `-`. This repositions the label to be at the center of the arrow, rather than based on the grid. Here's what it does to the previous diagram:

$$\begin{array}{ccc} A_1 \times A_2 \times A_3 \times A_4 & \xrightarrow{f_1} & B_1 \\ \downarrow{\scriptstyle g_1} & & \uparrow{\scriptstyle g_2} \\ A_1 & \xrightarrow{f_2} & B_2 \end{array}$$

```
\[\xymatrix{
    A_1 \times A_2\times A_3\times A_4\ar[r]^-{f_1}\ar[d]^{g_1} & B_1\\
    A_1\ar[r]^{f_2} & B_2\ar[u]_{g_2}
}\]
```

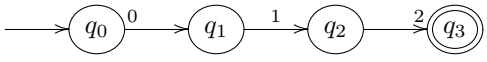
  That looks nicer.

- Alternatively, if you want the label to be right near the edge of the arrow, use `<` or `>`. These place the label right at the beginning and end of the arrow, respectively, as in the following example:




```
\[\atm\xymatrix{
    *{}\ar[r] &q_0 \ar[r]^<0 & q_1\ar[r]^1 & q_2\ar[r]^>2 &\accept{q_3}
}\]
```

  That's not always ideal, but fortunately there's a fix: one can use multiple `<<` or `>>` signs. Each additional sign moves the label 3 pt away from that end of the arrow. Here's what it looks like with the correction:

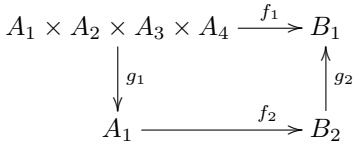



```
\[\atm\xymatrix{
    *{}\ar[r] &q_0 \ar[r]^<<0 & q_1\ar[r]^1 & q_2\ar[r]^>>2 &\accept{q_3}
}\]
```

It's a little bit counterintuitive that you add additional signs to go further away from the end that they're pointing to, so to speak. And often it takes a little fiddling to make this look good, but the results can be nice.

- Finally, for more explicit positioning, one can use `()`. `(`a`)` indicates that the label should be a of the way from the center of the source (i.e. `(0)`) to the center of the destination (i.e. `(1)`). For example, `(0.3)` will place the label one-third of the way along the grid. This is easier to adjust than some of the other options.

  Here's an example of what it looks like.

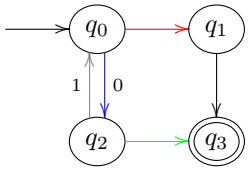



```
\[\xymatrix{
    A_1 \times A_2\times A_3\times A_4\ar[r]^-{f_1}\ar[d]^{g_1} & B_1\\
    A_1\ar[r]^(0.72){f_2} & B_2\ar[u]_{g_2}
}\]
```

6. Color

Color is far from essential for producing automata, but sometimes it's useful for drawing attention to a particular arrow or state or such. Like any use of color in LaTeX, you should make sure you call `\usepackage{xcolor}` somewhere in the preamble; for information on what colors are supported with which options, consult [1].

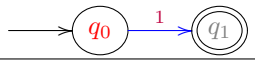
Coloring arrows is not difficult: the syntax is `\ar@[`*color*`][`*dir*`]`. In other words, everything is the same, but the color is named within the `@[...]` syntax. Here are some examples of it in action.




```
\[\atm\xymatrix{
    *{}\ar[r] & q_0 \ar@[red][r] \ar@<0.1cm>@[blue][d]^0 & q_1\ar[d]\\
    *{} & q_2\ar@<0.1cm>@[gray][u]^1 \ar@[green][r] & \accept{q_3}
}\]
```

Note, however, that color simply doesn't work on curved arrows, i.e. arrows which have been modified with `@(...)` or `@/.../`. That is, you will get a colored arrow, but everything else past that arrow in the figure, and in the rest of the document, will take that color by default. Word on the street has it that versions 3.8.4 and later fix this bug, but I haven't confirmed this; if you know more about it, please let me know. In any case, if you can avoid curved colored arrows, your code will be more portable. Color in XY-pic in general seems to be relatively finicky, but I have had no problems with coloring straight arrows.

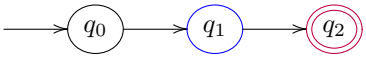
Coloring the text of states or of labels works just like normal LaTeX coloring, i.e. using `\textcolor{`*color*`}{`*text*`}`. Here's an illustration.




```
\[\atm\xymatrix{
    *{} \ar[r] & \textcolor{red}{q_0} \ar@[blue][r]^{\textcolor{purple}{1}} & \
       accept{\textcolor{gray}{q_1}}
}\]
```

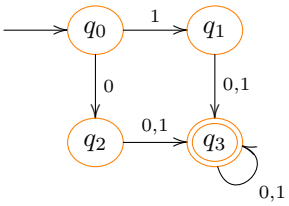
Finally, for automata one might want to color the frame. Recall that the default frame was defined as `++[o][F-]`; to add color, one changes this to `++[o][F-:`*color*`]`. For the accept state you'll want to write `++[o][F=:`*color*`]`, since it has a double frame. There are two ways I can think of to use this: for a single state you'll want to reset from the default by using the `*{...}` syntax.

```
\[\atm\xymatrix{
    *{} \ar[r] & q_0\ar[r] & *++[o][F-:blue]{q_1} \ar[r] &*++[o][F=:purple]{q_2}
}\]
```



Alternatively, one could use this to color every state in the automaton. This involves changing the `\entrymodifiers` syntax sightly, and then also handling the accept like above. Here's what this looks like.

```
\[\entrymodifiers={++[o][F-:orange]}\xymatrix{
    *{} \ar[r] & q_0\ar[d]^0 \ar[r]^1 & q_1\ar[d]^{0,1}\\
    *{} & q_2\ar[r]^{0,1} & *++[o][F=:orange]{q_3} \ar@(d,r)_{0,1}
}\]
```



If you're going to do this for a lot of automata, it might be worth defining macros for these.

7. Conclusion

There is so much more to XY-pic that I haven't covered, but this should be sufficient for typesetting good-looking automata. I hope it was useful.

If you find something unclear or wrong or such, please let me know; I can be contacted at `adebray@stanford.edu`. Also let me know if there's something you want to be able to do that I haven't included; this covers most of what I use to typeset automata, but other people probably use slightly different conventions that I should mention.

One thing I definitely intend to add is an appendix on fixing errors in these; LaTeX unfortunately doesn't have the clearest error messages, and XY has chosen to uphold this tradition. Stay tuned for what some common errors mean and how one might fix them.

Appendix A: Solutions to the Exercises

Exercise 2.1:

```
\[\xymatrix{
    q_0 \ar[r]^{a_1} \ar[dr]^{a_2} & q_1 & q_2 \\
    & q_3 \ar[ur]_{a_3}
}\]
```

Exercise 3.1:

```
\[\atm\xymatrix{
    *{}\ar[r] & q_0\ar[r]^a\ar[d]_b &q_1\ar[dr]^{a,b}\\
    *{} & q_2\ar[rr]^a &*{} & \accept{q_3}
}\]
```

Exercise 4.1

```
\[\atm\xymatrix{
    *{}\ar[dr]\\
    *{}& \ar[r]^0\ar[d]_1 &\ar[r]^0\ar[d]_1 & \ar@(r,ur)[]_0\ar[dl]_1\\
    *{} & \ar[r]^0\ar[d]_1 & \accept{}\ar@(dr,r)[]_{0,1}\\
    *{} & \ar[r]^0\ar@<0.5ex>[d]^1& \accept{}\ar@(r,ur)[]_{0,1}\\
    *{} & \ar@<0.5ex>[u]^1 \ar[ur]^0
}\]
```

References

[1] "Wikibooks: LaTeX." `en.wikibooks.org/wiki/LaTeX`. 24 October 2013.

[2] Rose, Kristoffer H. "XY-pic User's Guide." `http://www.math.ubc.ca/~cautis/tools/xypic.pdf`. 24 May 2012.

[3] Rose, Kristoffer H., and Ross Moore. "XY-pic Reference Manual." `http://cs.brown.edu/about/system/software/latex/doc/xyrefer.pdf`. 16 February 1999.