`hexpand`: Zsh-styled globbing module for Haskell

Zhiming Wang, March 17, 2016

Introduction

Globbing (and shell expansions in general) is essential in interactive shells and shell scripting. I write shell scripts all the time, and am looking into the possibility of doing some scripting in Haskell, so one of the first steps is to find a way to perform expansions in Haskell that is comparable in power and conciseness to the shell counterparts.

Among the various POSIX shells I know (or know to a limited extent), Zsh's expansion capabilities are simply unparalleled. The rich expansion facilities allow Zsh to magically accomplish the hard or impossible in other shells, usually concisely. For instance, if we want to select all Haskell source files in a directory tree and put them into an array sorted by mtime, in Zsh the solution is as simple as

```
hsfiles=( /path/to/tree/**/*.hs(om) )
```

Meanwhile, a robust solution for Bash is extremely complicated (as far as my Bash skill goes). This is just one of many examples. The full documentation of Zsh's expansion facilities can be found here,[1] or in the man page of `zshexpn(1)`.[2] It is hard to appreciate its vast power without actually reading the docs and putting the facilities to use.

Back to Haskell, we should note that a globbing package named `Glob`[3] have already existed on Hackage since 2008, which supports most of the basic glob operators available in Zsh (which is already superior to Bash). However, it understandably does not support glob qualifiers, glob modifiers, glob flags and such that make Zsh globbing (and expansions in general) really shine. Therefore, I aim to replicate part of Zsh's globbing capabilities in this project.

Unfortunately, due to time constraints, the outcome so far does not match the original plan, which was much more ambitious and technically challenging (see the section Future directions for some discussions on the original plan). However, I still managed to realize some extremely useful features that are not available in the existing tools.

Implementation

The idea is very simple. Currently there are three main stages:

1. Parsing a glob string (with glob pattern, qualifiers and modifiers);
2. Match building: generate the list of paths that match the glob pattern;
3. Postprocessing: filter and sort paths according to qualifiers, and transform path strings according to modifiers.

We use `attoparsec` in the parsing stage. The parsed glob string is represented in a new data type `Pattern`, which can be inspected from `src/Hexpand.hs`.

Features

Zsh offers a myriad of expansion features, and only a fraction of them have been implemented by now. Implemented features are listed below.

Glob operators

All default operators (operators that do not require `EXTENDED_GLOB` to be recognized) have been implemented:

- `*`: star (any string, including null);

- `?`: question mark (any character);
- `[...]`, `[^...]`, `[!...]`: character classes and inverted character classes (note that this operator is partially implemented, in that named character classes, e.g. `[:alnum:]`, are not implemented yet, let alone named character classes in the full Unicode range);
- `<[x]-[y]>`: number range;
- `(...)`: grouping;
- `|`: alternation.

Glob qualifiers

- `/`: directories only;
- `.`: regular files only;
- `@`: symlinks only;
- `-/`: directories and their symlinks;
- `-.`: regular files and their symlinks;
- `oc`: sorting, where *c* is one of
    - `n`: by name;
    - `L`: by file size;
    - `l`: by link count;
    - `a`: by atime;
    - `m`: by mtime;
    - `c`: by ctime.

Glob modifiers

- `a`: convert to absolute path;
- `A`: canonicalize (resolve) path;
- `h`: take dirname;
- `l`: convert to lowercase;
- `r`: drop file extension;
- `t`: take basename;
- `u`: convert to uppercase.

Miscellaneous

- Tilde expansion for current user: `~` expands to `$HOME`, `~/...` expands to `$HOME/...`.

Missing

There are some important yet currently missing features:

- Unicode support in glob strings. Glob strings are currently limited to ASCII due to our usage of `Data.Attoparsec.ByteString.Char8` for parsing. Note that Unicode paths can still be matched; it is codepoints beyond U+007F *in a glob string* that can cause trouble.

- `.` and `..` currently don't have special meanings in glob strings (they are matched literally for now).

- Recursive globbing with `**` or `***` are not yet implemented.

- Features exposed via Zsh `EXTENDED_GLOB` option (non-default), including globbing flags, are not yet implemented.

- A leading dot in a filename is not currently required to be matched explicitly (e.g., `~/*` will pick up `~/.zshrc`), which goes against the default in Zsh, and depending on use case may or may not be a blessing. This behavior is controlled by the `GLOB_DOTS` option in Zsh (or `DOT_GLOB` for Bash compatibility), which is off by default.

- There are many unimplemented qualifiers and modifiers, as well as some rough edges in the implemented parts.

Future directions

The original plan for this project was very ambitious: I wanted to implement parameter expansion too, which has even more variety than globbing (officially "filename generation" in Zsh), and enable mixing of parameter expansion and globbing to achieve much more (for instance, the following string can be used to find the various configuration files of `mpv`, assuming standard location: `${XDG_CONFIG_HOME:-$HOME/.config}/*.conf(N)`, which uses both parameter expansion and filename generation).

Of course, a technical issue immediately arises: Haskell does not have mutable parameters or variables to begin with, so what does "parameter expansion" even mean in this context? We can answer this question from two angles:

1. We can access environment variables (`System.Environment.getEnv`) and use them in parameter expansion; `XDG_CONFIG_HOME` and `HOME` in the example above are both environment variables.

2. While Haskell does not have mutable variables, it does have visible identifiers that could reference values. Consider the following hypothetical use case:

   ```haskell
   -- |Get the list of Haskell source files within a project with name pn.
   getHaskellSourceFiles :: String -> IO [FilePath]
   getHaskellSourceFiles pn = do
       projectsRoot <- expand "$HOME/dev/src/hs"
       glob "$projectsRoot/$pn/**/*.hs"
   ```

   This may not be a compelling case because we can just manually build an equivalent glob string with concatenation, but it is not hard to envision complicated use cases where a parameter expansion (with string manipulations, which an expansion library should be good at) one-liner can save many lines of code.

The latter is not easy in standard Haskell, though, because we essentially need

```haskell
readIdentifierValue :: String -> a
```

where the `String` argument is the identifier assigned in source code, which is almost arbitrary from the compiler's point of view.

However, Template Haskell comes to rescue here, because it allows us to access and manipulate Haskell ASTs, so we can turn a programmer-friendly string `$projectsRoot/$pn/**/*.hs` into more mundane syntax, e.g.,

```haskell
glob Dollar projectsRoot "/" Dollar pn "/**/*.hs"
```

where `Dollar` could be the constructor of a special indicator type used to mark the next argument as an identifier being expanded. We can then work from there.

Therefore, this is a promosing direction to work on. Of course, the less adventurous missing features in the subsection Missing should be implemented first.

Example

Usage of globbing should be obvious to the target audience. Nevertheless, we present a rather contrived example, available in the repository as `example/Libnames.hs`. What `Libnames.hs` does is to find all libraries (files beginning with `lib`) in the top level of `/usr/lib`, strip the `lib` prefixes as well as version numbers (if any) and extensions to obtain the library names, then print out the names after deduplication. When I run the script on my system (OS X 10.11.3) in a 80-column terminal window, I get

```
> stack runghc --package terminal-size example/Libnames.hs
ATCommandStudio           dbm                       netsnmpmibs
ATCommandStudioDynamic    des425                    netsnmptrapd
... 48 lines omitted ...
cupsppdc                  netsnmp                   z
curl                      netsnmpagent
curses                    netsnmphelpers
```

A Zsh implementation of the same task would be

```zsh
#!/usr/bin/env zsh
setopt histsubstpattern
libnames=( /usr/lib/lib*(.:t:s/[.-]*//:s/#lib//) )
print -c ${(u)libnames}
```

which is very concise.

Meanwhile, the library names generation part of the example Haskell script[4] is

```haskell
import Hexpand

import Control.Monad
import Data.List (elemIndex, group, sort)
import Data.Maybe (fromMaybe)

dedup :: (Ord a) => [a] -> [a]
dedup = map head . group . sort

getLibnames :: IO [String]
getLibnames = do
    libFilenames <- liftM2 fromMaybe (return []) (glob "/usr/lib/lib*(-.:t)")
    return $ dedup (map transform libFilenames)
  where
    transform f = drop 3 (dropSuffix f)
    dropSuffix f = take (min dashPos dotPos) f
      where
        charPos c s = fromMaybe (length f) (elemIndex c s)
        dashPos     = charPos '-' f
        dotPos      = charPos '.' f
```

This is not particularly satisfactory, but with glob pattern, qualifier and modifier, it is already much more pleasant than building from `System.Directory` or even `System.FilePath.Glob`.

Once the `s/l/r[/]` substitution modifier is implemented in `hexpand`, however, we will be able to replace `getLibnames` with

```haskell
getLibnames :: IO [String]
getLibnames = do
    libnames <- liftM2 fromMaybe (return [])
                                 (glob "/usr/lib/lib*(.:t:s/[.-]*//:s/#lib//)")
    return $ dedup libnames
```

so the module does have much space for features and improvements.

Contact

My email address is zmwangx@gmail.com or zmwang@stanford.edu.

My GitHub handle is @zmwangx.

---

1. http://zsh.sourceforge.net/Doc/Release/Expansion.html.↩
2. The HTML doc for Zsh expansions alone is 187 KiB in size (Zsh 5.2). Compare that to the full Bash Reference Manual, which is 738 KiB in total (Bash 4.3.42(1)).↩
3. https://hackage.haskell.org/package/Glob.↩
4. The rest of the example program implements a pretty printer that automatically columnizes output entries based on tty width — equivalent to Zsh's `print -c`. That part is not related to globbing.↩