

ICFP Programming Contest 2012

Lambda Lifting

ICFP Programming Contest Organising Team

July 13th 2012

1 Introduction

In recent years, functional languages such as Haskell and OCaml have been gaining in popularity (see for example the TIOBE Programming community index¹ which shows Haskell on a constant rise since 2006). Even though none of these languages have yet broken into the top 50, modern imperative languages often include a variety of features inspired by functional programming. Furthermore, the rise of multicore and parallel programming suggests a bright future for side-effect free, pure functional programming techniques. Even C++ has anonymous functions now, and they are slated for Java 8 to help support multicore!

While all of this is good news for advocates of functional programming languages, it has unfortunately led to a worldwide shortage of Lambdas, and unless something is done, current models predict that we will run out by October 2012. The good news is that a plentiful supply has been located underground, in the Lambda mines beneath the Lomond Hills near St Andrews in Scotland. If we can find a way to lift these Lambdas to the surface efficiently, we should have enough of a supply to take us well beyond the end of the Unix epoch.

We have already implemented the technology to survey underground, and we have maps of the mines. Your task is, given a map, to find a route for a Robot which will retrieve all of the Lambdas from the mine, avoiding any obstacles such as falling Rocks, and lift the Lambdas back to the surface.

Note: Our surveying technology is probably not perfect. It is possible we will discover some more things underground that we did not expect. We will let you know as the contest progresses. We will make at most *one* refinement during the first 24 hours, and at most *four* refinements during the contest overall. We will make *no* refinements in the 12 hours before either the lightning or full deadline.

Details of any refinements will be posted in the following ways:

- Via the contest web site <http://icfpcontest.org/>
- By email to all who have registered via EventBrite

2 Task Details

The task consists of implementing a program to read a mine description, and generate a sequence of commands to issue to a Robot to collect all of the Lambdas in the mine, then leave the mine. After each command is issued to the Robot, the following sequence occurs:









1. The Robot moves, if the move is valid, otherwise the Robot does nothing.
2. The map is updated according to the rules described below.
3. Ending conditions are checked.

These steps are described in detail below.

¹<http://www.tiobe.com/index.php/content/paperinfo/tpci/index.html>

2.1 Mine layout

A mine is an $n \times m$ grid of cells, with the convention that $(1, 1)$ is the **bottom left** and (n, m) is the **top right**. Each cell contains one of the following:

	Mining Robot (R in ASCII)		Wall (# in ASCII)
	Rock (* in ASCII)		Lambda (\ in ASCII)
	Closed Lambda Lift (L in ASCII)		Open Lambda Lift (O in ASCII)
	Earth (. in ASCII)		Empty ([SPACE] in ASCII)

Nothing can pass through a Wall, or outside the $n \times m$ grid of the mine. The mining Robot can excavate Earth, move into an Empty space, move onto and collect a Lambda, push Rocks, and enter an Open Lambda Lift. Rocks will fall if they are not supported by another object.

2.2 Robot Movement

In each step, the Robot executes a command, then the mine layout is updated. Commands which the Robot can execute are:

- **Move left**, L, moving the Robot from (x, y) to $(x - 1, y)$.
- **Move right**, R, moving the Robot from (x, y) to $(x + 1, y)$.
- **Move up**, U, moving the Robot from (x, y) to $(x, y + 1)$.
- **Move down**, D, moving the Robot from (x, y) to $(x, y - 1)$.
- **Wait**, W, which does nothing.
- **Abort**, A, which abandons mine exploration.

Moving from a location (x, y) to a new location (x', y') is valid if:

- (x', y') is Empty, Earth, Lambda or Open Lambda Lift.
 - If it is a Lambda, that Lambda is collected.
 - If it is an Open Lambda Lift, the mine is completed.
- If $x' = x + 1$ and $y' = y$ (i.e. the Robot moves right), (x', y') is a Rock, **and** $(x + 2, y)$ is Empty.
 - Additionally, the Rock moves to $(x + 2, y)$.
- If $x' = x - 1$ and $y' = y$ (i.e. the Robot moves left), (x', y') is a Rock, **and** $(x - 2, y)$ is Empty.
 - Additionally, the Rock moves to $(x - 2, y)$.

In all other situations, a move is invalid, and the Robot instead executes **Wait**. After a successful move, the grid location (x, y) becomes Empty.

2.3 Map Update

After Robot movement, the map is updated at each cell. The effect of map update is that Rocks fall if they are not supported, and they may also slide off other Rocks (left or right) and down the back of Lambdas (to the right).

Update works left to right, then bottom to top:

$$\begin{aligned} &(1, 1), (2, 1), \dots, (n, 1), \\ &(1, 2), (2, 2), \dots, (n, 2), \\ &\dots \\ &(1, m), (1, m), \dots, (n, m) \end{aligned}$$

Cell (x, y) is updated according to the following rules:

- If (x, y) contains a Rock, and $(x, y - 1)$ is Empty:
 - (x, y) is updated to Empty, $(x, y - 1)$ is updated to Rock.
- If (x, y) contains a Rock, $(x, y - 1)$ contains a Rock, $(x + 1, y)$ is Empty and $(x + 1, y - 1)$ is Empty:
 - (x, y) is updated to Empty, $(x + 1, y - 1)$ is updated to Rock.
- If (x, y) contains a Rock, $(x, y - 1)$ contains a Rock, either $(x + 1, y)$ is *not* Empty or $(x + 1, y - 1)$ is *not* Empty, $(x - 1, y)$ is Empty and $(x - 1, y - 1)$ is Empty:
 - (x, y) is updated to Empty, $(x - 1, y - 1)$ is updated to Rock.
- If (x, y) contains a Rock, $(x, y - 1)$ contains a Lambda, $(x + 1, y)$ is Empty and $(x + 1, y - 1)$ is Empty:
 - (x, y) is updated to Empty, $(x + 1, y - 1)$ is updated to Rock.
- If (x, y) contains a Closed Lambda Lift, and there are no Lambdas remaining:
 - (x, y) is updated to Open Lambda Lift.
- In all other cases, (x, y) remains unchanged.

The update rules are applied *exactly once* for each location (x, y) during the update phase. In particular, this means a Rock will only fall one space per update.

Note: During map update, all *reads* are from the old state, and all *writes* are to the new state. This can mean that rocks sometimes crash into each other, for example:

```
#* *#          #  #
#* *#  -----> #***#
#####          #####
```

2.4 Ending Conditions

Ending conditions are checked in the following order:

1. **Winning** condition: the Robot is at the same location as an Open Lambda Lift, which means the Lambdas are successfully lifted to the surface.
2. **Losing** condition: if the Robot is at location (x, y) and location $(x, y + 1)$ contains a Rock which was placed at that location in the most recent map update, then the Robot is destroyed.
3. **Abort** condition: the Robot executed the Abort command.

2.5 Input/Output Formats

Submitted programs should read a map description from standard input, and send a sequence of commands to standard output. The map description consists of m lines of up to n characters, describing the mine from top to bottom (i.e. the first line in the file is the row m , the last line is row 1) then left to right (i.e. the first character in each line is column 1). Characters represent the contents of cells as described above, i.e.:

- **R** for the Robot's initial location
- **#** for a Wall
- ***** for a Rock
- **** for a Lambda
- **L** for a Closed Lambda Lift
- **.** for Earth
- a space, for an Empty space

Note that input maps will never contain an Open Lambda Lift, and will contain *exactly one* robot and *exactly one* Closed Lambda Lift. The width of the mine is the number of characters in the longest line. Shorter lines are assumed to be padded out with spaces. For example, here is a simple 15×15 mine layout:

```
#####
#***...R.....#
#***...  ...*..#
#\...\  ...\\.#
#.....  ...*..#
#..      ..  ...#
#.....  ...  ...#
#.....  ...  ...#
#..  .      ..#
#...*.  ..  .....#
#.....  ..  .....#
#.\..  .....*#
#.....  .....#
#.....  .....#
#####L#
```

Programs should output a series of commands to the Robot as a sequence of characters: L for "Move left", R for "Move right", U for "Move up", D for "Move down", W for "Wait", A for "Abort". Any unrecognised characters (for example spaces or newlines) will be ignored. For example, the above map could be solved by:

```
DDDLLLLLLLURRRRRRRRRRRRRDDDDDDDLLLLLLLLLLLLDDRRRRRRRRRRRD
```

3 Scoring

A Robot exploring a mine will score points as follows:

- 25 points gained for every Lambda collected
- 1 point lost for every move made
- 25 extra points per Lambda collected on executing **Abort**.

- 50 extra points per Lambda collected on reaching the winning state.

We have provided a selection of maps on the contest web site (<http://icfpcontest.org/>) to test your programs. For scoring the contest itself, however, we have a selection of maps of various shapes and sizes which we will publish after the competition. These may be quite large.

3.1 Resource Limits

Programs will be given a limited (wall clock) time to find results:

- After **150** seconds, programs will be sent a `SIGINT`, after which they may still output a result.
- After a further **10** seconds, programs will be sent a `SIGKILL`, after which they will score 0.

We may increase these time limits for larger maps. Since all machines are different, we recommend you interpret the limits to mean that you should make your programs as fast as you can, and handle `SIGINT` appropriately.

We will also impose a limit on the length of routes we validate. For a $n \times m$ map, we will take at most the first $n \times m$ steps in the generated route.

3.2 Determining the winner

We will determine the winner as follows, in both the lightning and full divisions:

- We will run each entry on a number of small maps and rank them by aggregate score, using the scoring system described above. Entries scoring below the median score will be eliminated.
- We will then run each remaining entry on a number of larger maps and rank them by aggregate score (including the score from the first round). Again, entries scoring below the median score will be eliminated.
- Finally, we will run each remaining entry on a number of fiendish maps and rank them by aggregate score (including the scores from previous rounds). The entry with the largest score will be the winner.

We will announce the results of the first two elimination rounds in the weeks after the contest, and the overall winner at the International Conference on Functional Programming in Copenhagen.

You are free to create your own maps, and submit them to us if you wish. We may even use them to help judge the contest. We make no guarantee that the maps we provide will have a route that leads to the winning condition.

3.3 Judges' Prize

There is also a Judges' Prize, which will be decided by a vote amongst the judges, and announced at the International Conference on Functional Programming in Copenhagen. This is open to all entries in both the full and lightning divisions, whether they are eliminated in the early stages or not.

4 Submission

Your submission must be a single `.tgz` file, named `icfp-xxxxxxx.tgz` where `xxxxxxx` is the registration number, obtained from EventBrite, of the first team member to register. (To avoid confusion: this will be the *lowest* registration number of any team member. The registration number is the last 8 digits of your EventBrite order number (e.g. 946xxxxx or 947xxxxx).

To submit your entry, share it via Google Docs or Google Drive with one of the following accounts:

- `icfpcontestsubmission@gmail.com` for full submissions.
- `icfpcontestlightning@gmail.com` for lightning division submissions.

You can do this via the Google Docs web interface (<http://docs.google.com/>) by uploading the file, selecting the uploaded file then clicking **Share**. You may submit in either or both divisions, as you wish.

The `.tgz` file will be unpacked into a unique user's home directory, and should contain (at least) the following:

- An executable file `./install`, which is executed exactly once.
- An executable file `./lifter`, which may be generated by `./install`, and will be executed on each test map. It must read map data from standard input, and output a route on standard output.
- A file `./PACKAGES` listing the names of any non-standard packages required by either your `./install` or `./lifter` executables, one per line.
 - **Note:** If you are using the Debian testing environment, this file must instead be named `./PACKAGES-TESTING`.
- A directory `./src`, containing your source code. We will not try to compile this, but we will use it to help choose the Judges' Prize.
- A file `./README`, listing your team members and (optionally) describing how your entry works. We will use this to help choose the Judges' Prize.

Lightning contest deadline: 1200 GMT on Saturday July 14th

Full submission deadline: 1200 GMT on Monday July 16th

Good luck, and enjoy your Lambda Mining.

A Validation

We have implemented a validator for routes, which displays a score and the position (as an ASCII representation) at the end of the route. This validator is available on the web, for a limited number of input maps, at <http://www.undecidable.org.uk/~edwin/cgi-bin/weblifter.cgi>. You may use this to check that your implementation follows the same rules as our validator.

The validator also keeps track (anonymously) of high scores for each map, so that you can see how your routes compare to others.

We make no guarantees about the availability of this server over the contest, although we will try to make sure it stays up as far as possible. Please be kind to it! In particular, please try not to make more than one request every 10 minutes (on average). Any IP addresses found to be making excessive requests will be banned.