ICFP Programming Contest 2019
Worker-Wrappers against Bit Rot

Version 2.0

ICFP Programming Contest Organisers

June 21, 2019, 10:00am UTC

1 Introduction

With the tremendous success of functional programming in industry, it has been projected that by the year 2030 most of the code in the world will be written in functional languages using lambdas.

To cater to the growing need for lambdas, in 2012 the ICFP contest participants have developed procedures for lifting this precious resource from mines in Scotland.[1] Since then, lambdas have been excavated from mines all over the world, and in 2017, ICFP contestants came up with optimal ways to deliver lambdas to users via advanced punting strategies.[2]

The accelerated mining and delivery of lambdas allowed us to get rid of most legacy code — the infamous *bit rotting problem* was solved once and for all. However, eliminating legacy code creates a new problem: how to dispose of all the bit-rotten software? A plan has been devised to store it in the empty mines the lambdas were originally extracted from, which can now be converted into legacy waste silos.

To prevent bit rot from seeping into the soil, the mines, before they are turned into silos, need to be insulated by *wrapping* their entire surface in a decay-containing substance. To perform this task, scientists have designed robots called *worker-wrappers*. This year, we ask the ICFP community to help dispose of bit-rotten software by programming worker-wrappers. The goal of this contest is to determine, for a number of empty mines, the most efficient way to insulate them.

1.1 Contest Specification Updates

We expect the insulation demand to grow over the course of the contest, hence, the teams should expect some updates during the next 72 hours. Specification refinements will be made available at the following times:

- First update: **June 21, 5:00pm UTC** (7 hours into the contest)
- Second update: **June 22, 0:00am UTC** (14 hours into the contest)
- Third update: **June 22, 10:01am UTC** (immediately after the end of Lightning Division)

The details of the updates will be announced online in the following ways:

- via the contest website: `https://icfpcontest2019.github.io`
- in the contest Twitter account: `@icfpcontest2019`.

[1] `https://icfpcontest2012.wordpress.com`
[2] `https://icfpcontest2017.github.io`

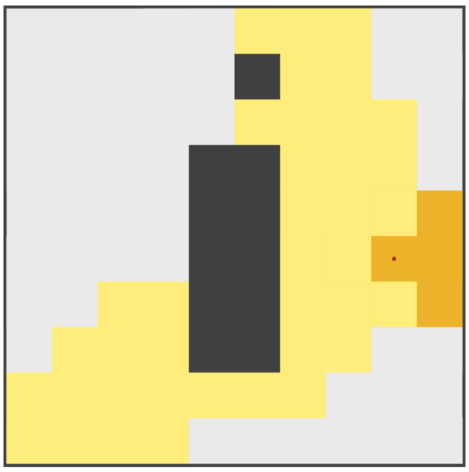


Figure 1: A map with two obstacles and a partial route of a worker-wrapper. The walls and solid obstacles are shown in dark color. The position of the worker-wrapper's body is depicted by a red dot; dark yellow squares show the current reach of its manipulators, while the light yellow squares show the parts of the surface that are already wrapped.

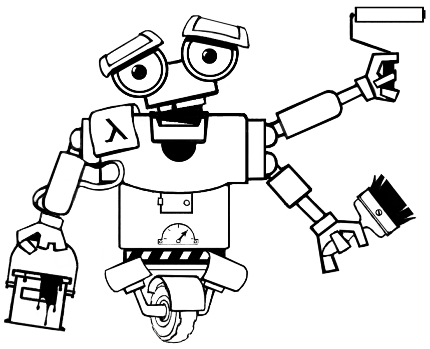
2 Maps and Worker-Wrapper Navigation

We have acquired a number of maps of mines that can be turned into silos for legacy code. Each map represents a single *worker-wrapper task*. The boundary of a map is encoded as a rectilinear polygon, described by the list of its vertices with non-negative integer coordinates. The list of vertices, therefore, defines the boundary of a map that is located "on the left" with respect to the direction induced by traversing the list.

The body of a worker-wrapper occupies a 1×1 square on the surface of the map, and its current position is described as the coordinates of the square's bottom-left corner. The initial position of the worker-wrapper is explicitly specified for each task. A map may contain a number of solid obstacles; each obstacle is encoded similarly to how the map is encoded. The obstacles do not touch each other or the boundary of the map, therefore, the worker-wrapper can reach any point within the mine that is not occupied by an obstacle.

An example of a 10×10 map with two obstacles is shown in Figure 1. The boundary of the map is defined by the list of vertices $[(0,0),(10,0),(10,10),(0,10)]$, while the obstacles are defined by the lists $[(4,2),(6,2),(6,7),(4,7)]$ and $[(5,8),(6,8),(6,9),(5,9)]$, correspondingly.

2.1 Basic navigation and wrapping rules



A standard worker-wrapper model **Wrappy-2019** (shown on the right) is shipped with three universal foldable robotic manipulators that are attached to its body and allow it to perform wrapping. Each robotic manipulator can wrap one 1×1 square of surface within its reach. Once a square is touched by a manipulator or the body of the worker-wrapper, it is considered wrapped. In order to cover the entire surface, at each unit of time the worker-wrapper can perform one of the following *actions*:

- move to an adjacent square of the surface,
- turn 90°, changing the relative position of its manipulators with respect to its body,
- do nothing.

The initial configuration of the manipulators is always the same and is described by the squares with coordinates $[(x+1,y),(x+1,y+1),(x+1,y-1)]$, where (x,y) is the location of the worker-wrapper's body (*cf.* dark yellow are in Figure 1). The manipulators *fold* when a worker-wrapper is next to a wall or an obstacle, thus making it possible to move through the map without damaging them, and unfold instantly when there is enough space to do so.

A *solution* of a task is a sequence of worker-wrapper's actions (moves, turns, *etc*), which, when executed, leaves no non-wrapped squares of surface inside the mine. The fewer units of time the worker-wrapper spends to complete the task, the better the solution.

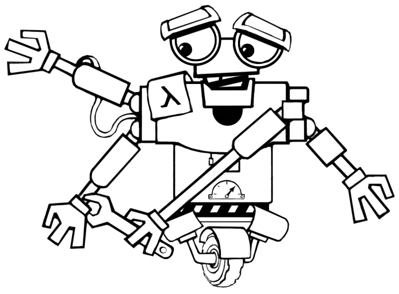
2.2 Boosters

Upon closer examination of the aquired mine maps, we have noticed some locations being marked. As it turned out, in the process of mining lambdas, a number of useful artefacts — so-called *boosters* — were left in the mines at those marked locations, lying on the floor. These boosters can be used to make the wrapping process much more efficient (and also more fun). A worker-wrapper can collect a booster by simply moving into the location containing it. Once collected, the booster can be used just once at any later point of time, with the effects described below. As of now, four kinds of boosters have been discovered:

2.2.1 Extension of the Manipulator

Code: B

Effect: Using this booster allows a worker-wrapper to permanently attach an additional manipulator, extending its wrapping range. A new manipulator can be only attached to a *side* of a worker-wrapper's body or of any of the already attached manipulators. In order to do so, when using the booster, a *relative* location of the new manipulator position (*wrt.* the current position of the worker-wrapper's body) should be provided.

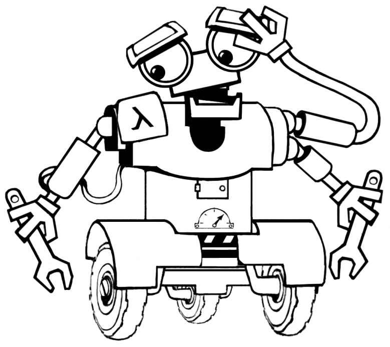


For instance, if the current position of the worker-wrapper is (x, y), and with its manipulators it can reach positions $[(x, y), (x + 1, y), (x + 1, y + 1), (x + 1, y - 1)]$, a new manipulator can be attached at relative positions $(1, 2)$, $(-1, 0)$, $(2, 1)$, *etc*, but *not* at a position $(3, 1)$ or $(0, 0)$. Once a manipulator is attached, *e.g.*, at $(1, 2)$, the worker-wrapper can reach positions $[(x, y), (x+1, y), (x+1, y + 1), (x + 1, y - 1), (x + 1, y + 2)]$, and if another manipulator is then attached at $(-1, 0)$, it will be able reach positions $[(x - 1, y), (x, y), (x + 1, y), (x + 1, y + 1), (x + 1, y - 1), (x + 1, y + 2)]$ The exact format of specifying an extension is provided in Section 3.2.

2.2.2 Fast Wheels

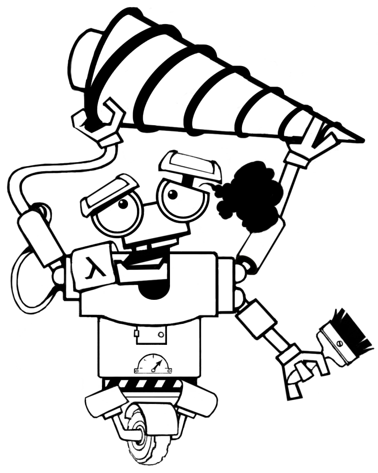
Code: F

Effect: This booster temporarily speeds-up the worker-wrapper, allowing it to make two moves in the same direction (up/down/left/right) in a single unit of time, assuming there is enough room to do so. If the there is a room for only one step (for instance, the worker-wrapper is just one square away from a wall), only one step will be performed. Unfortunately fast wheels also wear off fast, and their effect only lasts for **50 units of time**. Using a new pair of fast wheels while a pair is already active **extends the effect for additional 50 units of time** ~~for up to additional 50 units of time (*i.e.*, for the *maximum* of the remaining durations, not their sum)~~ **without increasing the speed any further**.



2.2.3 Drill

Code: L

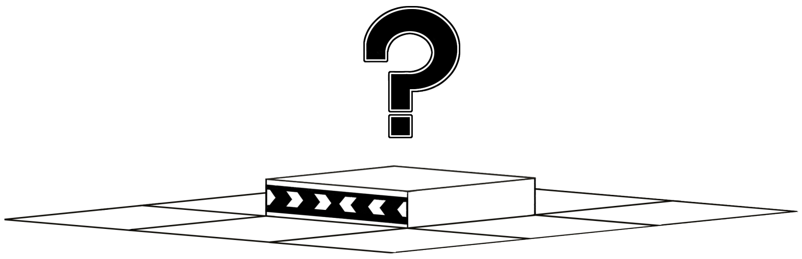


Effect: Using a drill booster allows a worker-wrapper to move through obstacles and even walls of the mine, creating tunnels of 1 square width. However, the drill cannot be used to move beyond the *bounding box* of the map (*i.e.*, to the locations with coordinates that are negative or exceed the maximal X/Y-coordinate of the map's vertices). Once started, the drill remains active for **30 units of time**, after which it wears off. The effect of using a second drill with one already activated follows the same rules as when using a second pair of fast wheels.

The drill booster *can* be used in combination with fast wheels and vice versa.

2.2.4 Mysterious Point

Code: X



Effect: The effect of this booster is currently unknown. Furthermore, it seems that it cannot be collected: the points on the map marked with this code contain some strange devices that are firmly mounted to the rock surface. Perhaps their purpose will become clear later.

2.2.5 Using boosters

Using each booster takes one time unit, during which the worker-wrapper cannot perform another action. Once the booster is used, it is considered spent and cannot be used again.

2.3 Rules of manipulator reachability

Worker-wrapper's manipulators are highly flexible, but have a limited reach, which should be taken into the account when calculating the solution. For instance, obstacles and corners of the mine can block them from reaching certain squares. According to the worker-wrapper specification, a square of surface can only be reached by a manipulator if its centre is *visible* from the centre of the square, corresponding to the location of the worker-wrapper's body. A point is considered visible from another if it can be connected by a straight segment without intersecting the sides of walls and obstacles (touching corners is allowed). Figure 2 shows an example of squares within and outside the reach of a worker-wrapper with an extended set of manipulators.

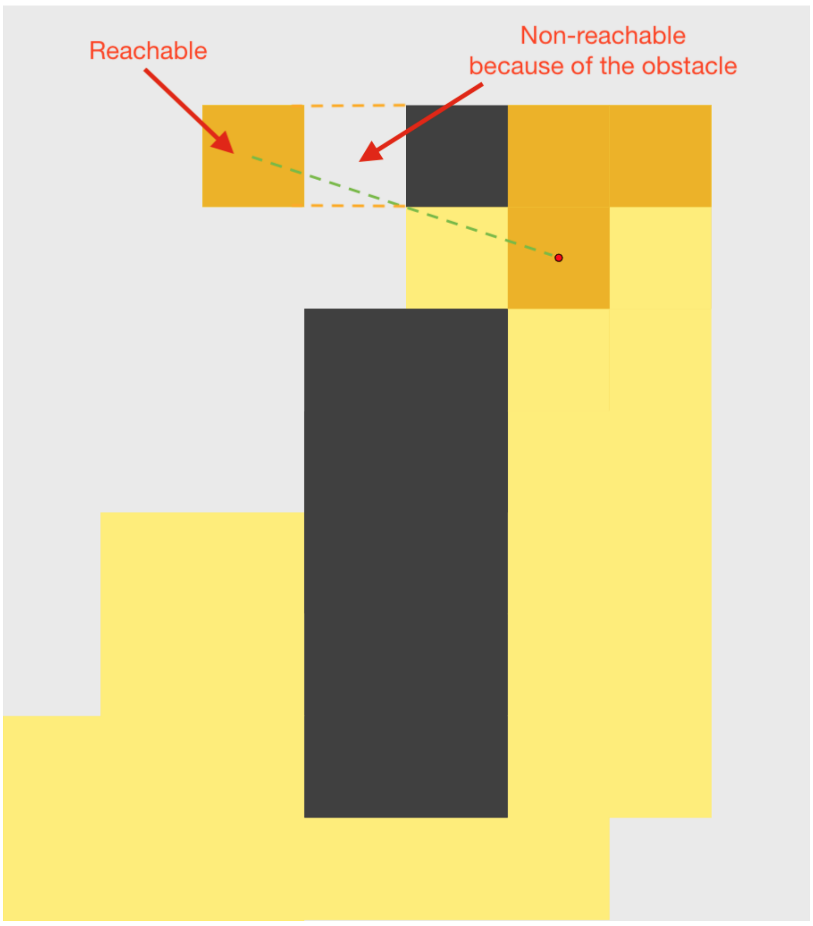


Figure 2: Worker-wrapper's reach is blocked by an obstacle.

3 Rules of the Contest

3.1 Task descriptions

Each worker-wrapper task is described by a single file *prob-NNN.desc* containing a four-tuple with the following components, separated by the character #:

- a contour map of the mine,
- an initial location of the worker-wrapper inside the mine,
- a (possibly empty) list of obstacles separated by semicolons,
- a (possibly empty) list of boosters and their locations separated by semicolons.

Booster kinds are denoted by their code, as defined in Section 2.2. The exact grammar of the task description is given below:

$$\begin{array}{rcl}
x, y & : & \mathrm{Nat} \\
\mathit{point} & ::= & (x, y) \\
\mathit{map} & ::= & \texttt{repSep}\ (\mathit{point}, \text{","}) \\
\text{BoosterCode} & ::= & \mathsf{B} \mid \mathsf{F} \mid \mathsf{L} \mid \mathsf{X} \\
\mathit{boosterLocation} & ::= & \text{BoosterCode}\ \mathit{point} \\
\mathit{obstacles} & ::= & \texttt{repSep}\ (\mathit{map}, \text{";"}) \\
\mathit{boosters} & ::= & \texttt{repSep}\ (\mathit{boosterLocation}, \text{";"}) \\
\mathit{task} & ::= & \mathit{map}\ \#\ \mathit{point}\ \#\ \mathit{obstacles}\ \#\ \mathit{boosters}
\end{array}$$

3.2 Encoding solutions

A solution for a task *prob-NNN.desc* is a sequence of actions encoded as a single-line text file named *prob-NNN.sol* for the corresponding number *NNN*. The actions are encoded as follows:

action	::=	W	(move up)
	|	S	(move down)
	|	A	(move left)
	|	D	(move right)
	|	Z	(do nothing)
	|	E	(turn manipulators 90° clockwise)
	|	Q	(turn manipulators 90° counterclockwise)
	|	B(*dx*, *dy*)	(attach a new manipulator with relative coordinates (*dx*, *dy*))
	|	F	(attach fast wheels)
	|	L	(start using a drill)
solution	::=	`rep` (*action*)	

A solution is *valid*, if it does not force the worker-wrapper to go through the walls and obstacles (unless it uses a drill), respects the rules of using boosters, and, upon finishing, leaves all reachable squares of the map wrapped.

3.3 Examples

An example task with valid solutions is available at

https://icfpcontest2019.github.io/download/part-1-examples.zip

A JavaScript checker and a visualiser are available for testing individual solutions:

Visualiser:	https://icfpcontest2019.github.io/solution_visualiser/
Checker:	https://icfpcontest2019.github.io/solution_checker/

3.4 Registration and Submission

The initial pack with 150 tasks to solve can be downloaded at

https://icfpcontest2019.github.io/download/part-1-initial.zip

To participate in the ranking, one needs to register the contest team to obtain a team-specific private identifier at:

https://icfpcontest2019.github.io/register/

Please write down your private ID. You will be using it throughout the contest.

Submissions should be archived as a single *.zip* file containing exactly the files *prob-NNN.sol* for the corresponding maps from the task archive (some solutions can be omitted). Submit your solution archive at:

https://icfpcontest2019.github.io/submit/

or via the command line, replacing the private ID and file path appropriately:

```
curl -F "private_id=34fbde" -F "file=@solutions.zip" https://monadic-lab.org/submit
```

If you are submitting via cURL, make sure there is no trailing slash after the submission URL.

Each team may only submit their solution once every 10 minutes.

To be considered for prizes, within *two hours of the end of the contest*, teams must update their profile with complete team information and submit a single *.zip* archive with their source code, a *README.txt* file (brief directions for judges to build/run the solution; description of the solution approach; feedback about the contest; self-nomination for judges' prize; *etc*), and any other supporting materials. This can be done at:

`https://icfpcontest2019.github.io/profile/`

3.5 Scoring

The team's score $score_{team,T}$ for a task T, for which a valid solution is provided, is computed as

$$score_{team,T} \triangleq \left\lceil 1000 \times \log_2 (X_T \times Y_T) \times \frac{t_{best}}{t_{team}} \right\rceil,$$

where X_T and Y_T are the maximal x/y-dimensions of T's map, t_{team} is a number of time units taken by the team's solution, and t_{best} is the minimum wrapping time among all teams' corresponding submitted solutions. Tasks for which a solution is not provided or is not valid yield a score 0.

3.6 Lightning Division

As traditional, the contest will have a Lightning Division spanning the first 24 hours. To be nominated for the Lightning Division prize, submit your solutions by **June 22, 2019, 10:00am UTC**.

3.7 Determining the Winner

We will use the same procedure to determine the winner in both the lightning and full divisions, ranking the teams by cumulative score, computed as the sum of scores for each task.

3.8 Live Standings

The Live Standings for the contest will be available at

`https://icfpcontest2019.github.io/rankings/`

The results of the live standings will be frozen one hour prior to the end of the Lightning Division. They will resume being updated starting **June 22, 2019, 1:00pm UTC**, and will be frozen again three hours prior the end of the contest.

3.9 The Judges' Prize

The judges' prize will be picked by the contest organisers. All entries in both the full and lightning divisions are eligible for the judges' prize.

Good luck, and happy wrapping!