CS4402 Constraint Programming
Practical 1: Human Resource Machine

School of Computer Science
University of St Andrews

2020-21

Summary

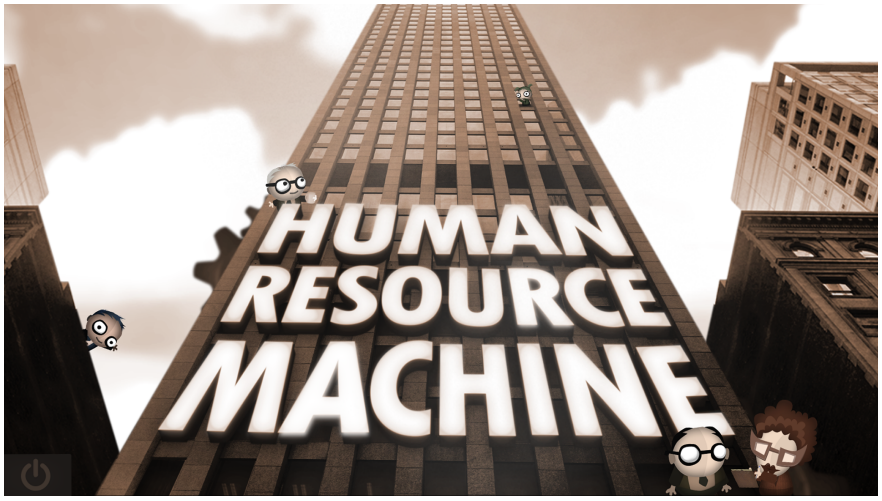
This Practical comprises 50% of the coursework component of CS4402. It is due on Tuesday 27th October at 21:00 (NB MMS is the definitive source for deadlines and weights). The deliverables consist of:

- A report, the contents of which are specified in Section 6 below.
- The Essence Prime constraint model for the basic specification.
- The Essence Prime constraint model(s), and possibly additional instances, for the extension part.

The practical will be marked following the standard mark descriptors as given in the Student Handbook (see link below).

Please read this document carefully. There are a number of requirements for the problem being modelled, and the way in which your solution and experiments should be reported.

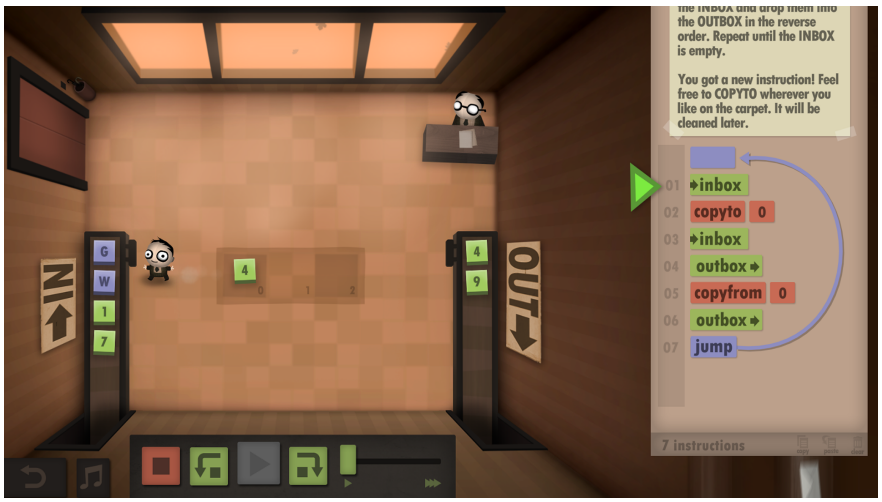
1 Background



This practical involves in modelling a puzzle game, **Human Resource Machine** (The Tomorrow Corporation, 2015). In each level, the player is presented with a task by their boss and the goal is to automate that task by programming an office worker. If successful, the player is promoted to the next level. If you are not familiar with this game, it would be a good idea to watch a playthrough video, such as this one:

https://youtu.be/cxg-Vu9_mRM

The office is, in fact, a simple computer. It has an inbox and an outbox (inputs and outputs), and a few slots on the floor to store items for later (registers/memory). The office worker can hold exactly one box in their hands at a time (accumulator). Boxes (data) display letters or numbers.



In each level the boss gives the player a task, like "Take everything from the INBOX, and put it in the OUTBOX!". The objective is to automate the task by programming the office worker with simple commands.

The player starts the game with just two commands, and gradually earns more as they are promoted. We will model nearly all commands the game has:

- `INBOX`: Go to the IN conveyor belt and pick up the box at the top. The box currently held in the worker's hands (if any) is discarded.
- `OUTBOX`: Go to the OUT conveyor belt and drop whatever box is in the worker's hands. This instruction *cannot* be performed when the worker is not holding anything.
- `COPYFROM`: This instruction is accompanied by a number. It tells the worker to go to the slot on the floor labelled with the specified number and copy the value in the slot into their hands. This instruction *cannot* be performed on an empty slot.
- `COPYTO`: Similar to the `COPYFROM` command, but instead places a copy of the value in the worker's hands into the specified slot. This instruction *cannot be* performed if the worker's hands are empty.
- `ADD`: This instruction is accompanied by a number. It tells the worker to go to the slot on the floor labelled with the specified number and then add the number in their

hands to the value in the slot. The sum will appear in their hands. This instruction *cannot* be performed if the worker's hands or the floor slot is empty.

- `SUB`: Similar to the `ADD` command, but subtracts the value in the slot from the value in the worker's hands. The difference appears in their hands. This instruction *cannot* be performed if the worker's hands or the floor slot is empty.
- `BUMP+`: This instruction is always accompanied by a number. It commands the worker to go to the slot indicated by the number, and add one to the value it contains. The resulting value replaces whatever the worker was holding. This instruction *cannot* be performed on an empty slot.
- `BUMP-`: As BUMP+, except that it subtracts one from the value in the slot.

For simplicity we do *not* incorporate the `JUMP` instruction in our model, since it adds significant complication.

2 Example

Consider a very simple task of increasing all input values by one and outputting them in order:

$$\texttt{input} = [1, 2]$$
$$\texttt{output} = [2, 3]$$

Following the task description, our first approach would be to repeatedly take a value from the input, copy it to a register slot, bump it and finally output it. A solution follows straightforwardly, requiring only one register and 8 steps:

```
INBOX, COPYTO 1, BUMP+ 1, OUTBOX,
INBOX, COPYTO 1, BUMP+ 1, OUTBOX
```

Note that the `BUMP+` instruction puts the incremented value in the worker's hands, so we do not need to perform a `COPYFROM`. However, we can do better.

The previous solution uses all values in the input, but we don't have this requirement. Reading the instructions carefully, note that the value in the register slot is not erased when using `BUMP+`, so we could reuse the value stored in the first register and shorten the solution to 6 steps:

```
INBOX, COPYTO 1,
BUMP+ 1, OUTBOX,
BUMP+ 1, OUTBOX
```

3 Modelling Human Resource Machine

The goal of this practical is to create a model that, given an input and expected output, can find a sequence of operators that solves the puzzle.

3.1 Parameter Format and Instances

For simplicity we will standardise the format of the parameter file, for which we shall use the identifiers `input` and `output`, together with some helpers:

```
language ESSENCE' 1.0
letting input=[1,2]
letting output=[2,3]
letting MAX_OPS=7
letting num_registers=1
```

where:

- `MAX_OPS` defines a bound on the maximum number of actions permitted. Necessary because the number of steps required to solve the problem is not known beforehand.
- `num_registers` describes the number of available floor slots (or registers) to the program.

As a hint, you will be able to use the following in the model:

```
given MAX_OPS: int(1..)
given num_registers : int(0..)
given input: matrix indexed by [int(1..input_length)] of int(0..100)
given output: matrix indexed by [int(1..output_length)] of int(0..100)
```

The identifiers `MAX_OPS` and `num_registers` will be positive integers. The other four identifiers: `input`, `output`, `input_length` and `output_length` will refer to the input and output sequences, and their respective lengths. As you can see, we are considering a limit of 100 in the domain of the values.

3.2 Approach to Modelling

Human Resource Machine is a **planning problem**, as discussed in the lectures and tutorials. The task in a planning problem is to find a sequence of actions to transform an initial state into a goal state. You should begin by reviewing your lecture notes on this topic and looking at the models for some other planning problems in the Savile Row examples directory, such as Sokoban or Peg Solitaire.

Planning problems are typically modelled as a sequence of states, where each element in the sequence represents the state of the world (i.e. the values in the registers and accumulator in this case). In addition, there is often a representation of an action linking two contiguous states. Think about the decision variables you will need to represent these states and actions, and the constraints necessary to connect them. As per the specification, there are some conditions preventing certain instructions being performed when a register or the accumulator is empty. These will need to be modelled carefully.

General advice: debugging constraint models can be difficult because a conflicting set of constraints will cause the solver to return no solution without there being an obvious cause. For this reason, you are encouraged to build up the set of variables and constraints in your model incrementally, continually testing them on small instances.

4 Empirical Evaluation

This section describes how you are to perform the empirical evaluation of your model. Please read and follow it carefully.

4.1 Supplied Instances and Instance Generator

Along with this specification you should find twenty instances on which to perform your experiments. These are separated into two directories, one with sixteen satisfiable instances, another with four unsatisfiable instances.

You will also find supplied an instance generator written in Java. It is run as follows:

```
java generator <in_length> <out_length> <seed>
```

and will produce a .param file of the format described above. The generator includes a rudimentary heuristic to estimate an upper bound on the number of actions needed.

4.2 Performing Your Empirical Evaluation

Your empirical work should be performed, analysed, and documented in your report as follows.

- Evaluate the performance of your model using the set of instances given, according to the instructions below. In doing so, use the default settings of Savile Row:

  ```
  ./savilerow hrm.eprime X.param -run-solver
  ```

  Where `X.param` is a placeholder for the concrete instance.

- For each instance, give the solver **10 minutes** to complete the search. Depending on your model, you may find that search is not completed for some of the instances provided. Record this in your report.

- For the instances where search is completed within the 10-minute budget, record the `SolverNodes`, `SolverSolveTime` and `Savile Row TotalTime`. As discussed in the tutorials, this information is available in the *.info* file after your model has been run, irrespective of whether the instance has a solution or not. Present it in a **table** in your report.

- For the instances with a solution, report the number of operators used in the solution and the sequence of operators found.

- Analyse your results:

  – Explain what you think the factors are that contribute to the difficulty of solving an instance of this problem with your model.

  - Represent graphically the solving time, search nodes and the factors that you think are relevant. Is there a correlation?

- Temporarily remove both `BUMP` operators from the model. Explain **why and how** the `SolverNodes`, `SolverSolveTime`, `Savile Row TotalTime` and the solution change on instance 5.param.

- Use the generator to explore the lengths to which your model will scale within a time budget of 10 minutes. Remember that not all instances of the same length are equally difficult, so consider using different seeds for each length combination you consider.

  - Record the results of the instances you generate and solve in your report.
  - Do both parameters contribute equally to hardness? If you think they do not contribute equally, explain why and which one you think contributes more.

5 Extensions

Already an expert?

Each room comes with optimization challenges that test how well your solution optimises for size and execution speed. Do a good job, and have fun! Management is watching...

– The boss

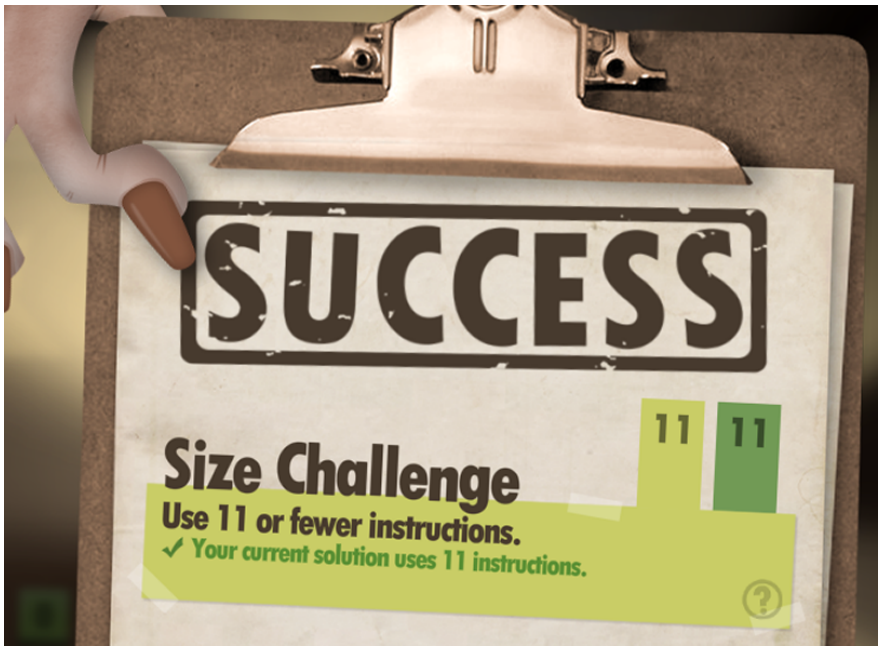
There follow some ideas for extension activities. These are not required, but attempting at least one extension activity is required to gain a mark above 17.

5.1 Extension: Solver variety

Select or generate a set of non-trivial instances and explore how the preprocessing options for Savile Row perform empirically (SR manual Section 2.5.4) and the different solver backends (SR manual Section 1.2).

Report which solvers or combinations of parameters work better, and which instances you have been able to solve that you were not able to before. Explain why you think this happens.

5.2 Extension: Optimisation



This extension deals with extending your model to minimise the number of operators that it uses.

- Explain how you approached the optimisation, and what extra elements you used.
- Re-run the provided instances and report:
  - the differences between the number of actions used with and without optimisation.
  - the `SolverNodes`, `SolverSolveTime` and `Savile Row TotalTime` differences with and without optimisation.

5.3 Extension: Getting Creative!

In this extension your task is to create new operators and see how they affect the solutions and the search.

- Explain the semantics of your new operators and their operands.
- Re-run the experiments and report the notable differences you see.
- Supported by the data you have gathered, give your opinion on the effect generally of adding new operators.

6 Report

Your report should have the following sections:

- Variables and Domains: Describe how you chose the variables and domains to represent the problem.
- Constraints: Describe each of the (sets of) constraints you have added to your model, and their purpose.
- Example Solutions: Demonstrate that your model is correct by showing the solution it produces for the following supplied instances: 3.param and 5.param. A similar style used in the examples given above is fine for this purpose.
- Empirical Evaluation: Describe your experimental setup, and record and analyse the output of the empirical evaluation described above.
- Extensions: Describe here the extensions you have attempted, if any. Models and empirical work should be reported as above, and compared with your original model.
- Conclusion: Explain what were the main bottlenecks and your experience during the modelling and evaluation parts.

7 Marking

The practical will be marked following the standard mark descriptors as given in the Student Handbook (see link below). There follows further guidance as to what is expected:

- To achieve a mark of 7 or higher: A rudimentary attempt at a model, incomplete or with several errors. Adequate evaluation and report.
- To achieve a mark of 11 or higher: A reasonable attempt at a model, mostly complete or complete with few flaws. Reasonably well evaluated and reported.
- To achieve a mark of 14 or higher: A good attempt at a model with only minor flaws, well evaluated and reported.
- To achieve a mark of 17: A fully correct model, very well evaluated and reported.
- To achieve a mark greater than 17: In addition to the requirements for a mark of 17, an attempt at the suggested extension activities.

8 Pointers

Consider these when doing the practical and deciding to submit it:

- **Mark Descriptors**: https://info.cs.st-andrews.ac.uk/student-handbook/learning-teaching/feedback.html
- **Lateness**: https://info.cs.st-andrews.ac.uk/student-handbook/learning-teaching/assessment.html
- **Good Academic Practise**: https://info.cs.st-andrews.ac.uk/student-handbook/academic/gap.html