Reverse engineering a Turing machine

Robin Visser

University of Amsterdam

Recap: Turing machine (definition)

Definition

A Turing machine is a 9-tuple $M = (Q, \Sigma, \Gamma, \vdash, \sqcup, \delta, s, t, r)$ where;

- Q is a finite set (the states)
- Σ is a finite set (the input alphabet)
- Γ is a finite set (the tape alphabet) containing Σ as a subset
- $\sqcup \in \Gamma - \Sigma$, the blank symbol
- $\vdash \in \Gamma - \Sigma$, the left endmarker
- $\delta : Q \times \Gamma \to Q \times \Gamma \times \{L, R\}$, the transition function
- $s \in Q$, the start state
- $t \in Q$, the accept state
- $r \in Q$, the reject state, $r \neq t$

Recap: Turing machine (step)

Transition function

$\delta : Q \times \Gamma \to Q \times \Gamma \times \{L, R\}$

TM *Step*:

- Read from the current cell
- Change state
- Write to the current cell
- Move (either left or right)

Recap: Turing machine (execution trace)

Proposition

Execution trace as a list of TM steps

Definition: Turing machine execution trace

A space-separated ordered list containing a finite amount of event representations.
Possible event representations are:

event representation	**event description**
$<$	Move head one cell to the left
$>$	Move head one cell to the right
$-$	Read (next event representation is input)
$+$	Write (next event representation is output)
<symbol literal>	A literal representation of a symbol $\in \Gamma$

Recap: Turing machine (execution trace, example)

Example

Consider a 4-state TM which erases its input "abaa":

event repr.	event descr.
$<$	Move left
$>$	Move right
$-$	Read
$+$	Write
$<$symbol literal$>$	symbol $\in \Gamma$

$- \vdash + \vdash >$

$- a + \sqcup >$

$- b + \sqcup >$

$- a + \sqcup >$

$- a + \sqcup >$

$- \sqcup + \sqcup >$

Note: Newlines added for readability!

Actual execution trace

$- \vdash + \vdash > - a + \sqcup > - b + \sqcup > - a + \sqcup > - a + \sqcup > -$ .....

Dissecting the execution trace: input extraction

- The input is the only parameter that determines what a certain TM is going to do, and it is embedded in the resulting execution trace.

Input extraction algorithm

- Emulate the TM using the execution trace
- Keep track of position
- Whenever the TM lands on a cell for the first time, store contents.

$\rightarrow$ **Any input read by the TM is extracted by the algorithm!**

Dissecting the execution trace: Input extraction

An execution trace may not contain the entire input

Example

Consider a TM that attempts to find a '1' in bitstrings

event repr.	**event descr.**
$<$	Move left
$>$	Move right
$-$	Read
$+$	Write
$<$symbol literal$>$	symbol $\in \Gamma$

$$
\begin{array}{c}
- \vdash + \vdash > \\
- 0 + 0 > \\
- 0 + 0 > \\
- 1 + 1 >
\end{array}
$$

Input could have been "001", "0010" or "001110010"

Dissecting the execution trace: output extraction

Definition

The output of a TM is defined as the longest possible finite string after the left endmarker whose last symbol is not '$\sqcup$' (the output could be empty)

Output extraction algorithm (direct approach)

- Emulate all the write operations of the TM on an empty tape

Output extraction algorithm (efficient approach)

- Start at the back of the execution trace
- Reverse emulate the TM
- Store only the last write operation on a given cell

Dissecting the execution trace: input/output completeness

- The extracted input is not necessarily complete
- The extracted output is not necessarily complete

Theorem (Input/Output Completeness)

An execution trace's input is complete if and only if the execution trace's output is complete

Assignment

Assume TM uses it's entire input $\rightarrow$ all traces are complete.

Reverse engineering: manually

Concept

Recreating a *functional* TM from multiple coherent execution traces.

Approach

1. Try to determine the design goal.
   (by using input/output extraction).
2. Try to understand the algorithm used to achieve this goal.
   (by visualizing trace execution step by step)
3. Mirror the algorithm step by step in a new TM.
   (Creating states/transitions where necessary)
4. Verify that the new TM produces the exact same execution traces.

Result

(Relatively) efficient TM with acceptable amount of states.

Reverse engineering: generic

Concept

Recreating *any* TM from multiple coherent execution traces.

Algorithm

1. Extract Sigma and Gamma.
2. Reverse engineer a single trace.
3. Reverse engineer another trace and combine the results.
4. Repeat step 3 until there are no traces left.
5. Verify that the new TM produces the exact same execution traces.

Result

(Relatively) inefficient TM with a lot of states.