- Start as early as possible, and contact the instructor if you get stuck.
- See the course outline for details about the course's marking policy and rules on collaboration.
- Submit your completed solutions to **Crowdmark**.

1. A queue automaton

[8] Let Σ be an alphabet, which is the alphabet for the languages of all the machines in this problem. A **queue automaton**, A, is like a pushdown automaton which accepts by final state, except that the stack is replaced by a queue. A **queue** is a tape allowing symbols to be written only to the left-hand end and read only from the right-hand end. Each write operation (we will call it a **push**) adds zero or more symbols to the left-hand end of the queue and each read operation (we will call it a **pull**) reads and removes one symbol from the right-hand end. As with a PDA, the input is placed on a separate read-only input tape, and the head on the input tape can move only from left to right. The input tape contains a cell with a blank symbol following the input, so that the end of the input can be detected. The queue is empty at the start of execution. Analogously to the case of a PDA, the transition function for a queue automaton is a function of the current state, the current input symbol (an alphabet symbol or ε) and the symbol currently being pulled from the queue. Each transition determines a new state, and pushes zero or more symbols onto the queue. In detail, if the states of A are Q, the alphabet for A is Σ and the queue alphabet of A is Γ, then the transition function, δ_A, for A is a function

$$\delta_A : Q \times (\Sigma \cup \{\varepsilon\}) \times \Gamma \to Q \times \Gamma^*.$$

A queue automaton accepts its input by entering a final state at any time.
Prove that every recursively enumerable language can be recognized by a queue automaton.

Copyright ©2023

Proof. To prove that the **queue automaton**, A (defined above), can accept any recursively enumerable language we will show that it can simulate an arbitrary Turing machine, T (defined below). As recursively enumerable languages are defined as languages recognized by a Turing machine, by showing A can simulate T and then that T can simulate A we garantee A can accept any recursively enumerable language.

- Define arbitrary Turing machine, T.
  - Let T' be an arbitrary **multi-tape** Turing machine with two tapes and two tape heads. We define a multi-tape machine for convenience since we know from **Theorem 21.1.1** that, *"if a language is decidable by a multi-tape Turing machine, then it is decidable by a one-tape Turing machine."*. As T' is arbitrary we define the usual ingredients where,

$$T' = (\Sigma', Q', F', q_0', \Gamma', B', \delta')$$

  - Next, we define the transition fuction δ for our multi-tape machine, where,

$$\delta : Q \times \Gamma^2 \to Q \times \Gamma^2 \times \{L, R\}^2$$

  - Finally, we define T as an arbitrary Turing machine with a single tape such that T accepts any language accepted by T' where,

$$T = (\Sigma, Q, F, q_0, \Gamma, B, \delta)$$

    with a transition function δ defined as:

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

- Prove equivalence between A and T.
  - **Simulate the queue automaton, A with the Turing machine, T'**
    * To set up our Turing machine, T' we begin with the input to A on the first tape. The first tape head will point at the first symbol of the input. The first tape will directly simulate the input tape described for A by reading one character and moving to the right at each step.
    * The second tape will be used to simulate the queue in A and will initially contain all blank, B, symbols. The second tape head will begin at an arbitrary position.
    * In order to effectively simulate the queue in A, the functions **push** and **pull** are simulated with the following algorithm:

    push To simulate the **push** operation from A, T will move the second tape head to the left, stopping at the first B symbol encountered. Then, the symbol being pushed in A is written to the second tape in T' replacing that B symbol.

    pull To simulate the **pull** operation from A, T' will move the second tape head to the right to the rightmost character on the tape before any B symbols. This can be achieved relatively easily by moving right to the first B symbol then moving one symbol to the left. Once read the rightmost symbol can be replaced by a B to complete the operation.

Copyright ©2023

    - As any operation in A can be carried out in T' we conclude that any arbitrary queue automaton A can be simulated on T'.

- **Simulate Turing machine, T, with the queue automaton, A.**
    - We will now use A to simulate an arbitrary automaton T defined above.
    - To simulate an arbitrary Turing machine we will use the queue in A to simulate the tape in T.
    - To achieve this two unique symbols ϕ and Δ are added to A's stack alphabet
    - We define ϕ as the symbol denoting the end of written symbols on T's tape. We will use this to recognize when the queue in A has stepped through each symbol on the tape.
    - The key idea is that the queue will be used as a circular representation of the tape whereby the right edge of the non-blank portion T's tape will end at the left character of ϕ and the left edge of the non-blank will begin at the character right of ϕ.
    - Next we define Δ as the symbol which will delimit the current symbol on the tape head.
    - To set up the queue in A to simulate the tape in T we begin by pushing Δ to the queue followed by each symbol of the input string, w and finally ϕ to denote the end of the tape.
    - At this point the queue will look like ϕ, w^R, Δ, since w was input on the left of the queue one character at a time and the queue pulls from the right while in the queue w appears to be in reverse order.
    - This is important because, as we will see, a move of the tape head right or left in T, whose tape is read left-to-right will be reflected in A whose queue will read right-to-left.
    - To simulate T there are four operations we must account for,
      1. When a symbol is read/written and the tape head is moved to the right:
         - This describes the transtion from T for $\delta(q, \alpha) = \delta(p, \beta, R)$ where $q, p \in Q$ and $\alpha, \beta \in \Gamma$
         - Follow these steps in A:
         - Pull and then push each character on the queue until a Δ is pulled.
         - Do not push the Δ symbol, instead pull the next symbol, α, from the queue.
         - Following the transition rules of T derive and Push β followed by Δ to the queue.
         - Δ now precedes the character to the right of β on T's tape.
      2. When a symbol is read/written and the tape head is moved to the left:
         - This describes the transtion from T for $\delta(q, \alpha_3) = \delta(p, \beta, L)$ where $q, p \in Q$ and $\alpha, \beta \in \Gamma$
         - Follow these steps in A:
         - Pull a symbol, α_1 from the queue and remember it in A's finite memory
         - This can be accomplished with parallel state paths using one path for each symbol in T's alphabet.

Copyright ©2023

- Pull another symbol, α_2 from the queue:
- If $\alpha_2 = \Delta$, pull a third symbol α_3 from the queue.
- Then, following the transition rule in T for α_3 where $\delta(q, \alpha_3) = \delta(p, \beta, L)$
- Push Δ then α_1 then β to the queue.
- Otherwise, if $\alpha_2 \neq \Delta$ push α_1 back to the queue and repeat the above algorithm remembering α_2 in A's finite memory as another symbol is pulled and checked for being Δ.
- Keep repeating until Δ is found.
- Eventually Δ will be found and the resulting order we pushed the symbols back to the queue will complete the transition function and the Δ symbol will preceed the character immediately to the left of β on T's tape.

3. When a symbol is written to the blank immediately to the left of the non-blank portion of the tape:
   - Pull then push each symbol on and off the queue until ϕ is found.
   - Push ϕ to the queue followed by the desired symbol to be written to T's tape.
   - As this symbol will now appear to the left of ϕ (our tape delimiter) it will effectively be the first symbol on the tape (left-most symbol) when we cycle through our queue.
4. When a symbol is written to the blank immediately to the right of the non-blank portion of the tape:
   - Pull then push each symbol on and off the queue until ϕ is found.
   - Push the desired symbol onto the queue followed by ϕ
   - As the symbol will now appear to the right of ϕ on the queue it will be the last symbol pulled before the tape delimiter which will simulate its placement at the end of the written portion (right-most) side of the tape in T at each cycle through the queue.

- <u>A can accept any recursively enumerable language</u>
  - We have simulated all operations of an arbitrary Turing machine T with our PDA A and simulated all operations of our PDA A with an arbitrary Turing machine T'.
  - By definition the class of recursively enumerable languages are those languages which can be recognised by a Turing machine.
  - Thus, creating an automaton which can simulate a Turing machine we guarantee that the machine accepts any language in the class of recursively enumerable languages.
  - Then, by showing this automaton can be simulated by a Turing machine we complete the equivalence
  - Since an arbitrary A can be simulated by a Turing machine and, further, an arbitrary Turing machine can be simluated by A we conclude that A can accept any language in the class of recursively enumerable languages as it is functionally equivalent to an arbitrary Turing machine.

$\square$

Copyright ©2023

2. Reductions

   Let $\Sigma = \{0, 1\}$.

[4] (a) Let L be a language over Σ such that $L \neq \emptyset$ and $L \neq \Sigma^*$. Let L_R be **any** recursive language over Σ. Prove that membership in L_R can be reduced to membership in L.

Proof. To prove that L_R reduces to L we describe an algorithm to use L_R as a decider for membership in L

- Let P_1 be the decision problem for membership in L_R,
- Let P_2 be the decision problem for membership in L.
- Define a TM M for L_R; since L_R is recursive, such a TM exists and will halt and accept or halt and reject on every finite input.
- Construct a new TM, M', which will run M
- In M', on any input use an epsilon transition from the initial state to test if the input given is empty
- For empty input reject and don't run M ($L \neq \emptyset$)
- For non-empty input run input on M
  - If M accepts then M' accepts then the input must be finite so $L \neq \Sigma^*$
  - If M rejects then M' accepts since again, the input must be finite so $L \neq \Sigma^*$
  - If M runs forever the input must be infinite since M will give an answer for any finite input as it represents a recursive language thus, M' runs forever and does not accept.
- Therefore, the reduction algorithm correctly maps instances of P_1 to instances of P_2 such that
- "yes" instances of P_1 are transformed into "yes" instances of P_2
- "no" instances of P_1 are transformed into "no" instances of P_2.
- This shows how we can use any recursive language L_R as a decider for membership in L

□

Copyright ©2023

[4] (b) Let L_{RE} be a recursively enumerable language over Σ. Let L_u be the **universal language** as defined in the lecture slides. (In detail, L_u is the set of pairs (e, w) such e is the identifier of a Turing machine, M, which accepts the input word w.) Prove that membership in L_{RE} can be reduced to membership in L_u.

Proof. We want to prove that membership in the recursively enumerable language L_{RE} can be reduced to membership in the universal language L_u.

- Let P_1 be the decision problem for membership in L_{RE},
- Let P_2 be the decision problem for membership in L_u.
- Recall: L_u is the language consisting of pairs (e, w) where e is the identifier of a Turing machine M, and M accepts the input word w.
- To prove L_{RE} reduces to L_u, we use the following algorithm to construct a Turing machine M' that performs the reduction.
  - Given an instance (e, w) for P_2, where e is a valid Turing machine identifier, and given $x \in \Sigma^*$ as an arbitrary input to M', M' behaves as follows:
    i. Ignore the input x.
    ii. Simulate the Turing machine M on input w.
    iii. If M accepts w, accept the input x (i.e., M' enters an accepting state).
    iv. If M rejects w, reject the input x (i.e., M' enters a rejecting state).
    v. If M runs forever on w, M' also runs forever on any input x.
- We consider M' then,
  - If $(e, w) \in L_u$ (i.e., M accepts w), then M' accepts any input x.
  - If $(e, w) \notin L_u$ (i.e., M rejects w or runs forever on w), then M' rejects any input x.
- Therefore, the reduction algorithm correctly maps instances of P_2 to instances of P_1 such that
- "yes" instances of P_1 (i.e., $w \in L_{RE}$) are transformed into "yes" instances of P_1 (i.e., $(e, w) \in L_u$)
- "no" instances of P_1 (i.e., $w \notin L_{RE}$) are transformed into "no" instances of P_2 (i.e., $(e, w) \notin L_u$).
- Since M' performs the reduction correctly, we have shown that membership in L_{RE} can be reduced to membership in L_u.

□

Copyright ©2023

3. An undecidable language

Let $\Sigma = \{0, 1\}$.

[4] (a) Give an explicit reduction from membership in the language $L_{\varepsilon+} = \{M \mid \varepsilon \in L(M)\}$ to membership in the language $L_{\varepsilon} = \{M \mid \{\varepsilon\} = L(M)\}$.

Proof. To prove that L_ε reduces to $L_{\varepsilon+}$ we give an explicit reduction.

- Let P_1 be the decision of L_ε-membership.
- Let P_2 be the decision of $L_{\varepsilon+}$-membership.
- Then, the following algorithm A reduces P_1 to P_2:
  - Let M be an arbitrary Turing machine such that $L(M) = L_{\varepsilon+}$
  - Construct a new Turing machine M' with the following properties:
    - M' is a non-deterministic Turing machine
    - Let $x \in \Sigma^*$ be an arbitrary input to M'
    - From the initial state in M' we use an ε-transition to branch to two states,
      i. The first branch will reject all input.
      ii. The second branch will follow an episilon transition and run machine M on the input x as described below
    - If $x = \varepsilon$ M' will run M with x as M's input and reject otherwise
    - If M accepts M' will accept.
    - If M rejects then M' will reject.
  - Now the following three cases need be considered.
    i. If M accepts then M' will accept ε and nothing else.
    ii. If M rejects then M' will accept no input
    iii. If M runs forever on input x then M' will run forever on input x
- Then the language $L(M')$ will be one of the following two things:

$$L(M') = \{\{\varepsilon\}, \text{ if } M \text{ accepts } x \\ \emptyset, \text{ if } M \text{ rejects } x\}$$

- Take M' as our corresponding instance for P_2 (L_ε-membership.)
- Then,
  - "yes" instances for P_1 map to "yes" instances for P_2
  - "no" instances for P_1 map to "no" instances for P_2
- Thus, A is a correct reduction from P_1 to P_2

□

Copyright ©2023

[4] (b) Prove that membership in the language $L_{\varepsilon+}$ from part 3a is undecidable. Do **not** use Rice's theorem.

Proof. We will prove that $L_{\varepsilon+}$ is undecidable by contradiction. Assuming $L_{\varepsilon+}$ is decidable we will show it can be used to solve a known to be undecidable problem, the halting problem.

- The Halting Problem:
  - Define $H(M)$ for TM M to be the set of inputs w such that M halts given input w, regardless of whether or not M accepts w.

$$\text{The Halting problem} = H(M) = \{(M,w)|w \in H(M)\}$$

- Assuming $L_{\varepsilon+}$ is decidable then there exists a TM, $M_{\varepsilon+}$, which, when given the input N where N is an arbitrary TM, will decide if $\varepsilon \in L(N)$.
- Now we will use $M_{\varepsilon+}$ to solve the halting problem above.
  i. Let (M,w) be an arbitrary instance of the decision problem expressed by $H(M)$
  ii. Construct a new TM, M' which takes an arbitrary $x \in \Sigma^*$ as input.
  iii. Temporarily ignore the input x
  iv. Inside M' run the universal Turing machine U, described in lecture, with the input (e,w) where e is a TM id for machine M and w is its input.
  v. Use the result from U as input for $M_{\varepsilon+}$
  vi. If $M_{\varepsilon+}$ returns true we know that M halts on w
  vii. Conversely, if $M_{\varepsilon+}$ returns false we know M does not halt on w
  viii. Thus, we have solved the halting problem using $M_{\varepsilon+}$ as a decider
- Since we know the halting problem is undecidable but can be solved with our algorithm we have reached a contradiction.
- Therefore, $L_{\varepsilon+}$ is undecidable.

□

Copyright ©2023