دانشگاه صنعتی شریف

دانشکده‌ی مهندسی کامپیوتر

درس نظریه‌ی زبان‌ها و ماشین‌ها

سوالات نمونه

پاسخ‌نامه‌ی مجموعه‌ی ۱۱ : زبان‌های تورینگ-تشخیص‌پذیر

استاد: دکتر علی موقر

تیم دستیاران درس - نیم‌سال دوم ۰۲ - ۰۱

۴ خرداد ۱۴۰۲

1 Turing Machines and Turing Recognizable Languages

1.1

We cross the first a, go to the first b and cross it, go back to first a and repeat, until all as are crossed. Then check that rest of the string (that is not crossed yet) contains only bs and if so, accept the string.

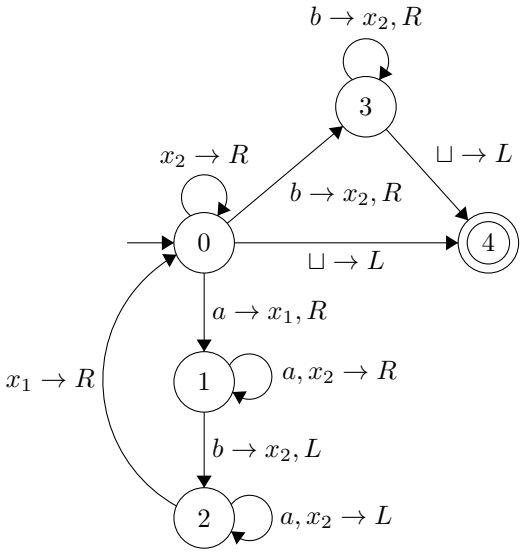


Figure 1: Turing machine for the language $L = \{a^m b^n \mid 0 \leq m \leq n\}$.

Transitions of the shape $a \to R$ which don't mention what character to write, just leave the character intact (or write the same character).

1.1.1

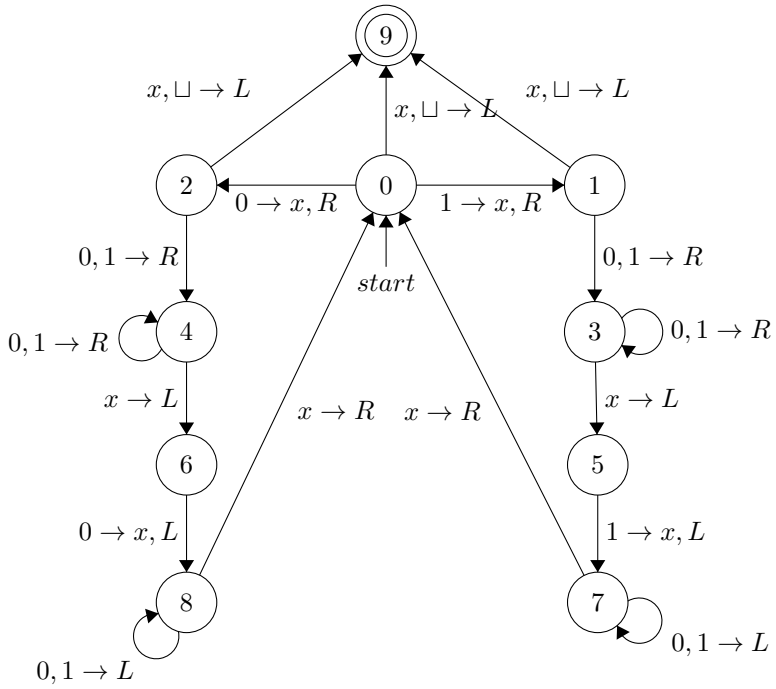


Figure 2: TM for the language $L = \{w \in \{0,1\}^* \mid w = w^R\}$

Transitions from states 1 and 2 to the final state handle the case where the string has odd length.

1.2

1.2.1

We show that we can simulate a Turing Machine with a 2-stack pushdown automaton and vice versa.

- For simulating a Turing Machine with a 2-stack pushdown automaton, first we push the characters of the input string (s) as we read them to the first stack. When we finish reading the input, we pop from the first stack and push to the second stack so that the characters are in the second stack in reverse order (we can push \$ to the stacks in the beginning so that we know when the stack is empty). This is the initial configuration $q_0 s$. Now we show how we go from one configuration to another configuration.
  Suppose we are in the configuration $C = xqy$; where q is the current state of the TM and its head points to the first character of y where y is in the second stack (in reverse order) and x is in the first stack.
  - $\delta(q, a) = (q', b, L)$: In this case, if top of the second stack is a, pop it and push b to it. Then pop from the first stack and push to the second one. And finally, go to the state

q' (traversing between states is done same as in the TM). Doing so, we end up in the configuration $C' = x[:-1]q'x[-1]by[1:]$.

- $\delta(q, a) = (q', b, R)$: If top of the second stack is a, pop it and push b to the first stack. Doing so, we reach the configuration $C' = xbq'y[1:]$.

We accept the input string as soon as we reach the final state of the TM since we have already read all the input string.

- Simulating a 2-stack pushdown automaton with a Turing Machine is easy. We simulate a non-deterministic 3-tape TM with our Turing Machine, where first tape contains the input, and second and third tapes represent the two stacks. If $(q', c, d) \in \delta(q, w, a, b)$ is one of the transitions in the 2-stack automaton, where it reads w from the input, and pop a and b from first and second stacks and push c and d to them respectively and go from the state q to the state q', then we act as follows in our 3-tape TM: We check the first head and if it points to w, we move it to the right. For each pop, we simply move the head of the corresponding tape to the left. And for each push, we write the character to the tape and move the head to the right. If any of a, b, c, or d is ϵ, we do not do the corresponding push or pop. We accept the input string if first head points to the blank character, and we are in the final state (Note that pushdown automata may have more than one final state, but we can add a new state as TM's final state and add a epsilon transition from each final state of the pushdown automaton to the new state).

So we have proved that the class of languages a Turing Machine can recognize is the same as the class of languages a 2-stack pushdown automaton can recognize.

1.2.2

If pushdown automaton has n stacks, then we can simulate it with a non-deterministic $n + 1$-tape TM in the similar manner as we did in the previous part with the first tape representing the input and the other tapes representing the stacks. So adding more stacks to the pushdown automaton does not change the class of languages it can recognize.

1.3

First, any regular TM can be considered a Turing Machine with ability to stay put (S) which simply doesn't use its S ability. Now we show that a TM with S can be converted to a TM without it. Suppose M is a TM with the ability to stay put:

$$M = (Q, \Sigma, \Gamma, \delta, q_0, q_{acc}, q_{rej})$$
$$\text{Where: } \delta : Q \times \Gamma \to Q \times \Gamma \times \{L, R, S\}$$

First we copy every state of M and add them to Q (we denote copy of state q as q').

$$Q' = \bigcup_{q \in Q} \{q'\}$$
$$M = (Q \cup Q', \Sigma, \Gamma, \delta', q_0, q_{acc}, q_{rej})$$
$$\text{Where: } \delta' : (Q \cup Q') \times \Gamma \to (Q \cup Q') \times \Gamma \times \{L, R\}$$

Now we convert the transition function δ to δ' as follows:
Suppose $\delta(q,a) = (r,b,X)$. If $X = R$ or $X = L$, then $\delta'(q,a) = \delta(q,a) = (r,b,X)$. Else if $X = S$, then $\delta'(q,a) = (q',b,R)$ and $\forall w \in \Gamma :\ \delta'(q',w) = (r,w,L)$.

Doing so, we have converted every transition of M with S, to a transition with R where it writes the desired character and moves the head to the right, going to a temporary state, then in the next transition, it moves back to left leaving that character intact.

So we have proved $L(M) = L(M')$.

1.4

We can see that applying homomorphism H on the string $w = w_1 w_2 \cdots w_n$ is equivalent to applying H on w_is separately, e.i. $H(w) = H(w_1)H(w_2)\cdots H(w_n)$.

1.4.1

We prove Turing Recognizable languages are closed under homomorphism by constructing a recognizer for $H(L)$ from a recognizer for L, M_L.

M' = "on input w:

1. For x in Σ^* (words are sorted first by size and then lexicographically):

   If $H(x) = w$ and $M_L(x)$ accepts, then accept.

1.4.2

Reference.
We show a decidable language and a homomorphism that converts it to an undecidable language. Alphabet of our language is $\Sigma = \{0,1,a,b\}$ and objects are encoded in strings with either $\{0,1\}$ binary representation or $\{a,b\}$ binary representation.

- $L = \{xy \mid x \in \{0,1\}^*, y \in \{a,b\}^*, x = \langle M,w\rangle, y = \langle n\rangle$ where n is an integer such that M halts on input w in n steps$\}$
- L is decidable. We can simply run M on input w for n steps and check if it halts.
- Consider the homomorphism H: $H(0) = 0$, $H(1) = 1$, $H(a) = H(b) = \varepsilon$.
- $H(L) = HALT$ which is undecidable

2 Linear Bounded Automata

2.1

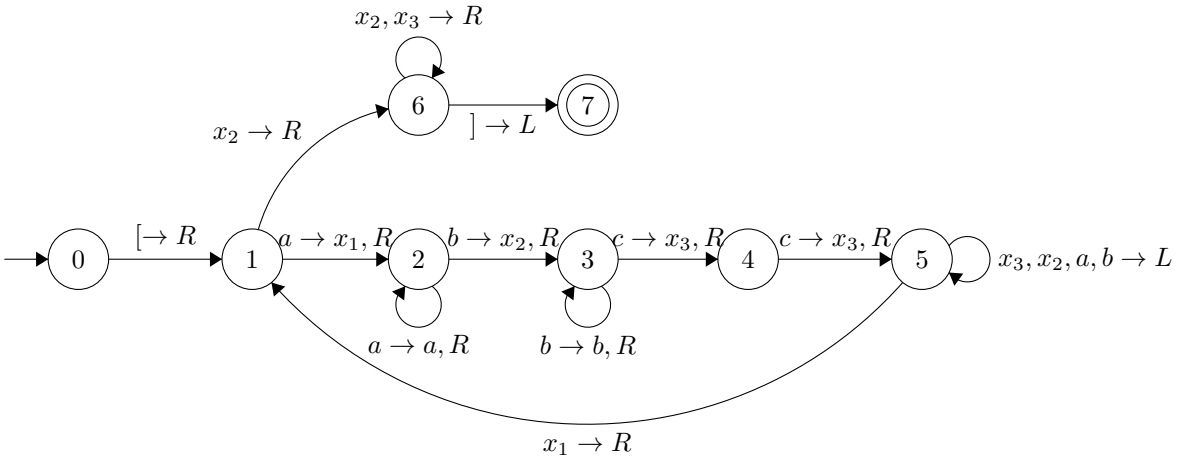


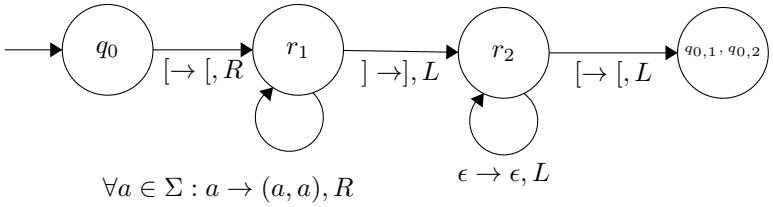
Figure 3: Linear Bounded Automaton for $L = \left\{a^n b^n c^{2n} \mid n \geq 1\right\}$

2.2

We will prove that context-sensitive languages are closed under union, intersection, and concatenation. Suppose M_1 and M_2 are linear bounded machines recognizing context-sensitive languages L_1 and L_2. (Throughout this proof, we suppose the alphabet of L_1 and L_2 are the same.)

$$M_1 = (Q_1, \Sigma, \Gamma_1, q_{0,1}, q_{acc,1}, q_{rej,1})$$
$$M_2 = (Q_2, \Sigma, \Gamma_2, q_{0,2}, q_{acc,2}, q_{rej,2})$$

- Union: We construct the linear-bounded automaton M for recognizing $L_1 \cup L_2$. We need to run both machines M_1 and M_2 on the tape and if any of them accepts the string, we accept it. We design states of M as Cartesian product of Q_1 and Q_2 (and some extra states that we will need). At first, the input string is written on the tape. We iterate through the tape and change every character a in the tape to (a, a) and come back to the beginning of the tape:



Then we mark both characters of the first pair on the input, denoting the initial positions of two heads. Then we do the following steps:

1. Run M_1 on first symbol of the current pair. Write the symbol on the first symbol of the current pair and move the head in the direction specified by its transition. Mark the first symbol of that pair. If we are in the state q_1, q_2, after this step we go to the state q_1', q_2 where q_1' is the state specified by M_1's transition.
2. Search for the marked symbol on the second symbol of pairs (e.g., search for it in the left direction and if not found, search for it in the right direction).
3. Run M_2 on the second symbol of the current pair. Move the head, mark the second symbol of that pair and change the (second part of) state similar to step 1.
4. Search for the marked symbol on the first symbol of pairs.
5. Go to step 1.
6. In any step, if we go to any of the states in the set $\{(q_{acc,1}, r) \mid r \in Q_2\} \cup \{(r, q_{acc,2}) \mid r \in Q_1\}$, we go to the state q_{acc} and accept the string.

So we have constructed an LBA recognizing $L_1 \cup L_2$ and proved its context-sensitive.

- Intersection: We do exactly same as the previous part, except the 6^{th} step; we accept the string only if both machines accept it. That means $q_{acc} = (q_{acc,1}, q_{acc,2})$.
- Concatenation: We construct M as follows:

  1. Non-deterministically (note that LBAs are non-deterministic), guess the first character of the second part of string (the part that is accepted with M_2), memorize it and change it to].
  2. Return to the beginning of the tape and run M_1 on the input string.
  3. If M_1 rejects, reject. If it accepts the string, we go to the first] symbol, change it back to the initial character, change the character before it to [and run M_2 on from there.
  4. If M_2 rejects, reject. If it accepts the string, we accept it.

  So $L_1 \circ L_2$ is context-sensitive.

3 Recognizability and Decidability

3.1

It's not decidable.
Proof by contradiction: Suppose it's decidable and S decides it. In this case, we design Turing Machine U deciding the language A_{TM}.
U = "On input $\langle M, w \rangle$ where M is a TM and w is a string:

1. Construct the following TM, M_1:
   M_1 = "On input x:
   1. If $x = 1$, accept.
   2. If $x = 2$, run M on w. Accept if M accepts w."
2. Construct the following TM, M_2:
   M_2 = "On input x:

1. If $x = 1$, accept.
2. If $x = 2$, accept."

3. Run S on $\langle M_1, M_2 \rangle$.
4. If S accepts, accept. If S rejects, reject."

But we know that A_{TM} is not decidable, so we have a contradiction.

3.2

Suppose M_A and M_B are deciders for A and B, respectively.

3.2.1

We design M, deciding $A \cup B$, as follows:
$M =$ "On input w:

1. Run M_A on w.
2. If M_A accepts, accept.
3. If M_A rejects, run M_B on w.
4. If M_B accepts, accept.
5. If M_B rejects, reject."

Note that since M_A and M_B are deciders, they both halt on any input.

3.2.2

We design U, deciding A^*, as follows:
$U =$ "On input w:

1. If $w = \varepsilon$, accept.
2. For $n \in \{1, 2, \ldots, |w|\}$:

   For all possible splits of w to $w_1, w_2, \ldots, w_n$:

   Run M_A on all the strings $w_1, \ldots, w_n$. If all of them accept, accept.
3. If none of the splits accept, reject."

Since U has to run M_A finitely many times, it halts eventually. So it's a decider.

3.3

Suppose M_A and M_B are recognizers for A and B, respectively.

3.3.1

We design M, recognizing AB, as follows:
M = "On input w:

1. Non-deterministically guess the first character of the second part of the string.
2. Run M_A and M_B on the first and second parts of the string, respectively.
3. If both of them accept, accept."

3.3.2

We want to prove that A is Turing Decidable if and only if A and $\bar{A}$ are both Turing Recognizable. One of the directions is easy. We know that every decidable language is recognizable too. And complement of a decidable language is decidable. So if A is decidable, then A and $\bar{A}$ are both recognizable.
For other direction, suppose M_1 and M_2 are recognizers for A and $\bar{A}$, respectively. We design M, deciding A, as follows:
M = "On input w:

1. Run both M_1 and M_2 on w in parallel.
2. If M_1 accepts, accept. If M_2 accepts, reject."

And running in parallel means having two tapes for simulating M_1 and M_2, taking turn for running one step of each machine.

We know that every string w is either in A or $\bar{A}$. So either M_1 or M_2 accepts w. So M halts. Therefore, it's a decider.

3.4

3.4.1

Suppose A is the language $\{w \in \{a,b\}^* \mid n_a(w) \geq n_b(w)\}$. For the input DFA D, we want to test that $L(D) \subseteq A$ or equivalently, $L(D) - A = L(D) \cap \bar{A} = \emptyset$.

We know that $\bar{A}$ is still a context-free language in this case:

$$\bar{A} = \{w \in \{a,b\}^* \mid n_a(w) < n_b(w)\}$$

And we can design following PDA for recognizing it:

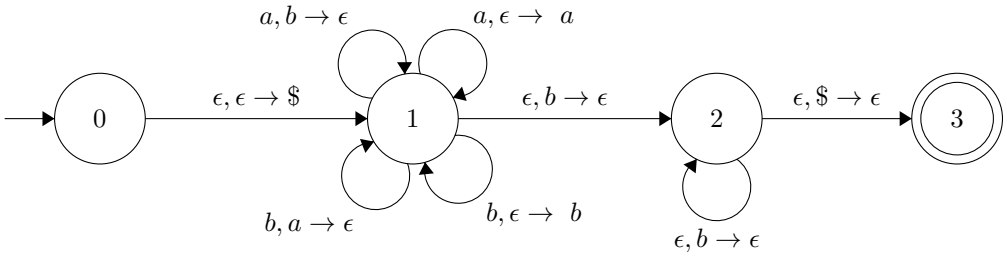


Figure 4: PDA for recognizing $\bar{A}$.

And we know that intersection of a context-free language and a regular language is a context-free language. Suppose G is our PDA for recognizing $\bar{A}$. We design PDA G', recognizing $L(D) \cap \bar{A}$, as follows:

$$\begin{aligned} G &= (Q_1, \Sigma, \Gamma, \delta_1, q_1, F_1) \\ D &= (Q_2, \Sigma, \delta_2, q_2, F_2) \\ \implies G' &= (Q_1 \times Q_2, \Sigma, \Gamma, \delta', (q_1, q_2), F_1 \times F_2) \\ \text{where } \delta'((r_1, r_2), a, b) &= \{((q, \delta_2(r_2, a)), s) \mid (q, s) \in \delta_1(r_1, a, b)\} \end{aligned} \tag{3.4.1}$$

And finally we know there is a decidable Turing Machine that checks whether language of a CFG is empty or not (denoted by E_{CFG} in the book). We use all these to design a decider for L as follows:

$M =$ "On input $\langle D \rangle$:

1. Construct PDA G for recognizing $\bar{A}$ as in Figure 4.
2. Construct PDA G' for recognizing $L(D) \cap \bar{A}$ as in Equation (3.4.1).
3. Convert PDA G' to an equivalent CFG C using the algorithm in Section 2.2 of the book.
4. Run E_{CFG} on C. If it accepts, accept. If it rejects, reject."

So we have constructed a decider for L.

3.4.2

(ALL_{CFG} was another language defined in the book!)
We use a similar algorithm for deciding this language:

$$L_2 = \{\langle D, C \rangle \mid D \text{ is a DFA and } C \text{ is a CFG such that } L(C) \subseteq L(D)\}$$

We need to make sure $L(C) \cap \overline{L(D)} = \emptyset$.

$M =$ "On input $\langle D, C \rangle$:

1. Convert DFA D to DFA D' which accepts $overline L(D)$, by swapping the final states and the non-final states.
2. Convert CFG C to the equivalent PDA G, by the algorithm in Section 2.2 of the book.
3. Construct PDA G' for recognizing $L(D') \cap L(G)$, as in Equation (3.4.1).
4. Convert PDA G' to an equivalent CFG C' using the algorithm in Section 2.2 of the book.
5. Run E_{CFG} on C'. If it accepts, accept. If it rejects, reject."

So we have constructed a decider for L_2.

4 Reducibility and Proving Undecidability

4.1

4.1.1

We want to show that $Inf_{TM} = \{\langle M \rangle \mid L(M) \text{ is infinite}\}$ is undecidable.
We construct the following machine F that computes a reduction f between A_{TM} and Inf_{TM}:

F = "On input $\langle M, w \rangle$:

1. Construct the following machine M' as follows:
   M' = "On input x:
   1. Run M on w.
   2. If M accepts, accept. If M rejects, reject."
2. Output $\langle M' \rangle$."

So $\langle M, w \rangle \in A_{TM}$ iff $f(\langle M, w \rangle) = \langle M' \rangle \in Inf_{TM}$.
Thus, we have the reduction $A_{TM} \leq_m Inf_{TM}$ and Inf_{TM} is undecidable.

4.1.2

Reference of the proof.
We want to prove that $INTRSCT_{CFG} = \{\langle G_1, G_2 \rangle \mid G_1 \text{ and } G_2 \text{ are CFGs such that } L(G_1) \cap L(G_2) = \emptyset\}$ is undecidable.
We prove it using the undecidability of post correspondence problem (PCP). Suppose we have a set of dominos $d_1, d_2, \ldots, d_n$, where $d_i = \left[\frac{w_i}{x_i}\right]$ for $i \in 1, \ldots, n$. Construct two Grammars (W, X) as below:

$$W \to w_1 W d_1 \mid w_2 W d_2 \mid \cdots \mid w_n W d_n \mid w_1 d_1 \mid w_2 d_2 \mid \cdots \mid w_n d_n$$
$$X \to x_1 X d_1 \mid x_2 X d_2 \mid \cdots \mid x_n X d_n \mid x_1 d_1 \mid x_2 d_2 \mid \cdots \mid x_n d_n$$

We know our dominos have a match, if and only if there is a string that can be generated by both W and X (they will have the same dominos on the right-hand side of the string. And the upper and lower parts of dominos form the same strings).
So given a pcp instance, we can map it to a problem of $\overline{INTRSCT_{CFG}}$ as follows:

F = "On input $\langle d_1, d_2, \ldots, d_n \rangle$:

1. Construct CFGs X and W as above.
2. Output $\langle X, W \rangle$."

We know that $\langle d_1, d_2, \ldots, d_n \rangle \in PCP$ iff $\langle X, W \rangle \in \overline{INTRSCT_{CFG}}$.
So $PCP \leq_m \overline{INTRSCT_{CFG}}$ and we know that PCP is undecidable. So $\overline{INTRSCT_{CFG}}$ and as a result $INTRSCT_{CFG}$ are undecidable.

4.1.3

$$PAL_{CFG} = \{\langle G\rangle \mid G \text{ is a CFG and } L(G) \text{ contains a pallindrome}\}.$$

Again, we use the undecidability of PCP to prove this part.
Suppose an instance of PCP, contains dominos $d_1, d_2, \ldots, d_n$, where $d_i = \left[\frac{w_i}{x_i}\right]$. We design the mapping function as follows:

$F =$ "On input $\langle d_1, d_2, \ldots, d_n\rangle$:

1. Construct CFG G with start variable S and rules as below:

$$S \to D_1 \mid D_2 \mid \cdots \mid D_n$$
$$\forall i \in 1, \ldots, n : D_i \to w_i S x_i^R \mid w_i x_i^R \qquad (R \text{ means reverse})$$

2. Output $\langle G\rangle$."

We know that the input instance has a match if and only if there is a palindrome string in $L(G)$. So $\langle d_1, d_2, \ldots, d_n\rangle \in PCP$ iff $\langle G\rangle \in PAL_{CFG}$, hence $PCP \leq_m PAL_{CFG}$. And we know PCP is undecidable. So PAL_{CFG} is undecidable.

4.2

Proof by contradiction. Assume T is decidable. Let M_T be a Turing machine that decides it. We can construct the following decider for A_{TM}:

$F =$ "On input $\langle M, w\rangle$:

1. construct machine M' as follows:
   $M' =$ "On input x:
   1. If $x = 12$, accept.
   2. If $x = 21$, Run M on w and if it accepts, accept. If it rejects, reject.
2. Run M_T on the input $\langle M', 12\rangle$.
3. If M_T accepts, accept. If M_T rejects, reject."

So we have designed a decider for A_{TM}. But we know A_{TM} is undecidable. So we have reached a contradiction and T is undecidable.

4.3

4.3.1

Reference of the proof.
Proof by contradiction: Assume P is a non-trivial property and $A_P = \{\langle M\rangle \mid M \text{has the property } P\}$ is decidable and H is a decider for it.

$$H(x) = \begin{cases} \text{accept} & \text{if the machine } x \text{ has the property } P \\ \text{reject} & \text{if the machine } x \text{ does not have the property } P \end{cases}$$

Since P is non-trivial, there is a machine Y that has the property P and a machine N that does not have it.
Define machine A as below:

$A =$ on input x:

1. If H accepts A, return $N(x)$.
2. If H rejects A, return $Y(x)$.

Now, if A has the property P, then $H(A) =$ accept and $A(x) \equiv N(x)$. And if A does not have the property P, then $H(A) =$ reject and $A(x) \equiv Y(x)$. Which is a contradiction in both cases. So A_P is undecidable.

4.3.2

First, we show that being finite is a non-trivial property for languages.

$Y =$ "On input x:

1. Reject.

$N =$ "On input x:

1. Accept.

Clearly the language of Y is finite and the language of N is infinite. Thus, being a non-trivial property, we can conclude that $FINITE = \{\langle T \rangle \mid L(T) \text{ is finite}\}$ is undecidable.