Decidability

We investigate the power of algorithms to solve problems. We demonstrate that certain problems can be solved algorithmically and others cannot. Our objective is to explore the limits of algorithmic solvability.

Coding decision problems as languages

Decision problems are problems that expect a Yes/No answer. E.g.

1. PRIME: Given a number, is it prime?

2. CYCLIC: Given a graph, is it cyclic?

3. DFA ACCEPTANCE: Given a DFA B and an input w, does B accept w?

We represent decision problems as languages.

E.g. the language representing CYCLIC is

$$\{ \langle G \rangle : G \text{ is a cyclic graph} \}$$

where $\langle G \rangle$ denotes an encoding of G as a string over $\Sigma = \{ 0, 1, (,), \# \}$(say).
The graph $(\{ 1, 2, 3 \}, \{ (1, 2), (2, 3), (3, 1) \})$ can be encoded as

$$(1\#10\#11)\#((1\#10)\#(10\#11)\#(11\#1)).$$

Decidability

We say that a decision problem is *decidable* if the corresponding language is (Turing-machine) decidable.

We say that a decision problem is *r.e.* (*recursively enumerable*) or *semi-decidable* if the corresponding language is r.e. (i.e. TM acceptable).

DFA Acceptance Problem

DFA Acceptance Problem: Given a DFA B and an input w, does B accept w?

The corresponding language is

$$A_{\mathrm{DFA}} \;=\; \{\, \langle B, w \rangle : B \text{ is a DFA that accepts input string } w \,\}.$$

Lemma. A_{DFA} is a decidable language.

Proof. We construct a TM M that decides A_{DFA}: On input $\langle B, w \rangle$ where B is a DFA and w an input

1. Simulate B on input w

2. If the simulation ends in an accept state, *accept.* If it ends in a non-accepting state, *reject.*

Convince yourself that it follows that A_{NFA} and A_{regexp} (acceptance problems

for NFA and regular expressions respectively) are also decidable.

Emptiness Problem for DFA: Given a DFA A, is $L(A)$ empty?

The corresponding language is

$$E_{\mathrm{DFA}} \;=\; \{\, \langle A \rangle : A \text{ is a DFA such that } L(A) = \emptyset \,\}$$

Lemma. E_{DFA} is a decidable language.

Proof. We design a TM T that uses a marking algorithm. On input $\langle A \rangle$ where A is a DFA:

1. Mark the start state of A.

2. Repeat until no new states get marked:

   Mark any state that has a transition coming into it from any marked state.

3. If no accept state is marked, *accept*; otherwise *reject*.

Equivalence Problem for DFA: Given two DFAs, are they equivalent?

The corresponding language:

$EQ_{\mathrm{DFA}} = \{\, \langle A, B \rangle : A$ and B are DFA s.t. $L(A) = L(B) \,\}$.

Lemma. EQ_{DFA} is a decidable language.

Proof. Key idea: For sets P and Q, $P \subseteq Q$ iff $P \cap \overline{Q} = \emptyset$. We use T, the algorithm for the Emptiness Problem E_{DFA}.

On input $\langle A, B \rangle$, where A and B are DFAs:

1. Construct $C = (A \cap \overline{B}) \cup (B \cap \overline{A})$. (DFAs are closed under union, intersection and complementation.)

2. Run T on input $\langle C \rangle$.

3. If T accepts, accept; if T rejects, reject.

Acceptance problem for CFG

Consider the language:

$$A_{\mathrm{CFG}} = \{\, \langle G, w \rangle : G \text{ is a CFG that generates } w \,\}$$

Lemma. A_{CFG} is a decidable language.

We use a fact: If G is in Chomsky normal form, any derivation of w has $2n - 1$ steps, where n is the length of w.

Proof. The TM S for A_{CFG} is: On input $\langle G, w \rangle$, where G is a CFG and w is a string:

1. Convert G to an equivalent one in Chomsky normal form.

2. List all derivations with $2n - 1$ steps, where n is the length of w.

3. If any of these derivations generate w, accept; otherwise, reject.

This is of course extremely inefficient, and will not be used in practice.

Emptiness Problem for CFG: Given a CFG G, is $L(G)$ empty?

Lemma. E_{CFG} is a decidable language.

Proof. The TM R for E_{CFG} is: On input $\langle G \rangle$, where $G = (V, \Sigma, R, S)$ is a CFG:

1. Mark all terminal symbols in G.

2. Repeat until no new variables get marked:

   Mark any variable A where G contains a rule $A \rightarrow U_1 \cdots U_k$ and each symbol $U_i \in \Sigma \cup V$ has already been marked.

3. If the start symbol is not marked, accept; otherwise, reject.

Equiv. Problem for CFG: Given two CFGs, are they equivalent?

Obvious attempt: Use the strategy for deciding EQ_{DFA}.

Unfortunately CFGs are neither closed under intersection nor complementation!

In fact EQ_{CFG} is *undecidable*.

Universal Turing Machines

Acceptance Problem for TM

$$A_{\mathrm{TM}} \;=\; \{\, \langle M, w \rangle : M \text{ is a TM that accepts input } w \,\}$$

Theorem. A_{TM} is r.e.

Proof. We define a TM U that accepts A_{TM}: On input $\langle M, w \rangle$ where M is a TM and w is a string

1. *Simulate M on input w*.

2. If M ever enters its accept state, accept; if M ever enters its reject state, reject.

Note that U loops on $\langle M, w \rangle$ if M loops on w. This is why U is not a decider.

If the algorithm had some way to determine that M was not halting on w, it could

reject.

Universal Turing Machines: Encoding Turing machines

A *universal* TM is a TM U that can simulate the actions of any Turing machine.
I.e. for any TM M and any input x of M, U accepts (respectively rejects or loops on) $\langle M, x \rangle$, if and only if, M accepts (respectively rejects or loops on) x.

This is equivalent to an interpreter of C written in C.

An encoding scheme over $\{0, 1\}^*$ for $\langle M, x \rangle$.

E.g. if the string begins with the prefix

$$0^n \, 1 \, 0^m \, 1 \, 0^k \, 1 \, 0^s \, 1 \, 0^t \, 1 \, 0^r \, 1 \, 0^u \, 1 \, 0^v$$

this might indicate that the machine has n states $(0, 1, \cdots, n-1)$; it has m tape symbols $(0, 1, \cdots, m-1)$, of which the first k are input symbols; the start, accept and reject states are s, t and r respectively, and the endmarker and blank symbols are u and v respectively. The remainder of the string can consist of a sequence of substrings specifying the transitions in δ.

E.g. $0^p 1 0^a 1 0^q 1 0^b 1 0$ might indicate that δ contains the transition

$$((p, a), (q, b, L)).$$

The exact details are unimportant.

Simulation. The tape of U is partitioned into 3 tracks:

- The top track holds the transition function δ_M of the input TM M.

- The middle track will be used to hold the simulated contents of M's tape.

- The bottom track will hold the current state of M, and the current position of M's tape head.

U simulates M on input x one step at a time, shuttling back and forth between the three tracks. In each step, it updates M's state and simulated tape contents and head position as dictated by δ_M. If ever M halts and accepts or halts and rejects, then U does the same.

Note: M's input alphabet and U's may be different.

There are languages that are not r.e.

There are only countably many TMs.

Observe that Σ^* is countable, for any alphabet Σ. Since $\langle M \rangle \in \Sigma^*$ for each TM M, we can enumerate the set of TMs by simply omitting those strings that are not encodings of TMs.

The set of languages over Σ, $\mathcal{P}(\Sigma^*)$, is uncountable.

Suppose there were an enumeration of $\mathcal{P}(\Sigma^*)$, namely, $L_1, L_2, L_3, \cdots$. Let $x_1, x_2, x_3, \cdots$ be an enumeration of Σ^*. Define a new language L by: for $i \geq 1$

$$x_i \in L \iff x_i \notin L_i$$

Now $L \in \mathcal{P}(\Sigma^*)$, but L is not equal to any of the L_is.

Thus $\mathcal{P}(\Sigma^*)$ is not in 1-1 correspondence with the set of Turing machines (there are more languages than TMs). It follows that there is some language that is not accepted by any TM.

A_{TM}: Acceptance Problem for TMs

Theorem. A_{TM} is undecidable.

Proof. Suppose, for a contradiction, TM H is a decider for A_{TM}. I.e.

$$H(\langle M, w \rangle) \;=\; \begin{cases} accept & \text{if } M \text{ accepts } w \\[1em] reject & \text{if } M \text{ does not accept } w \end{cases}$$

We construct a new TM D with H as a subroutine. D: On input $\langle M \rangle$, where M is a TM

1. Run H on input $\langle M, \langle M \rangle \rangle$

2. Output the opposite of H. I.e. if H accepts, *reject*, and if H rejects, *accept*.

Thus

$$D(\langle M \rangle) \;=\; \begin{cases} \text{\textit{reject}} & \text{if } M \text{ accepts } \langle M \rangle \\[2mm] \text{\textit{accept}} & \text{if } M \text{ does not accept } \langle M \rangle \end{cases}$$

Now we run D on input $\langle D \rangle$. We have

$$D(\langle D \rangle) \;=\; \begin{cases} \text{\textit{reject}} & \text{if } D \text{ accepts } \langle D \rangle \\[2mm] \text{\textit{accept}} & \text{if } D \text{ does not accept } \langle D \rangle \end{cases}$$

I.e. D accepts $\langle D \rangle$ iff D rejects $\langle D \rangle$, which is a contradiction. Thus neither D nor H can exist. $\qquad\square$

To summarize, A_{TM} is (r.e. but) undecidable.

A language that is not r.e.

We say that L is *co-r.e.* if its complement $\overline{L}$ is r.e.

Theorem. A language is decidable iff it is both r.e. and co-r.e.

Proof. Suppose L and $\overline{L}$ are acceptable by M and M' respectively. Define a TM N as follows: On input x

1. Write x on the input tapes of M and M'.

2. Run M and M' in parallel (e.g. by dove-tailing the two computations).

3. If M accepts, *accept*; if M' accepts, *reject*.

N is a decider for L because for any input x, either $x \in L$ or $x \in \overline{L}$: if the former then N must accept because M must halt and accept at some point, if the latter, then N must reject because M' must halt and accept at some point.

Corollary. $\overline{A_{\mathrm{TM}}}$ is not r.e. $\quad\square$

The Halting Problem: "Given M and x, does M halt on x?"

$$HALT_{\mathrm{TM}} \; = \; \{\, \langle M, w \rangle : M \text{ is a TM and } M \text{ halts on input } w \,\}$$

Theorem. The Halting Problem, $HALT_{\mathrm{TM}}$, is undecidable.

Proof. Suppose, for a contradiction, H is a decider for $HALT_{\mathrm{TM}}$. We use H to construct a TM S to decide A_{TM} as follows: On input $\langle M, w \rangle$ where M is a TM and w is an input:

1. Run TM H on input $\langle M, w \rangle$

2. If H rejects, *reject*.

3. If H accepts, run M on w until it halts.

4. If M has accepted, *accept*; if M has rejected, *reject*.

Since A_{TM} is undecidable, H cannot be a decider for $HALT_{\mathrm{TM}}$. $\qquad\square$

The Emptiness Problem for TM is undecidable

$E_{\mathrm{TM}} = \{\, \langle M \rangle : \; M$ is a TM and $L(M) = \emptyset \,\}$

Theorem. The Emptiness Problem for TM is undecidable.

Proof. Suppose, for a contradiction, E_{TM} is decidable by a TM E. We shall use E to construct a TM S that decides A_{TM}.

Definition of S: On input $\langle M, x \rangle$ where M is a TM and x an input

1. Construct a TM M_x defined as: On input y

    (a) If $y \neq x$ then *reject*.

    (b) Run M (input is x) and *accept* if M does.

2. Run E on input $\langle M_x \rangle$. If E accepts, *reject*, if E rejects, *accept*.

Now we have

$$M \text{ accepts } x \quad \Rightarrow \quad L(M_x) = \{\, x \,\} \quad \text{i.e. } S \text{ accepts } \langle M, x \rangle$$

$$M \text{ does not accept } x \quad \Rightarrow \quad L(M_x) = \emptyset \quad \text{i.e. } S \text{ rejects } \langle M, x \rangle$$

Thus S decides A_{TM}, which is a contradiction. $\qquad\qquad\qquad\qquad\square$

The Equivalence Problem for TM is undecidable

$EQ_{\mathrm{TM}} = \{\, \langle M_1, M_2 \rangle : \ M_1 \text{ and } M_2 \text{ are TMs and } L(M_1) = L(M_2) \,\}$

Theorem. The Equivalence Problem for TM is undecidable.

Proof. Suppose, for a contradiction, EQ_{TM} is decidable by Q. We define a TM that decides E_{TM} by: On input M

1. Run Q on input $\langle M, M_1 \rangle$ where M_1 is a TM that accepts nothing i.e. $L(M_1) = \emptyset$.

2. If Q accepts, *accept*; if Q rejects, *reject*.

This gives a contradiction. $\qquad\qquad\square$

Rice's Theorem

Rice's Theorem. Every non-trivial property of the r.e. sets is undecidable.

Reformulation Let P be any decision problem about TMs (expressed as a language) that satisfies the following properties:

(1) For any TMs M_1 and M_2 where $L(M_1) = L(M_2)$, we have $\langle M_1 \rangle \in P$ iff $\langle M_2 \rangle \in P$. I.e. the membership of a TM M in P depends only on the language of M.

(2) There are TMs M_1 and M_2 where $\langle M_1 \rangle \in P$ and $\langle M_2 \rangle \notin P$. I.e. P is non-trivial.

Then P is undecidable.

Proof of Rice's Theorem

Take a property P that satisfies (1) and (2). Wlog, assume that for any TM E, if $L(E) = \emptyset$ then $\langle E \rangle \notin P$. Take a TM K such that $\langle K \rangle \in P$. Given $\langle M, x \rangle$ where M is a TM and x is an input, we construct $N_{M,x}$ as follows: On input y

1. Save y on a separate track somewhere.

2. Write x on the input tape, and run M on input x.

3. If M halts on x, run K on y.

4. If K accepts, *accept.*

Either M halts on x or not.

If M does not halt on x, the simulation in 3. will not halt, and the input y of $N_{M,x}$ will not be accepted, and so, $L(N_{M,x}) = \emptyset$ i.e. $\langle N_{M,x} \rangle \notin P$.

On the other hand, if M halts on x, y is accepted by $N_{M,x}$ iff y is accepted by K i.e. $L(N_{M,x}) = L(K)$ i.e. $\langle N_{M,x} \rangle \in P$.

Thus

$$M \text{ halts on } x \quad \Rightarrow \quad L(N_{M,x}) = L(K) \quad \Rightarrow \quad \langle N_{M,x} \rangle \in P$$

$$M \text{ does not accept } x \quad \Rightarrow \quad L(N_{M,x}) = \emptyset \quad \quad \Rightarrow \quad \langle N_{M,x} \rangle \notin P$$

Thus we have reduced $HALT_{\mathrm{TM}}$ to P. Since $HALT_{\mathrm{TM}}$ is undecidable, so is P.

[Assume for a contradiction P was decidable, say by decider A, then we can construct a decider S for $HALT_{\mathrm{TM}}$ as follows: on input $\langle M, x \rangle$ construct $\langle N_{M,x} \rangle$ and run A on $\langle N_{M,x} \rangle$. If A accepts *accept*, if A rejects *reject*.]

$\square$

Application of Rice's Theorem

For an arbitrary TM M with $\Sigma = \{a, b\}$, the following problems are undecidable:

- whether M accepts $ababa$ (i.e. $ababa \in L(M)$)

- whether $L(M)$ contains two different strings of the same length;

- whether $L(M)$ contains any string of length five;

- whether $L(M)$ is a regular language;

- whether $L(M)$ is a context-free language;

Note The following properties of TMs are not properties of r.e. sets and Rice's Theorem is not applicable:

- M has at least 341 states;

- M halts on all inputs;

- M rejects $ababa$

- there exists a smaller machine equivalent to M.

TOTAL $= \{\langle M \rangle : M$ halts on all inputs$\}$

Theorem. TOTAL is neither r.e. nor co-r.e.

Proof. $HALT_{\mathrm{TM}}$ is not decidable, but it is r.e. Hence $\overline{HALT_{\mathrm{TM}}}$ is not r.e.

Therefore it suffices to reduce $\overline{HALT_{\mathrm{TM}}}$ to (i) TOTAL and to (ii) $\overline{\mathrm{TOTAL}}$.

(ii) Given $\langle M, x \rangle$ where M is a TM and x is an input, we set $N_{M,x}$ to be the following TM: On input y,

1. Run M on input x for up to $|y|$ steps.

2. If M has halted, then loop; if M has not yet halted, *accept*.

We have

$$\langle M, x \rangle \in \overline{HALT_{\mathrm{TM}}} \iff N_{M,x} \text{ halts on all input } y \iff N_{M,x} \in \mathrm{TOTAL}.$$

(ii) Given $\langle M, x \rangle$ where M is a TM and x is an input, we construct $N_{M,x}$ to be the following TM: On input y,

1. Erase y.

2. Write x on input tape, and run M on x.

3. If M accepts, *accept*; if M rejects, *reject*.

Note that either $N_{M,x}$ loops on all inputs, or it halts on all inputs. Now

$$\langle M, x \rangle \in \overline{HALT}_{\mathrm{TM}} \iff N_{M,x} \text{ loops on all inputs} \iff N_{M,x} \notin \text{TOTAL}.$$

Proving Undecidabilty

We have seen two main methods of proving undecidability results:

diagonalization and *reduction*.

Diagonalization.

E.g. *HALT*$_{\mathrm{TM}}$: Suppose H is a decider for *HALT*$_{\mathrm{TM}}$, construct new TM N using H as a subroutine: on input $\langle M \rangle$, run H on $\langle M, \langle M \rangle \rangle$, if H accepts let N loop for ever; if H rejects, let N accept. Then

$$N \text{ on } \langle N \rangle \text{ halts} \iff H \text{ on } N \text{ rejects} \iff N \text{ on } \langle N \rangle \text{ does not halt.}$$

Contradiction, therefore H does not exist.

Reduction. Let A be an undecidable problem over Σ_A^* and let B another problem over Σ_B^*.

If we can find a computable function $f : \Sigma_A^* \to \Sigma_B^*$ such that for all $w \in \Sigma_A^*$

$$w \in A \iff f(w) \in B$$

then B is undecidable as well. If B was decidable by M we could use it as a subroutine to decide A. On input $\langle w \rangle$ compute $\langle f(w) \rangle$, run M on $\langle f(w) \rangle$ and return the same answer.

Note Reduction works also for proving that a problem is not r.e.

The other method of proving that a problem is not r.e. uses the fact that a problem is decidable if and only if the problem and its complement are both r.e.

A puzzle: Given a collection of dominos such as e.g.

$$P \;=\; \left\{ \left[\frac{b}{ca}\right], \left[\frac{a}{ab}\right], \left[\frac{ca}{a}\right], \left[\frac{abc}{c}\right] \right\}$$

find a list of dominos from P (repetitions permitted) so that the string we get by reading off the symbols on the top is the same as the sting of symbols on the bottom. This list is called a *match*.

E.g. the following is a match taking dominos $2, 1, 3, 2$ and 4 in this order:

$$\left\langle \left[\frac{a}{ab}\right], \left[\frac{b}{ca}\right], \left[\frac{ca}{a}\right], \left[\frac{a}{ab}\right], \left[\frac{abc}{c}\right] \right\rangle$$

(Reading off the top string we get $abcaaabc$, the same as reading off the bottom one.)

A formal statement of the PCP

Instance: A collection P of *dominos*:

$$P = \left\{ \left[\frac{t_1}{b_1} \right], \left[\frac{t_2}{b_2} \right], \cdots, \left[\frac{t_k}{b_k} \right] \right\}$$

where $k \geq 1$, and all t_i and b_j are strings over Σ.

Question: Is there a *match* for P? I.e. a non-empty sequence $i_1, \cdots, i_l$ where each $1 \leq i_j \leq k$ and

$$t_{i_1} t_{i_2} \cdots t_{i_l} = b_{i_1} b_{i_2} \cdots b_{i_l}$$

Theorem. PCP is undecidable.

Examples

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

$$P = \left\{ \left[\frac{11}{111} \right], \left[\frac{111}{11} \right], \left[\frac{1100}{001} \right] \right\}.$$

Here is a match for P: $\langle 1, 1, 3, 2 \rangle$, i.e.

$$\left\langle \left[\frac{11}{111} \right], \left[\frac{11}{111} \right], \left[\frac{1100}{001} \right], \left[\frac{111}{11} \right] \right\rangle$$

2. For some collection of dominos, finding a match may not be possible. E.g.

$$P' = \left\{ \left[\frac{abc}{ab} \right], \left[\frac{ca}{a} \right], \left[\frac{acc}{bc} \right] \right\}$$

(because every top string is longer than the bottom string)

We modify *PCP* to

$$MPCP \quad = \quad \{P \text{ is an instance of PCP with a match}$$

$$\text{that starts with the first domino}\}$$

1. Suppose, for a contradiction, there is a TM R that decides the PCP. We use R to construct a TM S that decide A_{TM}. Let

$$M \; = \; (Q, \Sigma, \Gamma, \vdash, \sqcup, \delta, q_0, q_{\mathrm{acc}}, q_{\mathrm{rej}})$$

Take an input w for M. We construct an instance P' of the *MPCP* (over the alphabet $Q \cup \Gamma \cup \{\#\}$) that simulates M on w i.e.

$$P' \in MPCP \quad \Longleftrightarrow \quad \langle M, w \rangle \in A_{\mathrm{TM}}.$$

Idea. Writing a configuration (u_i, q_i, v_i) as $\#u_i\, q_i\, v_i$, a match of P' gives a string; the first part

$$\#q_0 \vdash w\#u_1\, q_1\, v_1\#u_2\, q_2\, v_2\#\cdots\#u_i\, q_i\, v_i\#\cdots\#u_n\, q_{\mathrm{acc}}\, v_n$$

corresponds exactly to an accepting configuration; the rest of the string

$$\#\cdots\#\cdots\cdots\#q_{\mathrm{acc}}\#\#$$

consists of progressively shorter segments of the form $\#\cdots$ that finally ends in $\#\#$.

2. We convert P' to P, an instance of *PCP* which still simulates M on w.

Definition of P'

Part 1. Put $\left[\dfrac{\#}{\#q_0 \vdash w_1 \cdots w_n \#} \right]$ into P' as the first domino $\left[\dfrac{t_1}{b_1} \right]$

Part 2. For each $a, b \in \Gamma$, $q, r \in Q$, if $\delta(q, a) = (r, b, R)$, put $\left[\dfrac{qa}{br} \right]$ into P'

Part 3. For each $a, b, c \in \Gamma$, $q, r \in Q$, if $\delta(q, a) = (r, b, L)$, put $\left[\dfrac{cqa}{rcb} \right]$ into P'

Part 4. For every $a \in \Gamma$, put $\left[\dfrac{a}{a} \right]$ into P'.

Part 5. Put $\left[\dfrac{\#}{\#} \right]$ and $\left[\dfrac{\#}{\sqcup\#} \right]$ into P'.

Part 6. For every $a \in \Gamma$, put $\left[\dfrac{aq_{\mathrm{acc}}}{q_{\mathrm{acc}}} \right]$ and $\left[\dfrac{q_{\mathrm{acc}}a}{q_{\mathrm{acc}}} \right]$ into P'.

Part 7. Add $\left[\dfrac{q_{\mathrm{acc}}\#\#}{\#} \right]$ to P'.

Consider the following TM M on input $w = ab$

$$M = (\underbrace{\{\, q_0, q_3, q_4, q_{\mathrm{acc}}, q_{\mathrm{rej}} \,\}}_{Q}, \underbrace{\{\, a, b, c \,\}}_{\Sigma}, \underbrace{\Sigma \cup \{\vdash, \sqcup\}}_{\Gamma}, \vdash, \sqcup, \delta, q_0, q_{\mathrm{acc}}, q_{\mathrm{rej}})$$

where δ is partly given by

$$\delta(q_0, \vdash) = (q_0, \vdash, R)$$
$$\delta(q_0, a) = (q_3, c, R)$$
$$\delta(q_3, b) = (q_4, a, R)$$
$$\delta(q_4, \sqcup) = (q_{\mathrm{acc}}, a, L)$$

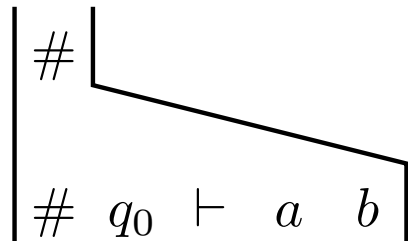An accepting computation:

$$(\epsilon, q_0, \vdash ab)$$
$$\rightarrow \quad (\vdash, q_0, ab)$$
$$\rightarrow \quad (\vdash c, q_3, b)$$
$$\rightarrow \quad (\vdash ca, q_4, \sqcup)$$
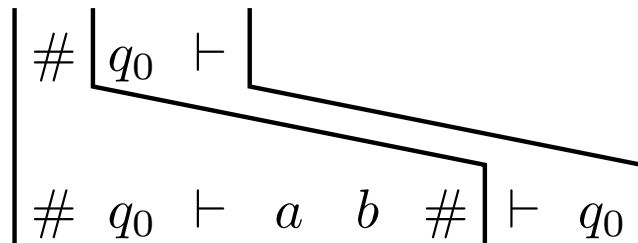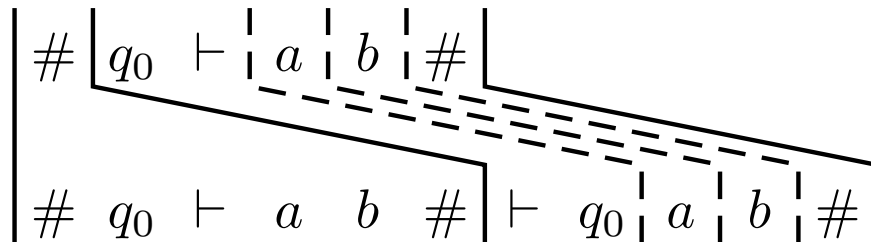$$\rightarrow \quad (\vdash c, q_{\mathrm{acc}}, ab)$$

part 1

$(\epsilon, q_0, \vdash ab)$

part 2

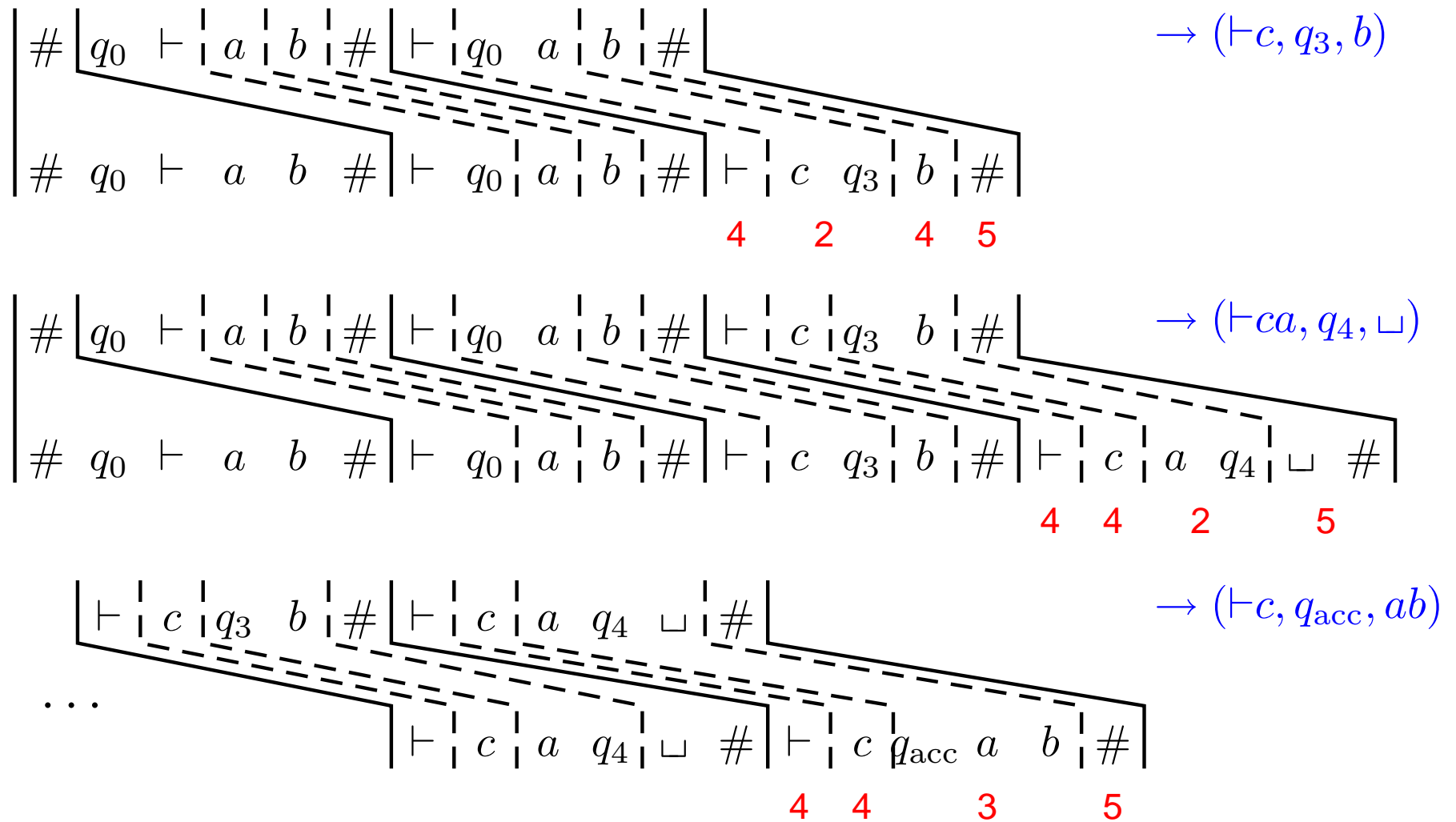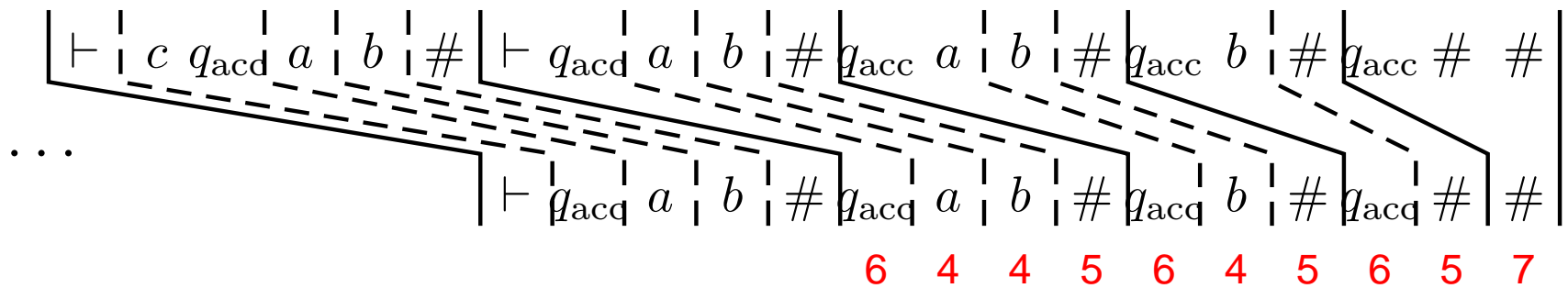$\delta(q_0, \vdash) = (q_0, \vdash, R)$

parts   4   4   5

$\rightarrow (\vdash, q_0, ab)$

$\rightarrow (\vdash c, q_3, b)$

$\rightarrow (\vdash ca, q_4, \sqcup)$

$\rightarrow (\vdash c, q_{\mathrm{acc}}, ab)$

An example (cont'd)

Tidying up.

$$\vdash c \; a \; q_4 \; \sqcup \; \# \; \vdash c \; q_{\mathrm{acd}} \; a \; b \; \#$$

$\cdots$

$$\vdash c \; q_{\mathrm{acc}} \; a \; b \; \# \; \vdash q_{\mathrm{acd}} \; a \; b \; \#$$

4   6   4   4   5

$$\vdash c \; q_{\mathrm{acd}} \; a \; b \; \# \; \vdash q_{\mathrm{acd}} \; a \; b \; \# \; q_{\mathrm{acc}} \; a \; b \; \# \; q_{\mathrm{acc}} \; b \; \# \; q_{\mathrm{acc}} \; \# \; \#$$

$\cdots$

$$\vdash q_{\mathrm{acd}} \; a \; b \; \# \; q_{\mathrm{acd}} \; a \; b \; \# \; q_{\mathrm{acd}} \; b \; \# \; q_{\mathrm{acd}} \; \# \; \#$$

6   4   4   5   6   4   5   6   5   7
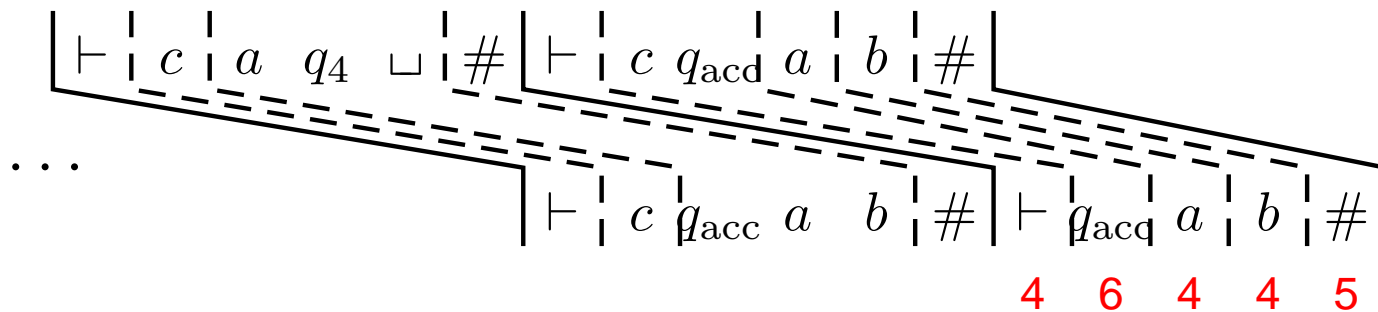
Finally we show how to convert P' to an instance of PCP which still simulates M on w. The idea is to build the requirement of starting with the first domino directly into the problem.

Notation. Take a string $u = u_1 \cdots u_n$. Define

$$\star u \quad = \quad * u_1 * u_2 * \cdots * u_n$$

$$u \star \quad = \quad u_1 * u_2 * \cdots * u_n *$$

$$\star u \star \quad = \quad * u_1 * u_2 * \cdots * u_n *$$

Suppose P' is the MPCP instance (over alphabet Θ)

$$\left\{ \left[\frac{t_1}{b_1} \right], \left[\frac{t_2}{b_2} \right], \left[\frac{t_3}{b_3} \right], \cdots, \left[\frac{t_k}{b_k} \right] \right\}$$

Now define P to be the PCP instance (over alphabet $\Theta \cup \{*, \diamond\}$)

$$\left\{ \left[\frac{\star t_1}{\star b_1 \star} \right], \left[\frac{\star t_2}{b_2 \star} \right], \left[\frac{\star t_3}{b_3 \star} \right], \cdots, \left[\frac{\star t_k}{b_k \star} \right], \left[\frac{\star \diamond}{\diamond} \right] \right\}$$

Note that the only domino that can start a match is the first one, because it is the only one where both the top and bottom start with the same symbol, namely, $*$.

The original symbols in Θ now occur in even positions of the match.

The domino $\left[\dfrac{\star \diamond}{\diamond} \right]$ is to allow the top to add the extra $*$ at the end of the match.

Concluding Remarks

Regular languages

– Finite automata, regular expressions and right-linear grammars

– Pumping Lemma and Myhill-Nerode Theorem

Contex-free languages

– Non-deterministic push down automata and context-free grammars

– Pumping Lemma for context-free languages

Decidable and recursively enumerable languages

– Turing machines [and unrestricted grammars]

– Halting Problem and other undecidable problems

L is regular $\Rightarrow$ L is context-free $\Rightarrow$ L is decidable $\Rightarrow$ L is r.e.

but none of the implications holds in the opposite direction.

Language	regular	context-free	semi-decidable/r.e.
Grammar	right-linear	context-free	unrestricted
rules	$A \to wB, A \to w$	$A \to \alpha$	$\alpha \to \beta$
Machine	DFA or NFA	NPDA	Turing machine
memory	finite	finite + one stack	unrestricted
Other Descr$^{\text{n}}$	Regular expression Myhill-Nerode-Thm		λ-calculus μ-recursive funct$^{\text{s}}$ RM, Hugs, Oberon, …
Example	$\{\, w : w \text{ contains } 10 \,\}$ $\{\, 0^n 1^m : n, m \geq 0 \,\}$	$\{\, w\,w^R \,\}$ $\{\, 0^n 1^n : n \geq 0 \,\}$	$\{\, 0^n 1^m 0^{n*m} : n \geq 0 \,\}$ $\{\, Mx : M \text{ halts on } x \,\}$ HP
Counterexample	$\{\, 0^n 1^n : n \geq 0 \,\}$	$\{\, 0^n 1^n 0^n : n \geq 0 \,\}$ $\{\, w\,w \,\}$	$\{\, M : M \text{ halts on every } x \,\}$ TOTAL
Application	Tokenizer Model checker	Parser	General computing