91.304 Foundations of (Theoretical) Computer Science

Chapter 5 Lecture Notes (Remainder)

David Martin (with modifications by Karen Daniels)
dm@cs.uml.edu
With thanks to Giam Pecelli

This work is licensed under the Creative Commons Attribution-ShareAlike License. To view a copy of this license, visit http://creativecommons.org/licenses/by-sa/2.0/ or send a letter to Creative Commons, 559 Nathan Abbott Way, Stanford, California 94305, USA.
Computation Histories

- Recall the notion of **configuration**: (state, head_position, tape_contents).
- In a deterministic TM every transition takes us from one configuration to the next.
- The idea of "**computation history**" is based on sequences of configurations.
Computation Histories

Definition 5.5: Let M be a TM and w a string:

- An **accepting computation history** for M on w is a sequence of configurations $C_1, C_2, ..., C_k$, where $C_1$ is the start configuration of M on w, $C_k$ is an accepting configuration of M, and each $C_i \rightarrow C_{i+1}$ according to the rules of M.

- A **rejecting computation history** for M on w is defined similarly, except that $C_k$ is a rejecting configuration for M.
Computation Histories

**Definition 5.6**: A *linear bounded automaton* is a standard TM whose tape head is not allowed to move beyond the tape squares containing the input.

**Note**: If the tape alphabet is larger than the input alphabet, the available memory can be increased by a constant factor - but the amount of memory remains a linear function of the length of the input.
Linear Bounded Automata

- Example (using left and right end markers): *(board work)*

- They are powerful: the following are LBAs:
  - Decider for $A_{DFA}$
  - Decider for $A_{CFG}$
  - Decider for $E_{DFA}$
  - Decider for $E_{CFG}$

- Every CFL can be decided by an LBA.
Computation Histories

**Why bother?** LBAs can be shown sufficient for the recognition of almost all "realistic" classes of languages, in particular CFLs:
- roughly, your C program will compile (actually, parse, but who's counting?) in a ("small") constant multiple of the number of bytes necessary to store it in memory.

**Definition:**
\[ A_{LBA} = \{ \langle M, w \rangle \mid M \text{ is an LBA that accepts } w \} \]

**Question:** is \( A_{LBA} \) decidable? The "general" problem \( A_{TM} \) is not...
Lemma 5.8: Let M be an LBA with q states and g symbols in the tape alphabet. Then there are exactly $q^ng^n$ distinct configurations of M for a tape of length n.

Proof. A configuration is uniquely determined
1. by the state M is in (q possibilities),
2. by the head position (n possibilities) and
3. by the tape contents ($g^n$ possibilities: $g$ possibilities for each of the n tape positions).
Computation Histories

**Theorem 5.9.** $\mathsf{A}_{\text{LBA}}$ is decidable.

**Proof.** We construct the following TM $D$:

1. On the first tape, store the input $\langle M, w \rangle$.
2. On the second tape, copy $w$.
3. On the third tape write $1^{qngn}$ (so, unary: $D$ extracts $q$ and $g$ from $M$, $n$ from $w$). Leave the head on the last $1$.
4. Using the second tape, simulate $M$ on $w$.
5. If $M$ halts in the accepting or rejecting state with the head of tape 3 on a 1, accept or reject.
6. After each time an instruction of $M$ is simulated, write a blank on tape 3 and move that head left.
7. If the head of tape 3 reaches $\$, reject.
Computation Histories

Proof (continued). The reason why this works is that after qng^n configurations we **must** be repeating ourselves, and thus have fallen into a loop.

QED
Computation Histories

**Theorem 5.10:**

\[ E_{LBA} = \{ \langle M \rangle \mid M \text{ is an LBA where } L(M) = \emptyset \} \]

is undecidable. \((\not\in \Sigma_0)\)

**Theorem 5.13:**

\[ \text{ALL}_{CFG} = \{ \langle G \rangle \mid G \text{ is a CFG and } L(G) = \Sigma^* \} \]

is undecidable. \((\not\in \Sigma_0)\)
Computation Histories

**Proof of 5.10** We'll show that $\text{NA}_{\text{TM}} \leq_m \text{E}_{\text{LBA}}$. (textbook reduces from $A_{\text{TM}}$)

- On input $\langle M, w \rangle$, the reduction will construct an LBA, say $B = B_{\langle M, w \rangle}$, such that $M$ doesn't accept $w$ iff $L(B_{\langle M, w \rangle})$ is empty. We will not run $B$ on anything, but we must show that we can **construct** such a $B$ via a Turing Machine from any $\langle M, w \rangle$.

- We construct $B$ so that it will accept input $x$ iff $x$ is an accepting computation history for $M$ on $w$. An accepting computation history is a finite string of the form $\# C_1 \# C_2 \# \ldots \# C_k \#$ where
  1. $C_1$ is the starting configuration of $M$ with input $w$
  2. for all $i$, the relation $C_i \vdash C_{i+1}$ holds, according to $M$
  3. $C_k$ is an accepting configuration of $M$. 
Computation Histories

- One can build the initial string $q_0w$ into $B_{(M,w)} \mu$ so the check can be performed.
- One can check that the last configuration contains the accepting state - just scan down the input string of configurations.
- One can check that each configuration follows legally from the previous one: check that they are identical except for the changes implied by the transition function of $M$. Move back and forth marking with a dot the positions just compared. When we find differences (they can be only over three adjacent characters), we check that they meet the requirements of the transitions of $M$. 


Computation Histories

- If all conditions are satisfied, then $B_{\langle M, w \rangle}$ accepts this proposed computation history $x$, otherwise it rejects.

It should be reasonably clear that

a) Given $\langle M, w \rangle$ as input, the reduction can produce a $B_{\langle M, w \rangle}$ as output with this behavior.

b) Such a $B_{\langle M, w \rangle}$ is always an LBA (why?).

And now:

$\langle M, w \rangle \in \text{NA}_{\text{TM}} \iff M$ doesn't accept $w$

$\iff$ there is no accepting computation history for $M$ on $w$

$\iff B_{\langle M, w \rangle}$ does not accept any input

$\iff B_{\langle M, w \rangle} \in \text{ELBA}$

QED
Computation Histories

**Theorem 5.13** \( \text{ALL}_{\text{CFG}} = \{ \langle G \rangle \mid G \text{ is a CFG and } L(G) = \Sigma^* \} \) is undecidable.

**Proof.** Will show that \( \text{NATM} \leq_m \text{ALL}_{\text{CFG}} \) using computation histories: given \( \langle M, w \rangle \) construct a CFG \( G_{\langle M, w \rangle} \) that generates all strings \( \iff M \) does not accept \( w \).

If \( M \) does accept \( w \), \( G_{\langle M, w \rangle} \) will not generate one string: the accepting computation history for \( M \) on \( w \).

We must show that we can construct this \( G_{\langle M, w \rangle} \) via a TM (the reduction) when it is given \( \langle M, w \rangle \) as input.

Again, an accepting computation history of \( M \) on \( w \) is a finite string of the form \( \# C_1 \# C_2 \# \ldots \# C_k \# \) where

1. \( C_1 \) is the starting configuration of \( M \) with input \( w \)
2. for all \( i \), the relation \( C_i \vdash C_{i+1} \) holds, according to \( M \)
3. \( C_k \) is an accepting configuration of \( M \)
Computation Histories

For a string not to be an accepting configuration history for M on w, at least one of the three conditions must fail. If M does not accept w, no such computation history exists, thus all strings fail, and $G_{\langle M, w \rangle}$ would generate all strings.

We construct the grammar $G_{\langle M, w \rangle}$ using the results that establish equivalence between CFGs and PDAs (all the constructions are algorithmic - thus TM performable, so we are OK).
Computation Histories

The reduction constructs a PDA $D_{(M,w)}$. It is nondeterministic, so at start we have 3 branches. This machine will accept strings that are not accepting computation histories:

1. One branch checks if the beginning of the input is $C_1$. Accepts if not.

2. One branch checks if the end $C_k$ contains $q_{\text{accept}}$. Accepts if not.

3. One branch scans the input until it decides (non-deterministically) that is has reached a $C_i$. It pushes the contents of $C_i$ onto the stack, and verifies that the stack contents match $C_{i+1}$ except for the modifications due to the transition. Accepts if not.
Computation Histories

The only problem is that we would be trying to match potentially unbounded strings such as $C_i \# C_{i+1}$, and that's something we can't do in a PDA, because stacks are LIFO structures. Yet if adjacent configurations were written in reverse order of each other, it would be easy to do.

This turns out to be a non-issue. For each normal accepting computation history

\[
# \ C_1 \ # \ C_2 \ # \ C_3 \ # \ C_4 \ # \ ... \ # \ C_k \ #
\]

there is another string that is the same except that it reverses every other component:

\[
# \ C_1 \ # \ C_2^R \ # \ C_3 \ # \ C_4^R \ # \ ... \ # \ C_k \ #
\]

So our PDA $D_{\langle M, w \rangle}$ will just assume that its input is in this form rather than the ordinary form of a computation history. Then we convert our PDA into the CFG $G_{\langle M, w \rangle}$, and we're done. \textbf{QED}
Using Similar Strategy...

☐ $\text{EQ}_{\text{CFG}}$ is undecidable.

- Textbook Exercise 5.1, p. 211.
- Recall:
  ☐ $\text{EQ}_{\text{CFG}} = \{ < G, H > \mid G \text{ and } H \text{ are CFGs and } L(G) = L(H) \}$
Landscape at end of Chapter 5

Each point is a language in this Venn diagram.
A natural undecidable language

- **Post’s Correspondence Problem** concerns *dominos* of the form

  \[
  \begin{array}{ccc}
  a & a & cb \\
  b & ab & aa \\
  \end{array}
  \]

  where each domino has a “top” and a “bottom” chosen from an underlying alphabet \( \Sigma \).

- An **instance** of the problem is a *finite* set \( P = \{ [t_1, b_1], [t_2, b_2], \ldots, [t_k, b_k] \} \) of dominos
  - Each \( t_i \) and \( b_i \) is a *nonempty* string over \( \Sigma \) (in other words, a member of \( \Sigma^+ \))
  - Think of each instance \( P \) as an alphabet
  - You can form *domino strings* over the alphabet \( P \)
Post’s Correspondence Problem

- Let \( \Sigma = \{a,b,c\} \) and \( P_1 = \{[b,ca], [a,ab], [ca,a], [abc,c]\} \). Sample domino string over \( P_1 \):
  - \( \text{ca abc} \)
  - \( \text{a abc} \)
  - \( \text{a c} \)

- A domino string is a match if the concatenation of characters on its top row is equal to the concatenation of characters on its bottom row.
- Repetitions of dominos are permitted.
- The example above is not a match, because \( \text{caabc} \neq \text{ac} \).
- But we can find a match using this instance \( P_1 \). One thing we know for sure: each match must begin with same character on top & bottom…
Post’s Correspondence Problem

- An instance $P$ contains a match if there is some nonempty domino string formed from $P$ that is a match.

- **Definition**
  \[ PCP = \{ <P> \mid P \text{ is an instance that contains a match} \} \]

- **Theorem 5.15** $PCP \notin \Sigma_0$
Proof outline

**Proof** Define the **Modified PCP**: 

\[ \text{MPCP} = \{ <P, d_1> \mid P \text{ is an instance that contains a match that begins with the domino } d_1 \in P \} \]

We’ll show that \( A_{TM} \leq_m \text{MPCP} \leq_m PCP \). Most of the work is in the first reduction, which we’ll call \( f \).

The idea is that \( <M,w> \in A_{TM} \) exactly when \( M \) goes from its initial configuration on input \( w \) into an accepting one:

\[ <M,w> \in A_{TM} \iff q_0 \xrightarrow{*}^* q_{acc} \text{ for some } x, y \in \Gamma^* \]

We’ll construct the reduction \( f \) so that it transforms \( <M,w> \) into an instance of MPCP in which a **match actually describes an accepting computation history of \( M \) on \( w \)**. The match is not a one-for-one rewriting of the computation history, but it's quite close.
Definition of reduction $f$

A. $f$: input $x$

B. if $x$ is not of form $<M,w>$ where $M$ is a TM, then print out some fixed string that we know isn’t in MPCP and halt

C. modify $M=(Q,\Sigma,\Gamma,\delta,q_0,q_{acc},q_{rej})$ so that:
   A. it still recognizes the same language as before
   B. but it never attempts to move left of its leftmost tape cell (substitute right-then-left motion instead)

D. construct $P=\{d_1,\ldots,d_k\}$ using rules 1-7 (on next slides…)

E. print $<P,d_1>$ and halt
Rule 1 of 7: initial configuration

- This is the initial domino and it describes M's initial configuration:
  \[ d_1 = \begin{array}{c} \# \text{ old config on top} \\ \# q_0 w \# \text{ new config on bottom} \end{array} \]

- This is the only domino that is bottom-heavy, meaning that it has more characters on the bottom than on the top.

- To get a match, there must also be dominos in the kit that are top-heavy to compensate.

- But most of the dominos will be perfectly balanced.

- This is the only domino that depends on the string w. The rest are either constant or depend on M alone.
Example TM $M$

$\Sigma = \{1,2,3\}$
$\Gamma = \{1,2,3,\$, $\sqcup\}$

Here $\Gamma$ on transition label represents any character of tape alphabet.
Also, $\Gamma / R$ represents $\Gamma \to R$. 

Unlabeled transitions point to $q_{rej}$

Transitions out of $q_{acc}$ and $q_{rej}$ are never actually taken...
Rule 1:  
#  
#q_0w#  

Example kit for  
<\text{M},w>
Rule 2 of 7: right moves

- Suppose $\delta(q, a) = (r, b, R)$. Then we want to ensure that a matched set of dominos can include this computation history snippet:
  
  ... qa ... $\vdash$ ... br ...

  (state is now $r$, $a$ is now $b$, moved to the right)

- So, for all $a, b \in \Gamma$ and $q \in Q-\{q_{\text{rej}}\}$ and $r \in Q$ and $\delta(q, a) = (r, b, R)$, we add the following domino to the set $P$:

  \[
  \begin{array}{l}
  qa \quad \text{old config} \\
  br \quad \text{new config}
  \end{array}
  \]

- Note: If $q_{\text{rej}}$ appears in a configuration, then we want to prevent a match, so we exclude dominos that might help make a partial match grow towards a full match.

- Note: If $q_{\text{acc}}$ appears in a configuration, then we'll use rules 6 and 7 to proceed towards a match.

- Note: $\# \notin \Gamma$. It's in the domino alphabet but not the TM alphabet.
Rule 1:
#
#q_0w#

Rule 2:
q_01  q_02  q_03  q_0$  q_0 \sqcup
Right moves
$q_1$  $q_1$  $q_1$  $q_1$  $q_1$
$q_11$  $q_1 \sqcup$  $q_1$
2$q_1$  3$q_1$  $q_{\text{rej}}$
$q_{\text{acc}}1$  $q_{\text{acc}}2$  $q_{\text{acc}}3$  $q_{\text{acc}}$  $q_{\text{acc}} \sqcup$
1$q_{\text{acc}}$  2$q_{\text{acc}}$  3$q_{\text{acc}}$  $q_{\text{acc}}$  $q_{\text{acc}} \sqcup$

Example kit for
$<M,w>$
Rule 3 of 7: left moves

- Suppose $\delta(q, a) = (r, b, L)$. Then we want to ensure that a matched set of dominos can include this computation history snippet:

  ... cqa ... $\vdash$ ... rcb ...

  (state is now r, a is now b, moved to the left, c is unchanged)

- So, for all $a, b, c \in \Gamma$ and $q \in Q - \{q_{\text{acc}}, q_{\text{rej}}\}$ and $r \in Q$ and $\delta(q, a) = (r, b, L)$, we add the following domino to the set $P$:

  \[
  \begin{array}{c}
  \text{cqa} \quad \text{old config} \\
  \text{rcb} \quad \text{new config}
  \end{array}
  \]

- Note that we don't care what c is; we add in one domino for each choice of c.

- Also, remember that M never tries to move left when it's at the left end of the tape. So there always is some character c to the left of the tape head when moving left, and so these dominos account for all of the possible left moves.

---

Why the lack of symmetry between right and left move cases?
Rule 1: #
#q_0 w#

Rule 2: q_0 1 q_0 2 q_0 3 q_0 $ q_0 \sqcup
Right moves $q_1 $q_1 $q_1 $q_1 $q_1
q_1 1 q_1 \sqcup q_1$
2q_1 3q_1 $q_{rej}

Rule 3: 1q_1 2 2q_1 2 3q_1 2 $q_1 2 \sqcup q_1 2
Left moves q_1 11 q_1 21 q_1 31 q_1 $1 q_1 \sqcup 1
1q_1 3 2q_1 3 3q_1 3 $q_1 3 \sqcup q_1 3
q_{acc} 13 q_{acc} 23 q_{acc} 33 q_{acc} $3 q_{acc} \sqcup 3

Example kit for <M,w>
Rule 4 of 7: the faraway tape

- The left move and right move dominoes determine what happens in the immediate vicinity of the tape head. The rest of the tape is unaffected by the transition.
- So, for every $a \in \Gamma$, add this domino to the kit:
  
  $a$
  
  $a$
  
  $a$
  
  $a$
Rule 1: 

#q_{0w}#

Rule 2: 

$q_01 \quad q_02 \quad q_03 \quad q_0$ \quad \text{q}_0 \downarrow

Right moves 

$q_11 \quad q_1 \downarrow \quad q_1$

$2q_1 \quad 3q_1 \quad \text{q}_{\text{rej}}$

$q_{\text{acc}}1 \quad q_{\text{acc}}2 \quad q_{\text{acc}}3 \quad q_{\text{acc}}$ \quad q_{\text{acc}} \downarrow

1$q_{\text{acc}} \quad 2q_{\text{acc}} \quad 3q_{\text{acc}} \quad \text{q}_{\text{acc}} \downarrow \quad \text{q}_{\text{acc}}$

Example kit for $\langle M, W \rangle$

Rule 3: 

Left moves 

$q_{11} \quad q_{12} \quad q_{13} \quad q_{1}$ \quad q_{1}$\downarrow$

$q_{12} \quad q_{13} \quad q_{14} \quad q_{15} \quad q_{16}$ \quad q_{17} \quad q_{18} \quad q_{19} \quad q_{10} \quad q_{11}$

Rule 4: 

Copying 

1 \quad 2 \quad 3 \quad \$ \quad \text{\downarrow}$
Rule 5 of 7: the # separators

- Add these two dominos to the kit:

  | # | # | old config |
  | # | □ | # | new config |

- These allow the separators to be carried along.

- The second domino allows a blank to be added to the right of the tape.
Rule 1: 
# 
#q0w#

Rule 2: 
q01 q02 q03 q0$ q0 \sqcup 
Right moves $q_1$ $q_1$ $q_1$ $q_1$ $q_1$

q1$ q1 \sqcup q1$

2q1 3q1 $q_{rej}$

q_{acc}1 q_{acc}2 q_{acc}3 q_{acc}$ q_{acc} \sqcup 

1q_{acc} 2q_{acc} 3q_{acc} $q_{acc} \sqcup q_{acc}$

Example kit for <M,W>

Rule 3: 
1q12 2q12 3q12 $q_12 \sqcup q_12$
Left moves q111 q121 q131 q1$1 q1 \sqcup 1$

1q13 2q13 3q13 $q_13 \sqcup q_13$

q_{acc}13 q_{acc}23 q_{acc}33 q_{acc}$3 q_{acc} \sqcup 3$

Rule 4: 
1 2 3 $ \sqcup$
Copying 1 2 3 $ \sqcup$

Rule 5:
Separators # # 
# \sqcup #
Rule 6 of 7: accepter

- For every $a \in \Gamma$, add these two dominos to the kit:
  
  $a q_{acc} \quad q_{acc} a \quad \text{old config}$
  
  $q_{acc} \quad q_{acc} \quad \text{new config}$

- This means that when $q_{acc}$ appears in a configuration, the adjacent characters can be squeezed out, one configuration at a time, leading towards a match.

- These are the top-heavy dominos to match the initial, bottom-heavy one.
Rule 1: 

#q_0w#

Rule 2: 

q_01 q_02 q_03 q_0$ q_0 ⊥

Right moves

$q_1$ $q_1$ $q_1$ $q_1$ $q_1$

q_11 q_1 q_1$

2q_1 3q_1 $q_{rej}$

q_{acc}1 q_{acc}2 q_{acc}3 q_{acc}$ q_{acc} ⊥

1q_{acc} 2q_{acc} 3q_{acc} $q_{acc} ⊥ q_{acc}$

Example kit for <M,w>

Rule 3: 

1q_{12} 2q_{12} 3q_{12} $q_{12}$ $q_{12}$

Left moves

q_{11} q_{12} q_{13} q_{1}$ q_{1}$

1q_{13} 2q_{13} 3q_{13} $q_{13}$ $q_{13}$

q_{acc}13 q_{acc}23 q_{acc}33 q_{acc}$3 q_{acc} ⊥3

Rule 4: 

1 2 3 $ ⊥$

Copying

1 2 3 $ ⊥$

Rule 5: 

Separators # # #

Rule 6: 

1q_{acc} q_{acc}1 2q_{acc} q_{acc}2 3q_{acc} q_{acc}3 $q_{acc} q_{acc}$ $q_{acc} ⊥ q_{acc} q_{acc}$

Acceptors q_{acc} q_{acc} q_{acc} q_{acc} q_{acc} q_{acc} q_{acc} q_{acc} q_{acc} q_{acc}
Rule 7 of 7: finisher

☐ Add this single domino to the kit:

$q_{acc}##$

#

☐ You just need it. You'll see why...
Rule 1: #
#q_{0W}#

Rule 2:
\begin{align*}
q_01 & q_02 & q_03 & q_0$ & q_0 \uparrow \\
\text{Right moves} & \quad & \quad & \quad & \quad \\
$q_11 & q_1 \uparrow & q_1$ & \quad & \quad & \quad & \quad \\
2q_1 & 3q_1 & $q_{\text{rej}} & \quad & \quad & \quad & \quad \\
\end{align*}

Example kit for $<M, W>$

Rule 3:
\begin{align*}
1q_12 & 2q_{\text{acc}} & 3q_{\text{acc}} & q_{\text{rej}} & q_{\text{acc}} \uparrow \\
\text{Left moves} & \quad & \quad & \quad & \quad \\
q_11 & q_121 & q_131 & q_1$1 & q_1 \uparrow 1 \\
\end{align*}

Rule 4:
\begin{align*}
1 & 2 & 3 & $ & \uparrow \\
\text{Copying} & \quad & \quad & \quad & \quad \\
1 & 2 & 3 & $ & \uparrow \\
\end{align*}

Rule 5:
\begin{align*}
\text{Separators} & \quad # & \quad # \\
& \quad # & \quad \uparrow # \\
\end{align*}

Rule 6:
\begin{align*}
1q_{\text{acc}} & q_{\text{acc}}1 & 2q_{\text{acc}} & q_{\text{acc}}2 & 3q_{\text{acc}} & q_{\text{acc}}3 & q_{\text{acc}} & q_{\text{acc}}$ & q_{\text{acc}} \uparrow \\
\text{Acceptors} & \quad q_{\text{acc}} & q_{\text{acc}} & q_{\text{acc}} & q_{\text{acc}} & q_{\text{acc}} & q_{\text{acc}} & q_{\text{acc}} & q_{\text{acc}} \\
\end{align*}

Rule 7:
\begin{align*}
q_{\text{acc}} & \quad # & \quad # \\
\text{Finisher} & \quad # \\
\end{align*}
Example TM

\[ \Sigma = \{1, 2, 3\} \quad \Gamma = \{1, 2, 3, $, \sqcup\} \]

- \( \Gamma / $, R \) from \( q_0 \) to \( q_1 \)
- \( 3 / 3, L \) from \( q_0 \) to \( q_{\text{acc}} \)
- \( 1 / 2, R \) from \( q_1 \) to \( q_1 \)
- \( 2 / 1, L \) from \( q_1 \) to \( q_{\text{acc}} \)
- \( \sqcup / 3, R \) from \( q_1 \) to \( q_{\text{rej}} \)
- \( $ / $, R \) from \( q_{\text{acc}} \) to \( q_{\text{rej}} \)
- \( \Gamma / R \) from \( q_{\text{acc}} \) to \( q_{\text{acc}} \)
- \( \Gamma / R \) from \( q_{\text{rej}} \) to \( q_{\text{rej}} \)

Unlabeled transitions point to \( q_{\text{rej}} \)

Transitions out of \( q_{\text{acc}} \) and \( q_{\text{rej}} \) are never actually taken...
Examples

- On board. 〈M,3123〉 and 〈M,13〉
MPCP ≤ₘ PCP via g

1. g: input <P, t₁/b₁> where P is a PCP instance and t₁/b₁ ∈ P

□ if input is not in this form then print some string known not to be in PCP and halt

2. Assuming P={t₁/b₁, ..., tₖ/bₖ},
   let Q = { *t₁/∗b₁*, *t₁/b₁*, *t₂/b₂*,
   *t₃/b₃*,
   ⋮,
   *tₖ/bₖ*,
   +◊/◊ }

   where +, ◊ are not in Σ and
   *abc = +a+b+c (plusses before)
   abc* = a+b+c+ (plusses after)
   *abc* = +a+b+c+ (before & after)

3. Print <Q> and halt

---

different from textbook, where * and 5-pointed star are used
Purpose of construction

- Each domino has some + characters on the top
  - Only one of them has + as the first character on the top and bottom: the one that includes $t_1$ and $b_1$
- Starting with this domino $*t_1/*b_1*$, you end up with an excess of + characters on the bottom
  - The only one that can balance it is $+\diamond$/ $\diamond$
- Otherwise, the + characters line up between each character and don’t interfere
- The effect is that $P$ contains a match starting with $t_1/b_1$ exactly when $Q$ contains any match whatsoever (since any match in it must begin with $*t_1/*b_1*$)
- Hence $\text{MPCP} \leq_m \text{PCP via } g$
Landscape at end of Chapter 5

Each point is a language in this Venn diagram.