文脈自由言語の普遍性判定問題
Universality Problem for Context-Free Languages

y.*

2018 年 9 月 3 日

最終更新日: 2018 年 9 月 3 日

概要

プッシュダウンオートマトンによって認識される言語のクラスを文脈自由言語という．与えられた言語 L が文字列全体 Σ^* に一致するかどうかを判定する問題を普遍性判定問題という．本稿では文脈自由言語に関する普遍性判定問題が決定不能であることを証明する．

Keywords: 文脈自由言語 (context-free language), プッシュダウンオートマトン (pushdown automaton), 文脈自由文法 (context-free grammar), 普遍性判定問題 (universality problem), 等価性判定問題 (equality problem), 包含判定問題 (inclusion problem).

1　プッシュダウンオートマトンと文脈自由言語

まず，プッシュダウンオートマトンについて簡単に復習しておく．詳細は Sipser [3] などの教科書を参照のこと．プッシュダウンオートマトンを知らない読者は以下の定義は飛ばして先に例 1.2 を読むことをおすすめする．

定義 1.1 (プッシュダウンオートマトン [3, 定義 2.13], 文脈自由言語). **(非決定性) プッシュダウンオートマトン** ((nondeterministic) pushdown automaton) とは，以下のデータからなる 6 つ組 $A = (Q, \Sigma, \Gamma, \delta, q_0, F)$ である:

1. 状態集合と呼ばれる有限集合 Q,
2. 入力アルファベットと呼ばれる有限集合 Σ,
3. スタックアルファベットと呼ばれる有限集合 Γ,
4. 遷移関数 $\delta\colon Q \times (\Sigma \cup \{\varepsilon\}) \times (\Gamma \cup \{\varepsilon\}) \to \mathcal{P}(Q \times (\Gamma \cup \{\varepsilon\}))$ (ここで ε は空文字列，$\mathcal{P}$ は冪集合を表す),
5. 開始状態 $q_0 \in Q$,
6. 受理状態の有限集合 $F \subseteq Q$.

プッシュダウンオートマトン A の計算は形式的には次のように定義される．入力文字列 $w \in \Sigma^*$ に対して，以下の条件を満たすような w の分解 $w = w_0 w_1 \cdots w_m$ $(w_i \in \Sigma \cup \{\varepsilon\})$ と状態のある列 $r_0, r_1, \ldots, r_m \in Q$ とスタックの内容のある列 $s_0, s_1, \ldots, s_m \in \Gamma^*$ が存在するとき，A は w を受理するという:

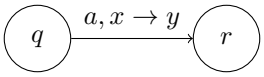
* http://iso.2022.jp/

1. $r_0 = q_0$ かつ $s_0 = \varepsilon$,
2. 各 i に対し，ある $t \in \Gamma^*$ があって $\delta(r_i, w_i, x) \ni (r_{i+1}, y)$ かつ $s_i = xt, s_{i+1} = yt$ である．
3. $r_m \in F$.

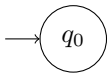
A が w を受理しないとき，A は w を拒否するという．

感覚的なことを言えば，$\delta(q, a, x) \ni (r, y)$ というのは，A の状態が q，未読文字列の先頭の文字が a，スタックのトップの文字が x であるとき，「文字 a を読み込み (“消費” し)，A の状態を r に変更し，さらにスタックのトップの文字を x から y に書き換え」てもよいことを表す．ここで，$x = \varepsilon$ のときは「スタックに y をプッシュする」ことを意味し，$y = \varepsilon$ のときは「スタックから x をポップする」ことを意味し，さらに $x = y = \varepsilon$ のときは「スタックの内容を変化させない」ことを意味する．w に対してうまく計算の経路を選んで受理状態に至ることができるとき，A は w を受理する．入力文字列を読み込み終わったときに受理状態になかった場合だけでなく，途中で可能な遷移がなくなり，入力文字列を消費しきれなかった場合にも A は w を拒否することに注意せよ．[*1]

プッシュダウンオートマトンを作る際に集合を使って書き下すのは骨が折れる (し，何より読みづらい) ので，以下ではもっぱら**状態遷移図** (state diagram) を用いる．状態遷移図とは，Q の各状態 q を頂点とした (ラベル付きの有向) グラフであって，$\delta(q, a, x) \ni (r, y)$ のときに



のように辺を描いたものである．また，開始状態は



のように表し，受理状態 $q \in F$ は

のように二重丸で表すものとする．

アルファベット Σ 上の**言語** (language) とは，Σ^* の部分集合のことをいう．プッシュダウンオートマトン A に対し，A **の言語** (language of A) $L(A)$ を $L(A) := \{ w \in \Sigma^* \mid A \text{ は } w \text{ を受理する} \}$ とおく．A は $L(A)$ を**認識する** (recognize) ともいう．言語 L があるプッシュダウンオートマトン A によって $L = L(A)$ と表されるとき，L を**文脈自由言語** (context-free language) という．

例 1.2 ([3, 例 2.16]). $\Sigma = \{\mathtt{a}, \mathtt{b}, \mathtt{c}\}, \Gamma = \{\mathtt{x}, \mathtt{\$}\}$ とする．状態遷移図 1 から定まるプッシュダウンオートマトン A が認識する言語は $L(A) = \{ \mathtt{a}^i\mathtt{b}^j\mathtt{c}^k \mid i, j, k \geq 0 \text{ かつ } [i = j \text{ または } i = k] \}$ である．実際，A はまず a と同じ数だけの x をスタックに積んだ後，$i = j$ であるか $i = k$ であるかを非決定的に “推測” し，$i = j$ の場合には b の数がスタックに積まれた x の数と一致するかを検査し，$i = k$ の場合には c の数がスタックに積まれた x の数と一致するかを検査する．スタックアルファベットの$はスタックの “底” を検出するための記号である．

[*1] このような非決定性に関しては「Turing 機械の変種」[2] の非決定性 Turing 機械も参照のこと．ただし，Turing 機械の場合と異なり，非決定性プッシュダウンオートマトンは決定性プッシュダウンオートマトンより計算能力が真に強いことが知られている．

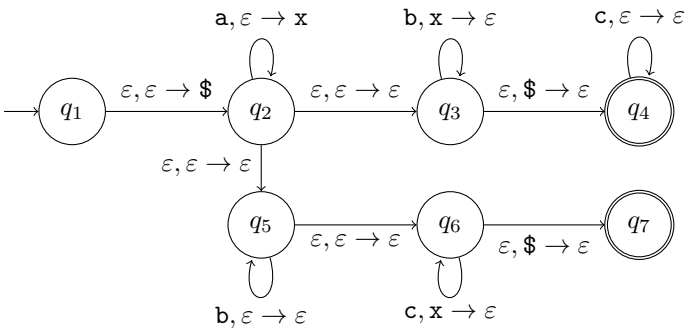


図 1　プッシュダウンオートマトン A

例えば，入力文字列 $w = \texttt{aaabbccc} \in L(A)$ に対する A の受理計算履歴は表 1 のようになる．

表 1　A の aaabbccc に対する受理計算履歴

ステップ数	状態	未読文字列	スタックの内容
0	q_1	aaabbccc	ε
1	q_2	aaabbccc	$
2	q_2	aabbccc	x$
3	q_2	abbccc	xx$
4	q_2	bbccc	xxx$
5	q_5	bbccc	xxx$
6	q_5	bccc	xxx$
7	q_5	ccc	xxx$
8	q_6	ccc	xxx$
9	q_6	cc	xx$
10	q_6	c	x$
11	q_6	ε	$
12	q_7	ε	ε

プッシュダウンオートマトン A が与えられると，A から定まる文脈自由言語 $L = L(A)$ に関する決定問題を考えることができる．最も基本的な問題は，与えられた文字列が L に属するかどうかを判定する問題であるが，これは決定可能であることが知られている．

問題 1.3 (文脈自由言語の所属判定問題 (membership problem for context-free languages)).
Input: 文脈自由言語 L と文字列 w [*2]
Question: $w \in L$ か？

定理 1.4 ([4, 定理 4.7]). 文脈自由言語の所属判定問題 1.3 は決定可能である．

証明の概略. $L = L(A)$ なるプッシュダウンオートマトン A をとると，A に対応する**文脈自由文法** (context-

[*2] 以降，文脈自由言語 L が入力されると書いたら，実際には L を定めるプッシュダウンオートマトンの記述が入力されるとみなす．

free grammar) G を構成することができる．さらに，G を等価な **Chomsky 標準形** (Chomsky normal form) に変換することができる．このとき，G による w の導出は存在すればちょうど $2|w|-1$ ステップであるので (ここで $|w|$ は w の長さを表す)，G による $2|w|-1$ ステップの導出を全て列挙し，その中に w が現れるかを確かめればよい． □

所属判定の次に基本的な問題として，A が何らかの文字列を受理するかどうか，すなわち $L(A)$ が空集合かどうかを問う問題がある．これも決定可能であることが知られている．

問題 1.5 (文脈自由言語の空性判定問題 (emptiness problem for context-free languages)).
Input: 文脈自由言語 L
Question: $L = \emptyset$ か？

定理 1.6 ([4, 定理 4.8]). 文脈自由言語の空性判定問題 1.5 は決定可能である．

証明の概略. $L = L(A)$ なるプッシュダウンオートマトン A をとると，A に対応する文脈自由文法 G を構成することができる．G の各変数記号 U に対し，「終端記号のみからなる文字列を U から導出できるならば U に印を付ける」という操作を，新しく印を付けることのできる変数がなくなるまで繰り返す．最終的に，開始変数に印が付いていれば受理し，そうでなければ拒否する． □

これとは対照的に，A があらゆる文字列を受理するかどうかという問題は決定不能になることを次節で証明する．

2 普遍性判定問題

本節では，与えられたプッシュダウンオートマトン A があらゆる文字列を受理するかどうかという決定問題 (普遍性判定問題) が決定不能になることを示す．

問題 2.1 (文脈自由言語の普遍性判定問題 (universality problem for context-free languages)).
Input: 文脈自由言語 L
Question: $L = \Sigma^*$ か？

定理 2.2 ([4, 定理 5.13]). 文脈自由言語の普遍性判定問題 2.1 は決定不能である．

証明のアイデア. 証明には停止問題の決定不能性を用いる．停止問題の決定不能性の証明は「Turing 機械の定義と停止問題」[1] を参照のこと．停止問題への入力を M, w とする．このとき，プッシュダウンオートマトン A_M を

$$M \text{ が } w \text{ を受理する} \iff L(A_M) \neq (\Sigma^A)^* \tag{1}$$

という条件を満たすように構成する (ここで Σ^A は A_M の入力アルファベットである)．M が w を受理すると仮定して，その受理計算履歴を $C_0, C_1, \ldots, C_s$ とするとき，入力文字列 $C_0\#C_1\#\cdots\#C_s\#$ を拒否するように A_M を設計したい．つまり A_M に，与えられた文字列が M の w に対する受理計算履歴でないことを検出させたい．#で区切られた文字列 $w^A = C_0\#C_1\#\cdots\#C_s\#$ (各 C_i は ε かもしれない) が受理計算履歴でないのは，w^A が次のいずれかの “欠陥” を有するときである:

1. C_0 が開始状況でない．

2. ある i について，C_i が計算状況を表していないか，または C_i から C_{i+1} に M の遷移関数によって正しく遷移できていない．
3. C_s が受理状況でない．
4. w^A が#で終わっていない．

A_M は入力文字列 w^A が上の欠陥のうちのどれを持つのかを，非決定性によって “推測” する．実際に w^A に欠陥があれば A は w^A を受理する．

上では受理計算履歴を#でつなげて表したが，実際には A_M がスタックをうまく活用できるように，各計算状況は

$$\underbrace{\longrightarrow}_{C_0}\#\underbrace{\longleftarrow}_{C_1^{\mathcal{R}}}\#\underbrace{\longrightarrow}_{C_2}\#\underbrace{\longleftarrow}_{C_3^{\mathcal{R}}}\#\cdots\#\underbrace{\qquad}_{C_s}\#$$

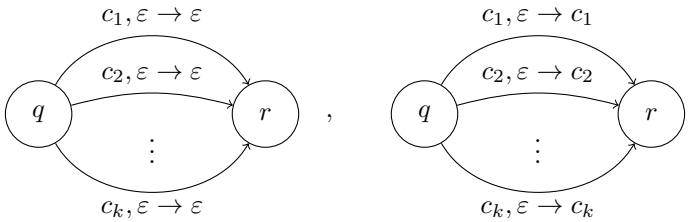
のように交互に逆向きに配置されるとする (ここで，$C^{\mathcal{R}}$ は C を逆順に並べた文字列を表す)． □

証明. 停止問題への入力を $M = (Q, \Sigma, \Gamma, \delta, q_0, q_{\text{accept}}, q_{\text{reject}}), w = w_1 w_2 \cdots w_n$ とする．ただし，w が 2 文字未満のときは末尾に␣を付け加えて $n \geq 2$ となるようにしておく．証明を簡潔にするため，以下の条件を仮定する．

- M はヘッドがテープの左端を見ている状態でさらにヘッドを左に動かそうとすることはない．
- 各計算状況 C_i は
$$a_{-l} \cdots a_{-1} q a_0 a_1 \cdots a_r$$
という形ではなく，
$$a_l \cdots a_{-1} \overset{q}{a_0} a_1 \cdots a_r$$
という形の文字列であるとする ($\overset{q}{a_0}$ は一つの文字)．
- M のヘッドが右に行きすぎた場合だけでなく，C_{i+1} は必ず C_i の末尾に␣を付け加えた形になっていなければならない (つまり，$|C_{i+1}| = |C_i| + 1$ となっている場合に限り正しい遷移とみなす)．

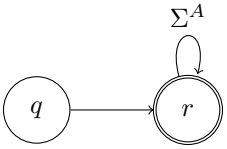
このとき，条件 (1) を満たすプッシュダウンオートマトン $A_M = (Q^A, \Sigma^A, \Gamma^A, \delta^A, q_0^A, F^A)$ を構成する．$\Gamma^Q := \{ \overset{q}{a} \mid q \in Q \setminus \{q_{\text{reject}}\}, a \in \Gamma \}$ とおき，$\Sigma^A := \Gamma \cup \Gamma^Q \cup \{\#\}, \Gamma^A := \Gamma \cup \Gamma^Q \cup \{\$\}$ とおく．A_M への入力文字列を w^A で表す．構成の方針として，A_M の全体を一度に構成するのではなく，機能ごとに分割して小さな**ガジェット** (gadget) を設計し，後でひとつにまとめることにする．

以下では煩雑な状態遷移図をいくつも描くので，楽をするためにいくつかの省略記法を導入しておく．まず，スタックの内容を変化させないような遷移 $\xrightarrow{a, \varepsilon \to \varepsilon}$ は単に $\xrightarrow{a}$ と書くことにする．また，多重辺は 1 本の矢印にまとめ，矢印の上のラベルを集合で表すことにする．例えば，$\Gamma \cup \Gamma^Q = \{c_1, c_2, \ldots, c_k\}$ であるとするとき，

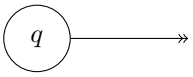


という 2 つの遷移をそれぞれ

のように表す．さらに，受理が確定する状態への遷移

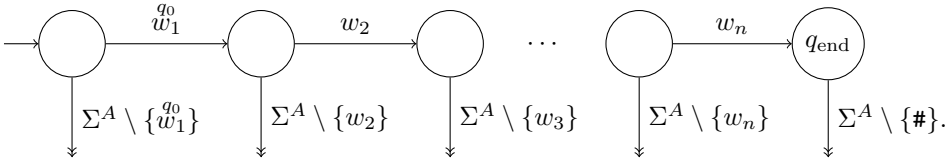


を単に



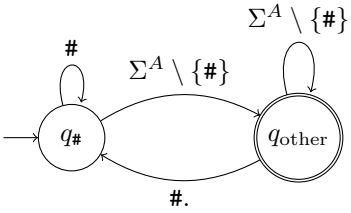
のように省略して書くことにする．

C_0 が開始状況になっていないことを検査するガジェット A_{InitConf}. A_{InitConf} は，入力 w^A が正しい開始状況 $\overset{q_0}{w_1}w_2 \cdots w_n\#$ から始まっていないことを検査する役割を持つ:



A_{initConf} に w^A が入力されると，先頭から 1 文字ずつ $\overset{q_0}{w_1}w_2 \cdots w_n\#$ と比較され，1 文字一致するごとに A_{initConf} は状態が右へ移動することを強制される．w_n まで一致するとついには q_{end} へ到達する．それに加えてさらに次の#まで一致していれば，A_{InitConf} はどうあがいても入力文字列を消費できなくなり，その結果 w^A を拒否することを余儀なくされる．しかし一方で，もし一箇所でも $\overset{q_0}{w_1}w_2 \cdots w_n\#$ と異なっている文字があれば，その時点で受理状態へ “脱出” することができる．

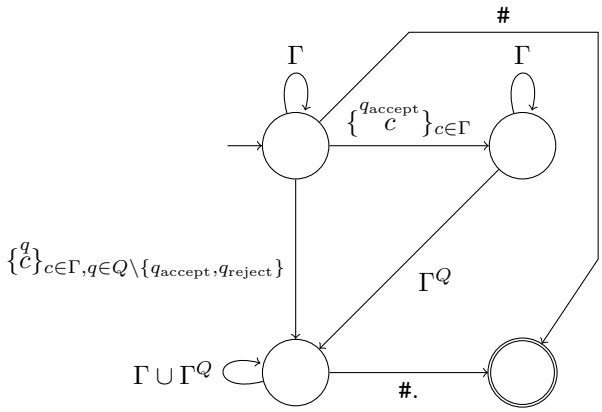
#で終わっていないことを検査するガジェット A_{ValidEnd}. A_{ValidEnd} は w^A が#で終わっているかどうかを検査する役割を持つ:



現在の状態に関係なく，読み込んだ文字が#ならば状態が $q_\#$ になり，それ以外の文字ならば q_{other} になる．ここでは非決定性を使う必要はない．

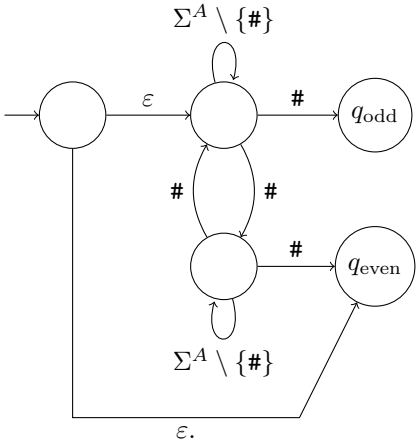
C_s が受理状況になっていないことを検査するガジェット A_{AccConf}. A_{AccConf} は未読文字列が $C_s\#$ の形であって，なおかつ C_s が受理状況でないことを検査する役割を持つ (C_s の向きに関係なく同じ動作でよ

い):



A_{AccConf} は，入力に欠陥があればその時点で状態が下段に移動する．逆に入力に欠陥がない場合には下段に移動することはできず，よって受理することはできない．

#の偶奇によって処理を分岐するガジェット A_{Parity}. A_{Parity} は，w^A の先頭から何文字かを#境界まで消費して，未読文字列を後続のガジェットに引き渡す役割を持ち，このとき消費した#の個数の偶奇によって処理を引き渡す先を変化させる:

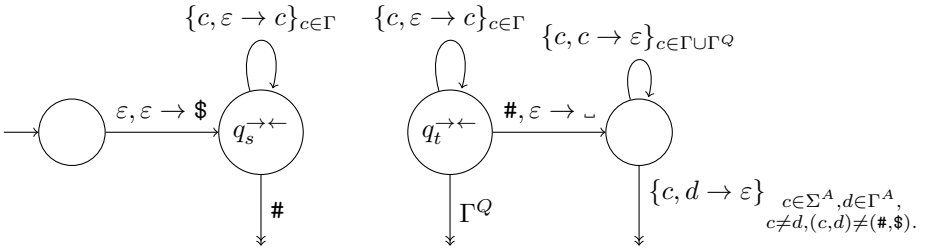


A_{Parity} は消費した#の個数が偶数個 (0 個も含む) ならば状態 q_{even} にいることができ，奇数個ならば q_{odd} にいることができる．

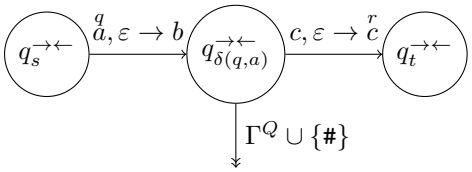
C_i が状況でないか，C_i から C_{i+1} に正しく遷移していないことを検査するガジェット $A^{\rightarrow\leftarrow}_{\mathsf{ValidTrans}}, A^{\leftarrow\rightarrow}_{\mathsf{ValidTrans}}$.
これが最も重要なガジェットであり，スタックを本質的に使用する唯一のガジェットでもある．

$A^{\rightarrow\leftarrow}_{\text{ValidTrans}}$ は未読文字列が $C_i\#C^{\mathcal{R}}_{i+1}\#$ から始まっている (つまり i が偶数である) とき，C_i が計算状況を表していないか，または C_i から C_{i+1} に正しく遷移していないことを検査する役割を持つ．また $A^{\leftarrow\rightarrow}_{\text{ValidTrans}}$ は未読文字列が $C^{\mathcal{R}}_i\#C_{i+1}\#$ から始まっている (つまり i が奇数である) とき，C_i が計算状況を表していないか，または C_i から C_{i+1} に正しく遷移していないことを検査する役割を持つ．この検査の基本的な戦略は「C_i を読み込みながら，遷移後の計算状況をスタックに積み込み，実際の C_{i+1} と 1 文字ずつ比較する」というものである．

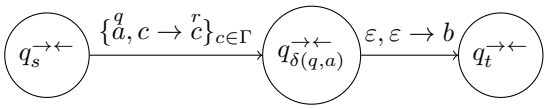
まず，$A^{\rightarrow\leftarrow}_{\mathrm{ValidTrans}}$ を構成する．以下の部品をベースにして，これに辺を付け加えて構成していく:



$\delta(q,a) = (r,b,\mathrm{R})$ となるような全ての $q, r \in Q \setminus \{q_{\mathrm{reject}}\}, a, b \in \Gamma$ に対して，$A^{\rightarrow\leftarrow}_{\mathrm{ValidTrans}}$ に

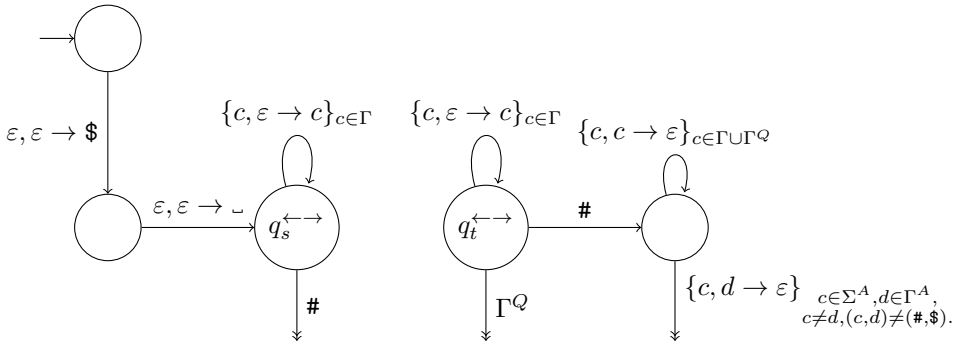


という頂点と辺を追加する．同様に，$\delta(q,a) = (r,b,\mathrm{L})$ となるような全ての $q, r \in Q \setminus \{q_{\mathrm{reject}}\}, a, b \in \Gamma$ に対して，$A^{\rightarrow\leftarrow}_{\mathrm{ValidTrans}}$ に

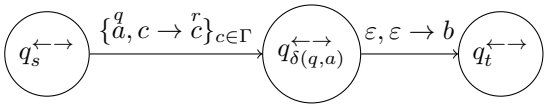


という頂点と辺を追加する．

次に，$A^{\leftarrow\rightarrow}_{\mathrm{ValidTrans}}$ を構成する．先程と同様に，以下の部品をベースにして，これに辺を付け加えて構成していく:

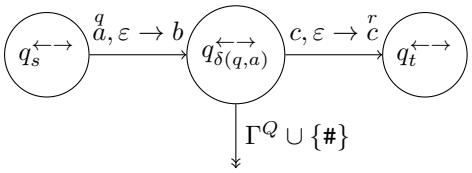


$\delta(q,a) = (r,b,\mathrm{R})$ となるような全ての $q, r \in Q \setminus \{q_{\mathrm{reject}}\}, a, b \in \Gamma$ に対して，$A^{\leftarrow\rightarrow}_{\mathrm{ValidTrans}}$ に



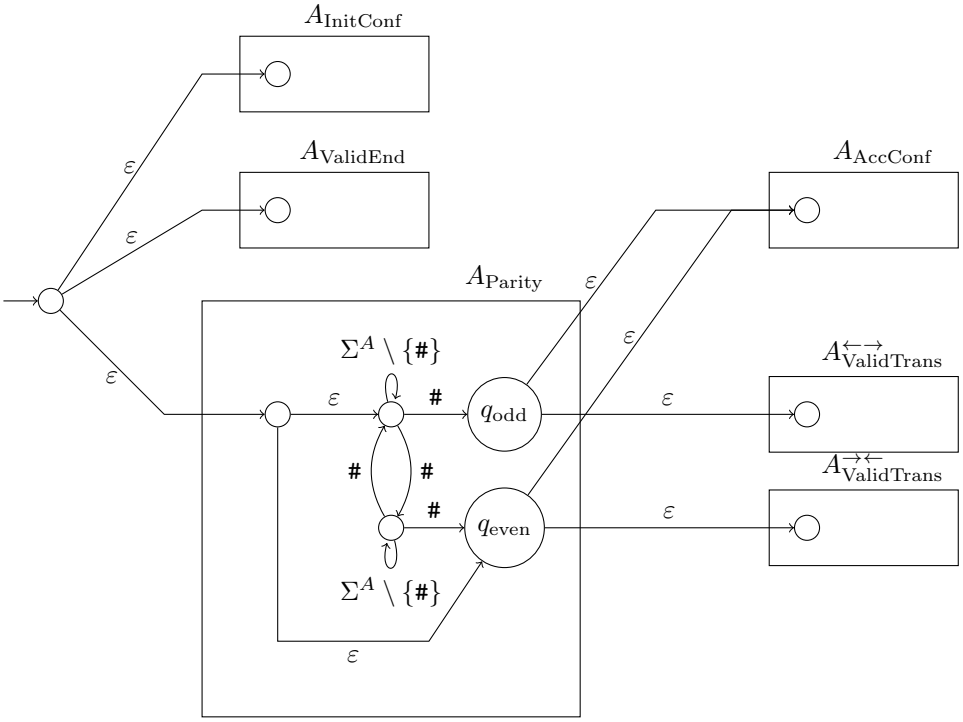
という頂点と辺を追加する．同様に，$\delta(q,a) = (r,b,\mathrm{L})$ となるような全ての $q, r \in Q \setminus \{q_{\mathrm{reject}}\}, a, b \in \Gamma$

に対して，$A^{\leftrightarrow}_{\mathrm{ValidTrans}}$ に



という頂点と辺を追加する．

これらのガジェットをまとめて，以下のように A_M を作る．



この A_M が条件 (1) を満たすことは，A_M の作り方から

$$
\begin{aligned}
M \text{ が } w \text{ を受理する} &\iff M \text{ の } w \text{ に対する，“欠陥” のない受理計算履歴が存在する} \\
&\iff \text{受理計算履歴 } w^A \text{ が全てのガジェットで拒否される} \\
&\iff \text{受理計算履歴 } w^A \text{ が } A_M \text{ に拒否される} \\
&\iff L(A_M) \neq (\Sigma^A)^*
\end{aligned}
$$

となることからわかる． □

例 2.3. M を「Turing 機械の定義と停止問題」 [1] で定義した $\{\, \mathtt{a}^n\mathtt{b}^n \mid n \geq 0 \,\}$ を判定する Turing 機械とする．また，停止問題への入力を $(M, \mathtt{aabb})$ とする．このとき，上で構成した A_M は

$$
\overset{q_0}{\mathtt{a}}\mathtt{abb\#}\,\mathtt{bb}\overset{q_1}{\mathtt{a}}\,\mathtt{\#}\,\mathtt{a}\overset{q_1}{\mathtt{b}}\mathtt{b}\,\mathtt{\#}\,\mathtt{b}\overset{q_1}{\mathtt{b}}\mathtt{a}\,\mathtt{\#}\,\mathtt{abb}\overset{q_1}{\sqcup}\,\mathtt{\#}\,\overset{q_2}{\mathtt{b}}\mathtt{ba}\,\mathtt{\#}\,\mathtt{a}\overset{q_3}{\mathtt{b}}\,\mathtt{\#}
$$

$$
\mathtt{b}\overset{q_3}{\mathtt{a}}\,\mathtt{\#}\overset{q_3}{\sqcup}\mathtt{ab}\,\mathtt{\#}\,\mathtt{b}\overset{q_0}{\mathtt{a}}\,\mathtt{\#}\,\mathtt{b}\overset{q_1}{\sqcup}\,\mathtt{\#}\,\overset{q_1}{\mathtt{b}}\mathtt{b}\,\mathtt{\#}
$$

$$
\mathtt{b}\overset{q_2}{\mathtt{b}}\,\mathtt{\#}\,\overset{q_3}{\sqcup}\,\mathtt{\#}\,\overset{q_0}{\sqcup}\,\mathtt{\#}\,\overset{q_{\mathrm{accept}}}{\sqcup}\,\mathtt{\#}
$$

という入力文字列を拒否するので $L(A_M) \neq (\Sigma^A)^*$ となる．

文脈自由言語が普遍性問題が決定不能であることを用いると，以下の 2 つの問題も決定不能であることが簡単にわかる．

問題 2.4 (文脈自由言語の等価性判定問題 (equality problem for context-free languages)).
Input: 文脈自由言語 L_1, L_2
Question: $L_1 = L_2$ か？

問題 2.5 (文脈自由言語の包含判定問題 (inclusion problem for context-free languages)).
Input: 文脈自由言語 L_1, L_2
Question: $L_1 \subseteq L_2$ か？

系 2.6. 文脈自由言語の等価性判定問題 2.4 と包含判定問題 2.5 は決定不能である．

証明. どちらの問題も $L_1 = \Sigma^*$ とおけば普遍性判定問題が帰着される． □

参考文献

[1] y., Turing 機械の定義と停止問題 (2018), http://iso.2022.jp/math/undecidable-problems/files/turing-machine-and-the-halting-problem.pdf.

[2] y., Turing 機械の変種 (2018), http://iso.2022.jp/math/undecidable-problems/files/variants-of-turing-machine.pdf.

[3] M. Sipser (太田和夫・田中圭介 監訳, 阿部正幸・植田広樹・藤岡淳・渡辺治 訳), 計算理論の基礎 [原著第 2 版] 1. オートマトンと言語, 共立出版, 2008.

[4] M. Sipser (太田和夫・田中圭介 監訳, 阿部正幸・植田広樹・藤岡淳・渡辺治 訳), 計算理論の基礎 [原著第 2 版] 2. 計算可能性の理論, 共立出版, 2008.

変更履歴

2018/09/03　公開