

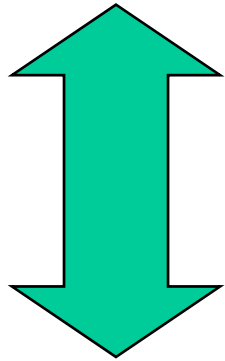
## 6. チューリングマシンの符号化と 計算不可能性

## 6-1. TMの符号化

これまで、チューリングマシンで様々な“計算”が行えることを見てきた。ここでは、チューリングマシンは、一種の“数”であることをみていく。

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

TMの数学的定義



$$\langle T \rangle = 11110 \cdots 0110 \cdots 01 \in \{0,1\}^*$$

TMの符号化

# アイデア

1: を区切り記号をして用いる。

0: を一進数の“数”として用いる。

## 一進数

一種類の記号で数を表す。

10進数

1

2

3

4

5

$n$

一進数

0

00

000

0000

00000

$0^n$

# 状態の符号化

状態を符号化する。  
ここで、状態の名前は重要ではなく、  
状態の数だけが重要である。

$$Q = \{q_1, q_2, \dots, q_l\}$$



$\dots 10^l 1 \dots$

両脇の1は、  
(アルファベット等)  
他の集合との境目  
を意味する。

# アルファベットの符号化

アルファベットを符号化する。  
ここでも、記号の名前は重要ではなく、  
記号の数だけが重要である。

入力アルファベット

$$\Sigma = \{a_1, a_2, \dots, a_m\} \longleftrightarrow \dots 10^m 1 \dots$$

テープアルファベット

$$\Gamma = \{a_1, a_2, \dots, a_m, a_{m+1}, \dots, a_n\} \longleftrightarrow \dots 10^n 1 \dots$$

前半は入力  
アルファベットを  
表す。

## 受理状態の符号化

受理状態の集合を符号化する。  
ここでは、状態の添え字の集合に注目して、  
符号化する。

受理状態集合

$$F = \{q_{i_1}, q_{i_2}, \dots, q_{i_k}\}$$



$$\dots 110^{i_1} 110^{i_2} 11 \dots 110^{i_k} 11 \dots$$

# 初期状態と空白記号の符号化

初期状態は、常に、

$$q_1 \in Q$$

であるとするれば、符号化の必要がない。

空白記号は、常に、

$$B = a_n \in \Gamma$$

であるとするれば、符号化の必要がない。

# 状態遷移関数の符号化

状態遷移関数を符号化する。

ヘッドの移動方向を次のように符号化する。

$$R = 0$$

$$L = 00$$

これにより、一つの状態遷移関数を次のように符号化する。

$$\delta(q_{j_1}, a_{j_2}) = (q_{j_3}, a_{j_4}, 0^{j_5})$$



$$c = 0^{j_1} 10^{j_2} 10^{j_3} 10^{j_4} 10^{j_5}$$

空集合以外の状態遷移関数を符号化する。

$$\delta = \{\delta_1, \delta_2, \dots, \delta_h\}$$

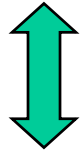


$$\dots 11c_1 11c_2 11 \dots 11c_h 11 \dots$$



これらをまとめて、

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

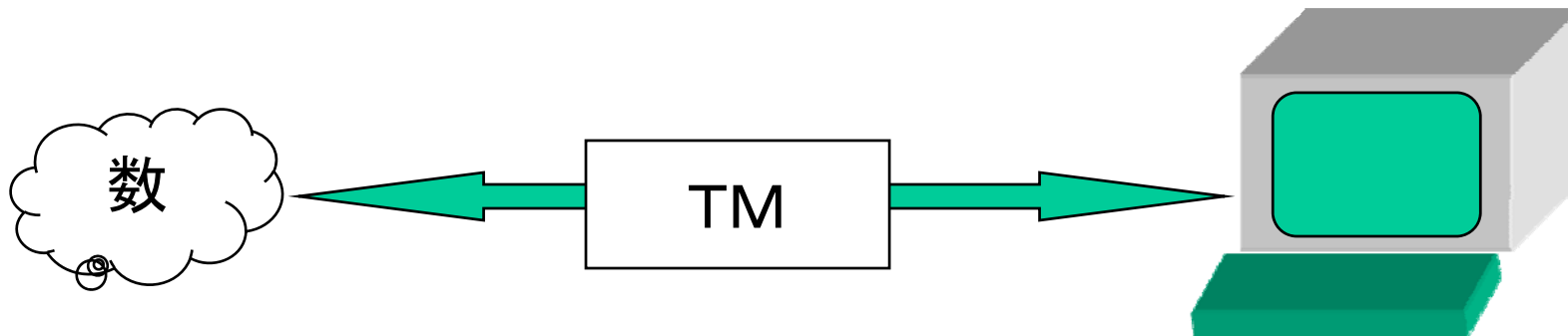


$\langle T \rangle =$

$11110^l110^m110^n1110^{i_1}110^{i_2}11\dots110^{i_k}111c_111c_211\dots11c_h111$

$\in \{0,1\}^*$

このように、TMは数とみなせる。  
また、TMは  $\{0,1\}^*$  上の文字列ともみなせる。



## 練習

次のTM  $T$  を符号化せよ。

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

$$Q = \{q_0, q_1, q_2, q_3, q_4\}$$

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

$$\Gamma = \{0, 1, X, Y, B\}$$

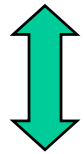
$$F = \{q_4\}$$

$\delta$	0	1	$X$	$Y$	$B$
$q_0$	$(q_1, X, R)$			$(q_3, X, R)$	
$q_1$	$(q_1, 0, R)$	$(q_2, Y, L)$		$(q_1, Y, R)$	
$q_2$	$(q_2, 0, L)$		$(q_0, X, R)$	$(q_2, Y, L)$	
$q_3$				$(q_3, Y, R)$	$(q_4, B, R)$
$q_4$					

# 入力テープの符号化

入力記号列も、次のように符号化できる。

$$x = a_{i_1} a_{i_2} \cdots a_{i_p}$$



$$\langle x \rangle = 10^{i_1} 10^{i_2} \cdots 10^{i_p} \in \{0, 1\}^*$$

## 6-2. 万能チューリングマシン

符号化によって、どのようなTMも、 $\{0, 1\}^*$ の文字列で表現できることがわかった。

また、入力記号列も、 $\{0, 1\}^*$ の文字列で表現できることもわかった。

このように符号化されているTM  $T$ を入力として、 $T$ の動作をシミュレートするTM  $U$ を設計することができる。つまり、任意のTM  $T$ と入力  $x$ の組に対して、その符号が与えられたときに、 $T$ の動作をシミュレートするTMを**万能TM**という。

# 万能TM

次のような、5テープTM  $U$ でどんなTM  $T$ の動作でもシミュレートすることができる。

$\langle T, x \rangle = \langle T \rangle \circ \langle x \rangle$	Uへの入力
100...01001... ↑	Tのテープ
11001000100001001011...	Tの状態遷移関数 $\delta$
0000	Tの状態
0000100000	ワークテープ

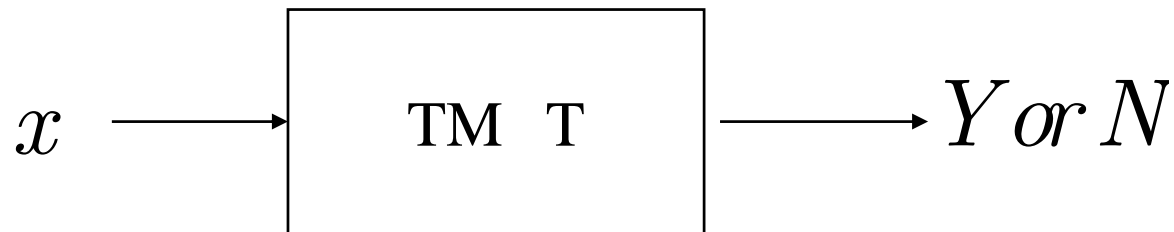
このように、他の任意のTMをシミュレートするTMを  
**万能TM (Universal Turing Machine, UTM)**という。

# UTMの動作

- (i) UはTの現在の状態が受理状態かチェックする。  
(状態テープの内容が、符号化の一部と同じかチェックする。)
- (ii) 状態テープの内容をワークテープにコピーし、  
Tのヘッド位置の記号を1に続けてコピーする。  
このとき、状態遷移関数 の前半部の書式になる。
- (iii) ワークテープの内容と一致する状態遷移関数を  
入力テープから検索する。(見つからない場合は、  
非受理とする。)
- (iv) (iii)が見つかったら後半部にしたがって、第2テープ、  
第4テープ内容を更新する。(シフト動作が必要となるが、  
実現可能である。)

# TMとUTM

ここでは、TMやUTMを一種のブラックボックスをみなす。



## 6-3. TMの限界(計算の限界)

TMは、現在のコンピュータが実行できるものは、すべて実行することができる。  
コンピュータによって、実現されている様々なソフトウェアを考えると、TMで何でも計算可能のように思えてしまう。  
しかし、TMで受理できない言語が存在する。  
これは、コンピュータには、原理的に限界があることを示している。



## 計算の表

TMは、 $\{0, 1\}^*$  の文字列であり、  
入力も、 $\{0, 1\}^*$  の文字列である。

いま、 $\{0, 1\}^*$  の文字列すべてを次のように並べることができる。

$$\sigma_1 = 0, \sigma_2 = 1, \sigma_3 = 00, \sigma_4 = 01, \dots, \sigma_\infty$$

ここで、 $\sigma_i$  を縦横に配置して2次元の表をつくる。

$(i, j)$  成分は、TM  $\sigma_i = \langle T \rangle$  が入力  $\sigma_j = \langle x \rangle$  を  
受理するときに○、その他は×とする。

(なお、 $\sigma_i$  がTMの符号化にそっていなければ×とする。)

このようにして、2次元の表を構成することができる。

# 入力記号

チューリングマシンの符号化

	$\sigma_1$	$\sigma_2$	...	$\sigma_j$	...	
$\sigma_1$	×	×	×	×	×	×
$\sigma_2$	×	...				
⋮						
$\sigma_i$	<i>O</i>	<i>O</i>	×	<i>O</i>	×	
⋮						

上の方の行は  
符号化にそって  
ないので×が多い

## TMで認識不可能な言語

計算の表に基づいて次のような言語を構成できる。

$$L_{\overline{TM}} = \{\sigma_i \mid TM\sigma_i = \langle T \rangle \text{が} \sigma_i = \langle x \rangle \text{を受理しない。}\}$$

この言語は、計算の表において、  
対角成分が×となるような  $\sigma_i$  すべてからなる。

このとき、次の命題が成り立つ。

言語  $L_{\overline{TM}}$  はいかなるTMによっても認識されない。

## 証明(対角線論法)

$L_{\overline{TM}}$ を認識するTM  $T$ が存在するとする。(背理法の仮定。)  
 $T$ の符号化したものを  $\sigma_T$  とする。すなわち、 $\sigma_T = \langle T \rangle$ 。  
計算の表より、  
 $(\sigma_T, \sigma_T)$   
の要素は0か×である。

場合1: 0のとき

このときは、

$\sigma_T = \langle T \rangle$  は列  $\sigma_T = \langle x \rangle$  を受理するので、 $L_{\overline{TM}}$  には含まれない。しかし、 $T$ は受理しており矛盾である。

場合2: ×のとき

今度は、 $\sigma_T$  が  $L_{\overline{TM}}$  に入るのに $T$ は受理しない。  
よって、こちらも矛盾である。

以上より、 $L_{\overline{TM}}$  を認識する TM は存在しない。 *QED* 20

# TMにおける停止能力

必ず停止するTMでは認識できないが、  
そのような制限のないTMでなら認識できる  $\{0,1\}$  上の  
言語  $L_{\overline{halt}}$  が存在する。

## 証明

$L_{\overline{halt}} = \{\sigma_i \mid TM\sigma_i = \langle T \rangle \text{は、列}\sigma_i\text{に対して停止して受理しない。}\}$   
とする。このとき、 $L_{\overline{halt}}$  が命題の言語であることを示す。

この言語は、必ず停止する言語では認識できない。  
背理法(対角線論法)により証明する。  
必ず停止するTM  $T$ が存在すると仮定する。(背理法の仮定)  
このとき、符号化によって、 $\sigma_T = \langle T \rangle$ を得る。  
このとき、列  $\sigma_T$  を入力しても停止する。  
すると、前の議論と同様にして、 $T$ が  $\sigma_T$  を受理しても、  
受理しなくても矛盾が生じる。

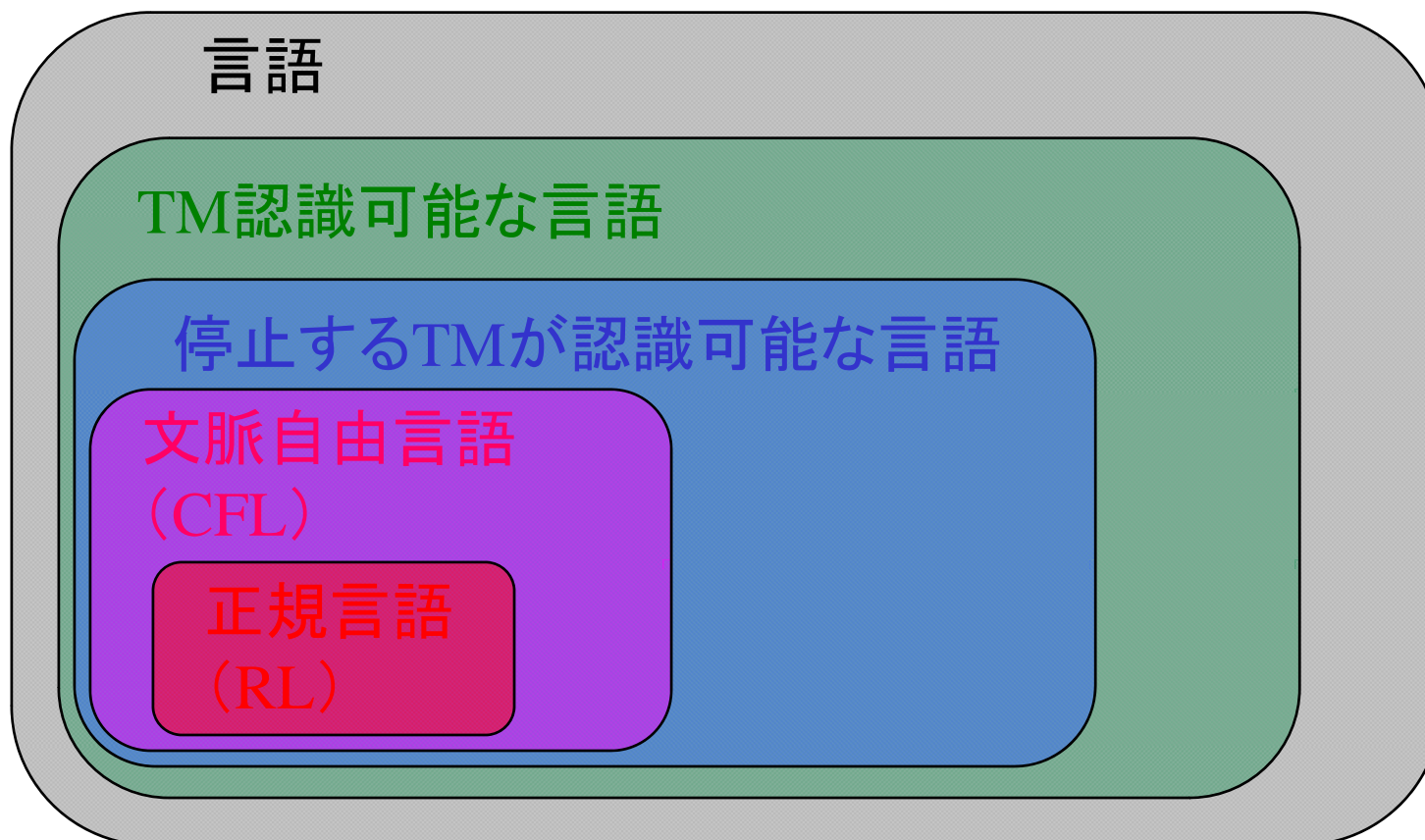
一方、必ず停止するという条件をはずすと、次のように  
簡単に認識可能。

$T$ の列  $\sigma_T$  に対する動作をシミュレートして、  
非受理状態で停止したときのみ受理をする。

以上より、必ず停止するTMは、TMより能力が小さい。

# 言語間の関係

言語には、TM認識不可能なもの、TM認識可能なもの、  
文脈自由文法、正規言語がある。  
これらは、真の包含関係を形成する。



## 6-4. 言語と問題

### 問題(クラス)と問題例(インスタンス)

ここで、計算機で解く問題について再考する。

問題といった場合に、次のような3つのタイプが考えられる。

素因数分解についての「問題」

- (1) 781167は合成数かどうか判定せよ。
- (2) 整数 $n$ が与えられたとき、それが合成数かどうか判定せよ。
- (3) 整数 $n$ の素因数をすべてもとめよ。

問題のインスタンス

列挙の問題  
といいます。

インスタンスの  
集合(クラス)



# 判定問題

ここからは、(2)のタイプの問題を扱いたい。  
(2)のように{yes,no}で答えられる問題を  
「判定問題」(decision problem)といいます。

判定問題を記述する際には、

- 1.問題の名称、
2. 問題例の集合、
- 3.yesとなるべき条件

の3つを記述する必要がある。

これらの記述によって、初めて問題が定義される。

ここで、2, 3は簡単に、

2'インスタンス(の代表)

3'質問

に置き換えることもある。

## 判定問題の例

名称: 合成数の問題

インスタンス: 整数  $n$

問:  $n$  は合成数か?

名称: 最大値の問題

インスタンス: 実数の集合  $S = \{a_1, a_2, \dots, a_n\}$  と

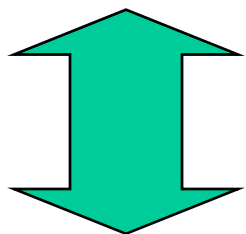
添え字  $i \in \{1, 2, \dots, n\}$

問:  $a_i$  は  $S$  の最大値か?

# 言語と問題

判定問題は、yesとなるインスタンスの集合を言語とみなせば、一意に表現可能である。

名称: 合成数の問題  
インスタンス: 整数 $n$   
問:  $n$ は合成数か?



$$\begin{aligned}L_Y &= \{c \in N \mid c \text{は合成数}\} \\ &= \{4, 6, 8, 9, 10, \dots\}\end{aligned}$$

このように、問題は言語に読み替え可能。問題を解くとは、対応する言語を認識するTMを作ること。

## 6-5. 停止性問題

次のような問題を考える。

名称: TMの停止性問題

インスタンス:  $\langle T, \sigma \rangle$

問: TM  $T$ は  $\sigma$  に大して、停止するか？

この問題は、無限ループの自動判別等に利用可能で、大変有用であるが、計算不可能である。つまり、自動無限ループ判定ソフトは原理的に、実現不可能である。

## 定理

TMの停止性問題は非可解である。(計算不可能である。)

## 証明

$L_{halt} = \{\sigma_i \mid TM\sigma_i = \langle T \rangle\}$ は、列 $\sigma_i$ に対して停止して受理しない。

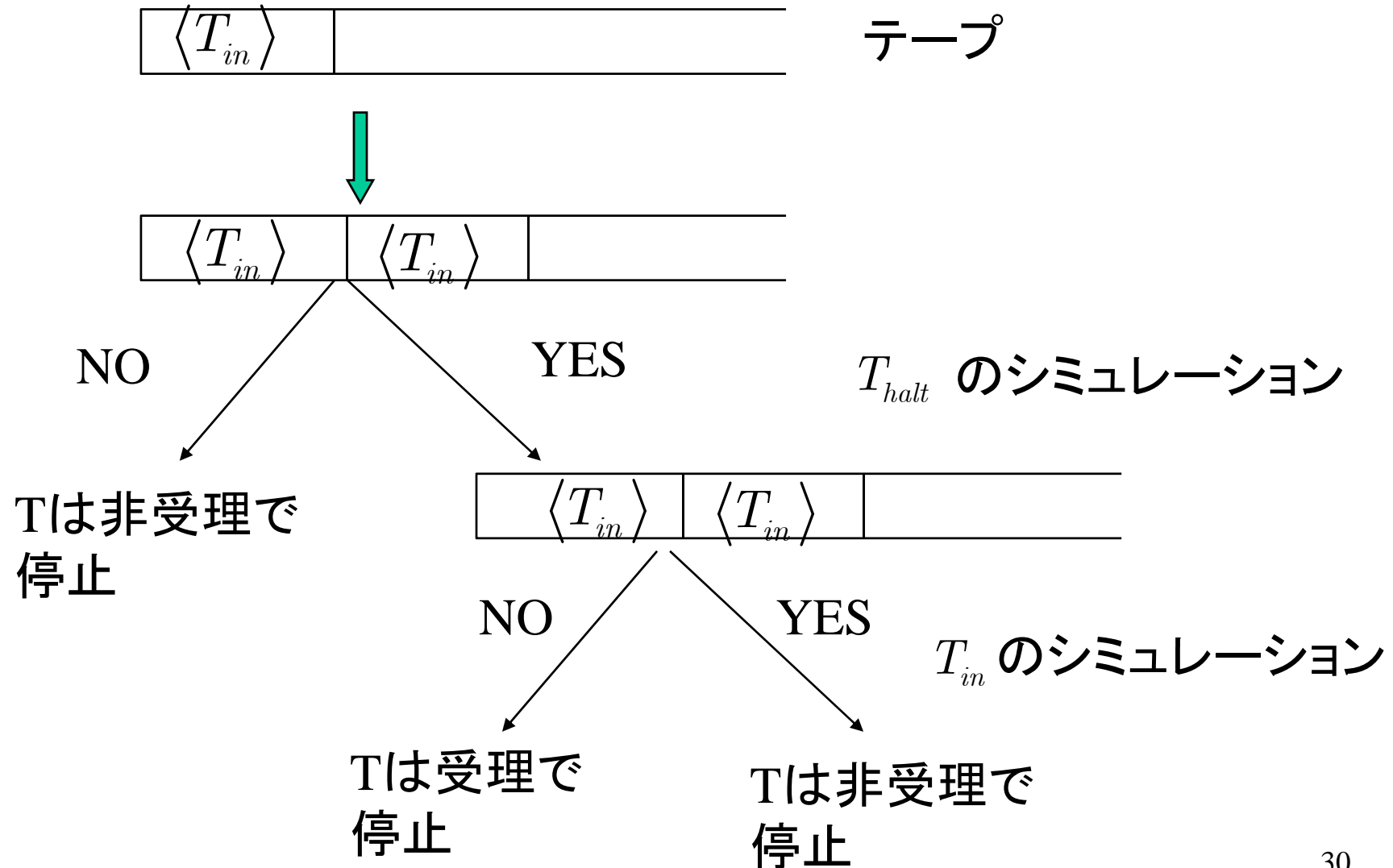
は、前にみてきたように停止保証TMでは認識できない。

この言語を利用して、命題を背理法によって示す。

TMの停止性問題を解くTM  $T_{halt}$  が存在すると仮定する。  
(背理法の仮定)

$T_{halt}$  をUTMの構成の要領でシミュレートすることができる。  
このことを利用してTを構成する。

入力  $\langle T_{in} \rangle$  に対して、コピーを作り文字列  $\langle T_{in} \rangle \langle T_{in} \rangle$  を構成する。



もし、 $\langle T_{in}, T_{in} \rangle$  が必ず停止することがあらかじめ判別できれば、 $\langle T_{in} \rangle$  をシミュレートすることによって、 $L_{halt}$  の言語を認識できるTM  $T$ が構成できる。  
しかも、 $T$ は必ず停止する。  
これは、 $L_{halt}$  を認識する停止保証TMが存在しないことと矛盾する。

以上より、もし停止性判定問題を解くTM  $T$  は存在しない。

*QED*

## 6-6. 停止性問題の別証明

一般のプログラムの停止を判定するような、プログラムは存在しない。

### 証明

プログラムPとPへの入力Dを引数とするような次のようなプログラムが存在するとする。

halttester1 (Program P, Data D);

入力: プログラムPと、そのプログラムへのデータD

出力: PへDを入力したときに、

停止するなら        yes

停止しないなら     no

を出力する。(必ず停止する)



DataのDとしては、どのようなデータでもかまわないはずである。  
よって、Dとしてプログラム自身を常にとるような関数を  
構成できる。

halttester2(Program P);

入力: プログラムP

出力: PへPを入力したときに、

停止するなら      yes

停止しないなら    no

を出力する。(必ず停止する)

つまり、halttester1(P,P)と同様の動作をする。

これは、halttester1(P,D)が存在すれば、容易に構成できる。  
(単に、機能を限定させているだけである。)

次に、halttester2(P)を元に、次のような関数を構成する。

```
funny1(Program P);
```

入力: プログラムP

出力: halttester2(P)がyesなら、無限ループ  
halttester2(P)がnoなら、停止

具体的には、次のような関数を構成すれば良い。

```
funny1(Program P){  
    if(halttester2(P){  
        for(;;);  
    }  
    else{  
        printf("HALT¥n");  
    }  
}
```

このとき、プログラム `funny1()` に、  
引数として、`funny1()` を与えたときの動作を考える。

すなわち、`funny1(funny1);` が停止するかどうかを考える。

場合1:

`haltester2(funny1)` が `yes` と出力する場合、  
このときは、  
`funny1()` の作り方から明らかに停止しない。  
これは、`funny1` が停止すると判断していることと矛盾する。

場合2:

`haltester2(funny1)` が `no` と出力する場合、  
このときは、  
`funny1()` の作り方から停止する。。  
これは、`funny1` が停止しないと判断していることと矛盾する。

このようにいずれの場合も矛盾が生じる。