TME 5、6和7：确定性的图灵机

(定义，例子，在Python中编程)

- 2.1版（2023年1月） -

Mathieu.Jaume@lip6.fr

本节介绍的代码在文件Turing.py中。在所有关于图灵机的TME中，*与课程和教程中不同的是*，图灵机的带子被假定为左边和右边都是无限的。在图灵机的实现中、符号H由字符'Z'代表。

1 单带确定性的图灵机

1.1 代表性

在Python中，单段确定性图灵机由四元组M = (d,q0,qok,qko)表示，其中q0、qok和qko分别代表初始状态、接受状态和拒绝状态，d是代表过渡函数的列表，包含对每个过渡期$q_1 \xrightarrow{a/a_{12},x}_M q_2$形式的元素（（q1,a1），（q2,a2,x））。

由于过渡函数是由一个列表定义的，我们定义了一个函数assoc_f，给定一个代表函数f和元素x的列表，如果x属于f的域，则返回$f(x)$，否则返回None（我们这里假定f的域的元素是
这相当于假设图灵机的状态也可以与原子平等相媲美）。

缩写：assoc_f

```
def assoc_f(lf,x):
    """ list[alpha*beta] * alpha -> beta """
    for (xf,yf) in lf:
        如果xf == x:
            return
    yf return None
```

例1 机器被建造：

$$M_0 = \big(\underbrace{\{q_0, q_1, q_2, q_3\}}_{Q}, \underbrace{\{A, B, a, b\}}_{\Sigma}, \underbrace{\{A, B, a, b, \mathrm{H}\}}_{\Gamma}, \delta, q_0, q_2, q_3\big)$$

其中δ的定义是：

$$\delta(q_0, A) = (q_1, A, R) \quad \delta(q_0, a) = (q_3, a, R) \quad \delta(q_0, b) = (q_3, b, R)$$
$$\delta(q_0, B) = (q_3, B, R) \quad \delta(q_1, A) = (q_3, A, R) \quad \delta(q_1, B) = (q_2, B, R)$$
$$\delta(q_1, a) = (q_1, b, R) \quad \delta(q_1, b) = (q_1, a, R)$$

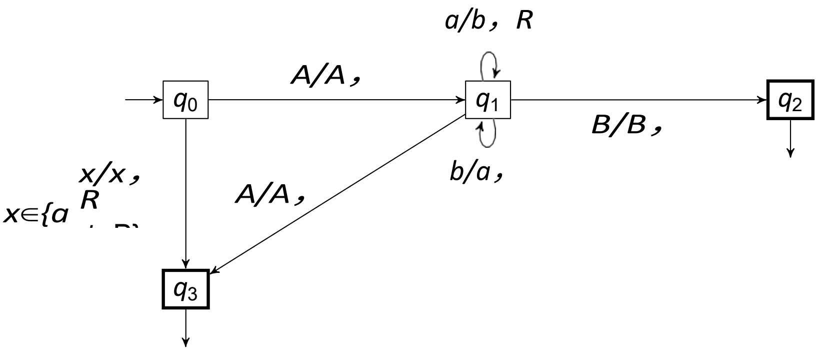

图1 - 图灵机*M*的图形表示$_0$ (例子1)

图1中给出了M_0 的图形表示。

代表例1的*M机器*$_0$ （如图1所示）的过渡函数的 列表写为 （*i*表示状态q_i ）：

示例：$_{M0}$机器的过渡函数

```
l_M_ex1 = [((0, "A"), (1, "A", "R")), ((0, "a"), (3, "a", "R")), ((0, "b"), (3, "b", "R")、
           ((0, "B"), (3, "B", "R")), ((1, "A"), (3, "A", "R")), ((1, "B"), (2, "B", "R")), ((1,
           "a), (1, "b", "R")
```

例1的 机器$_{M0}$（如图1所示） 。

```
M_ex1 =(l_M_ex1,0,2,3)
```

1.2 单段确定型图灵机的 执行情况

配置 图灵机*M*的*配置*是*M*的当前状态、磁带（即其内容）和磁带上读头的位置等数据。让$w_1\ |q|w_2$ *表示M*的配置，当*M*的当前状态为$q \in Q$时，字
$w\ w_{12} \in \Gamma^*$ 被写在磁带上，读头被定位在第一个字母上
的单词w_2 。给定一个写有$w \in \Sigma^*$ 的磁带，图灵机*M*的*初始配置*是$|q_0\ |w$。一个*接受*（或*拒绝*
）的*配置*是一个形式为$w_1\ |q_{ok}\ |w_2$ （或$w_1\ |q_{ko}\ |w_2$ ）的配置。

我们定义了一个函数print_config_1，用于显示图灵机的配置，它来自一个代表磁带的列表L，一个指定磁带上读头位置的整数t（t≤len(L)），机器的状态q∈*Q*以及
机器的状态q_{ok} 和q_{ko} 。当达到状态q_{ok} （resp. q_{ko} ）时，显示的配置
包含ok（resp. ko）而不是状态。

显示单 频 机 的配置

```
def print_config_1(L,t,q,qok,qko):
    for s in L[:t]:
        print(s,end='')
    print("|",end='')
    if q == qok:
        print("ok",end='')
    elif q == qko:
        print("ko",end='')
```

```
    否则:
        print(q,end='' )
    print("|",end='')
    for s in L[t:]:
        print(s,end='')
    print(" " )
```

例子：显示一个单 频 机器的配置

```
>>> print_config_1([1,2,3,4,5,6,7,8,9],0,"q","q2","q3")
|q|123456789
>>> print_config_1([1,2,3,4,5,6,7,8,9],4,"q2","q2","q3")
1234-56789
>>> print_config_1([1,2,3,4,5,6,7,8,9],9,"q","q2","q3")
123456789|q|
```

执行 图灵机M在配置C中的一个执行步骤产生了配置C'，它被表示为$C \sim_M C'$，定义为：

- $C' = u|q'|acv$ 如果$C = ua|q|bv$且$\delta(q, b) = (q', c, L)$

  例子 如果$\delta(q, l) = (q', c, L)$ 那么从配置$C = motsur|q|leruban$，我们得到配置$C' = motsu|q'|$rceruban：

---	m	o	t	s	u	r	c	e	r	u	b	a	n	---

$\underbrace{motsu}_{w1}\ \underbrace{rceruban}_{w2}$

↑ $\boxed{qr}$

- $C' = uc|q'|v$ 如果$C = u|q|bv$且$\delta(q, b) = (q', c, R)$

  例子 如果$\delta(q, l) = (q', c, R)$ 那么从配置$C = wordsur|q|leruban$，我们得到配置$C' = wordsurc|q'|$eruban：

---	m	o	t	s	u	r	c	e	r	u	b	a	n	---

$\underbrace{motsurc}_{w1}\ \underbrace{eruban}_{w2}$

↑ $\boxed{qr}$

一个机器M在一个词$w \in \Sigma^*$，表示为M（w）↓，如果存在一个序列有限的$C_0, C_1, ---, C_n$ 的配置，表示为$C_0 \sim^*_M C_n$，这样：

— $C_0 = |q_0|w$是初始配置

— 对于所有$0 \leq i < n$，$C_i \sim_M C_{i+1}$

— C_n是一个接受或拒绝的配置 $M(w)\downarrow^{ko}$

在一个词$w \in \Sigma^*$，运行图灵机M可以导致四种不同的情况：

— 机器停在一个接受的配置中，用$M(w)\downarrow$来表示。^{ok}

— 机器停在拒绝配置中，用$M(w)\downarrow$表示。^{ko}

— 机器达到了一个配置，从这个配置中无法访问。可以

— 的 "无限循环 "机器，用M（w）↑表示。

在下文中，为了减轻对δ的书写，并简化对图灵机可能执行的描述，当$\delta(q, b)$未被定义时，我们将*C转化*为一个拒绝的构型。因此，在以下图灵机的表述中，没有从状态$q\in Q$和符号$x\in\Gamma$的转换，意味着图灵机的

被认为有默认的过渡$q \xrightarrow{a/a,R} q_{ko}$.根据这一惯例，给定

一个词$w\in \Sigma^*$，所以有三种情况：M（w）$\downarrow^{ok}$，M（w）$\downarrow^{ko}$，M（w）$\uparrow$。

实施 我们定义了一个函数exec_MT_1，它可以通过显示M在执行过程中的连续配置来执行一台图灵机M（确定性的，只有一盘磁带）。这个函数的参数是一个图灵机M，一个代表初始磁带的列表L和一个指定磁带上读头初始位置的整数i0。
(它对应于列表L的一个索引)。这个函数返回一个三联体（b, i_F, L），其中b*是*一个布尔值，表示计算是否成功（即计算是否在接受q_{ok}的状态下结束）、
i_F是计算结束时读头的位置，L是计算结束时代表磁带的列表。最后，由于假设磁带是无限长的，但列表L是有限的，这个函数将添加L中计算所需的元素（在左边或右边）（每个添加的元素将用符号H初始化，在实现中用'Z'编码）。当然，如果图灵机M进行的计算没有以提供的磁带结束，exec_MT_1函数的执行也不会结束。

待完成：执 行 一个确定性的单频机器

```
def exec_MT_1(M,L,i0):
    # M: 单段确定型图灵机 L: 代表初始段的列表
    # i0: 读头的初始位置
```

例子：执行机器M_0的例子1

```
>>> exec_MT_1(M_ex1,["A","a","b","a","a","B"],0)
|0|AabaaB
A|1|abaaB
Ab|1|baaB
Aba|1|aaB
Abab|1|aB
Ababb|1|B
AbabbB|ok|Z
(True, 6, ['A', 'b', 'a', 'b', 'B', 'Z'] )

>>> exec_MT_1(M_ex1,["B","a","b","a","a","B"],0)
|0|BabaaB
B|ko|abaaB
(False, 1, ['B', 'a', 'b', 'a', 'B'] )
```

在下面的例子中，将只给出exec_MT_1函数的执行结果（不显示产生的显示）。

例2 (L_1 = $\{w\in\{a, b\}^* \mid |w|_a = |w|_b\} \in \mathbf{R}$)
语言L_1是递归的，存在一个图灵机M_1，使得L(M_1)=L_1。

要完成：机器M_1，使L(M_1) = $\{w\in\{a, b\}^s \mid |w|_a = |w|\}_b$

```
l_ex2 = ...
M_ex2 =(l_ex2,...)
```

几个图灵机可以接受L_1。因此，计算结束时磁带的内容和读头的位置可能会因所采取的方法而不同，但所有可能的M_1版本必须产生相同的布尔值（表明计算是否成功）。

机器M的执行实例$_1$，这样L(M_1) = $\{w\in \{a, b\}^s \mid |w|_a = |w|_b\}$

```
>> exec_MT_1(M_ex2,["Z"],0)
(True, 1, ['Z', 'Z'] )
>>> exec_MT_1(M_ex2,["a","b","a","b","a","a","Z"],0)
(False, 6, ['X', 'Y', 'X', 'Y', 'X', 'a', 'Z'] )
>>> exec_MT_1(M_ex2,["a","b","b","a","b","a","Z"],0)
(True, 7, ['X', 'Y', 'X', 'Y', 'X', 'Y', 'Z', 'Z'] )
>>> exec_MT_1(M_ex2,["b","b","a","b","a","a","Z"],0)
(True, 7, ['X', 'X', 'Y', 'X', 'Y', 'Z', 'Z'] )
>>> exec_MT_1(M_ex2,["b","b","a","Z"],0)
(False, 3, ['X', 'X', 'Y', 'Z'] )
```

例3 (L_{isneg} = $\{w\in\{0, 1\}^{A} \mid w$以1结尾$\}$ $\in\mathbf{R}$)

语言L_{isneg}是递归的，包含负相对整数的2's complement二进制表示，当最小有效位被赋予左[1]。我们定义一个机器M_{isneg}，如果$w\in L_{isneg}$，则$M_{isneg}(w)\downarrow^{ok}$；如果$w\notin L_{isneg}$，则$M_{isneg}(w)\downarrow^{ko}$。 我们希望这台机器不改变磁带的内容，并替换掉播放头

在计算结束后，对w的第一个符号。

待完成： M_{isneg}机器

```
d_isneg = ...
M_isneg =(d_isneg,...)
```

M_{isneg}机器的执行实例

```
>>> exec_MT_1(M_isneg,["1","0","1","0","0","Z"],0)
(False, 1, ['Z', '1', '0', '1', '0', 'Z'] )
>>> exec_MT_1(M_isneg,["0","0","1","0","1","Z"],0)
(True, 1, ['Z', '0', '0', '1', '0', '1', 'Z'] )
```

2 图灵机的组成

要构造一个执行某种计算的图灵机M，可能需要将这种计算分解为子计算，定义执行每个子计算的图灵机，并将它们组合起来以获得机器M。在本节中，我们将介绍对应于编程语言的三种基本构造的组合。

2.1 序列

设$M_1 = Q_1, \Sigma, \Gamma, \delta_1, q^1, q^1_0\ q^1$ 好 寇和$M_2 = Q_2, \Sigma, \Gamma, \delta_2, q^2, q^2, q^2_0$ 好 寇单磁带决定性图灵。我们希望从磁带L上运行机器M_1，读头在位置i_1，然后从计算M_1得出的磁带上运行机器M_2（读头在计算结束时由机器M_1定位）。如果M_1的计算失败，那么M_2就不会被执行，而且序列的计算也会失败。

[1] 回顾一下，一个有n个比特的二进制字（最重要的比特在左边）$a_{n-1}\cdots a_0$代表相对整数$-a_{n-1}2^{n-1} + \sum_{i=n-2} a_i2^i$使用2的补码。一个二进制字代表因此，当其最重要的位为1时，是一个负的相对整数。

由于我们有一个图灵机执行模拟器，可以返回最终的磁带和磁带上播放头的最终位置，所以很容易定义一个函数，依次模拟两个图灵机的执行：

模拟2台机器依 次 执行

```
def exec_seq_MT_1(M1,M2,L,i1):
    (b,i2,L2)=exec_MT_1(M1,L,i1)
    if b:
        返回 exec_MT_1(M2,L2,i2) 否则:
        返回（b,i2,L2）。
```

也可以从机器M_1和M_2中构造一个新的图灵机M，它可以执行与M_1和M_2的序列计算相对应的计算。这种构造的原理如图2所示，机器M的正式定义是：

$$M = Q_1 \cup Q_2, \Sigma, \Gamma, \delta, q_0^1, q^2_{好}, q^2_{寇}$$

与：

$$\delta = \delta_2 \cup \{q_1 \xrightarrow{a/a_{12},d}_M q_2 \mid q_1 \xrightarrow{a/a_{12},d}_{M1} q_2 \text{ 和 } q_2 \neq q^1_{好} \text{ 和 } q_2 \neq q^1_{寇}\}$$
$$\cup \{q_1 \xrightarrow{a/a_{12},d}_M q_0^2 \mid q_1 \xrightarrow{a/a_{12},d}_{M1} q^1_{好}\}$$
$$\cup \{q_1 \xrightarrow{a/a_{12},d}_M q^2_{寇} \mid q_1 \xrightarrow{a/a_{12},d}_{M1} q^1_{kb}\}$$

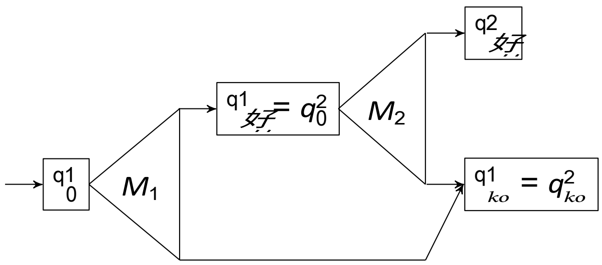

图2-图灵机组成：序列

实施 构建的机器的状态应以$(1, q_i)$的形式表示，状态$q_i \in Q_1$，$(2, q_i)$表示状态$q_i \in Q_2$。

待完成：按2台机器的顺序组成

```
def make_seq_MT(M1,M2):
    # M1,M2: 1段决定性的图灵机
```

例4（用二补表示的整数的相反数）。

假设w是代表相对整数k的2的补码的二进制字。要计算-k的二进制表示（在2的补码中），只需计算w的补码，然后进行二进制加法w+1[2]。因此，这里是一个依次执行以下机器的问题

2. 回顾一下，一个只包含1符号的二进制字在2的补码中代表相对整数-1。注意到对于任何二进制词w

来说，布尔和w+w总是产生一个只包含1个符号的二进制词，注意到$_{kw}$（resp. $_{kw}$）是代表w（resp. w）的相对整数，我们有$_{kw}$+$_{kw}$=-1，我们最后得到$_{-kw}$=$_{kw}$+1

M_{comp} 和 M_{succ} 在附录中定义。由此产生的机器被表示为M_{opp}。 例如，考虑相对整数5，其2的补码是二进制字w=0101。w的补码是w = 1010，那么w的后继者是1011，对应于相对整数-5的2补码。通过使最小有效位出现在所使用的图灵机磁带的左侧，我们可以通过运行模拟器找到这个结果

由exec_seq_MT_1函数实现：

例子：-5和-(-5) 的计算

```
>> exec_seq_MT_1(M_compl_bin,M_succ_bin,["1", "0", "1", "0"],0)
(True, 1, ['Z', '1', '1', '0', '1', 'Z'] )
>>> exec_MT_1(M_opp_int_bin,["1","1","0","1"],0)
(True, 1, ['Z', '1', '0', '1', '0', 'Z'] )
```

将要完成：建造莫普机器

```
M_opp_int_bin = ...
```

莫普机执行的例子

```
>>> exec_MT_1(M_opp_int_bin,["1","0","1","0"],0)
(True, 1, ['Z', '1', '1', '0', '1', 'Z'] )
>>> exec_MT_1(M_opp_int_bin,["1","1","0","1"],0)
(True, 1, ['Z', '1', '0', '1', '0', 'Z'] )
```

这也可以通过构建机器M_{opp}，并运行它来实现。

2.2 有条件的

On souhaite construire une machine de Turing M dont l'exécution correspond à l'exécution de la conditionnelle «if C then P_1 else P_2» à partir des trois machines de Turing M_C, M_1 et M_2 telles que :

- $M_C = Q_C, \Sigma, \Gamma, \delta_C, q^C, q^C, q^C$ 是一台图灵机，当w时，$M_C(w)\downarrow^{ok}$ 验证了属性C，当W没有验证属性C时，$M_C(w)\downarrow^{ko}$
- $M_i = Q_i, \Sigma, \Gamma, \delta_i, q_0^i, q^i, q^i$ (for $i \in \{1, 2\}$) 是一台执行计算 P_i

由于我们有一个图灵机执行模拟器，可以返回最终的磁带和磁带上播放头的最终位置，所以很容易定义一个函数来模拟条件 "if C then P_1 else P_2 "的执行情况，从三个图灵机M_C、M_1 和M_2。机器M_1 和M_2 从磁带上运行，并且该磁带上读头的位置来自机器M的计算$_C$。

模拟3台机 器 的条件执行情况

```
def exec_cond_MT_1(MC,M1,M2,L,i0):
    (bc,ic,Lc)=exec_MT_1(MC,L,i0)
    if bc:
        返回 exec_MT_1(M1,Lc,ic) 否
    则:
```

也可以从机器M_C、M_1 和M_2 中构建一个新的图灵机M，执行条件 "如果C则P_1，否则P_2"。这种构造的原理如图3所示，机器M的正式定义为：

$$M = Q_C \uplus Q_1 \uplus Q_2, \Sigma, \Gamma, \delta, q_0^C, q^2, q^2$$

与：

$$\delta = \delta_2 \cup \{ q_1 \xrightarrow{a/a_{12},d}_M q_2 \mid q_1 \xrightarrow{a/a_{12},d}_{M_C} q_2 \text{ 和 } q_2 \neq q^C_{好} \text{ 和 } q_2 \neq q^C_{寇} \}$$

$$\cup \{ q_1 \xrightarrow{a/a_{12},d}_M q0^1 \mid q_1 \xrightarrow{a/a_{12},d}_{M_C} qC_{好} \}$$

$$\cup \{ q_1 \xrightarrow{a_1/a_2,d}_M q0^2 \mid q_1 \xrightarrow{a/a_{12},d}_{M_C} qC_{寇} \}$$

$$\cup \{ q_1 \xrightarrow{a/a_{12},d}_M q_2 \mid q_1 \xrightarrow{a/a_{12},d}_{M_1} q_2 \text{ 和 } q_2 \neq q^1_{好} \text{ 和 } q_2 \neq q^1_{寇} \}$$

$$\cup \{ q_1 \xrightarrow{a/a_{12},d}_M q^2_{好} \mid q_1 \xrightarrow{a/a_{12},d}_{M1} q^1_{ok} \}$$

$$\cup \{ q_1 \xrightarrow{a/a_{12},d}_M q^2_{寇} \mid q_1 \xrightarrow{a/a_{12},d}_{M1} q^1_{kb} \}$$

$$\cup \{ q_1 \ldots \}$$

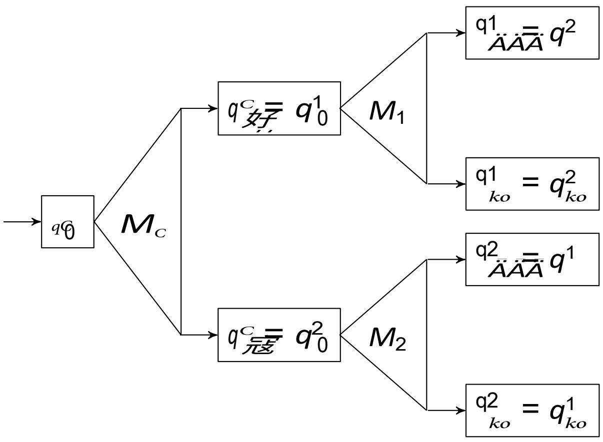

图3 - 图灵机的构成：条件性

实施 构建的机器的状态应该用成对的形式表示，$(0, q_i)$表示状态$q_i \in Q_C$，$(1, q_i)$表示状态$q_i \in Q_1$，$(2, q_i)$表示状态$q_i \in Q_2$。

待完成：有条件的由机器组成

```
def make_cond_MT(MC,M1,M2):
    # MC, M1, M2: 一段式确定性图灵机
```

例5（用二补表示的整数的绝对值）。

图灵机M_{abs}，允许从其二进制表示中计算出相对整数的绝对值，其最小有效位在左边，可以从例3的机器M_{isneg}，附录中定义的机器M_{opp}和机器M_{id}构造。例如，使用

exec_cond_MT_1模拟器，我们可以计算出
整数5和-5的绝对值，如下所示。

例子：计算|5|和|-5|

```
>> exec_cond_MT_1(M_isneg,M_opp_int_bin,M_id,["1", "0", "1",
"0"],0) (True, 1, ['Z', '1', '0', '1', '0', 'Z'] 。)
>> exec_cond_MT_1(M_isneg,M_opp_int_bin,M_id,["1", "1", "0",
"1"],0) (True, 1, ['Z', '1', '0', '1', '0', 'Z'] 。)
```

将要完成的工作：建造M_{abs}

```
M_abs = ...
```

执行M的例子$_{abs}$

```
>>> exec_MT_1(M_abs,["1","0","1","0"],0)
(True, 1, ['Z', '1', '0', '1', '0', 'Z'] )
>>> exec_MT_1(M_abs,["1","1","0","1"],0)
(True, 1, ['Z', '1', '0', '1', '0', 'Z'] )
```

这些结果也可以通过构建机器M_{abs} ，并运行它来获得。

2.3 循环

On souhaite construire une machine de Turing M dont l'exécution correspond à l'exécution de la boucle «while C do P» à partir des deux machines de Turing M_C, M_P telles que :

- $M_C = Q_C, \Sigma, \Gamma, \delta_C, q^C_0, q^C_{好}, q^C_{寇}$ 是一台图灵机，当w时，$M_C(w)\downarrow^{ok}$ vérifie la propriété C et $M_C(w)\downarrow^{ko}$ lorsque w ne vérifie pas la propriété C
- $M_P = Q_P, \Sigma, \Gamma, \delta_P, q^P_0, q^P_{好}, q^P_{寇}$ 是一台执行计算的机器 P

由于我们有一个图灵机执行模拟器，可以返回最终的磁带和磁带上播放头的最终位置，所以很容易定义一个函数，从两个图灵机的执行中模拟 "while C do P "的执行。
图灵M_C 和M_P。M_P 机器从磁带上运行，其位置是
该磁带上的读数来自机器M_C 的计算。

模拟2台机 器 的循环执行情况

```
def exec_loop_MT_1(MC,M,L,i0):
    (bc,ic,Lc)=exec_MT_1(MC,L,i0)
    if bc:
        (bM,iM,LM) =
        exec_MT_1(M,Lc,ic) if bM:
            返回 exec_loop_MT_1(MC,M,LM,iM)
        否则:
            返回 (False,iM,LM)
    否则:
```

也可以从机器M_C 和M_P 中构建一个新的图灵机M，执行循环 "while C do P"。这种构造的原理如图4所示，机器M的正式定义为：

$$M = Q_C \uplus Q_1 \uplus Q_2, \Sigma, \Gamma, \delta, q^C_0, q^2_{ok}, q^2$$

与：

$$\delta = q_1 \xrightarrow{a/a_{12},d}_M q_2 \mid q_1 \xrightarrow{a/a_{12},d}_M \quad {}_2 \models q^C_{好} \quad 寇 \quad q_2 和 q$$

$$\cup \quad q_1 \xrightarrow{a/a_{12},d}_M q^P_0 \mid q_1 \xrightarrow{a/a_{12},d}_M \quad qC \quad P \quad 好的$$

$$\cup \quad q_1 \xrightarrow{a/a_{12},d}_M q_2 \mid q_1 \xrightarrow{a/a_{12},d}_M \quad C \quad q_2 和 q_2 \quad \xrightarrow{}_M \quad 好$$

$\vdash q^{P}$

$$\cup \quad q_1 \xrightarrow{a/a_{12},d}_{M} q_0 \P\, q_1 \quad {}^{a/a_{12}}_{,d} \qquad qP$$

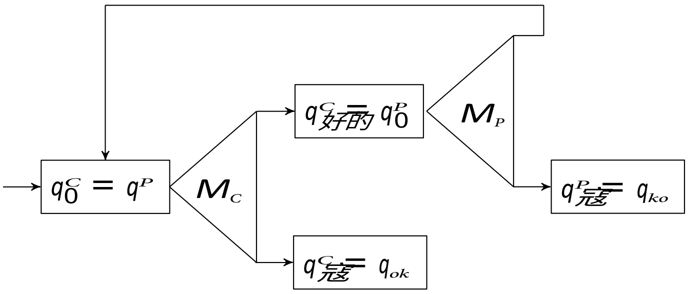

图4 - 图灵机组成：循环

实施 构建的机器的状态应以(0, q_i)的形式表示，状态q_i ⍰ Q_C ，(1, q_i)表示状态q_i ∈ Q_P 。

待完成：由机器组成的循环

```
def make_loop_MT(MC,M):
    # MC,M: 1段决定论的图灵机
```

例6（将读头定位在第一个符号1上）。
我们希望定义一个图灵机M_{foo1} ，该机在给定字母表Γ={0，1，H}上的磁带后，如果输入词的第一个符号1存在，则将读头放在该词的尾部符号H上，否则就放在该词的尾部符号上。这台机器的行为可以通过结构指定：

"当字符读数=0时，将读头向右移动

测试所读字符是否为符号0的机器$M_{=0}$ 和将读头向右移动一个位置的机器M_{right} 在附录中定义。然后，机器M_{foo1} 可以直接构造如下。

将要完成的工作：建造机器M_{foo1}

```
M_foo1 = ...
```

执行M的例子$_{foo1}$

```
>>> exec_MT_1(M_foo1,["1","0","1","0"],0)
(True, 0, ['1', '0', '1', '0'] )
>>> exec_MT_1(M_foo1,["0","0","1","0","1"],0)
(True, 2, ['0', '0', '1', '0', '1'] )
>>> exec_MT_1(M_foo1,["0","0","1"],0)
(True, 2, ['0', '0', '1', 'Z'] )
>> exec_MT_1(M_foo1,["0", "0", "0"],0)
(True, 3, ['0', '0', '0', 'Z', 'Z'] )
```

例7（用二补表示的整数的相反数）。

例4中定义的M机$_{opp}$ ，分两步进行：遍历字w，计算w，然后用1对w*进行*二进制加法。可以用不同的方式来进行这两个操作。事实上，使用前面的例子和附录中定义的图灵机，我们可以验证这种计算可 以 按如下方式进行：

如果$M_{isneg}(w)\downarrow^{ok}$
那么$_{Mpropag}(w)$
否则$M_{id}(w)$舒
；而$M_{=0}(w)\downarrow$
做
严先生(w)
Mfoo0 Mfoo2 Mfoo1 $Mfoo$
如果$M_{=H}(w)\downarrow^{ok}$
那么$M_{id}(w)$
；否则$M_{right}(w); M_{comp}(w)$
、
foo4 M
x
M_f_oo3

进行这一计算过程的机器M_{foo}，可以通过构建以下机器并将其组成得到。

将要完成的工作： 建造机器M_{foo0}, M_{foo2}, M_{foo3}, $M_{=H}$, M_{foo4}, M_{foo}

```
M_foo0    =
...M_foo2  =
...M_foo3  =
...M_eq_Z  =
...M_foo4  =
...M_foo   =
```

执行M的例子$_{foo}$

```
>>> exec_MT_1(M_foo,["1","0","1","0"],0)
(True, 1, ['Z', '1', '1', '0', '1', 'Z'] )
>> exec_MT_1(M_foo,["1", "1", "0", "1"],0)
(True, 1, ['Z', '1', '0', '1', '0', 'Z'] )
>> exec_MT_1(M_foo,["0", "0", "1"],0)
(True, 1, ['Z', '0', '0', '1', '0', 'Z']
)
```

3 多波段确定型图灵机

为了方便图灵机的设计，可以考虑具有多个频段的机器。然而，这种可能性并没有增加单频图灵机的表达能力[3]。一个*确定性的k频图灵机M*是
一台有k个磁带的机器，上面有k个读/写头移动。因此，过渡函数是Γ的k个符号单元：

$$\delta : Q \times \Gamma^k \to Q \times \Gamma^k \times \{L, R, S\}^k$$

其中S（表示*停留*）表示有关磁带的读/写头不移动。
因此，$\delta(q, (a_1, --, a_k) = (q^r, (a^r_1, --, a^r_k), (d_1, --, d_k))$ 表示在 状态q中，如果每个头
如果读数i指向符号a_i，那么这个符号就被符号a^r_i所取代（在
i-i 机器移动到状态q^r，读头i执行位移d_i。我们注意到：

$$q \xrightarrow{(a_1,---,a_n)/(a'_1,---,a'_n),(d_1,---,d_n)}_M q^r$$

这种过渡。

3. 任何多波段图灵机M都可以由单波段图灵机M'来模拟，这样如果M在其所有输入上停止，那么M'也在其所有输入上停止。

实现 k 波段确定性图灵机的原理与单波段机器的原理相同。它是一个元组M=(d,q0,qok,qko)，其中d是一个代表过渡函数的列表。在这个列表中，每个过渡由一个元素表示，其形式为((q,(a1,...,an)),(q',(a'1,...,a'n),(d1, ...,dn))。一个k-磁带机的配置是机器所处状态的数据，*k*个磁带的内容和这些磁带上*k*个读头的位置。显示这样的配置实际上等同于显示一个单磁带机的*k*个配置。下面的函数可以实现这种显示（L是磁带的列表，所以它是一个列表，T是播放头位置的列表：T[i]是第*i*个磁带L[i]上播放头的位置，所以在这个磁带上读取的字符是L[i][T[i]]）。

显示一个k - b a n d 机器的配置

```
def print_config_k(L,T,q,qok,qko,k):
    for i in range(k):
        print_config_1(L[i],T[i],q,qok,qko)。
```

模拟k带机执行的功能与模拟单带机执行的功能类似：在每个步骤中，必须对*k*带中的每一个进行操作，现在有可能让播放头在一个步骤中保持在同一位置。

待完成：执行K-带决定性的机器

```
def exec_MT_k(M,k,L,T):
    # M: 具有k条带的确定性图灵机 # k: 带的数量
    # L: 初始磁带表示的列表 # T: K个读头的初始位置
```

例8 ($L_1 = \{w \in \{a, b\}^* \mid |w|_a = |w|_b\}$)
再考虑一下例2中的*L*语言$_1$。我们想建立一个机器M^2，在两条带子接受这种语言。 1
支持这种语言的双波段图灵机M^2，其原理很简单：

— 第一个波段是输入词出现的波段，这个波段不是从来没有修改过，它只是在计算过程中被遍历。

— 第二条带子用来计算已经遇到的符号数*a*和已经遇到的符号数*b*之间的差异；在初始状态下，一个特殊的符号*X*被放置在这个带子上。

当机器遇到的*a符号*多于*b符号*时，它就处于*q状态*$_1$，在这个状态下，每遇到一个新的*a符号*，第二个磁带的读头就向右移动。在这个状态q_1，如果读到的字符是符号*b*，那么读头就向左移动，如果不指向符号*X*，就向右移动，机器进入状态q_2，这相当于机器的状态，当多于
*b符号*的遍历比*a符号的*遍历多。机器在 状态*q*中 的 行为$_2$
与q_1的对称性。在字的末尾，第二个磁带的读头指向符号*X*，当且仅当输入的字（在第一个磁带上）包含尽可能多的
符号比*b*符号多。这个机器如图5所示。

2带机M^2_1，这样L(M^2_1) = $\{w \in \{a, b\}^* \mid |w|_a = |w|_b\}$ (q_{ok} = 3和q_{ko} = 4)

```
d2_ex1 = [((0,("a","Z")),(1,("a","X"),("R","R"))),
          ((0,("b","Z")),(2,("b","X"),("R","R"))),
```

```
((0,("Z","Z")),(3,("Z","Z"),("S","S"))),
```

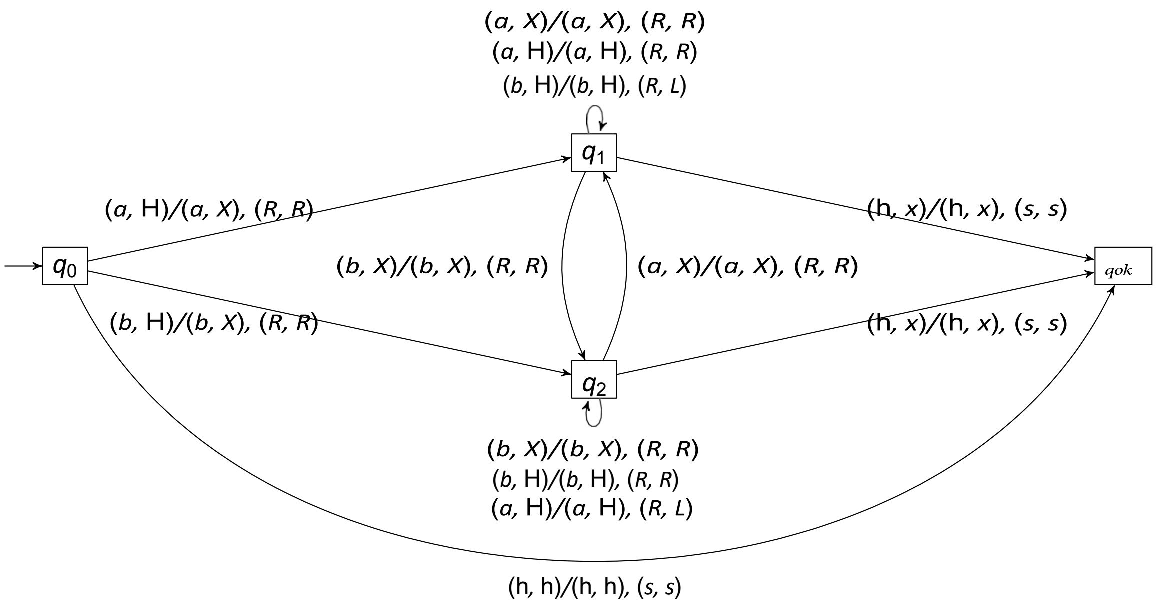

图5 - 图灵机M^2 来自例8的图灵机M

```
         ((1,("a","X")),(1,("a","X"),("R","R"))),
         ((1,("a","Z")),(1,("a","Z"),("R","R"))),
         ((1,("b","Z")),(1,("b","Z"),("R","L"))),
         ((1,("b","X")),(2,("b","X"),("R","R"))),
         ((1,("Z","X")),(3,("Z","X"),("S","S"))),
         ((2,("a","X")),(1,("a","X"),("R","R"))),
         ((2,("a","Z")),(2,("a","Z"),("R","L"))),
         ((2,("b","Z")),(2,("b","Z"),("R","R"))),
         ((2,("b","X")),(2,("b","X"),("R","R"))),
         ((2,("Z","X")),(3,("Z","X"),("S","S")))]
M2_ex1 = (d2_ex1,0,3,4)
```

机器M的执行实例^2，这样L(M^2) =$_1${w∈ {a, b}^{s} | |w|$_a$ = |w|$_b$ }

```
>>> exec_MT_k(M2_ex1,2,[["Z"],["Z"]],[0,0])
(True, [0, 0], [['Z'], ['Z']] )
>>> exec_MT_k(M2_ex1,2,[["a","b","b","a","a","a","b","b","Z"],["Z"]],[0,0])
(True, [8, 0], [['a', 'b', 'b', 'a', 'b', 'Z'], ['X', 'Z', 'Z'])
>>> exec_MT_k(M2_ex1,2,[["a","b","b","a","a","a","b","a","Z"],["Z"]],[0,0])
(False, [8, 2], [['a', 'b', 'b', 'a', 'Z'], ['X', 'Z', 'Z'] )
```

例9（Palindrome语言）

让L_{palin} = {w∈{0, 1}^{A} | w = w˜}。我们希望构造一个图灵机M^2 佩
，在
两个接受这种语言的乐队。

待完成： 机器M^2 佩林 2-条纹

```
d2_palin_bin = ...
M2_palin_bin =(d2_palin_bin, ...)
```

执行M的例子 2

```
>>> exec_MT_k(M2_palin_bin,2,[["Z"],["Z"]],[0,0])
(True, [0, 0], [['Z', 'Z'], ['Z', 'Z'])
>>> exec_MT_k(M2_palin_bin,2,[["0","Z"],["Z"]],[0,0])
(True, [2, 0], [['Z', 'X', 'Z'], ['Z', 'X', 'Z'])
>>> exec_MT_k(M2_palin_bin,2,[["1","0","1","Z"],["Z"]],[0,0])
(True, [4, 0], [['Z', 'X', 'X', 'Z'], ['Z', 'X', 'Z'] )
>>> exec_MT_k(M2_palin_bin,2,[["1","0","0","1","Z"],["Z"]],[0,0])
(True, [5, 0], [['Z', 'X', 'X', 'X', 'Z'], ['Z', 'X', 'X', 'Z'] )
>>> exec_MT_k(M2_palin_bin,2,[["1","1","1","0","1","Z"],["Z"]],[0,0])
(False, [2, 3], [['Z', 'X', '1', '1', '0', '1', 'Z'], ['1', '1', '0', 'X', 'Z'] )
>>> exec_MT_k(M2_palin_bin,2,[["1","0","1","0","1","Z"],["Z"]],[0,0])
(True, [6, 0], [['Z', 'X', 'X', 'X', 'X', 'Z'], ['Z', 'X', 'X', 'Z'] )
```

例10（单数到二数的转换）。

我们希望建立一个两段式图灵机$M_{1\to 2}$，可以将单数表示的自然整数转换为二数表示的自然整数。最初，第一段包含单数字（由I符号组成）作为输入，前面是符号

而第二个条纹是空的。当机器完成后，第一个条纹没有变化，第二个条纹包含二进制字（由0和1符号组成），前面是$符号。

要完成的任务： $M_{1\to 2}$ 2带式机器

```
d2_un_to_bin = ...
M2_un_to_bin =(d2_un_to_bin,...)
```

$M_{1\to 2}$机器的执行实例

```
>>> exec_MT_k(M2_un_to_bin,2,[["$"],["Z"]],[0,0])
(True, [1, 1], [['$', 'Z'], ['$', '0'])
>>> exec_MT_k(M2_un_to_bin,2,[["$","I"],["Z"]],[0,0])
(True, [2, 1], [['$', 'I', 'Z'], ['$', '1'])
>>> exec_MT_k(M2_un_to_bin,2,[["$","I","I","I"],["Z"]],[0,0])
(True, [4, 1], [['$', 'I', 'I', 'Z'], ['$', '1', 'Z'] )
>>> exec_MT_k(M2_un_to_bin,2,[["$","I","I","I","I"],["Z"]],[0,0])
(True, [5, 1], [['$', 'I', 'I', 'I', 'Z'], ['$', '1', '0', '0', 'Z'] )
>>> exec_MT_k(M2_un_to_bin,2,[["$","I","I","I","I","I"],["Z"]],[0,0])
(True, [6, 1], [['$', 'I', 'I', 'I', 'I', 'Z'], ['$', '1', '0', '1', 'Z'] )
>>> exec_MT_k(M2_un_to_bin,2,[["$","I","I","I","I","I","I"],["Z"]],[0,0])
(True, [7, 1], [['$', 'I', 'I', 'I', 'I', 'Z'], ['$', '1', '1', '0', 'Z'] )
```

A 图灵机在TME中使用

例11（二进制字的补数）。

让f_{comp}是计算一个二进制字的补码的函数。我们定义实现这个函数的机器M_{comp}，并在计算结束后将读头重新定位在最左边的位上。例如：

$$\begin{array}{ccc} 1001011H & & 0110100H \\ \uparrow & \leadsto & \uparrow \\ q_0 & & q_{ok} \end{array}$$

图6所示的机器M_{comp}， 允许通过用符号0替换每个符号1，用符号1替换每个符号0来扫描二进制字。在这个运行结束时，达到状态q_1，这使得读头可以放回字的开头。

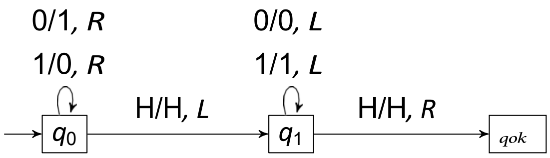

图6 - 图灵机$\mathcal{M}_{comp}$ (例子11)

机器M_{comp} (q_{ok} = 2 和 q_{ko} = 3)

```
d_compl_bin = [((0, "0"),(0, "1", "R")), ((0, "1"),(0, "0",
                                            "R"))、
              ((0,"Z"),(1,"Z","L")), ((1,"0"),(1,"0","L")),
              ((1,"1"),(1,"1","L")), ((1,"Z"),(2,"Z","R"))]
```

执行$_{Mcomp}$机器的例子

```
>>> exec_MT_1(M_compl_bin,["0","1","0","1","1","Z"],0)
(True, 1, ['Z', '1', '0', '1', '0', 'Z'] )
```

例12（一个整数的二进制表示法的继承者）

让f_{succ}是一个函数，给定一个二进制字w，其最重要的位在右边，执行二进制加法w+1。我们定义机器$\mathcal{M}_{succ}$，它实现这个函数并将读头重新定位在最左边的位上（在计算结束时，磁带的右端对应于最有意义的位或符号H）。例如：

H		1(H)	100H		110(H)	111H		0001(H)
†	~	†	†	~	†	†	~	†
q_0		q_{ok}	q_0		q_{ok}	q_0		q_{ok}

$\mathcal{M}$机$_{succ}$， 如图7所示，允许从最小有效位开始扫描二进制字：一旦遇到0符号，就会被1符号取代，剩下的就是将读头重新定位在最小有效位上，否则，只要读到的符号是1字符，就会被0符号取代，并且传播携带的

是在下一个位上进行的。进位传播的状态（这也是初始状态）是q_0，在频段结束前完成计算时达到的状态是q_1，在频段结束时完成计算时达到的状态是q_2，也就是说，当一个额外的位被加入到

在磁带的最后。

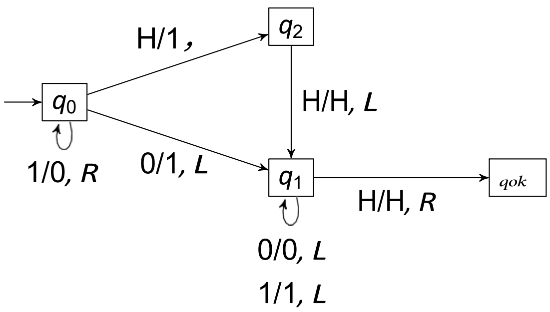

图7 - 图灵机M_{succ} (例子12)

机器M_{succ} (q_{ok} = 3 和 q_{ko} = 4)

```
d_succ_bin = [((0, "0"), (1, "1", "L")), ((0, "1"), (0, "0",
                                                  "R")) 、
              ((0,"Z"),(2,"1","R")), ((1,"0"),(1,"0","L")),
              ((1,"1"),(1,"1","L")), ((1,"Z"),(3,"Z","R")),
              ((2,"Z"),(1,"Z","L"))]
```

$_{Msucc}$机器执行的例子

```
>> exec_MT_1(M_succ_bin,["Z"],0)
(True, 1, ['Z', '1', 'Z'] )
>> exec_MT_1(M_succ_bin,["0"],0)
(True, 1, ['Z', '1'] )
>> exec_MT_1(M_succ_bin,["1"],0)
(True, 1, ['Z', '0', '1', 'Z'] )
>>> exec_MT_1(M_succ_bin,["0","1","0"],0)
(True, 1, ['Z', '1', '1', '0'] )
>>> exec_MT_1(M_succ_bin,["1","0","1","0"],0)
(True, 1, ['Z', '0', '1', '1', '0'] )
>> exec_MT_1(M_succ_bin,["1", "1", "1"],0)
(True, 1, ['Z', '0', '0', '1', 'Z'] )
```

例13（身份函数）

身份函数可以由图灵机M_{id}来实现，其初始状态为状态q_{ok}。

计算身 份 函数的机器

```
M_id = ([],0,0,1)
```

例14 (L_c = {c})

语言L_c包含一个由单一符号c组成的单字。定义了一个图灵机$M_{=c}$，它决定读头读到的字符是否是符号c，在这种情况下进入接受状态q_{ok}（否则机器会进入拒绝状态q_{ko}）。在这两种情况下，读头都被重新定位在被测试的字符上。为了定义这样的机器，我们需要一个包含所有可能的字符的字母表Γ。

带。例如，图8显示了当Γ={0，1，H}时图灵机$M_{=0}$。

因此，M 机的转换$_{=c}$，取决于字母表Γ，为了不需要

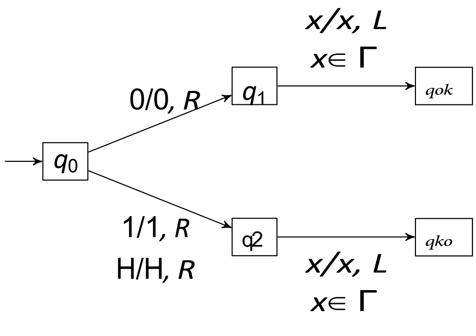

图8 - 图灵机$M_{=0}$（例14）

为了对不同的字母 "手动 "重建这个机器，我们定义了函数make_test_eq，它可以从符号c和字母表中自动进行这个构造。

从字母表Γ构建$_{M=c}$机器

```
def make_test_eq(c,alphabet):
    d = []
    for x in alphabet:
        if c==x:
            d = d + [((0,c),(1,c, "R"))
        否则:
            d = d + [((0,x), (2,x, "R"))
        d = d + [((1,x),(3,x, "L")), ((2,x),(4,x,
    "L")] M = (d,0,3,4)
    返回M
```

例子：从字母表{0，1，H}构造机器$_{M=0}$

```
M_eq_0 = make_test_eq("0",["0", "1", "Z"])
```

用于测试所读字符是否与符号*c*不同的*M*机$_{/=c}$，只需将*M*机的接受和拒绝状态颠倒一下就可以得到$_{=c}$。

从字母表Γ中构建*M*机器$_{/=c}$

```
def make_test_neq(c,alphabet):
    (d,q0,qok,qko) = make_test_eq(c,alphabet)
    return (d,q0,qko,qok)
```

例15（将读头向右移动一个位置）

机器M_{right}，将读数头向右移动一个位置，可以通过以下由字母 "Γ "设置参数的函数获得。

从字母表Γ中构建*M*机器$_{right}$

```
def make_MTright(alphabet):
    d = [] 。
    for a in alphabet:
        d = d + [((0,a),(1,a, "R")]
    M = (d,0,1,2)
    返回M
```

机器M_{right}，从字母表Γ={0，1，H}中选择。

```
M_Right_bin = make_MTright(["0", "1", "Z"))
```

例16（符号位的传播）

如果*w*是一个相对整数的2's complement二进制表示，其最小有效位在左边，那么在二进制字*w的*右端复制位的函数对应于符号[4]位的传播。我们定义了图灵机$_{Mpropag}$，它实现了这个功能，并在计算结束后将读头重新定位在字的第一位上。

图9所示的$_{Mpropag}$机的原理与TME 5中的M_{isneg}机相似。当机器检测到字的最后一位是0（或1）符号时，一个0（或1）符号被添加到字的末尾，然后读头被重新定位在字的第一位上。

4. 给定一个*n*位的二进制字$_{an-1}$ - - $_{a0}$代表一个相对整数*k*的2's complement，最重要的位在左边，可以用一个*n*+1位的二进制字$_{anan-1}$ - - $_{a0}$来代表这个相同的整数*k*，重复最重要的位，即$_{an}$=$_{an-1}$。

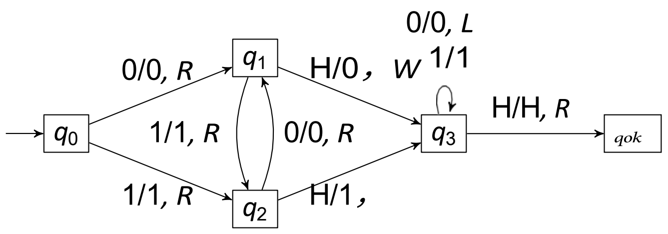

图9 - $_{Mpropag}$图灵机（例子16）

$_{Mpropag}$机器（q_{ok} = 4和q_{ko} = 5）

```
d_prop1 = [((0, "0"), (1, "0", "R")), ((0, "1"), (2, "1", "R")), ((1, "1"),
           (2, "1", "R")), ((1, "Z"), (3, "0", "L")、
           ((2,"0"),(1,"0","R")), ((2,"1"),(2,"1","R")), ((2,"Z"),(3,"1","L")),
           ((3,"0"),(3,"0","L")), ((3,"1"),(3,"1","L")), ((3,"Z"),(4,"Z","R"))]
M_prop1 =(d_prop1,0,4,5)
```

$_{Mpropag}$机器执行的例子

```
>>> exec_MT_1(M_prop1,["0","0","1","0","Z"],0)
(True, 1, ['Z', '0', '0', '1', '0', '0'] )
>>> exec_MT_1(M_prop1,["1","0","1","0","1","Z"],0)
(True, 1, ['Z', '1', '0', '1', '0', '1', '1])
>>> exec_MT_1(M_prop1,["1","Z"],0)
(True, 1, ['Z', '1', '1'] )
```