图灵机

1　快速查看一有多少个步骤？

另请参见视频讲座录音：快速回顾-有多少个步骤？在画布上。我们已经看到了如何通过计算步骤数来分析程序的运行时间。例如：

```
空白 g6（char[]p）（
  Elapse（8 步）；
  为(nati=0；i<p.length()；i++){
    Elapse（5 步）；
    为(natj=i；j<p.length()；j++){
    }
      Elapse（2 步）；
  }
}
```

记住，一个“步骤”应该是一个固定的时间量。

这种分析方法应用广泛、使用方便。但是，我们如何在代码的每个部分中获得基本的步骤计数呢？它们只是假设（或猜测）。例如，许多人通过假设两个值的每次比较都是“一步”来分析排序算法。这当然是一个有用的假设，但它忽略了一个事实，即比较两个大数字比比较两个小数字需要更长的时间。

为了严格地推理运行时间，并避免掩盖任何时间成本的风险，我们需要一个精确的计算模型来完全指定步骤。我们将看到的模型是图灵机(TM)，它是由艾伦·图灵在 1936 年发明的。TM 对步骤的构成持非常保守的观点，所以它可以作为一个黄金标准。如果你的算法在图灵机上是快速的，那么它是无可争议的快！

2 复杂性符号

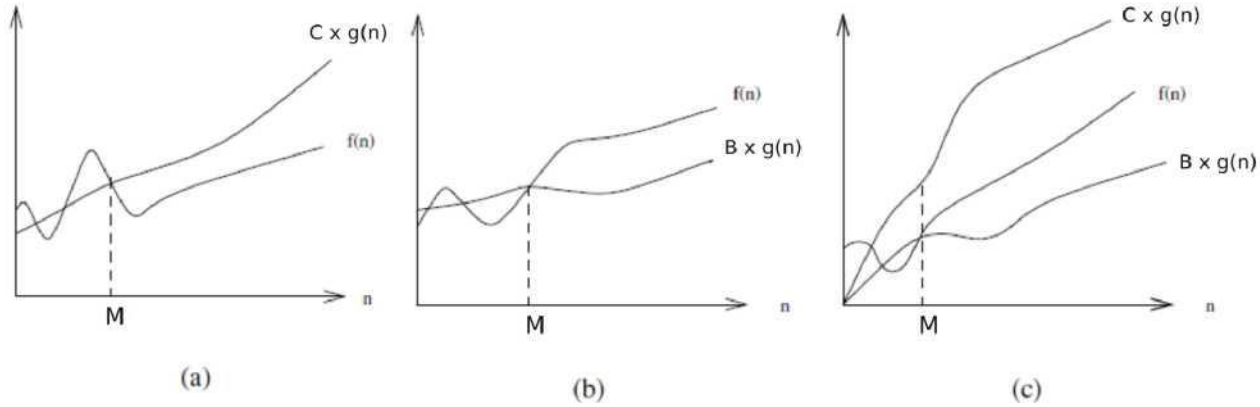
参见视频讲座录音：画布上的复杂性符号。设 f 和 g 是从 N 到非负实数的函数。

(a) 我们说 fe0(g)，或非正式地“f(n)是 0(g(n))”，当 g 是 f 的上界，直到一个常数因子。Thatis：有这样的数字>，对于 n>M，我们有 f(n)<Cxg(n)。

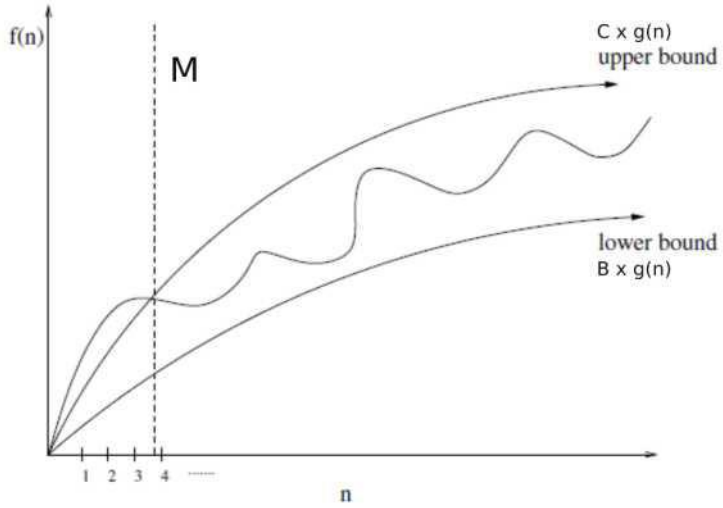
(b) 我们说 feQ(g)，或非正式地“f(n)是 Q(g(n))”，当 g 是 f 到一个常数因子的下界。也就是说：有数字 M 和 B>0，这样对于 n>M，我们有 Bxg(n)<f(n)。

(c) 当上述两个条件都成立时，我们说 fe0(g)，或非正式地“f(n)为 0（g(n)）”。也就是说：有数字 M 和 C 和 B>0，这样对于 n>M，我们有 Bxg(n)<f(n)<Cxg(n)。

下图说明了上述复杂度符号。



有了这些符号，我们就可以更精确地了解复杂性了。例如，如果我们说最坏情况下的运行时间是 0(n)，它实际上可能是线性的，但如果我们知道它是 0(n)那么它真的不比二次好，因为它将在复杂度的下界和上界内。对于 n>M 有效的上下界平滑了复函数的行为，我们希望有一个紧界，以确保我们的估计是精确的，如下所示：



3 什么是图灵机？

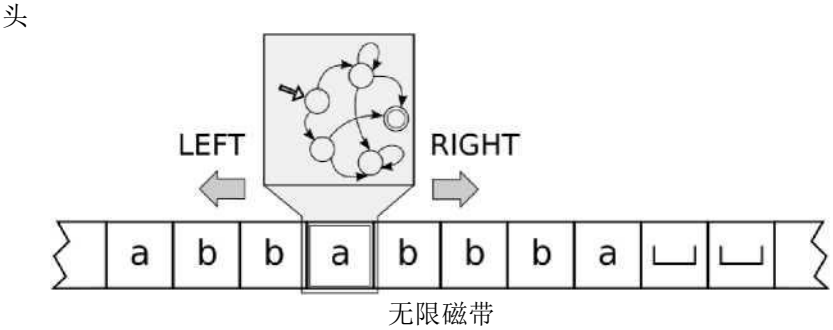
参见视频讲座录音：什么是图灵机？在画布上。

“图灵机可以做真正的计算机能做的一切。然而，即使是图灵机也不能解决某些问题。类似于自动自动机，但具有无限和无限制的内存，图灵机是一个更精确的通用计算机模型在非常真实的意义上，这些问题超出了计算的理论极限。

图灵机模型使用一个无限的磁带作为其无限的内存。它有一个磁带头，可以读取和写符号，并在磁带上移动。

最初，磁带只包含输入字符串，并且在其他地方均为空白。如果机器需要存储信息，它可以将这些信息写入磁带上。为了阅读它所写的信息，机器可以把它的头移回去。机器继续计算，直到它决定产生一个输出。通过输入指定的接受和拒绝状态来获得接受和拒绝的输出。如果它没有进入一个接受或拒绝的状态，它将永远持续下去，永远不会停止。”(Sipser，2013)。

简单地说，图灵机是机械计算的一个简单的形式模型，一个通用的图灵机可以用来计算任何函数，它可以由任何其他图灵机计算。图灵机有有限多的状态(比如 DFA)，但它也有一个外部内存：一个无限的磁带，分成单元格。这台机器的一个头位于磁带的一个单元格上。图灵机可以在磁带上读写符号，并在每一步之后向左或向右移动（或保持在原地）。与 DFA 不同，一旦图灵机进入接受或拒绝状态，它就会停止计算并停止。下图显示了一个图灵机的一般表示：



无限磁带

在给出一个图灵机之前，我们首先指定两个有限集：

- 磁带字母表 T，其中包括一个“空白”字符”（也如上面所示）。在任何时候，每个单元格都包含 T 中的一个字符，并且只有有限数量的非空白字符。我们通常取 T={a，b，=}。非空白字符的集合被称为输入字母表(D。
- *返回值的集合 V。*

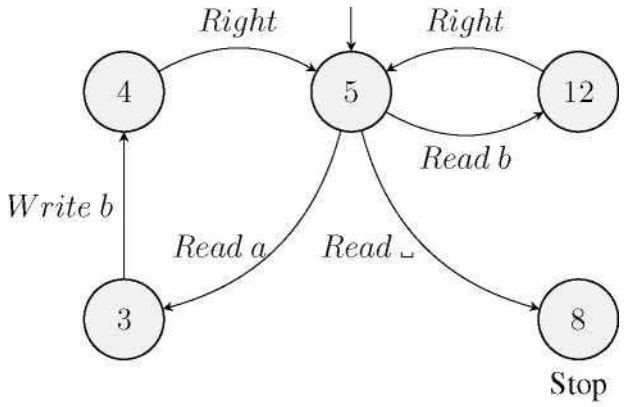
对于 V={true，false}，可用的说明如下：

- 读取，这可能会导致 a 或 b 或
- 写一个
- 写 b
- 写”
- 向左移动
- 向右移动
- 不操作，什么都不做
- 返回 True（接受）
- 返回 False（拒绝）

如果 V 是单例的，那么返回指令通常只是写的停止。请务必注意以下几点：

- 从一开始，他的头在哪里？
- 头到底应该在哪里？

下面的机器在一个空白磁带上的 a，b 块的最左边的单元格上启动。它向右移动，将每个 a 转换为 b，并



例如： 5 Read b

贝巴 12 右边的

bbbbb 5 阅读 a

ba 3 写 b

4 右边的

5 读 b

在单元格的右侧单元格上停止。

下面的机器向右移动三步，永远等待。

无操作

3.1 奇偶校验示例

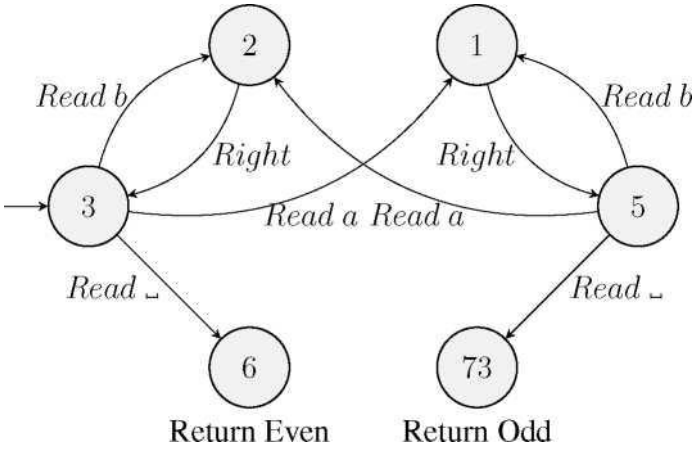
右边的 *右边的* *右边的*

参见视频讲座录音：画布上的奇偶校验示例。

以下“奇偶校验”机器具有：

磁带字母表：T={“，a，b}，返回集：V={偶，奇}

它从一个空白磁带上的 a，b 块的最左边的单元格开始，以块右边的方格结束，表示 a 的数目是偶数还是奇数。



更正式地说，图灵机由以下部分的 T 和 V 组成：

- X 状态的有限集
- 一个初始状态 peX。
- 一个过渡函数 5 从 X 到

ₓ（阅读说明并更改状态）

+TxX（写入指令和更改状态）

+X （向左移动头部并改变状态）

+X （向右移动头部并改变状态）

+X （不运行）

+V(从集合 V 中返回一个值)

上述校验校验 TM 可以正式描述为：

X> ({3, 2,6, 73, 5,1},

P>{3}, *5a{3-—、阅读(a|~、1，b|~、2，=i_|6)2I——iRight3*

6——1 甚至返回

5i-我阅读(ai_i2，bi_i1，=|~i 73)

73i——我返回奇怪的

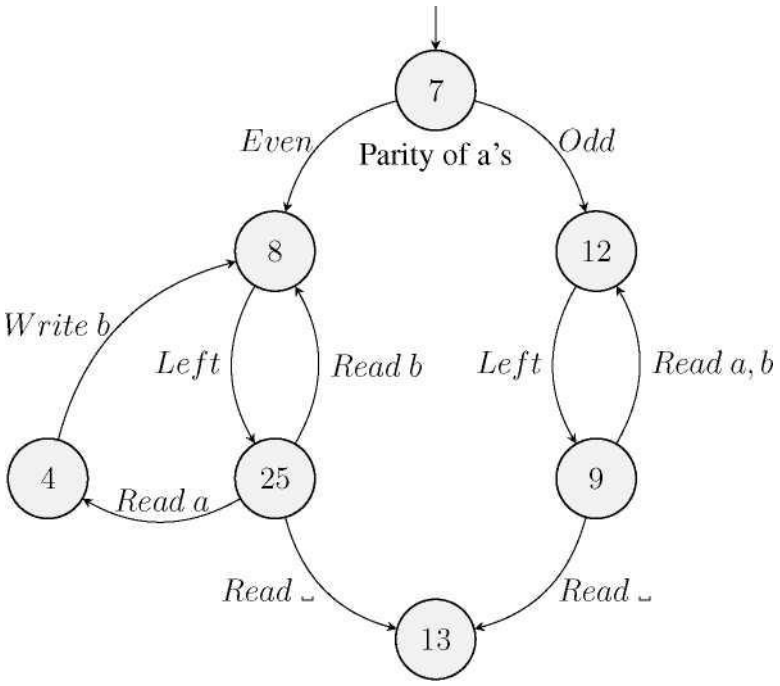
1i——i 右 5

})

3.2 宏

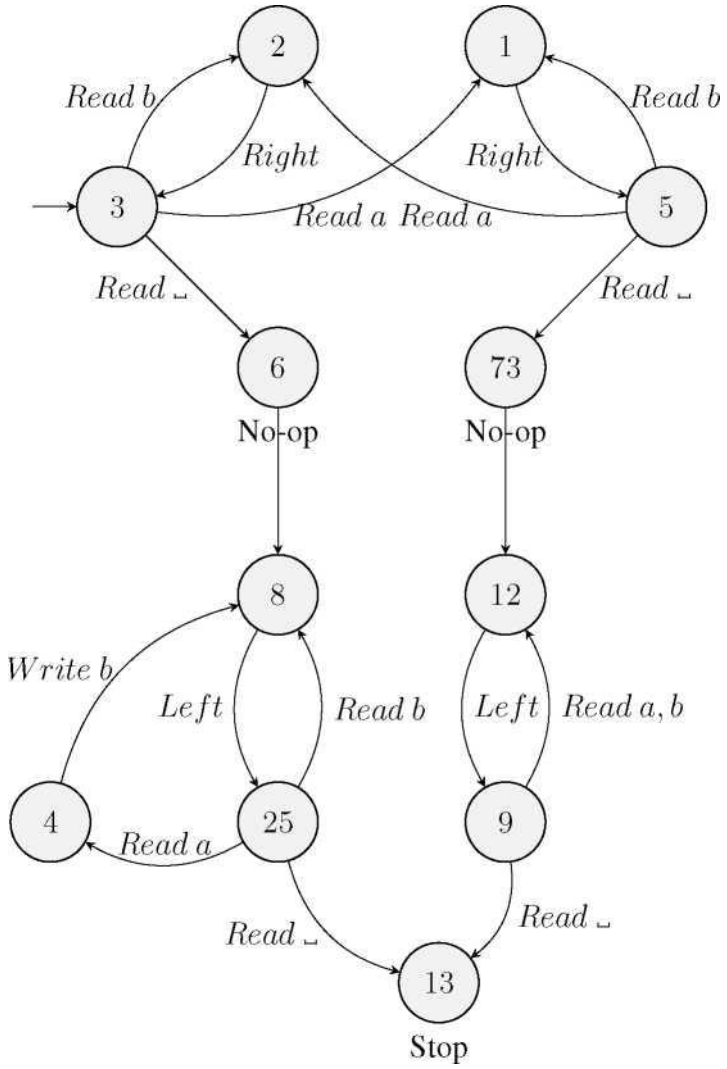
参见视频讲座录制：在画布上的图灵机上使用宏。

编写程序的一种方便的方法是使用宏，宏是一个缩写整个程序的单一指令。为了从带有宏的程序中获得完整的程序，我们需要扩展所有的宏。下面是一个例子，使用我们在上面看到的奇偶校验检查器：



停止

我们可以看到状态 7 是一个宏，它缩写了 a 的奇偶校验，即 a 的数是偶数还是奇数。为了获得完整的程序，我们展开了这个宏，这意味着我们用奇偶校验检查器的定义替换了“a 的奇偶校验”。任何指向宏的东西，现在都将指向定义的初始状态。同样地，如果具有宏的状态是初始状态，则定义的初始状态是扩展程序的初始状态。例如，指向起始状态 7 的箭头现在将指向宏定义的起始状态 3。奇偶校验器的每个返回指令都被一个 No-op 取代，从而导致主程序的适当的下一个状态，即下一个状态将是宏之后的 V 产生的状态。



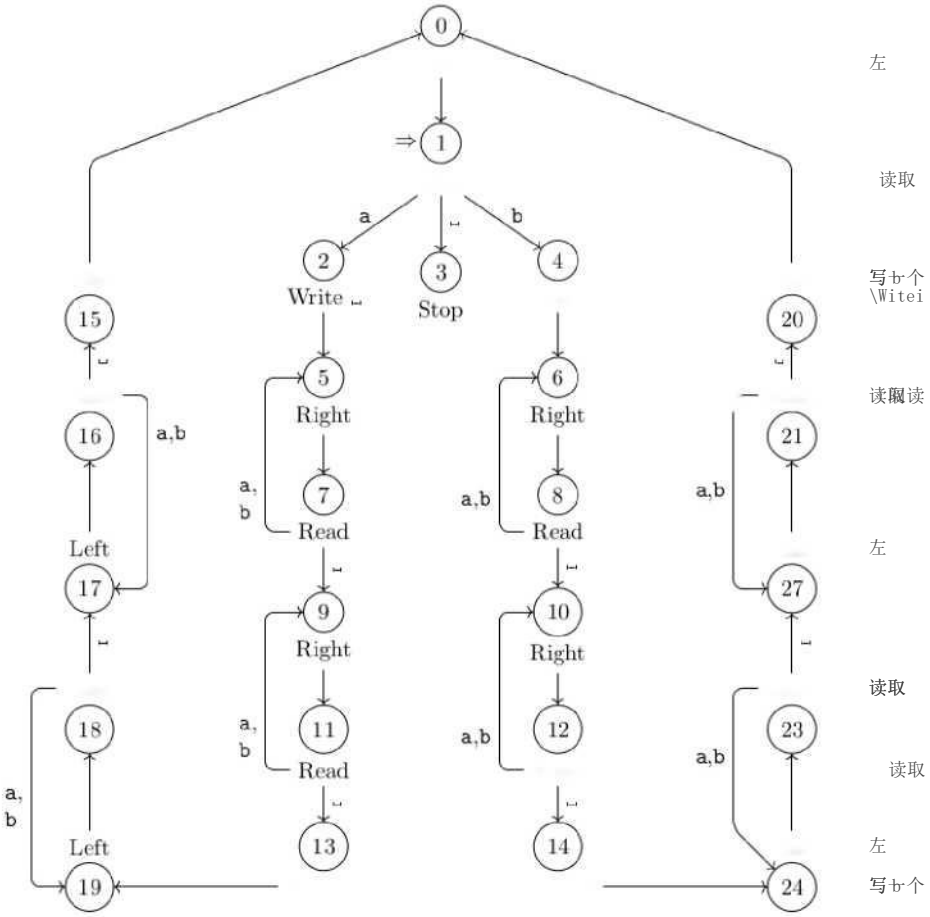
3.3 黑客攻击

参见视频讲座录音：在画布上入侵图灵机。

在实施一些程序时，我们可能需要诉诸于黑客攻击的方法。例如，我们想构建一个图灵机，它从一个空白磁带上的 a，b 块的最右边字符开始，并将输入字符串的反向副本放在右边，在原始字符串和反向字符串之间有一个空白字符。机器应停止，头部位于原块左侧的空白单元上。例如，在一开始就是：abbab

我们希望得到以下输出：Labba“babba”

下面的图灵机实现了所需的程序：



注意此机器如何强制使用空白来记录当前正在复制的位置。所复制的字符是 a 或 b 是否包含在该状态中。我们可以根据初始块的长度来计算出这个程序的精确步数。本 TM 所采取的步骤的总体概述如下：

- 记录当前的字符，无论是 a 还是 b
- 更换空白（黑客！）
- 向右移动两步，并写入这个字符
- 然后回去用 a 或 b（无论之前有的东西）替换空白
- 向左移动一步

一记录当前的字符，无论是 a 还是 b
一用一个空白文件来替换它（黑客！）
一向右移动到中心空白字符
一向右移动到下一个空白处，并写入此字符
一向左移动到中心空白字符
一向左移动到我们刚刚删除的角色
一用 a 或 b（任何之前有的东西）替换空白区域
一向左移动一步，如果它为空白，则停止
一否则，请重新开始此循环。

运行时间显然是二次的：1+2+3+…+n 乘以一个常数！可以证明，复制-反向任务不能比这更快地解决。

4 图灵机变体

参见视频讲座录音：在画布上的图灵机变体。
正如我们所看到的，因为图灵机的概念是如此的保守，程序可能是复杂的，运行缓慢。让我们考虑一些更自

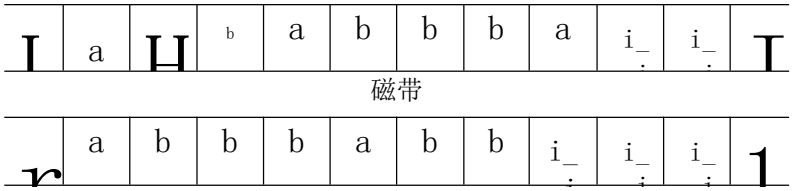
由的变体。

4.1 辅助字符

假设，除了输入的字母表和空白字母之外，我们还有一个有限的辅助字符集。程序可能会假设它们最初没有出现，并且必须保证最终它们不会出现，但在执行过程中它们可以被使用。例如，假设输入字母为{a，b}，辅助字母为{a‘，b’}。然后我们可以编写一个更直接的程序来复制-反向，使用 a！表示当前正在复制的 a，b‘表示当前正在复制的 b（而不是像以前那样使用空白）。

4.2 辅助/多组机器

一个双磁带图灵机有一个主磁带和一个辅助磁带，每盘磁带上都有一个头。对双磁带 TM 的输入提供在主磁带上，并且程序可以假定该辅助磁带最初是空白的，并且必须确保其最终它是空白的。原始的单磁带 TM 及其合理的变体都具有相同的能力，即它们能够识别同一类的语言。可用的指令有写主 x、写辅助 x、读主、



主头

左

辅助磁带

副头

右

读辅助、左辅助、左主、左辅助、右主、右辅助、不操作和返回 v。

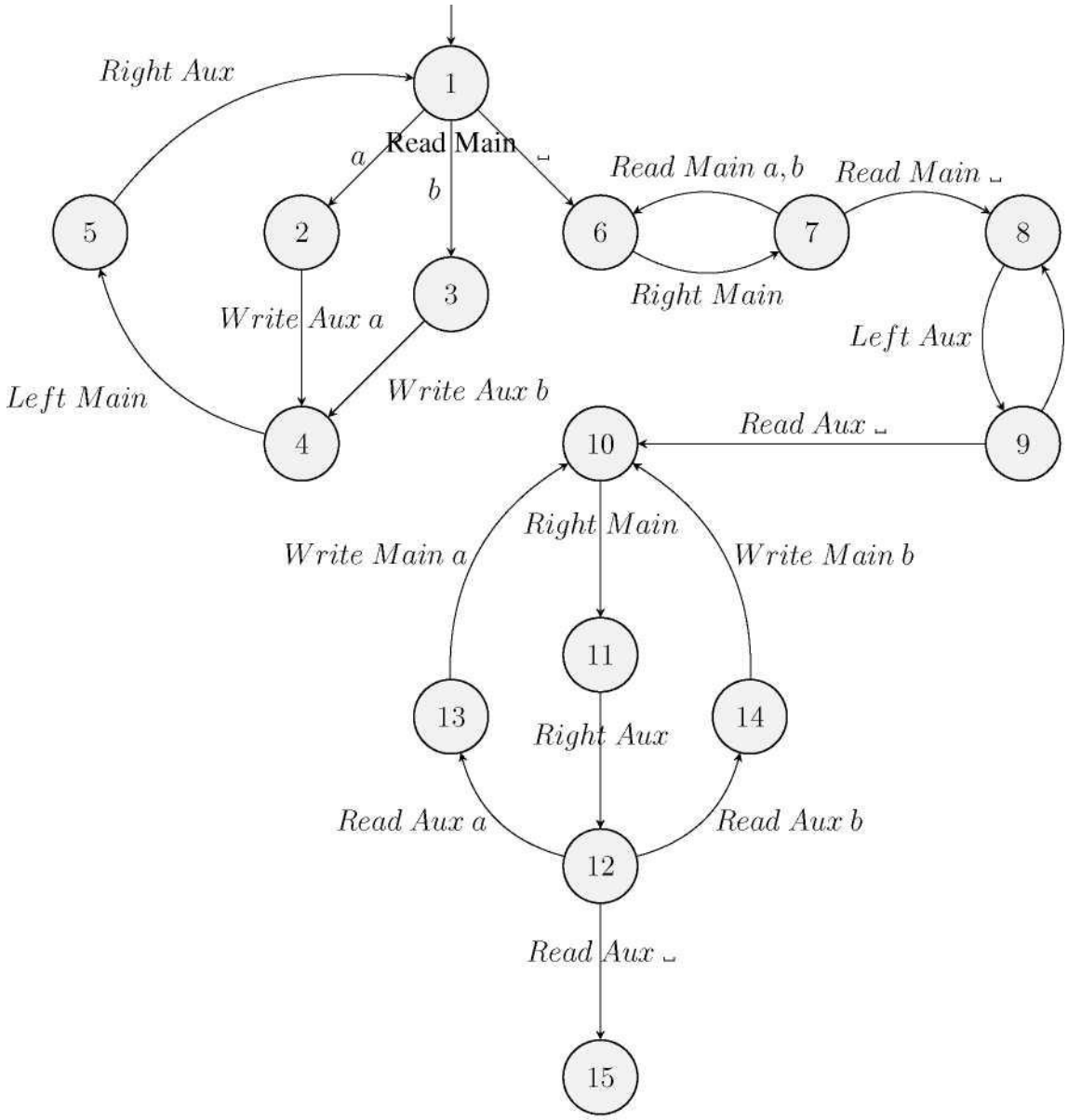
有关现有说明的详情如下：

左

右

- 读取主音，这可能会导致 a 或 b 或“
- 读取 Aux，这可能会导致 a 或 b 或
- 写主 x(x=a 或 b 或
- 写入 Auxx(x=a 或 b 或=)
- 左主
- 左辅助
- 右主
- 右 Aux
- 无操作的主控件，它什么都不做

- 不操作辅助，它什么都不做
- 返回 True（接受）
- 返回 False（拒绝）



Move left through 2 blocks on main tape and stop

下面的两盘磁带 TM 展示了如何在线性时间内解决复制-反向问题：

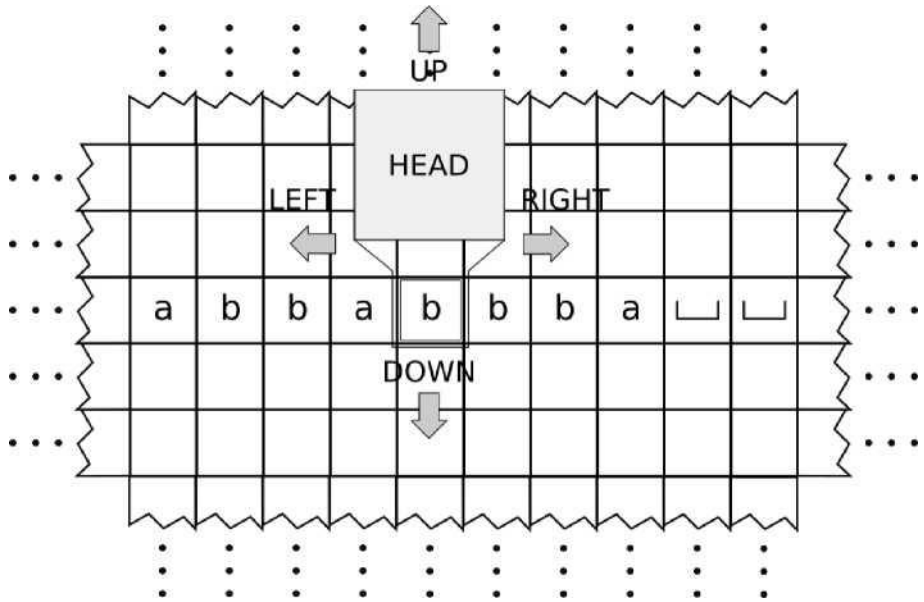
例如，如果这两盘磁带 TM 以：ˆabbab 开始“…在主磁带上（主头在第一个空白，在串之后）和辅助磁带上的六六六=（辅助头在第一个空白，在整个空磁带上）。在第一阶段，从状态 1 到状态 5，TM 从主磁带(a 和 b)中读取，并将它们写入辅助磁带，同时将主磁头向左移动，辅助磁头向右移动。一旦它从主带中读取空白，通过移动主头向右和辅助头向左，重置两个头，直到两者都读取空白（状态 6 到 9）。
在第二阶段，从状态 10 到 14，它将两个头向右移动，并从辅助磁带中读取(a 和 b，现在是相反的顺序)，并将它们写入主磁带。一旦它从辅助磁带中读取一个空白，TM 就知道它已经用程序完成了，并且可以根据需要重置它的两个头。我们还可以在重置过程中擦除辅助磁带（这里省略）。我们希望在主磁带上得到以下输出：

读取 Auxa，b

Labba“babba”

4.3 二维图灵机

一个二维图灵机（2D-TM）有一个无限的薄片，而不是一个磁带。可用的说明是写 x、读取、右、左、上、下、不操作和返回 v。程序可能假定最初的图纸除一行之外是空白的，并且必须确保最后除该行之外是空白的。下图显示了一个二维压结机的表现示意图。



5 总结

在这个讲义中，我们快速回顾了复杂性符号，并理解了研究图灵机的必要性。我们已经看到了一个图灵机的一般模型，它包含一个头，一个无限的磁带，并且可以在执行一个程序时左右移动它的头。我们已经理解，TM 是一个简单的机械计算的形式模型，它可以做一个真正的计算机所能做的一切。我们研究了一些图灵机的例子，包括奇偶校验检查、宏和黑客攻击。我们还讨论了图灵机的变体，包括辅助字符、多磁带和二维 TMs。我们已经理解，“附加反向副本”问题可以通过以下方式来解决：

- 二次时间 O)在一个 TM。
- 2 个磁带 TM 上的线性时间 O(n)。

你认为我们可以在线性时间内的 TM 上解决这个问题吗？答案是否定的！你认为哪一种机器的计算模型更合适吗？你可能会说，2 磁带 TM 是不现实的，因为它允许两个头之间的即时通信，可能相距很远。

我们已经看到，同样的问题可以在不同的机器上以不同的复杂性解决，例如，在一台机器上的二次在另一台机器上是线性的。然而，我们还没有看到这一点：

- 在一种机器上可以用多项式时间解决，但在另一种机器上不能解决。
- 一个问题可以用一种机器来解决，但不能用另一种机器来解决。

实际上，这些事情不可能发生在我们看过的所有机器上。所以，对于康斯坦斯来说，我们使用 TM 还是 2D-TM 很重要，但对于 Polly 来说，这并不重要，因为它的 O(n)还是 O(n 仍然是多项式。

6 进一步的阅读/参考

- Sipser，M.（2013）第三章：教会学习论文，计算理论导论，第三版，参与学习定制出版，梅森，美国