**2 文法作为生成工具**

**2.1 语言是一个无穷集合**

就像日常用语一样，在计算机科学中，“文法”是用来“描述”一种“语言”的。然而如果从表面上看是具有误导性的，因为计算机科学家和普通的说话者对这三个术语含义的理解略有不同。为了建立我们的术语并划定话语世界，我们将从最后一个开始研究上面的术语。

**2.1.1 语言**

对于大部分人来说，语言首先是一种交流方式，几乎无意识地使用语言，当然在激烈的辩论中也是如此。交流是通过空气振动或通过文字符号携带信息来实现。仔细观察，语言信息会分解成句子，这些句子由单词组成，而单词在书写时又由符号序列组成。语言在所有三个构成层面上都可以不同。字母可能略有不同，如英语和爱尔兰语，或者非常不同，如英语和汉语。语言往往差别很大，甚至在人们熟知的语言中，人们将马称为“un cheval”或“ein Pferd”。句子结构的差异经常被低估。即使是紧密相关的荷兰人，也经常有一种近乎莎士比亚式地语句：“lk geloof je niet”，“i believe you not”，而更多相关的语言很容易找到像匈牙利语“Penzem van”，“money-my is”的结构，而英语中则是“I have money”。

计算机科学家对这一切都有非常抽象的看法。是的，一种语言有句子，而这些句子具有结构。不管它们是否传达了某种信息，都不是他所关心的，但是信息可能来自于它们的结构，因此把这些信息称为句子的“意义”是完全正确的。句子由单词组成，被称为token，每个单词都可能带有一些信息，这就是它对整个句子意义的贡献。但是，语言不能再被分解了，这并不使计算机科学家担心。凭借他对可伸缩的解决方案和多层次技术的热爱，他宣称，如果单词最终有了结构，它们就是用不同语言写成的句子，其中字母就是token。

形式语言学的从业者，从此被称为形式语言学家（将他与“正式语言学家”区分开来，后者的详述留给读者想象），再次从抽象的角度来看待这个问题。语言就是句子的集合，每个句子都是符号的序列，没有意义，没有结构，任何一个句子都属于一个语言，或者不属于。符号的唯一属性是它有一个标识，在任何语言中都有一定数量的不同符号，即字母表，而且数量是有限的。为了方便起见，我们将这些符号写为a，b，c，。。。，但是✆，✈，❐，。。。只要有足够的符号就行。单词序列意味着每个句子中的符号都有固定的顺序，我们不应该将它打乱。单词集合意味着这是一个无序的集合，并且去除了重复项。一个集合可以通过花括号括起来，并写入它的对象来写下一组。所对形式语言学家来说，“a, b, ab, ba”是一种语言，并且{a,aa,aaa,aaaa,...}也是，尽管后者有将在后面解决的计数问题。根据计算机科学家在句子/单词和单词/字母之间看到的对应关系，形式语言学家也把一个句子称为单词，并且说“单词ab属于语言{a, b, ab, ba}”。

现在让我们考虑这些紧凑而强大的想法的含义。

对于计算机科学家来说，语言可能是一个无限大的句子集合，每个句子都是具有结构的记号组成。记号以及对应的结构配合来描述句子的语义，这就是句子的意义。结构和语义都是新的，也就是说，在形式模型中不存在，计算机科学家的责任就是提供和操作它们。对于计算机科学家来说，3+4\*5是一个“个位数算法”语言的句子（“个位数”以避免有无数个符号），它的结构可以通过插入括号来显示：(3+(4\*5))，它的语义是23。

对于语言学家来说，语言是一组可能相互关联的句子。每个句子都是以一种结构化的方式，包含在现实世界中具有意义的单词。结构和单词一起赋予了句子传达的意义。单词拥有结构并由字母组成，这写字母和一些结构配合赋予这个单词意义。强调语义与现实世界的关系，以及句子与单词和单词与字母两个层次的整合是语言学家的领域。“轮子猛烈地旋转”是一个句子，而“圆圈睡红”是没有意义的。

形式语言学家持有他对语言的看法，因为他想要研究语言最基本的属性。计算机科学家坚持他的观点，因为他想要一个清晰的、透彻的、明确的方法来描述计算机中的物体以及计算机间的通信，这是一个与人类不同的通信伙伴。语言学家持有他对语言的看法，因为它给了他一个严格控制的看似混乱但也无限复杂的对象：自然语言。

**2.1.2 文法**

凡是学过外语的人都知道，语法是用来描述和教授这门语言的规则和例子。良好的语法对常称之为语法或句法的句子/单词级别和称之为形态学的单词/字母级别进行了仔细的区分。语法包含像“pour que is followed by the subjunctive, but parce que is not”。形态学包含的规则类似于“英文名词的复数形式是通过追加-s形成的，除非单词以-s，-sh，-o，-ch或-x结尾，在这种情况下追加-es，或当这个词有一个不规则的复数”。

我们暂时跳过计算机科学家关于语法的观点，并立即转向正式语言学家的观点。他的观点非常抽象，且与外行人的观点类似：语法是任何精确的、有限的、完整的对语言（即句子集合）的描述。这实际上是学院风格的语法，带有模糊性。虽然这个定义具有充分的普遍性，但事实证明它太笼统，因此相对无力。它包括诸如“可能由Chaucer创作的句子集合”的描述。理想地说，这定义了一个集合，但是我们没办法创建这个集合或者测试一个给定的句子是否属于这种语言。这个特殊的例子，“本来可以”并不让形式语言学家担心，但有更加接近他的研究领域的例子。“π的十进制展开式中最长连续出现数字7的块”描述了一种语言，其中最多有一个单词（该单词将仅包含数字7），并且它的定义是准确的，即有限且完整的。然而这有一个不好的问题，人们无法找到这个词：假设某人在十亿数字之后找到了一百个由7连续组成的块，那么在这之后还有可能找到一个由更多7组成的块。另一个问题是，人们甚至不知道这个最长的块是否存在。很可能，当π的十进制展开位数越来越多，人们可能会发现更长的由7组成的块，由不断增长的间距间隔。π的小数展开的理论可能可以回答这样的问题，但是没有这样的理论存在。

由于这些原因，形式语言学家放弃了对语法的静态的、理想的观点，以获得更具建设性的语法观点，即生成语法：生成语法是一种以精确地、固定大小的方法来构造语言中的句子。这意味着，按照既定规则，必须可以无歧义地构建语言中的每个句子（以有限数量的动作）。这并不意味着，给定一个句子，规则告诉我们该如何构造这个特定的句子，它只是可能这样做。规则的形式有多种，其中一些比另一些更方便。

计算机科学家基本持有相同的观点，通常还要求规则可以暗示如何构建一个句子。

**2.1.3 无穷集合的问题**

上述语言的定义可以产生无限的符号序列，而语法则是产生这些句子的有限方法，这就产生了两个尴尬的问题：

1. 有限的规则如何产生无限的句子？

2. 如果一个句子只是一个序列而没有结构，并且一个句子的含义是从其句子中衍生的，那么我们该如何获知这个句子的意思？

这些问题有很长且很复杂的答案。我们首先关注第一个问题，然后把这本书的主体放在第二个问题上。

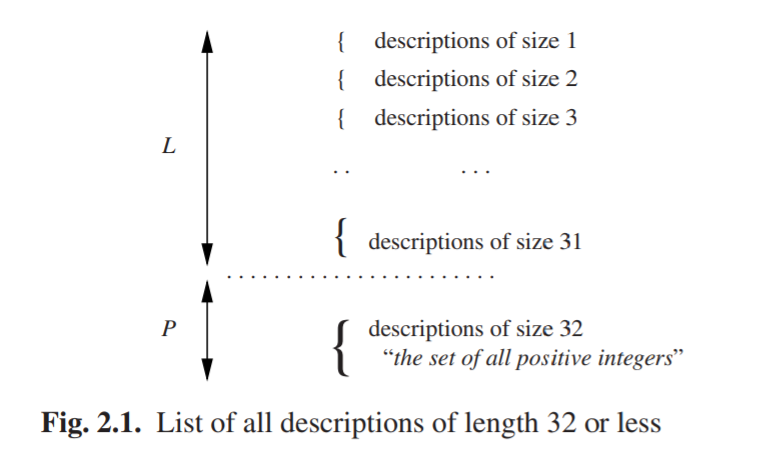
**2.1.3.1 从有穷描述中产生无穷的集合**

事实上，从单个有限描述中获得无限集合没有问题：“所有正整数的集合”是一个非常有限大小的无限大集合描述。然而，这个想法有些令人不安，因此我们将重新修改我们的问题：“是否所有的语言都可以用有限的描述来描述？”，正如上面所说的，答案是否定的，但是证明并非无意义。它是非常有趣并且非常有名，有必要在这里提出一个大纲。

**2.1.3.2 描述是可枚举的**

证明基于两个观察和一个技巧。第一个观察是可以列出描述并给出一个数字，如下所示。首先是长度为1的所有描述，即只有一个字母长，并按照字母顺序对它们进行排序。这是我们列表的开始。确切地说，根据我们接受的描述，对于第一个描述，可能有0个，或27个（所有字母和空格），或95个（所有可打印ASCII字符）或类似的东西。这与下面的讨论无关。

然后，我们对大小为2的所有描述按字母顺序排列，以给出列表中的第二个块。以此类推长度为3或4，以及更长的单词。这为列表中的每个描述分配了一个位置。例如描述“the set of all positive integers”，它的大小是32，不包括引号。为了找到他们的位置，我们需要计算有多少描述少于32个字符，命名为L。然后我们必须生成关于长度为32的描述，并对其排序，命名为P，并添加L和P。这将给出一个巨大的数字，但它确保描述都在列表的一个明确的位置上。



这里应该指出两点。首先是只按照字母顺序列出所有描述，而不考虑它们的长度，通常不这样做：已经有无限多的以“a”开始的描述，并且没有以较高字母开头的描述可以在列表上得到一个数字。第二个是没有必要这么做。这只是一个思想试验，它让我们在不能进行实地考察时，能够对一个系统的行为进行研究。

此外，列表上还有许多无意义的描述。事实证明，这对论据并不重要。重要的是，所有有意义的描述都在列表中，且上述论点确保了这一点。

**2.1.3.3 语言是无限长的位串**

我们知道语言中的单词（句子）是由一组有限的符号组成的，这个集合被称为字母表。我们假设字母表中的符号是有序的。然后语言中的单词也可以排序，我们用Σ来表示字母表。

最简单的语言是将字母表中的字母组成的所有单词进行结合。对于字母表 Σ ={a, b}，我们可以得到语言{ , a, b, aa, ab, ba, bb, aaa, . . . }。我们称这个语言为Σ\*。这将在后面解释原因，暂时它只是一个名字。

Σ\*始于“{, a”，语言中的第一个单词是一个空单词，由0个a和0个b组成。并没有理由去掉它，但是如果写下来，它很容易被忽视，所以我们将它写成ε，而不管字母表（字母表中没有该字符）。所以 Σ∗ = { ε, a, b, aa, ab, ba, bb, aaa, . . . }。在一些自然语言中，动词“to be”的现在时形式是空的词，形成了“I student”，意思是“I am a student”。俄语和希伯来语就是这样的例子。

由于字母表Σ中符号是有序的，因此我们可以使用于前一节中相同的技术列出语言Σ\*中的单词：首先排序所有大小为0的单词，然后为所有大小为1的单词排序，等等。这实际上是Σ\*中已使用的顺序。

语言Σ∗具有一个有趣的属性，就是所有使用字母表Σ的语言都是它的子集。这意味着，给定另一个比Σ更小的语言，称为L，我们可以遍历Σ∗中的单词列表，并将所有在L中出现的单词取出来，这将覆盖L中的所有单词，因为Σ∗包含Σ中的任何可能单词。

假设我们的语言L是“包含b以外的所有单词的集合”。L是集合{a, aa, aab, aba, baa, . . . }。在列表开头加上记号，看起来如下图所示：

ε

✔ a

b

✔ aa

ab

ba

bb

✔ aaa

✔ aab

✔ aba

abb

✔ baa

bab

bba

bbb

✔ aaaa

... ...

根据字母表的顺序，空白和记号列表就足够识别和描述该语言。为了方便起见，我们将空白记为0，将空格记为1，就好像它们是计算机中的位一样，我们可以写出L = 0101000111010001···（Σ∗ = 1111111111111111···）。应该注意的是，这适用于任何语言，像L这样的正式语言，像Java这样的编程语言，或者像英语这样的自然语言。在英语中，位串中的连续的1是非常少的，因为几乎没有任何的任意排序的单词是一个好的英语句子（并且根据我们是在句子/单词或单词/字母水平，几乎没有任何的任意排列的字母序列是一个好的英语单词）。

**2.1.3.4 对角化**

上一节将无限比特串0101000111010001···添加到描述“包含b字母以外的所有单词的集合”，同样的，我们可以将位串添加到所有描述中。某些描述可能不会产生一种语言，我们可以将任意的无限位串添加到其中。由于所有的描述都可以放在有单独编号的列表中，所以我们可以得到如下：

Description Language

Description #1 000000100···

Description #2 110010001···

Description #3 011011010···

Description #4 110011010···

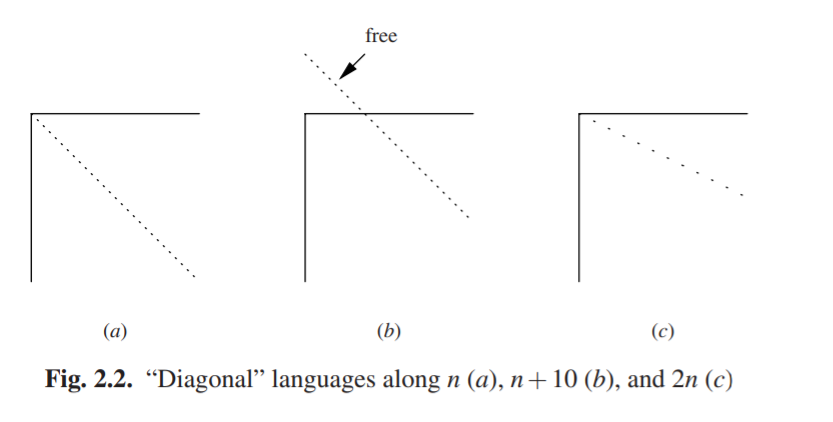
Description #5 100000011···

Description #6 111011011···

... ...

左边所有的描述，定义了右边所有的语言。我们现在断言存在许多不在上述语言列表中的语言：上述列表并不完整，尽管描述列表是完整的。我们将用康托尔的对角论证法来证明这一点。

考虑语言C=100110...，它具有第n位比特与描述#n的第n位比特不同的属性。C的第一个比特是1，因为描述#1的第1位比特是0。C的第二个比特是0，因为描述#2的第二位比特是1，等等。C是通过沿着西北到东南的对角线，并复制遇到的位的反面来形成的。如图2.2(a)。语言C不可能在列表中！语言C不可能在第一行，因为它的第一个位与第一行的第一个位不同，根据定义，它也不可能在第n行，因为它的第n位为第n行的第n位不同。



因此，尽管我们已经详尽地列出了所有可能地描述，我们至少有一种语言没有在列表中描述。而且还有更多的语言没有在列表中。例如，构造语言，其n+10位与描述#n的第n+10位不同，它也不可能在列表中，因为对于n>0，其描述的语言的第n+10位都与第n行的第n+10位不同。这意味着第1到9比特不起作用，可以任意选择，如图2.2(b)，这会产生不在列表中的另外2^9=512种语言。我们还可以进一步这样做。假设我们构造了一门语言，它的第2n位与描述#n的第2n位不同，显然它也不可能在列表中，现在每一个奇数位都没有指定，并且可以自由选择。这使我们可以自由创造无限数量的语言，其中没有一种允许有限的描述。参见图2.2中的对角线，简而言之，对于每一种可以描述的语言，都有无限的无法描述的语言。

对角化技术在大多数关于计算机科学理论的书籍中描述得更为正式，参见例如Rayward-Smith[393, pp. 5-6]，或Sudkamp[397, Section 1.4]。

**2.1.3.5 讨论**

上面的演示向我们展示了几件事情。首先，它显示了将语言作为形式对象的能力。虽然上述的内容需要大量的放大和证明才能作为证明的资格（还有一点需要澄清，为什么定义语言C的上述描述本身不在描述列表中，参见问题2.1，它使我们能够获得对不可评估的属性的洞察）。

其次，它表明了我们只能描述所有可能语言中的一个很小的子集（甚至不是一小部分），这无限的语言，永远超出了我们可触及的范围。

第三，我们已经证明了，虽然有无穷多的描述和无穷多的语言，但是这些无穷大并不相等，后者比前者更大。Cantor称这些无穷大为ℵ0和ℵ1，以上只是他证明ℵ0<ℵ1的一个特例。

**2.1.4 用有穷的规则描述语言**

构造一组对象的好方法是从一个小对象开始。“2是偶数，两个偶数的和还是偶数”有效地生成了所有偶数的集合。形式主义者会补充“并且没有其他数字是偶数”，但我们知道这一点。

假设我们想要生成所有类型名为“Tom, Dick and Harry”的枚举集合，其中除了最后两个，所有名称都以逗号分隔。我们不接受“Tom, Dick Harry”或“Tom and Dick and Harry”，但是我们不反对重复：“Grubb, Grubb and Burrowes”是可以的。虽然这些句子都不是正常英语的完整句子，我们依旧叫它们句子，因为这是我们用名词枚举的小型语言。一个简单的规则是：

0. Tom是一个名字，Dick是一个名字，Harry是一个名字

1. 一个名字是一个句子

2. 句子后面跟着逗号，并且名字也是一个句子

3. 在完成之前，如果句子以“逗号 名字”结尾，则用“and 名字”替换

尽管这对一起合作的读者来说是有效的，但是依旧存在几个问题。特别是第三条。例如句子没有以“逗号 名字”结尾，而是以“逗号 Dick”结尾，名字只是代表真实名称的符号，这些符号不能出现在真正的句子中，最后必须用第0条中给出的真实名称代替。同样，规则中的“句子”一词代表所有实际句子的符号。这里涉及两种符号：出现在完整句子中的真正的符号，如“Tom”，“Dick”，逗号和“and”，还有一些中间符号，如“句子”和“名字”，他们不能再一个完整句子中出现。第一种对应于上面解释的单词或标记，它们的专业术语是终结符（或简称终结）。中间的符号被称为非终结符，一个没有启发性的术语。为了区分它们，用小写字母书写终结符，而用大写字母书写非终结符。非终结符在语言上下文中称为变量或句法类别。

为了强调规则的生成特性，我们将用“Y可能被X替换”来代替“X是Y”：假如“tom”是一个名字的实例，则无论任何地方有一个名字，我们都可以用“tom”代替。这告诉我们：

0. 名称可以用“tom”、“dick”、“harry”代替

1. 句子可以用名字代替

2. 句子可以用句子、名字代替

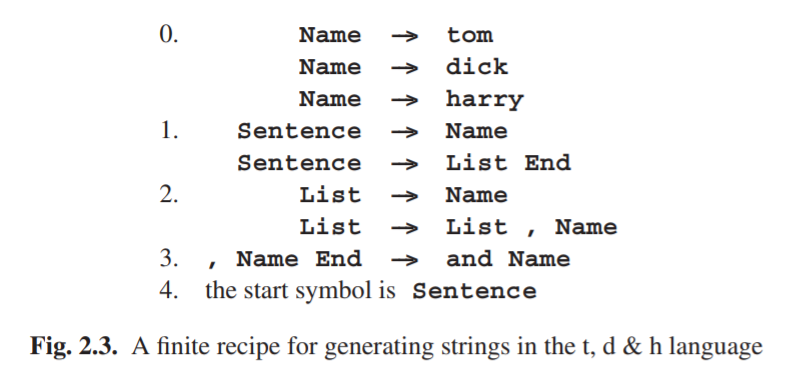
3. 名字被替换前，出现在句子末尾的“, 名字”必须用“and 名字”代替

4. 只有当句子不再包含非终结符时，句子才结束

5. 我们用句子开始替换程序

第0-3条描述替换规则，而第4条和第5条不同。第四条不是针对文法的，它通常是有效的且是必要规则之一。第五条告诉我们从哪里开始生成。这个名称很自然地被称为初始符，它是每个文法所必须的。

第3条依旧看起来不是那么清晰，大多数规则都有“可能被替换”，但是这个规则有“必须被替换”，且它指的是“句子的结尾”。其余的规则通过替换来完成，但问题是我们如何测试句子的结尾。这可以通过在它之后添加一个结束标记来解决。如果我们将结束标记设置为非终结符，除了从“, 名称”到“and 名称”的替换外任何地方都无法使用，我们会自动执行除非进行替换测试，否则不会完成句子的限制。为了简洁起见，我们写 ---> 而不是“可以替换”，因为终结符和非终结符现在被确定为技术对象，所以我们将用类似打字机的字体来书写他们。在--->之前的部分称为左侧，之后的部分称为右侧，规则的结果如图2.3。



这是一个简单且相对精确的形式，规则同样简单：从开始符开始，一直替换直到右侧没有非终结符。

**2.2 形式文法**

上面的规则形式，根据规则进行替换，足以作为正式语法的基础。类似的形式，通常称为“重写系统”，在数学家中有着悠久的历史，且已经在印度使用了几个世纪（参见Bhate和Kak[411]）。图2.3的具体形式首先被Chomskey[385]广泛研究。他的研究是几乎所有形式语言、解析器、一部分编译器构造和语言学的基础。

**2.2.1 形式文法的正式定义**

由于形式语言是数学的一个分支，这个领域的工作是使用一种特殊的符号来完成的。为了展示它的风格，我们将给出一个语法的正式定义，然后解释为什么它描述了一个如图2.3中所示的语法。所使用的形式体系对于正确性证明是必不可少的，但不是用于理解这些原则。在这里展示仅仅是给人一种印象，或填补空白。

定义2.1：生成文法是一个四元组(VN, VT, R, S)，如下所示：

1. VN和VT是符号的有穷集合，
2. VN交VT为空
3. R是（P,Q）对的集合，而

（3a）P属于VN与VT并集元素的至少一个元素的组合且

（3b）Q属于VN与VT并集元素的组合

（4）S属于Vn

四元组只是一个由四个可识别部分组成的对象，它们是按照非终结符，终结符，规则和初始符的顺序排列。上面的定义并没有说明这一点，所以这是老师要解释的。非终结符集合称为VN，终结符集合称为VT。对于我们的文法，有：

VN = {Name, Sentence, List, End}

VT = {tom, dick, harry, ,, and}

（注意，逗号“，”在终结符集中）

VN和VT的交集必须为空，由空集符号表示。因此非终结符和终结符没有一个共同符号，这是可以理解的。

R是所有规则的集合，P和Q分别是左侧和右侧。每个P必须由一个或多个非终结符序列组成，每个Q必须由零个或多个非终结符或终结符序列组成。对于我们的文法，有：

R = {(Name, tom), (Name, dick), (Name, harry),

(Sentence, Name), (Sentence, List End), (List, Name),

(List, List , Name), (, Name End, and Name)}

注意两个不同的逗号。

初始符S必须是VN的一个元素，也就是说它必须是一个非终结符：

S = Sentence

这结束了我们对正式语言学的考察，简而言之，正式语言的数学是一门语言，一门必须学习的语言，它可以非常简洁地表达什么和怎样，但是却很少提供关于原因的信息。考虑这本书的翻译和注释。

**2.2.2 从形式文法生成句子**

图2.3中的文法是我们语言的短语结构语法（通常缩写为PS语法）。有一种更简洁的语法，其中相同左侧的多个右侧对象被组合在一起，然后用竖线“|”分开。这个分隔符属于形式定义，就像箭头--->一样，可以被读作“或者”。由竖线分隔的右侧对象也被称为替代品。在这个更简洁的形式中，我们的文法变成了：

0. Name ---> tom | dick | harry

1. Sentences ---> Name | List End

2. List ---> Name | Name , List

3. , Name End ---> and Name

带有下标s的非终结符是开始符号（下标标记的是符号，而不是规则）。

现在让我们从这个文法中生成初始示例，只根据上述规则使用替换。我们获得下列连续形式的句子：

中间类型 使用的规则 说明

Sentence 初始符

List End Sentence ---> List End 规则1

Name , List End List ---> Name , List 规则2

Name , Name , List End List ---> Name , List 规则2

Name , Name , Name End List ---> Name 规则2

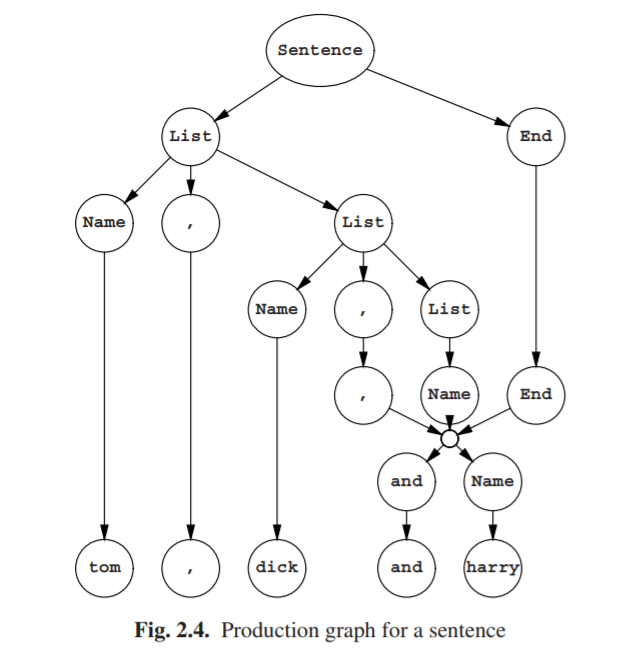
Name , Name and Name , Name End ---> and Name 规则3

tom , dick and harry 3次规则0

中间形式被称为句型。如果一个句型不包含非终结符则称为句子，并且属于生成的语言。

通过使用“图形”在相应符号之间绘制连接线，可以使生产过程更加可视化。图是由一组边连接起来的一组节点。一个节点可以被看做是纸上的一个点，边是一条线，每条线连接两个节点。一个点可能不止是一条线的端点。图中的节点通常是“被标记”的，这意味着它们被赋予了名称，而不仅仅是一个点。假如边是箭头，则图是一个有向图，如果边是线条，则图是无向的。几乎所有用于解析技术的图都是有向的。

与上述生产过程对应的图如图2.4所示。这样的图片称为生产图或句法图，描述了最终句子的句法结构（关于给定的文法）。我们看到生产图通常是扇面向下的，但偶尔也会看到因重写一组符号而导致的星型结构。



图中的环是一条从带着箭头的N开始，返回到N的路径。生产图不能包含环。我们将在后面看到，为了得到了一个环，我们需要在生产图中有一个非终结符节点N，该节点直接或间接生成子节点N。但是由于生产过程中总是为其生产的节点生成一个副本，所以它无法生成一个已经存在的节点。因此生产图总是“无环”的，有向无环图称为dag。

文法产生“tom, dick, harray”是不可能的，因为任何试图产生多个名字的尝试都会带有一个“End”，而唯一可以摆脱它的方法（我们必须摆脱它，因为它是一个非终结符）是被规则3吸收，这将产生“and”。令人惊讶的是，我们成功实现了“必须替换”的概念，而这个系统只用“可能替换”。仔细观察，我们发现我们已经将“必须替换”分为“可能替换”和“不能是非终结符”。

出了我们的标准例子，文法也可以产生许多其他的句子。例如：

harry and tom

harry

tom, tom, tom and tom

以及无限的其他的句子。如果硬是要产生不包含“and”的错误形式的句子，将会导致如下句型：

tom, dick harry End

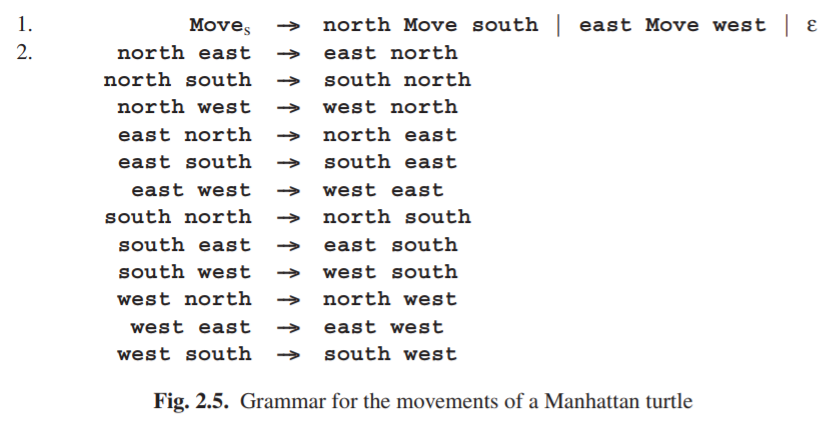
这并不是句子，也不适合任何产生式规则。这种形式被称为盲区。正如产生式规则中的向右箭头一样，规则不能被反向使用。

**2.2.3 形式文法的表现力**

形式文法的主要属性是它有产生规则，可以用来重写部分句子结构的一部分和初始符，它是所有句型的母亲。在产生式中，我们发现了非终结符和终结符，完整的句子只包含终结符。而其余的部分由文法作者和句子的生产者决定。

这是一个令人印象深刻而简洁的框架，这立即产生了一个问题：这足够吗？这很难说，但是如果这都还不够，我们就再没有更具表现力的东西了。尽管听起来很奇怪，但人类已知的所有其他生成集合的方法都被证明与短语结构文法相当或更弱。当然，生成一个集合的一个显而易见的方法是写一个程序来生成它，但是任何可以由程序生成的集合都被证明可以由短语结构文法来生成。甚至还有更加神秘的方法，但是都被证明不更具表现力。另一方面，也没有证据表明不存在这种更强大的方法。但是，鉴于许多不同的方法都遇到各种障碍，很难找到更强大的方法。参见Révész [394, pp 100-102]。

作为表现力的另一个例子，我们将给出一个曼哈顿乌龟运动的文法。曼哈顿乌龟在平面上移动，只能往北、东、南或西移动一个方块的位置。图2.5的文法生成了返回到它自己起点的所有路径。关于规则2，应该注意的是，许多作者要求左侧至少有一个非终结符。这个限制总是可以通过添加新的非终结符来实施。



简单往返东北西南方向的产生式如图2.6所示（名称缩写为一个字母）。注意规则1中的空白选择（ε），导致了图中第3个M消失。