习题课 第二章复习内容

彭怡腾 PB19000071

一 引入

为什么我们需要谓词逻辑?

因为我们希望更细致准确的表达更多的逻辑结构和推理。

例如:

x1：所有人都会死

x2：苏格拉底的父亲是人

--》

x3：苏格拉底的父亲会死

这个推理是正确的吗？是的

但是它没有办法在命题演算L中表达，为什么？因为在L中，我们没有办法很好的表达诸如“是”，“所有”这样子的关系，我们如果希望表达上述的推理，我们只能将其每个语句整体进行形式化，那么推理过程将变为$\{x_1, x_2\} \vdash x_3$，显然这个推理在L中是不成立的。

因此你不难想到，我们需要对之前的演算做出一些的**扩展**和**改进**，从而使得我们得到的这个新的工具能表达更多的内容。

那么或许有的同学就会想，那么我们得到的这个新的工具就是完备的吗，能表达所有的逻辑吗？答案当然是否定的，在这里，我们所学习的是一阶谓词逻辑，往上走还有高阶逻辑，LTL（线性时间逻辑），CTL（分支时间逻辑），CTL*等等，尽管这些逻辑或多或少都有自己无能为力的地方，但它们已经尽可能的在尝试去表达更多的、其他的逻辑没有覆盖到的内容，而新的逻辑也在不断的被提出，作为它们的补充扩展，这个方向便也这样子在不断的发展完善。

二 概念和语法层面

谓词演算公式集

就好像我们在学习算术运算时，首先要知道什么是数字，有了数字之后我们还需要把它们组合成十位数，百位数，甚至更多位数。然后，我们还需要知道什么是加减乘除的符号，或许一开始我们还不知道怎么利用它们进行运算，但至少第一步我们先需要认识，什么是数字，什么是符号，这里也是类似的。首先，我们要认识我们在谓词演算中能用来操作的最小单位是什么，其次是介绍谓词演算中的符号和其他的一些东西。

表示命题的最小单位：原子公式

正如同命题演算的最小单位是单个命题变元，谓词演算也有它的最小单位：原子公式。

我们首先来看看原子公式集的定义：

定义 2 (原子公式集) 原子公式集是指

$$Y = \bigcup_{i,n} \left(\{R_i^n\} \times \underbrace{T \times \cdots \times T}_{n\text{ 个 }T} \right)$$

即

$$Y = \{(R_i^n, t_1, \cdots, t_n) \mid R_i^n \in R, t_1, \cdots, t_n \in T\}.$$

以后常把原子公式 $(R_i^n, t_1, \cdots, t_n)$ 写成 $R_i^n(t_1, \cdots, t_n)$.

这个定义乍一看并不好理解，什么是R，什么又是T？如果你对代数结构有印象，你会发现这里其实是很多个**积集**，不记得什么是积集了也没有关系，我们可以这样子理解：

每个**原子公式**是这样子的一个元组$\left(R_i^n, t_1, \ldots, t_n\right)$，第一个元素是谓词，后面n个元素是项。

它一般也写作$R_i^n\left(t_1, \ldots, t_n\right)$，这样子写，想必对于各位计算机专业的同学来说就更好理解了，它就类似于一个n元函数，每个t_i是这个**谓词**R_i^n的参数。

那么我们不由得引出下面两个问题，什么是**项**？什么又是**谓词**？

谓词

首先，我们刚刚也说过，谓词演算的**最小单位**是原子公式，那么我们回忆一下命题演算的最小单位：命题变元，它有什么特点：即使它作为一个单独的公式，我们也依然可以判断它的真假。

那么原子公式同样也作为一个最小单位，它也应该具备相同的特点。我们来看一个例子：

x > y * z

在这个例子中，x，y，z单独拿出来都无法判断真假，即使是y * z也不可以，因此在这里，我们就可以把>看作一个谓词：R_1^2，这里上标的2代表参数的个数，代表有两个参数，下标的1代表序号，代表这是有两个参数的第1个谓词。因此，将y*z记作s后，这个命题我们可以写作：$R_1^2(x, s)$

项

依然是上面那个例子，我们在最后将y*z记作了s，但其实我们不难想到，类似于谓词的形式化表示，我们完全可以用更细粒度的方式去表达y*z这个形式。

运算符由此诞生，运算符记作f_i^n，和谓词类似上标的n代表参数的个数，下标的i代表n个参数的运算符中，该运算符的序号。

而上述例子中的y和z，就是*这个运算符的参数，它们也是项，因此我们可以定义最小的项：**变元**和**常元**，这两个概念并不用多讲，它们就类似于编程语言中变量和常量一般，代表着某些可以变化的个体对象和确定的个体对象。

经过运算符得到的结果是项，运算符的参数是依然是项，因此这是一个递归的定义，通过这个递归的定义，以及变元和常元的概念，我们就可以定义项集：

(i) 个体变元 $x_i(\in X)$ 与个体常元 $c_i(\in C)$ 都是项.

(ii) 若 $t_1,\cdots,t_n$ 是项，则 $f_i^n(t_1,\cdots,t_n)$ 也是项 $(f_i^n\in F)$.

(iii) 任一项皆如此形成，即皆由规则 (i)，(ii) 的有限次使用形成.

当运算符集 $F=\varnothing$ 时，规定项集 $T=X\cup C$.

按上述规则，当 $F\neq\varnothing$ 时，项集 T 可如下分层:

$$T=T_0\cup T_1\cup T_2\cup\cdots\cup T_k\cup\cdots$$

其中

$$\begin{aligned}
T_0 &= X\cup C=\{x_1,\ x_2,\ \cdots,\ c_1,\ c_2,\ \cdots\},\\
T_1 &= \{f_1^1(x_1),\ f_1^1(x_2),\ \cdots,\ f_1^1(c_1),\ \cdots\\
&\quad f_2^1(x_1),\ \cdots,\ f_2^1(c_1),\ \cdots\\
&\quad \cdots\cdots\\
&\quad f_1^2(x_1,\ x_1),\ \cdots\\
&\quad \cdots\cdots\\
&\quad f_1^3(x_1,\ x_1,\ x_1),\ \cdots\\
&\quad \cdots\cdots\},\\
T_2 &= \{f_1^1(f_1^1(x_1)),\ \cdots,\ f_1^2(x_1,\ f_1^1(x_1)),\ \cdots\},\\
&\quad \cdots\cdots
\end{aligned}$$

第 k 层项 (T_k 的元素) 含有 k 个运算符. 换句话说，第 k 层项由零层项 (个体常元、个体变元) 经 k 次运算得来. 更简单地说: 项集 T 是由 $X\cup C$ 生成的 F 型代数 (即以 F 为运算集的代数系统).

小结

接下来对上面的内容做一个小结，首先，我们知道了**项**，它不是谓词演算的最小单位，但它类似于谓词的参数，因此也必不可少。我们还知道了**运算符**，它能将我们的项进行扩充，使得我们拥有更强大的表达项的方式。然后是**谓词**，谓词演算以此命名，它的重要性可见一斑，由谓词和项构成的一个个**原子公式**，组成了我们谓词演算的最小单位。

谓词演算公式集

这里我们直接看定义

建立谓词演算公式集前，先列出我们所采用的这种形式语言的字母表如下:

个体变元 $x_1,\ x_2,\ \cdots$	(可数个)
个体常元 $c_1,\ c_2,\ \cdots$	(可数或有限个)
运算符 $f_1^1,\ f_2^1,\ \cdots,\ f_1^2,\ f_2^2,\ \cdots$	(可数或有限个)
谓词 $R_1^1,\ R_2^1,\ \cdots,\ R_1^2,\ R_2^2,\ \cdots$	(可数或有限个，至少一个)
联结词 $\neg,\ \rightarrow$	
全称量词 $\forall$	
左右括号、逗号 (,), ,	

形成谓词演算公式集的过程与命题演算类似，不同的是:

第一，现由原子公式集 Y 出发，而不是由命题变元集出发;

第二，除了一元运算 $\neg$ (否定) 与二元运算 $\rightarrow$ (蕴涵) 外，现还增加了可数个一元全称量词运算 "$\forall x_i$" $(i=1,2,\cdots)$.

谓词演算公式的形成规则是:

(i) 每个原子公式是公式.

(ii) 若 p,q 是公式，则 $\neg p$, $p\rightarrow q$, $\forall x_i\,p\,(i=1,2,\cdots)$ 都是公式.

(iii) 任一公式皆如此形成，即皆由规则 (i)，(ii) 的有限次使用形成.

在第一部分中，四句话是在覆盖原子公式的定义，前一节我们已经了解过了。

第二部分表达了我们可以使用的符号，逗号是为了原子公式提供的，不过多赘述，**否定**和**蕴含**这两个符号我们之前已经接触很多了，唯一的一个新符号是∀符号，这个我们之后会经常打交道。

第三部分中给出了公式的定义，也是前面那么多铺垫之后终于引出的内容。

我们将上述的公式集定义为**K(Y)**，注意，这里是**K(Y)**,而非之后的谓词演算**K**，这种感觉就好像是，这里的K(Y)还是静态的公式，还没有通过后面定义的运算为其注入活力，因此务必区分这两个符号。

接下来，我们通过上面的三个符号又引出四个新的符号，方便我们更便捷的表达一些逻辑结构：

除了 $\neg, \rightarrow$ 和 $\forall x_i$, 还可在 $K(Y)$ 上定义新的运算 $\vee, \wedge, \leftrightarrow$ 及 $\exists x_i$ (存在量词运算), 定义式是:

$$
\begin{aligned}
p \vee q &= \neg p \rightarrow q, \\
p \wedge q &= \neg (p \rightarrow \neg q), \\
p \leftrightarrow q &= (p \rightarrow q) \wedge (q \rightarrow p), \\
\exists x_i p &= \neg \forall x_i \neg p.
\end{aligned}
$$

注意公式 $\forall x(p \rightarrow q)$ 和公式 $\forall x p \rightarrow q$ 的区别. 前者 $\forall x$ 的作用范围 (简称“范围”) 是 $p \rightarrow q$, 而后者 $\forall x$ 的范围是 p.

这里需要注意的是，我特意将紧跟其后的一句话也截了下来，它不难理解，但有的同学容易忽略，请务必注意。

至此，我们便完成了最基础的一些定义，这些定义我写的篇幅较多，原因是将这些定义全部理顺之后，看后面的内容也较容易理解，因此后面的内容可能会在一定程度上加快叙述的进度。

任意符号

给出了任意符号，自然要给出与其相关的一些扩展的定义，实际上很多题目就是在考察大家有没有将这些定义**翻译**成形式化语言的能力。

下面我们逐个考察课本中的三个定义的关键点：

定义 1 (变元的自由出现与约束出现) 在一个公式里，个体变元 x 的出现如果不是在 $\forall x$ 中或在 $\forall x$ 的范围中，则叫做自由出现，否则叫做约束出现.

这个定义非常需要注意的一点是$\forall x$这个里面的x也是约束的，这里很多同学会忽略。

定义 2 (闭式) 公式若不含自由出现的变元，则叫做闭式.

例如，公式 $\forall x_1(R_1^2(x_1, x_2) \rightarrow \forall x_2 R_1^1(x_2))$ 不是闭式，因为 x_2 在其中自由出现一次. 公式 $\forall x_1 (R_1^2 (x_1, c_1) \rightarrow \forall x_2 R_1^1 (x_2))$ 是闭式.

这个定义是在上述定义的基础上进行的，没有什么多注意的地方。

定义 3 (项 t 对公式 p 中变元 x 是自由的) 用项 t 去代换公式 p 中自由出现的个体变元 x 时，若在代换后的新公式里，t 的变元都是自由的，则说 t 对 p 中 x 是可自由代换的，简称 t 对 p 中 x 是可代换的，或简称 t 对 p 中 x 是自由的.

换句话说，用 t 代换 p 中自由出现的 x 时，若 t 中有变元在代换后受到约束，则说 t 对 p 中的 x 是“不自由的”(或“不可自由代换的,”或“不可代换的”).

下面两种情形，t 对 p 中 x 是自由的:

1° t 是闭项;

2° x 在 p 中不自由出现.

此外，在任何公式中，x_i (作为项) 对 x_i 自己总是自由的.

这个定义有点绕口，有部分同学理解起来有一定的难度，这里我们用一种计算机的视角去看这个定义：

```
bool val = 自由
step1：判断p中有没有自由的x
step2：若有自由的x，把t替换进去，判断是否有t中变元被约束，若有，val=约束
```

那么我们就不难理解，为什么特意提到的第二点：x在p中不自由出现，也会使得t对x自由了，因为我们在第一步判断的时候就失败了，即，当p中没有自由的x给t换的时候，自动判断为t对p中x自由。

谓词演算

定义

有了上面的符号和公式，我们终于可以定义谓词演算语法层面的运算方法了，这里直接给出两个定义再做说明：

谓词演算 K 是用公式集 $K(Y)$ 来定义的，与命题演算 L 的定义方式类似.

定义 1 (谓词演算 K) 谓词演算 K 是指带有如下规定的“公理”和“证明”的公式集 $K(Y)$.

1° “公理”

取 $K(Y)$ 中以下形状的公式作为“公理”.

(K1) $p \to (q \to p)$;

(K2) $(p \to (q \to r)) \to ((p \to q) \to (p \to r))$;

(K3) $(\neg p \to \neg q) \to (q \to p)$;

(K4) $\forall x p(x) \to p(t)$, 其中项 t 对 $p(x)$ 中的 x 是自由的;

(K5) $\forall x(p \to q) \to (p \to \forall x q)$, 其中 x 不在 p 中自由出现.

以上给出的是五种公理模式，其中 $p, q, r, p(x)$ 都是任意的公式.

2° “证明”

设 p 是某个公式，Γ 是某个公式集. p 从 Γ 可证，记作 $\Gamma \vdash p$, 是指存在着公式的有限序列 $p_1, \cdots, p_n$, 其中 $p_n = p$, 且对每个 $k = 1, \cdots, n$ 有

(i) $p_k \in \Gamma$, 或

(ii) p_k 为公理，或

(iii) 存在 $i, j < k$, 使 $p_j = p_i \to p_k$ (此时说由 p_i, $p_i \to p_k$ 使用 MP 得到 p_k)，或

(iv) 存在 $j < k$, 使 $p_k = \forall x p_j$. 此时说由 p_j 使用“Gen”(“推广”) 这条推理规则得到 p_k. x 叫做 Gen 变元 (Gen 是 Generalization 的缩写).

我们先找一下熟悉的内容，熟悉的内容有这样一些：

1、K1，K2，K3和L中的L1，L2，L3形式上是一样的。但我们同样也要认识到，它们的本质上已经发生了一些区别，这里公式p的定义和L中公式p的定义已经大不一样了，因此这里的相似只是形式上的相似，切不可混为一谈，但也是从这里，我们可以看出一点K是在L上扩展的特征。

2、“证明”中，前三条的性质形式上是一样的，这里我又使用了形式上这个词，大家应该能体会这个意思。

我们接下来关注不一样的地方：

1、首先我们关注K5，K5理解起来很容易，x不在p中自由出现，那么也就是说$\forall x$至始至终都没有约束到p中的任何x，那么我们完全可以将它提到后面来。

2、K4，K4这条规则非常重要，它将一般的对象进行了特殊化，这就好像是我们将一个类（一般的），进行了实例化（变成了特殊的），我们当然不想在这个实例化的过程中引入新的约束，因为这或许会改变命题的意思，因此最后的那个条件是必要的。

3、“证明”中多了一条Gen规则，它和K4直观上是互补的两个规则，K4将一般特殊化，而Gen则将特殊一般化。[2]中给了一个很不错的解释的例子：

我们需要花些时间来理解这个规则，因为这个规则看起来似乎是从 ϕ 的特殊情况得到一般情况 $\forall x\phi$。x_0 不出现在框外的任何地方的辅助条件使得我们能够实现这个过渡（从特殊到一般——译者注）。

为了理解这一点，考虑下面的推理。如果想证明可以通过挤压的方式将手中的乒乓球压扁，你可以说：“给我一个乒乓球，我会将它压扁”，然后，给你一个乒乓球，你压扁它。但是，怎么才可以使我们相信你可以通过这种方式把任何一个乒乓球压扁呢？当然，不可能把所有的乒乓球都给你，又如何确信你可以压扁呢？好的，现在假设你压扁的那个乒乓球是一个任意的或“随机”的，即它一点也不特殊，就像你事先“准备好的”球；这足以使我们相信你可以通过这种方式把任何一个乒乓球压扁！我们的规则是说，如果你可以证明对没有任何特殊性的 x_0，ϕ 是真的，那么你能够证明，对任意的 x，ϕ 是真的。

换个角度考虑，只有在没有一个假设包含 x 作为自由变量的方式得到 ϕ，那么，从 ϕ 到 $\forall x\phi$ 的证明步骤才是合法的。任何以 x 为自由出现的假设均对这样的 x 加了约束。例如，假设 bird(x)将 x 约束到了“鸟的领域范围之内”，这样我们用此公式所能证明关于 x 的所有内容，只能是约束到鸟类的一个陈述论断，而不是我们可能想到的其他内容。

这边我再做进一步的阐述：我们将pj中的所有自由出现的x，用一个随机的变元x0代替，这个x0是我们新引入的，它不在这次证明的其他任何地方出现。这样子进行了代换之后的公式，和原来公式的性质应该是一样的，因为我们是用一个不在其他任何地方出现x0，代换了pj中自由的x。那么接下来考虑到我们这个x0是一个随机的，或者说，是任意的。那么我们就可以将原来的pj进行一个推广，就有理由认为∀xpj也是可以推导出来的，即Gen规则。

K和L的关联

定理 1 设 $x_1,\cdots,x_n$ 是命题演算 L 的命题变元，$p(x_1,\cdots,x_n)\in L(X_n)$. 我们有

$$\vdash_L p(x_1,\cdots,x_n) \quad\Rightarrow\quad \vdash_K p(p_1,\cdots,p_n),$$

其中 $p_1,\cdots,p_n\in K(Y)$, $p(p_1,\cdots,p_n)$ 是用 $p_1,\cdots,p_n$ 分别代换 $p(x_1,\cdots,x_n)$ 中的 $x_1,\cdots,x_n$ 所得结果.

定义 2 (命题演算型永真式，简称永真式) 若 $p(x_1,\cdots,x_n)\in L(X_n)$ 是命题演算 L 中的永真式，则对任意 $p_1,\cdots,p_n\in K(Y)$, $p(p_1,\cdots,p_n)$ 叫做 K 的命题演算型永真式，简称为永真式.

这两个定理和定义很好理解，但是我们需要注意的是，即使是课本，也是没有直接说L里面成立的公式就能直接在K里面成立的，而是通过这两个定义进行了连接，因此大家也不要把K和L混为一谈，K是L的推广，但是同样是一个独立的逻辑。

演绎定理

定理 2 (演绎定理)

1° 若 $\Gamma\vdash p\to q$, 则 $\Gamma\cup\{p\}\vdash q$;

2° 若 $\Gamma\cup\{p\}\vdash q$, 且证明中所用 Gen 变元不在 p 中自由出现，则不增加新的 Gen 变元就可得 $\Gamma\vdash p\to q$.

这里和L中的演绎定理形式上没有什么区别，但是切记这里的2有前提条件，原因是证明中用到了K5，我们刚刚有说到K5直观上为什么是对的，如果演绎定理2使用的时候不考虑前提条件，那么就是没有考虑到K5的前提条件，这样子的证明是有漏洞的。

等价可替换性和前束范式

定理 1 (子公式的等价可替换性) 设公式 q 是公式 p 的子公式: $p=\cdots q\cdots$. 用公式 q' 替换 p 中的 q (一次替换) 所得结果记为 $p'=\cdots q'\cdots$. 则有

$$\Gamma\vdash q\leftrightarrow q' \quad\Rightarrow\quad \Gamma\vdash p\leftrightarrow p'.$$

这里得单独拿出来提一下这个等价可替换性，**等价可替换性是K中的，L中似乎只有一个等值可替换**，大家切忌张冠李戴，虽然这种代换是很爽，但是切忌随意代换，代换的时候写清楚依据，不能是“我觉得这里没问题所以我换了”，这种情况，我们是形式化的证明，对规范性和证明依据要求比较严格，因为形式化的验证在计算机这边很多是跟安全有关的，因此这种情况下，每一步都要保证严格的准确，不能纯靠感觉。

定义 1 (前束范式)　前束范式，指形如

$$Q_1 x \cdots Q_n y\, p$$

的公式，其中 $Q_1, \cdots, Q_n$ 表示量词符号 $\forall$ 或 $\exists$, 尾部 p 是不含有量词的公式.

前束范式的形式不唯一，难度不大，但是需要关注，考试的时候可能较繁琐，需要一定的细心，不过多赘述，大家知道有这个东西，会按照课本上的步骤按部就班的变化就好。

小结

让我们简单回忆一下刚刚语法层面的路线，首先我们发现命题演算的能力不足，因此我们添加了“谓词”，然后我们希望更精细的表达项，因此我们添加了“运算符”，然后我们还希望表达“所有”和“存在”的概念，因此我们添加了全称量词和存在量词。

这样子我们就有了类似于算数里面的数字和符号，但这还不够，我们还需要运算的法则。于是我们又添加了谓词演算K的公理和规则，**在这个体系下**，我们可以认为公理都是正确的，因此通过这些正确的公式，我们通过运算的规则——证明，从而可以得到很多的新的公式，这些由公理和规则推出得到的公式在我们这个体系下同样也是正确的。

于是当我们试图用语法层面的内容去证明一个实际的问题的时候，思路可能就变成了，首先将一个实际的问题抽象成一个公式，接下来利用我们这个体系的公理和规则去尝试推导这个公式，如果能推导出来，则代表这个公式**在谓词演算的体系下**是正确的，从而一定程度上能说明实际问题的正确性。

三 语义层面

多说一句，这里将语义层面排序排到了语法层面的后面，是因为课本上是这样子的顺序，因此我们大体上还是根据课本的顺序来，但是我个人认为可能先谈语义层面的更为合适，因为这更直观。语义层面更适合于人去求解，而语法层面更适合计算机去求解，这也是很多同学在命题演算里面会觉得列真值表比较轻松的原因，但是语义层面对于计算机来说开销有点大：10个变量的真值表就已经有2^10的规模了。而语法层面，依据公式去尝试或者利用一些其他的算法去符号化的求解公式就好，因此一定程度上开销更小。

通往目的的路不止一条，证明逻辑成立的方法也不限于一种。那么现在，就请抛开之前学的语法层面的公式，定理，让我们换一个角度重新来看谓词逻辑。

解释域和项解释

没有了那些抽象的公理和规则，我们应该如何证明一个公式的正确性？我们可以从引入谓词演算的那一步找到答案：从现实中来，回到现实中去。我们首先将现实中的东西一步步的抽象，抽象成了**谓词，运算符，变元，常元。**那么现在我们希望将这些一般的东西重新特殊化，以判断它们的正确性。

于是很自然的，我们需要去解释我们的那些抽象符号的含义，解释域的概念随即产生：

定义 1 (K 的解释域) 设非空集 M 具有以下性质:

1° 对 K 的每个个体常元 c_i, 都有 M 的元素 $\overline{c_i}$ 与之对应:

$$c_i \mapsto \overline{c_i},\ \ \overline{c_i} \in M;$$

2° 对 K 的每个运算符 f_i^n, 都有 M 上的 n 元运算 $\overline{f_i^n}$ 与之对应:

$$f_i^n \mapsto \overline{f_i^n},\ \ \overline{f_i^n} \text{ 是 } M \text{ 上的 } n \text{ 元运算};$$

3° 对 K 的每个谓词 R_i^n, 都有 M 上的 n 元关系 $\overline{R_i^n}$ 与之对应:

$$R_i^n \mapsto \overline{R_i^n},\ \ \overline{R_i^n} \text{ 是 } M \text{ 上的 } n \text{ 元关系}.$$

又是一如既往的符号化的定义，但是它想表达的东西其实很浅显，那就是，我们希望将**谓词，运算符，常元**，都在代数结构M中找到对应的对象与之对应。

谓词，运算符，常元，我们还少了什么？变元。

为什么变元需要单独拿出来讲？我个人的理解，还是用程序设计中类和对象的思想，我们构造解释域的这个过程，就好像将公式（这个类）进行了实例化，而变元则是这个类里面那些函数的形参，一些基本的属性，方法都是可以固定下来的，正如公式里面不变的谓词，运算符，常元一样，而变元本身是代表着变化，因此我们需要做进一步的解释。

项解释的定义如下：

(1) $\varphi(x_i) = \varphi_0(x_i),\ \varphi(c_i) = \overline{c_i}$.

若 $\varphi(t_1), \cdots, \varphi(t_n)$ 已有定义，则令

(2) $\varphi(f_i^n(t_1, \cdots, t_n)) = \overline{f_i^n}(\varphi(t_1), \cdots, \varphi(t_n))$,

即，对给定的解释域，每个变元都有了确定的解释，便有了确定的项解释。这是一个归纳的定义，在部分归纳证明的过程中，可能会利用到(2)，即利用到项解释的**保运算性**。

这里我们确实可以给固定的变元一个固定的解释，但是我们难免想到，对任意符号约束下的变元怎么办？比如：所有人都会死，这里我们怎么去解释这个“所有人”？我们需要**变元变通**的定义：

定义 2(项解释的变元变通) 设 x 是某个给定的个体变元，y 是任意的个体变元，且 $\varphi, \varphi' \in \Phi_M$ 满足条件

(3) $y \neq x \ \Rightarrow\ \varphi'(y) = \varphi(y)$,

则把 φ' 叫做 φ 的 x 变通. (此时 φ 与 φ' 互为对方的 x 变通.)

这个定义直接说来就是，需要关注的变元x的解释进行变化，而其他变元的解释完全不变，这就叫做x的变通。这个定义最大的作用就是，我们有了一个解释任意的工具，我们只需要令其他的解释不变，对任意符号约束的变元进行变通，就可以较好的解释“任意”这个性质了。

公式的解释

经过上面的定义，我们可以很好的解释原子公式了，但是这还不够，回忆语法层面的定义，我们还需要对公式进行解释，这个解释可以由原子公式的解释归纳而来。

定义 1 (公式的赋值函数)　设 M 是给定的解释域，p 是 K 中任一公式. 由公式 p 按下面的方式归纳定义的函数 $|p|: \Phi_M \to \mathbf{Z}_2$ 叫做公式 p 的赋值函数.

对任一项解释 $\varphi \in \Phi_M$,

(i) 当 p 为原子公式 $R_i^n(t_1, \cdots, t_n)$ 时，令

$$|p|(\varphi) = \begin{cases} 1, & \text{若 } (\overline{t_1}, \cdots, \overline{t_n}) \in \overline{R_i^n}, \\ 0, & \text{若 } (\overline{t_1}, \cdots, \overline{t_n}) \notin \overline{R_i^n}; \end{cases}$$

(ii) 当 p 是 $\neg q$ 或 $q \to r$ 时，令

$$|\neg q|(\varphi) = \neg |q|(\varphi),$$

$$|q \to r|(\varphi) = |q|(\varphi) \to |r|(\varphi);$$

(iii) 当 p 是 $\forall x q$ 时，令

$$|\forall x q|(\varphi) = \begin{cases} 1, & \text{若 } \varphi \text{ 的任一 } x \text{ 变通 } \varphi' \text{ 都使 } |q|(\varphi') = 1, \\ 0, & \text{若存在 } \varphi \text{ 的 } x \text{ 变通 } \varphi' \text{ 使 } |q|(\varphi') = 0. \end{cases}$$

第一条中，“属于”这个符号需要多注意一下，这里是用集合的角度去看待谓词，例如我们可以考虑R_1^1这个谓词，它解释为“是学生”，那么这个集合中就包含所有的学生对象，接下来我们只需要查看项的解释是否在这个集合中，就可以知道对应的原子公式是否成立了。

第二条不用多说，又是一个保运算性的形式。

第三条值得多关注一下，这里就是我们之前定义的**变元变通**派上用场的时候了，我们固定其他的变元不动，只变动约束的x的指派，从而得以判断这个解释是否对任意的x都为真。

除了这三条之外，这个定义的整体也是很有趣的，它将我们之前对项的解释扩展到了公式层面，但是它又不完全是一种解释了。从直观的角度理解，我们将公式整体做出了解释，于是可以判断公式的真假。这是一种映射的感觉，因此这里我们称之为公式的赋值函数，它将我们每一个单独的项解释，映射到了公式整体二元的真或假上。

闭式的语义特征

这一节实则是上面两节的扩展，因此在课本中被单独拿出来讲。

一个直观上明显的事实是: 项的解释只与在该项中出现的变元的指派有关，与其他变元的指派无关；公式的真值也只与在该公式中出现的自由变元指派有关，而与其他变元的指派无关. 这就是下面的命题 1.

命题 1　设 M 是 K 的解释域，$\varphi, \psi \in \Phi_M$.

1° 若对项 t 中的任一变元 x 都有 $\varphi(x) = \psi(x)$, 则 $\varphi(t) = \psi(t)$.

2° 若对公式 p 中任一自由出现的变元 x 都有 $\varphi(x) = \psi(x)$, 则 $|p|(\varphi) = |p|(\psi)$.

这里要关注的一点是，通过命题1和上面的解释，我们可以知道，**公式的真值只和自由变元的指派相关**，如果熟记了上面关于任意符号解释的定义，我们不难理解这一点。

被**任意符号**约束的变元的真值是由**变元变通**决定的，也就是说，被约束的变元在解释的时候就是需要在不断的**变化**的指派中，依然使得公式为真，这个公式的赋值函数，也即公式的解释才使得公式为真。但如果是希望证明解释使公式为假，那么取一个特例就好。这也是符合我们的常识的，当有人告诉你：所有的xxx都xxx的时候，你取个反例就可以打他的脸了。

但是相对的，我们也应该知道，当有人说：存在xxx会xxx的时候，这时候你想打他的脸就麻烦了，你得证明所有的情况都是不成立的。这在我们的谓词逻辑的解释中也是一样的，我们可以看到，存在符号定义时是将任意符号取反了的，也就是说，当我们希望证明存在符号的解释为0时，我们应该像证明任意符号取1一样，证明所有的情况都不成立。

请大家务必注意上面这一段，因为在最后一次作业中非常多的同学这里做得都有问题。

接下来看这个定义：

定义 1 (公式在解释域中恒真与恒假)　公式 p 在解释域 M 中恒真，记作 $|p|_M = 1$, 是指对任一 $\varphi \in \Phi_M$, $|p|(\varphi) = 1$;

公式 p 在解释域 M 中恒假，记作 $|p|_M = 0$, 是指对任一 $\varphi \in \Phi_M$, $|p|(\varphi) = 0$;

在解释域 M 中非恒假公式叫做在 M 中可满足公式.

两个定义需要关注，一个是这个**恒真/恒假**的书写形式，在题目中碰见时不要误当作是公式的赋值函数。另一个是可满足公式，这个大家在考试和作业中可能碰的比较少，但是还是需要知道，如果以后的领域涉及这个方向，可能会经常和这个词打交道，这个词不难理解，就是能够找到一个项解释使得公式的解释为真，因此公式“可以满足”。

接下来我们用这个命题来看看大家对前面的这些定义有没有很好的理解：

命题 2　$|p|_M = 1 \quad \Leftrightarrow \quad |\forall x\, p|_M = 1.$

如果你可以一下子理解这个命题为什么正确的，那么前面的内容你应该理解得就比较到位了。首先这里左项是一个恒真式，也就是说，对于任意的项解释，这个公式都解释为真，而我们前面说过，公式的真值只和自由变元有关，那么也就是说，对于p中自由的x，所有的项解释都为真，那么进一步的，对于任意的一个项解释，它的x的变元变通当然也为真，从而可以推出右项也是恒真的。

语义推论

$$\Gamma \models p$$

这个形式想必大家都不陌生，我们在命题演算的语义层面也经常见到，但是还记得我上面说过的吗，谓词演算是扩展，但是也是一个新的东西了，因此我们需要重新定义这些概念。

定义 1 (模型)　设 M 是 K 的一个解释域. M 是公式集 Γ 的模型，指 Γ 的每个公式都在 M 中恒真:

$$r \in \Gamma \quad \Rightarrow \quad |r|_M = 1.$$

$\Gamma = \varnothing$ 时任何解释域都是 Γ 的模型.

首先我们定义了模型的概念，模型这个概念就好像是前提条件。Γ是一个公式集，这个公式集里面放的就是我们已知的前提条件抽象出的公式，我们当然需要他们在现在给我的这个解释域中保持恒真（要是给的前提条件不一定是真的，那就太恐怖了），因此基于这个考虑，我们很自然的可以得到模型的定义。

定义 2 (语义推论)　公式 p 是公式集 Γ 的语义推论，记作 $\Gamma \models p$, 指 p 在 Γ 的所有模型中都恒真，即: 在使 Γ 的每个成员都恒真的解释域中，p 也恒真；或者说，Γ 的任何模型也都是 $\Gamma \cup \{p\}$ 的模型.

定义 2 也可写成

$$\Gamma \models p \quad \Leftrightarrow \quad \text{当每个 } r \in \Gamma \text{ 都有 } |r|_M = 1 \text{ 时，也有 } |p|_M = 1.$$

在定义了上面的模型，也就是定义了“前提条件在固定解释域下的恒真”之后，我们终于可以给出上面那个形式的定义了。这个定义通俗来说就是，在任何使得我们的前提条件恒真的解释域中，右侧这样子写出的p，也就是我们推导出的结论，也应该恒真。

定义 3 (有效式与满足公式)　$\varnothing \models p$ 时，p 叫做 K 的有效式，记为 $\models p$.

若 $\neg p$ 不是有效式，则 p 叫做 K 的可满足公式.

由有效式的定义可知

$$\models p \quad \Leftrightarrow \quad p \text{ 在 } K \text{ 的所有解释域中恒真.}$$

命题 1　K 中 (命题演算型) 永真式都是有效式.

这里可满足公式的概念前面已经出现过了，想表达的内容是一样的。有效式的概念也不难理解：在没有前提条件的情况下，需要对所有的解释域都恒真，这样子才是一个有效式。这里唯一需要多看一眼的地方是，需要将永真式和有效式进行区分，有效式是谓词演算中的术语，而永真式是命题演算中的术语，当心张冠李戴。

命题 2 $\Gamma \models p$ 且 $\Gamma \models p \to q \quad \Rightarrow \quad \Gamma \models q.$

这个性质是一个很有趣也很方便的性质，它的形式上就仿佛语法层面的MP，但是它不是MP，还记得我们在刚刚介绍语义层面的时候所说的吗，它和语法层面是两条路，因此在这条路上我们没有MP可用了，而这个命题则告诉我们，我们虽然名义上没有了MP，但是它在这条路上，依然成立，我们依然可以用类似的形式，只不过它换了个名字，叫做**命题2**。所以大家如果希望在谓词演算的语义层面也使用类似MP的形式的话，请记得，不要写MP，它是命题2。

小结

让我们也回忆一下语法层面做的工作，首先我们希望对每一个原子公式做出解释，但是首先我们只能对那些不变的符号做出解释，所以我们定义了**解释域**，它能对谓词，运算符，常元做出解释，进一步，我们希望对变元做出解释，因此我们又定义了**项解释**，最后，我们希望能对我们的公式做出解释，于是我们又在公式符号的基础上归纳定义了公式的解释——**赋值函数**，但是这样子还不够，我们希望知道哪些公式是在所有解释域中都恒真的，因此我们有了**有效式**这个概念，然后我们还希望通过前提推出结论，于是又有了**模型**——使得前提公式都为真的解释域，以及**语义推论**——它形式化的代表了从恒真的前提推导出恒真的结论的这个过程。

四 可靠性和完全性

这里课本上用了两节来证明，我们此处不过多的讲解证明过程（那太困难了，而且也没过多的必要，感兴趣的同学可以去深究），就应用的层面来讲，我们只需要知道这样两件事情：

首先，语义层面和语法层面不是一个东西，不要**直接**把它们里面的一些性质和处理方法混为一谈。也就是说，如果题目是⊢，但是你却不假思索的使用╞里面的方法去证明，这代表你可能没有理解它们的不同，可能会进行扣分。

其次，我们利用可靠性和完全性可以打通这两个层面的桥梁，使得可以将一个层面的证明问题转化为另一个层面的，注意，这里是转化，你依然不可以混着使用，你可以将问题不停的转化，转化到哪个层面就用哪个层面的方法，但是你不应该同时使用两个层面的方法。

总的来说，我个人的感觉就好像是，我们使用乘法公式也可以算乘法，使用电路也可以计算乘法，但是这两者虽然输入一样，结果一样，里面处理的手段方法却是不同的。同样的，我们需要认识到语义和语法的不同，但同时也应该认识到，很厉害的一点，它们殊途同归。

五 题外话

1、或许很多同学以后再也不会过多的接触这个方向的内容，但希望大家可以从这门课中学到两个重要的思路，一个是发现问题，解决问题的思路，我们发现了L中的不足，因此扩展到了K，虽然K也不够尽善尽美，但是至少我们迈出了新的一步，让形式化的工具更加完善了一点。第二个是这种严谨证明的思想，以直观的理解为辅助，但是在走出每一步的时候都有着严格可信的依据，这种思想或许能在很多方面帮到我们。

理论和技术在不断的发展迭代，但是思想总是闪着光的，也衷心希望大家在学习其他课程的时候多多关注课程的思想以及对自己的启发。

2、建议的复习顺序：熟悉课本基础概念>作业题>课后其他习题>其他的资料。当然，大家有不同的复习思路，这里仅供参考~

3、这个课程相关的方向有：形式化方法，软件测试，编程语言，信息安全……也欢迎感兴趣的同学作进一步的研究。

六 参考内容

[1] 数理逻辑(第2版) 汪芳庭

[2] 面向计算机科学的数理逻辑-系统建模与推理

[3] 陈小平老师-数理逻辑-2020年春相关课件

[4] 黄文超老师-形式化方法导引-2022年春相关课件