《计算范畴论》导读

甄景贤 generic.intelligence@gmail.com

October 20, 2018

恶补数学好几年之后，最近看懂了很多书。《Computational category theory》是 Rydeheard & Burstall 1988 年的书，但至今还没有类似的课本。

我觉得这本书非常重要，因为它涉及到 **unification algorithm**。现时人工智能中最关键的问题似乎是如何从 propositional logic（命题逻辑）过渡到 relational 或 first-order logic（一阶谓词逻辑），特别是如何找到高效率的学习算法。关於 logic-based AI 的基础可以看看《AI — a modern approach》这本书。

Unification algorithm（同一化算法）是 谓词逻辑 推导的核心算法。将这个算法加进 命题逻辑 的推导算法（它叫消解法，resolution），就可以得到 谓词逻辑 的推导算法。换句话说：unification + resolution = first-order deduction.

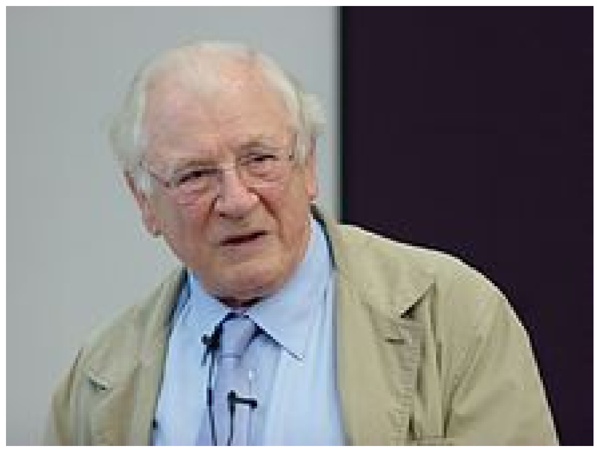
Unify 的意思是：将两个 **逻辑项**，透过 **variable substitution** 变成「一样」。Variable substitution 是 谓词逻辑 的本质，**量词** $\forall$ 和 $\exists$ 就是作用在这些变量上。所有初中生都懂得如何做「变量代入」，但它其实是一个很麻烦的动作，没有了它，谓词逻辑 就变成 命题逻辑。虽然所有数学家都知道什么是 variable substitution，但它的精确描述，到了 1920-30 年代才开始出现。例如 Schönfinkel 和 Curry 创造了 combinatory logic，Church 创造了 λ-calculus)，他们的目的之一就是揭示「代入」的机制。

Unification 的例子：

$$\text{loves}(X, Y) = \text{loves}(\text{john}, \text{mary}) \quad \text{with } \{X/\text{john}, Y/\text{mary}\} \tag{1}$$

（如果逻辑里面有 function symbols 可以更复杂）

Unifcation 算法最初由 Jacques Herbrand (1930) 提出，后经 J A Robinson (1965) 发明 resolution 算法，再将它们结合，应用到自动推理。



在经典逻辑 AI 中，unification 是核心算法之一，但它的时间复习性并不是瓶颈。笔者曾经研发过 higher-order unification，有开源代码在 GitHub.

以下是一个简单的 unification algorithm in Lisp，作者是 Peter Norvig：

```
;;; -*- Mode: Lisp; Syntax: Common-Lisp; -*-
;;; Code from Paradigms of Artificial Intelligence Programming
;;; Copyright (c) 1991 Peter Norvig

;;;; File unify.lisp: Unification functions

(requires "patmatch")

(defparameter *occurs-check* t "Should we do the occurs check?")

(defun unify (x y &optional (bindings no-bindings))
  "See if x and y match with given bindings."
  (cond ((eq bindings fail) fail)
        ((eql x y) bindings)
        ((variable-p x) (unify-variable x y bindings))
        ((variable-p y) (unify-variable y x bindings))
        ((and (consp x) (consp y))
```

```
        (unify (rest x) (rest y)
               (unify (first x) (first y) bindings)))
       (t fail)))

(defun unify-variable (var x bindings)
  "Unify var with x, using (and maybe extending) bindings."
  (cond ((get-binding var bindings)
         (unify (lookup var bindings) x bindings))
        ((and (variable-p x) (get-binding x bindings))
         (unify var (lookup x bindings) bindings))
        ((and *occurs-check* (occurs-check var x bindings))
         fail)
        (t (extend-bindings var x bindings))))

(defun occurs-check (var x bindings)
  "Does var occur anywhere inside x?"
  (cond ((eq var x) t)
        ((and (variable-p x) (get-binding x bindings))
         (occurs-check var (lookup x bindings) bindings))
        ((consp x) (or (occurs-check var (first x) bindings)
                       (occurs-check var (rest x) bindings)))
        (t nil)))

(defun subst-bindings (bindings x)
  "Substitute the value of variables in bindings into x,
  taking recursively bound variables into account."
  (cond ((eq bindings fail) fail)
        ((eq bindings no-bindings) x)
        ((and (variable-p x) (get-binding x bindings))
         (subst-bindings bindings (lookup x bindings)))
        ((atom x) x)
        (t (reuse-cons (subst-bindings bindings (car x))
                       (subst-bindings bindings (cdr x))
                       x))))

(defun unifier (x y)
 "Return something that unifies with both x and y (or fail)."
 (subst-bindings (unify x y) x))
```

从深度学习的角度考虑，问题是如何将 神经网络 算法 融合到 逻辑算法？表面上看，这是两件截然不同的东西。笔者思考这个问题很多年，也提出过一些方案，但并不特别成功。为了更明白 unification 的机制，我看了一些从范畴论角度处理 unification 的理论。

最早用范畴论角度研究 unification 的人是 Joseph Goguen (1941-2006)，他发现了 unification 对应於范畴论中的 **co-equalizer** 概念:

他写了一篇很详细的论文《What is unification? — a categorical view of substitution, equation and solution》(1989).

首先介绍一下什么是 **计算** 范畴论。范畴论很抽象（它关注在集合之间的**映射**，而不是集合里面的**元素**），但计算是具体的。关键是：用 **types** 代表 范畴论里的 objects，**functions** 代表范畴论里的 morphisms。换句话说，用 **函数式编程** 模拟范畴论，计算返回的结果是一些 functions。

> 众所周知，Haskell 的前身是 ML，ML = Lisp + type system. ML 也衍生了 OCaml = Objective Caml，而 Caml 的 CAM = categorical abstract machine. CAM 是 Cartesian-closed category 和 combinatory logic 的结合。（我暂时还不大清楚 CAM 的理论）

在 Rydeheard & Burstall 这本书里，用的语言是 ML。例如：

用 type `` `o `` 表示 objects，

用 type `` `a `` 表示 arrows，

那么 source 和 target 函数的类型就是

`` `a -> `o. ``

范畴论中，**co-equalizer** 的定义是：

$$\begin{tikzcd} X \arrow[r, shift left, "f"] \arrow[r, shift right, "g"'] & Y \arrow[r, "q"] \arrow[dr, "q'"'] & Q \arrow[d, dashed, "!"] \\ & & Q' \end{tikzcd} \tag{2}$$

换句话说，q 是 universal 的箭咀，令 $qf = qg$.

它的**对偶**，**equalizer** 的定义，“cone” 在左边：

$$\begin{tikzcd} Q \arrow[r, "q"] & X \arrow[r, shift left, "f"] \arrow[r, shift right, "g"'] & Y \\ Q' \arrow[u, dashed, "!"] \arrow[ur, "q'"] & & \end{tikzcd} \tag{3}$$

其中 $fq = gq$.

它们的意义不相同 (Awodey 2006) p.54-57：

An **equalizer** is a generalization of the idea of the **kernel** of a homomorphism, or an equationally defined “variety”, like the zero-set of a real-valued function. In other words, sets of elements x for which $f(x) = g(x)$ for $f, g : X \to Y$.

A **co-equalizer** is a generalization of a **quotient** by an equivalence relation.

在此只解释 co-equalizer：Define a relation on Y by $y_1 \rightsquigarrow y_2$ iff $\exists x \in X$ such that $f(x) = y_1$ and $g(x) = y_2$. Let $\simeq$ be the equivalence closure of $\rightsquigarrow$, and Q be the set of $\simeq$-equivalence classes. The quotient function $q : Y \to Q$ maps an element y to its equivalence class $[y]$ so that $qf = qg$.

在 R & B 书中 §3.4.2 讲述了 逻辑 **terms** 的范畴 $\mathcal{T}_\Omega(X)$，其中 X 是 the set of variables。A **term substitution** $f : X \to Y$ 是作用到这个范畴内的函数：

$$f : X \to \mathcal{T}_\Omega(Y) \tag{4}$$

Unification 作用在一些形如 $s = t$ 的 **equations** 上。

假设有一组 equations 用 index set I 指标：$\{s_i = t_i : i \in I\}$. 这组等式可以这样表示：

$$I \underset{g}{\overset{f}{\rightrightarrows}} X \tag{5}$$

where $f(i) = s_i, g(i) = t_i$. 换句话说 f 和 g 分别指向 等式的左右两端。

基於范畴论的算法的特点是：将函数 recursively **分拆** 成更细小的函数来计算。

Theorem 1. *If $q : X \to Q$ is the co-equalizer of the parallel pair:*

$$I \underset{g}{\overset{f}{\rightrightarrows}} X \xrightarrow{q} Q \tag{6}$$

and $r : Q \to R$ is the co-equalizer:

$$\begin{tikzcd} & X \arrow[dr, "q"] & & \\ I' \arrow[ur, "f'"] \arrow[dr, "g'"'] & & Q \arrow[r, "r"] & R \\ & X \arrow[ur, "q"'] & & \end{tikzcd} \tag{7}$$

then $rq : X \to R$ is the co-equalizer:

$$I + I' \underset{[g,g']}{\overset{[f,f']}{\rightrightarrows}} X \xrightarrow{rq} R \tag{8}$$

这分拆的意思是，例如：

$$\text{loves}(X, Y) = \text{loves}(\text{john}, \text{mary}) \tag{9}$$

则分拆成两条 equations：

$$\begin{aligned} X &= \text{john} \\ Y &= \text{mary} \end{aligned} \tag{10}$$

另外还有一个 theorem 将 term 分拆出 **sub-term**，在书里有解释，从略。

Conclusion

发觉原来范畴论对 unification 的表述，其实和最简单 naïve 的 algorithm（例如 Lisp 那个）基本上是完全一样的！不同的只是从更抽象的角度来看，如此而已。但这对於 将 unification 联系到其他数学结构上，或许会有启发。

知道了 unification 的结构之后或许会更容易将 神经网络 应用到逻辑上，虽然仍未有具体想法。其实，如果沿用 first-order logic 的 syntax，则整个系统完全和经典的 symbolic AI 没有分别，神经网络好像多此一举。所以，要真正能发挥到神经网络的功能，必需使用所谓「**分布式知识表述**，distributive representations」。这是我现时思考的方向。

References

Awodey (2006). *Category theory.*

Goguen (1989). “What is unification - a categorical view of substitution, equation and solution”. In: *Resolution of equations in algebraic structures* 1: algebraic techniques, pp. 217–261.

Robinson (1965). “A machine-oriented logic based on the resolution principle”. In: *Communications of the ACM* 5, pp. 23–41.

Rydeheard and Burstall (1988). *Computational category theory.*

Simmons (2011). *An introduction to category theory.*