数学的终极目标，是消除所有智力思考的需要。

— Alfred North Whitehead

Genifer 4.0 理论笔记

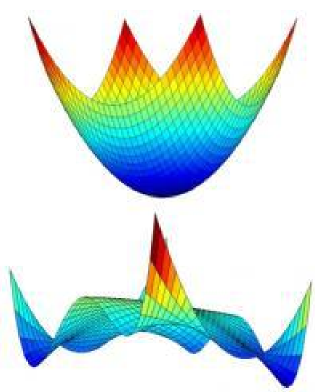
YKY (甄景贤)

27 May 2015

读者可能好奇我搞的 Genifer 4 是什么，其实没有很大不了，基本上是想将逻辑转移到**连续空间**，再应用连续逼近的技巧，例如梯度下降法 (gradient descent)。

最近一年时间，费了九牛二虎之力，才将逻辑变连续了（甚至这一步也未完善），但至少可以将 inductive learning 的问题变成数学上的 continuous optimization 问题。

然后才发觉，最快的 optimization 算法，是基於 convex （凸性）的技术。如下图的 fitness landscape，一个是凸的，另一个是不规则的：[1]



[1]from Charles H Martin's blog.

而如果没有了凸性，便要用到 genetic algorithms、branch-and-bound 之类的技巧。那些技巧我在 computer science 一早知道了，而且毋须用到连续性和梯度，令我失落地感到「多此一举」☹

后者虽然也可以很快，但它们属於 heuristics （窍门），也就是没有速度的保证。由於机器学习的复杂性极高，在没有保证下我们敢不敢尝试用那些 heuristics 呢？比方说，你愿不愿意投资几百万的赌注在 Genifer 3.0 上？

数学的确很神奇，它不断在进步，可惜进步得很缓慢。现时的一个前沿是将 convex 推广，但我暂时还未听过有什么突破（但我只是初学者，还未堪察过这範围的文献）。如果有的话，那将会是 ground-breaking，但这不是普通人能做到的，甚至大部分数学家也不能。

暂时可以做的大概是：

- 找一个 non-convex 但仍然可以快速解决的问题，然后将我们的问题转化成那问题。
- 改变原问题的 representation，这需要很有创意。甚或将 optimization 问题改变成 solve equation 的问题、等。
- 然后试试那 fitness landscape 会不会有好一点的性质；需要实验，把那些 error surface 描出来。
- 放弃 convex，仍然可以用其他 global optimization 的方法，如：遗传算法、branch-and-bound、convex-concave / DC programming、神经网络的 back-propagation、simulated annealing 等。但如上所述，有些技巧已经不需连续性，可以回到 Genifer 3。

馀下的篇幅，谈谈我关於逻辑连续化的尝试，日后可能还会有用（但也可能是 dead-end）。

1 逻辑的连续化

基本上我是想建立这个对应 (correspondence)：

KB (knowledge base)	$\Leftrightarrow$	连续空间
逻辑 rules	$\Leftrightarrow$	连续空间上的非线性算子
学习逻辑 rules	$\Leftrightarrow$	在泛函空间中寻找算子

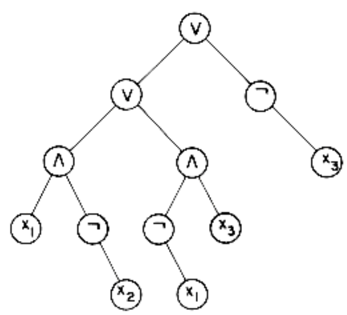
即使连续化之后还有 non-convex 的问题，暂且不提。单是连续化已经很难，因为传统逻辑的结构很复杂。

很多逻辑理论，用的是数学符号表述，但其实和数学的其他分支没有什么联系，只是一堆符号而已。如果要真正有用，必须从逻辑「搭桥」(bridge) 到其他数学分支上。我发觉最容易是通过 (抽象) 代数，因为有了代数表达式之后，可以较易看到和其他数学的关系。

2 逻辑是什么？

逻辑分为 **propositional logic** 和 **predicate logic**，前者 isomorphic to Boolean algebra 所以比较易懂，数学上也较简单；后者是很难用代数描述的，例如要用到 Tarski 所创的 cylindric algebra，在逻辑圈子以外很少人懂，而我正试图用另一个办法，其想法是基於组合逻辑 (combinatory logic) 和关系代数 (relation algebra)。[2]

首先很明显的是，逻辑 formulas 可以表达成树状结构，而树状结构也可以很容易表示为 sum-product 的代数结构，例如（甚至是二元树）：



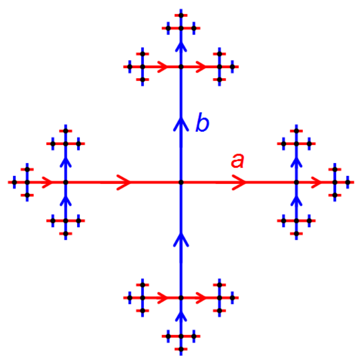
但在向量空间中表示会有问题，因为向量有加法，但乘法没有明显的候选人。例如，如果小明 · 爱 · 小娟 和 小强 · 爱 · 踢 · 足球 这两句不相关的句子在概念空间上的位置「相撞」或者不合理地接近，会是很不妥的。

[2]所谓组合逻辑的意思是：将谓词逻辑的 $R(a,b)$ 写作代数式的 Rab 或 aRb，例如 $loves(john, mary)$ 变成 $john \cdot loves \cdot mary$。关系代数是组合逻辑的一个特殊形式。

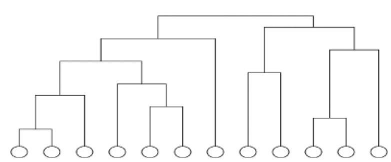
3 Cayley graph

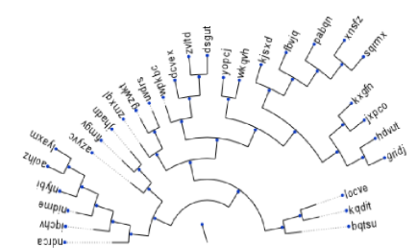
较早时我试过的一个做法是用 Cayley graph，但后来发觉不好用。

Cayley graph 的原理很简单，例如下图是 $\{a, b\}$ 两个元素产生的自由群 (free group)，乘积用线连起来：



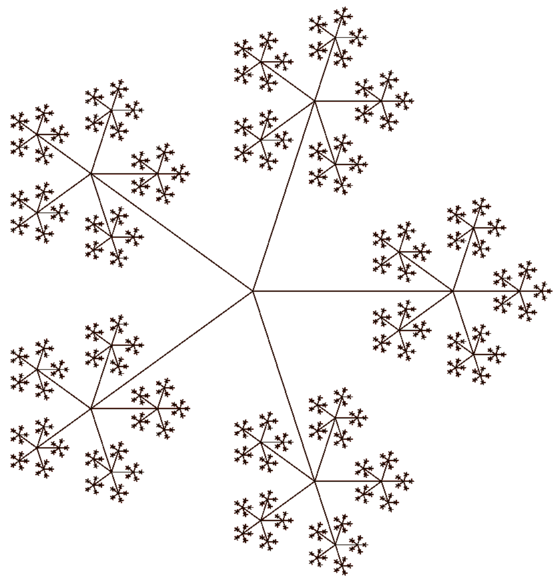
在人工智能中还有本体论 (ontology) 这个概念，例如 猫 $\subseteq$ 动物。本体可以表达成一个阶级分类 (hierarchical cluster)，这又可以变成圆形或球形：





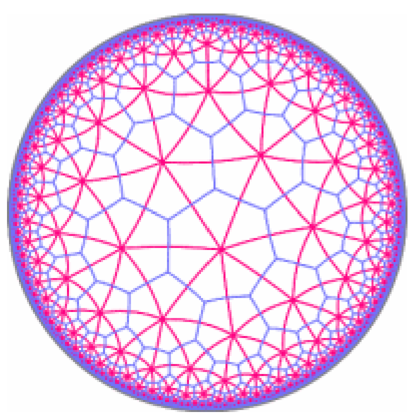
我企图将本体论和 Cayley graph 的想法结合起来：Cayley graph 中每个 node 是一个概念（例如 爱 或 足球），而每个概念也可以在球状的本体中找到位置。问题是本体球本身已经有 hierachical 结构，我想把球状本体「嵌入」到 Cayley graph 的節點中，似乎不易。

再者，当 Cayley graph 的分支数目增大时，看起来越来越「不自然」；例如这只是 $n=5$ 的情况：



原因是，Cayley graph 其实是一个 fractal structure！我原希望得到「连续」的效果，但在 Cayley graph 上从一个节点跳到另一个节点是**不连续**的。

虽则如此，Cayley graph 还有一个好处，就是它可以嵌入到 hyperbolic disc 上，这个圆盘有**双曲几何** (hyperbolic geometry) 的尺度（这在 M C Escher 设计的艺术中常见到，即圆盘越接近边缘的空间尺度越缩小）：



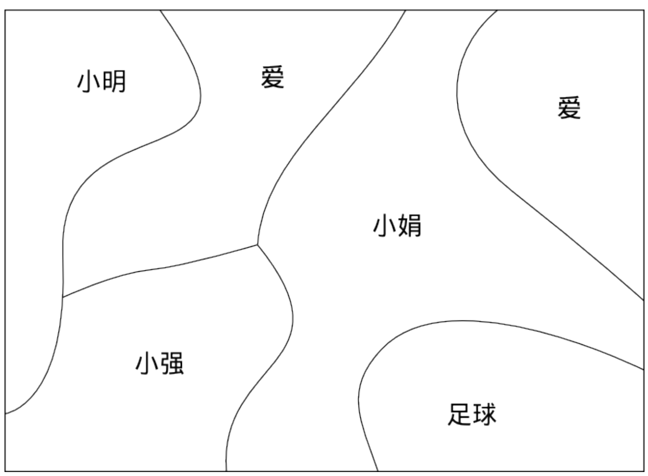
在双曲几何中直线变成圆弧，而神经网络将空间切割的方法也包含线性的 $Y = (\sum W_i X_i)$，或许可以制造一种 hyperbolic neuron？可惜因为不连续的问题，这方案暂时搁置。

4 Distributive vector representation (DVR)

这是一个较好的方案，似乎可以解决**连续性**的问题。
在 DVR 架构下，知识是用一个很长的 vector 表示：

$$K = \mathbf{v} = (v_1, v_2, ..., v_n)$$

这在神经网络中很常见。意思是：每个**概念**不是用单一的神经元 (即一个 v_i) 表达，而是用整个 layer 的神经元表达 (即 $\mathbf{v}$)。在这种表述下，一个概念是空间中的一个或多个 regions，它们可以是 disjoint 的。
例如，向量空间的维数 $n = 2$ 时，可以有这样的例子：



向量可以**相加**，相加的意思是叠加 (superimpose)，这概念也来自神经网络。例如 小明 · 爱 · 小娟 和 小强 · 爱 · 踢 · 足球 这两个 formulas 可以同时叠加在 K 上，表示这两句句子同时是真的。在概念之间互不「相撞」的情况下，这是可行的。
但仍有一个问题：当命题重复时，命题的真值不会变双倍（例如重复说「小明爱小娟」两次），换句话说，$a \wedge a = a$，但向量的加法会变双倍，除非我们选另一个**域**上的向量空间 (vector space over a field)，在这个域里 $1 + 1 = 1$ 或什么。这也是一个未解决的细节。
暂时也不理**乘法**的问题，第 7 节我们再讨论怎样表示乘积。

5 逻辑推导

逻辑的基本运算，是由一个 KB (**knowledge base**) 推导出新的逻辑句子，即 deduction。“KB” 这术语来自经典 AI，在新表述下简称作 K。

Deduction 的基本动作是 apply rules to facts。Deduction 可以用算子 T 表示：

$$T : K \mapsto K.$$

但 T 是**非线性**的。

T 对应於一个 logical rule。例如一个经典例子是「爷爷」的定义：

爷爷 (X,Z) ← 爸爸 (X,Y) ∧ 爸爸 (Y,Z)

这个 deduction 动作，首先要做 pattern matching ，然后才能 apply。

所以有这样的对应：

deduction	⇔	apply operators
pattern matching	⇔	filter: $K \to K$

举例说，我们已知这些事实：

爸爸 (小明, 小强)
爸爸 (小强, 大强)

而我们必须找到｛小明/X, 小强/Y, 大强/Z｝这个 substitution，才可以 apply 上面那 rule。找这个 substitution 的操作叫 pattern matching，经典的**归一化**算法 (unification algorithm) 可以做到。

这就是 T **非线性**的原因。因为对於不同的 K，T 的作用通常是 0，除非 pattern matching 成功，T 的作用才是非零。(当然，实际上我们会将 T 变 smooth，所以上面的「$= 0$」应该说是「接近 0」。)

在一个 AI 系统里有很多 rules，对应於 $T_1, T_2, ..., T_n$。分别将 T_i 作用在 KB 上，然后再叠加在一起，这就进行了推导的一步 (single step)。但由於 T_i 是非线性的，求和与 T_i 不可交换 (commute)。

单步的推导是：

$$K' = \sum T_i(K)$$

而 KB 的所有逻辑结论 (full logical consequence) 就是：

$$\begin{aligned} K^\infty &= \text{以上的单步重复无限次} \\ &= (\sum T_i)^\infty(K^0). \end{aligned}$$

6 学习

学习算法是人工智能的瓶颈，有了好的 inductive learner，其他细节只是细节（例如用 reinforcement learning 解决）。

逻辑的 inductive learning 有一个**双重结构**的特点，就是学习需要用到推导，但推导本身也是是一个复杂算法。

换句话说，先用推导得到 K^∞ （这本身需要算子的 iteration）：

$$K^\infty = (\sum T_i)^\infty (K^0).$$

然后将结果 K^∞ 和理想的 K^* 比较，得到误差，而我们目标是找出一套 T_i 令这误差最少。T_i 活在非线性算子的空间中。

逻辑上我们希望达到的是：

$$K^0 \cup \{T_i\} \models E$$

其中 E 是新的需要解释的例子或经验 (experience)。理想的答案是 $K^* = K^0 \cup E$。

误差 $\mathcal{E}$ 是推导出来的 K^∞ 减去理想的 K^*：

$$\mathcal{E} = K^\infty - K^* = (\sum T_i)^\infty K^0 - (K^0 + E).$$

如果 T_i 是可微的，梯度 $\partial\mathcal{E}/\partial T_i$ 存在，我们就可以用梯度下降法。

和一般常见的的优化算法比较，例如 Newton-Raphson 只是一个算子的 iteration，

$$x^\infty = T^\infty x^0$$

就是所需答案。

和传统神经网络的 back-propagation 比较，back-prop 的算法是：

（一粒神经元） $y_j = (\sum W_{ij} x_i)$

（多层结构） output $= y_0 \circ y_1 \circ \ldots \circ y_n$(input)

（update 法则） $\mathbf{W}' = \mathbf{W} + \alpha\, \partial\mathcal{E}/\partial\mathbf{W}$

两者比较：

- back-prop 的算子 $y_0 \circ y_1 \circ \ldots \circ y_n$ 是来自很多层的神经网络结构，所以 back-prop 可以看成是 error 在**空间**中的传播。

- 我们的 iteration 是在时间中发生的，但似乎没有本质上分别。

我现时认为问题是：那 K 的空间可能太大（即使维数不大），因为它包含所有逻辑式子，而且，如果没有任何 approximation 的话，这问题完全和逻辑学习的原问题一模一样，这新的做法不会有任何改进，那顶多是原问题在连续空间中的 relaxation。应该利用连续空间的 function approximation 来做点**一般化** (generalization)。

但逻辑中也有它的一般化结构。一般化的目的是**压缩**，也可以说学习就是压缩。逻辑学习本身就是一种复杂度很高的压缩法（所以它才那么慢），那 $A \subseteq B$ 的 subsumption 关系，造成一个阶层 (hierarchical) 分类结构，这分类法相当於把数据的复杂性取对数 (take the logarithm of the size of the knoweldge base)，就像二分搜索 (binary search) 那样。但因为逻辑学习有双重的 exponential 复杂度，可能单使用这个技巧还未够？

研究逻辑学习的 Stephen Muggleton 在 2002 年关於用**遗传算法**做逻辑学习说：

> 「在一阶逻辑学习系统里，验证假设的方法通常是呼叫一个逻辑证明器（例如 *Prolog* 解译器）去找出那假设对学习例子的正和负覆盖。已知道这是一阶概念学习中复杂而费时的一步。在基因学习里，这情况更差，因为每一代都有一群假设要验证。」[3]

7 乘法

暂时还不清楚怎样表达乘法，例如 小明 · 爱 · 小娟 这样的乘积。

其中一个方案由 Geoffrey Hinton 提出：每个概念是一个矩阵 (matrix)，如果概念 a 和 b 有关系 $a\,R\,b$，则代表它们的矩阵会服从 $AR = B$，而这些矩阵在矩阵空间中的位置是要学习得来的。但这方法似乎比较麻烦，因为要判断哪个元素跟著哪个元素，要靠矩阵乘法得出来。

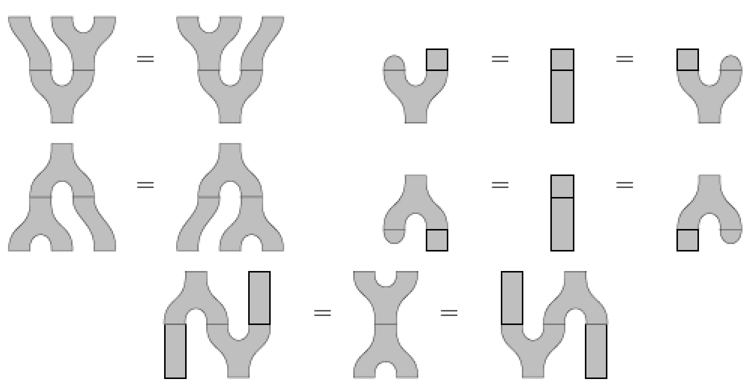
[3] *"The usual way for evaluating a hypothesis in first-order concept learning systems is to repeatedly call a theorem prover (eg Prolog interpreter) on training examples to find out positive and negative coverage of the hypothesis. This step is known to be a complex and time-consuming task in first-order concept learning. In the case of genetic-based systems this situation is even worse, because we need to evaluate a population of hypothesis in each generation. This problem is another important difficulty when applying GAs in first-order concept learning."*

第二个方案是使用张量积 (tensor product)，但张量积的 dimension 很大，而且随著乘积的长度增加。Paul Smolensky 的书《The harmonic mind》(2006) 第一卷有描述他发明的 distributive tensor product representation，我迟些有空再解释它。

第三个方案，出自《Quantum physics and linguistics》(2013) 这本论文集。他们使用的是幺半範畴 (monoidal categories)，那是一种又有乘积又可以组合 (compose) 的範畴。它的定义包括：

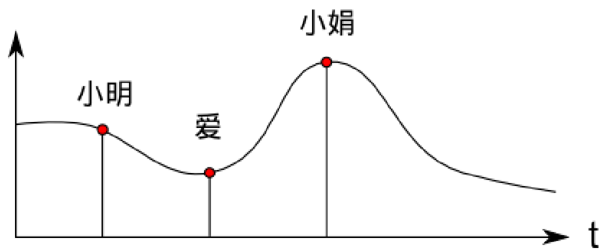
- objects: $X, Y, \ldots$
- morphisms: $f : X \to Y$
- composition: $f \circ g : X \to Z$
- grouping objects into: $X \otimes Y$
- a special object for the "empty system": I
- parallel composition: for $f_1 : X_1 \to Y_1$ and $f_2 : X_2 \to Y_2$
  $$f_1 \otimes f_2 : X_1 \otimes X_2 \to Y_1 \otimes Y_2$$

它的代数运算可以用一些扭 (braid) 图像地表示：



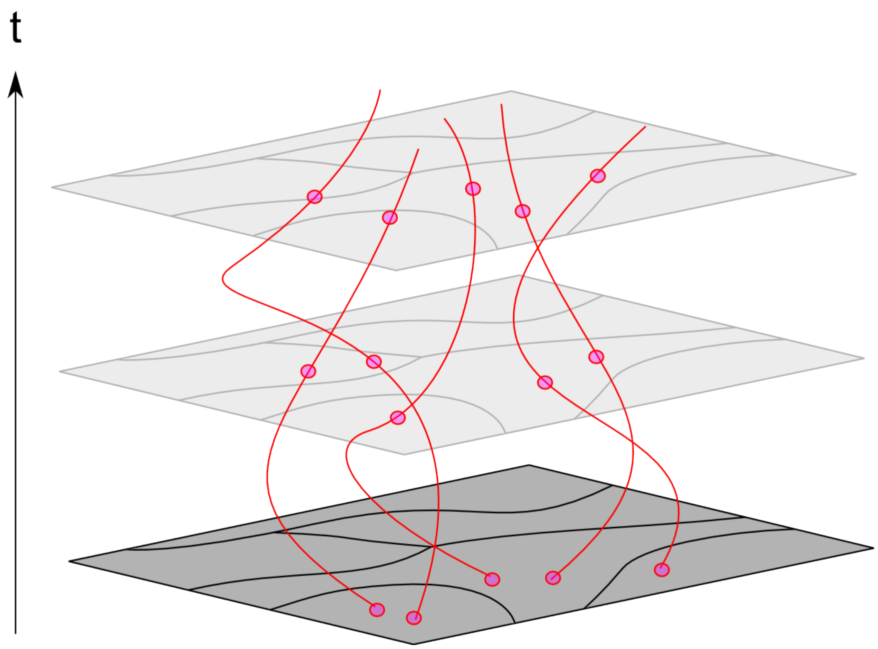
这似乎是是一个很 general 的结构（包含 tensor product？）这本书其中一个主旨就是说这个範畴能够涵盖所有自然语言的语法。但我暂时未有空研究它。

另一个方案是我创的，想法是：可以把下面这样的一个时间函数看成一个连续的时间**序列**：



而乘积也是一个 discrete 序列，可以把它连续化变成时间 t 的**连续曲线**。

这些曲线在 $K \times t$ 的空间中是像这样的：



这些曲线可以用 spline 表示，也就是是多项式 (polynomial)。这样或许可以用到 algebraic geometry （现代数学里很深奥的一个分支）里面的技巧。一个多项式可以用它的 coefficients 表示，那就是一串数，很方便。而多项式之间的变换也可以用多项式来描述，形成对偶空间 (dual space)，这类似於 logic 中，rule 可以作用在其他 rules 之上。

8 Formal power series

幂级数的形式是这样的：

$$\sum k_i a^i$$

但如果 a_i 不是普通的数，我们不能用普通加法求和，所以叫它做形式级数 (formal series)。

抽象地说，设 A 是一个**原子概念**的集合，例如 $\{$爱, 恨, 男人, 女人, ...$\}$，A^* 就是这些概念可以组成的所有句子。给这些句子乘上一些系数再加起来，例如：

0.8 小明 · 爱 · 小娟 +
0.3 小娟 · 爱 · 小明 +
0.9 小娟 · 爱 · 小强 +
...

系数 $k_i \in K$ 而 K 可以是任何半环 (semi-ring)，我们可以把 K 看成是**逻辑真假值**的半环。

所有如上的形式级数，记作 $K\langle\langle A\rangle\rangle$。

形式级数和有限自动机 (finite state machine, FSM) 有密切关系：FSM 可以用来辨认**形式语言** (formal languages)。一个形式语言 L 可以是任何 A^* 的子集。如果 L 包含某句句子，则给这句子系数 1，否则系数 0。於是我们得到一个代表那形式语言的形式级数。

并不是任何形式语言都可以被 FSM 辨认。

如果可辨认，则对於这语言内的每个字母 ("原子概念")，可以建立一个 $M_{n\times n}$ 矩阵，其 M_{pq} 元素视乎自动机由状态 p 到 q 有没有 transition（如果有是 1，没有是 0）。

这些矩阵的乘法满足概念的 monoid 乘法，例如: $a{\cdot}b{\cdot}c \times d{\cdot}e{\cdot}f = a{\cdot}b{\cdot}c{\cdot}d{\cdot}e{\cdot}f$。

在表示论 (representation theory) 里，我们说这些矩阵是这个 monoid 的 matrix representation。

根据表示论，一个环 K 乘上一个 monoid A 会得到一个 K-module，所以 $K\langle\langle A\rangle\rangle$ 可以看作是一个 (左边的) K-module。

Module 的存在代表 representation 的存在。（著名的犹太裔女数学家 Emmy Noether 开创用 module 研究 representations 的做法。）

而刚才也看到，FSM 可辨认 $\Rightarrow$ 可以建立 matrix representation。

所以，「FSM 可辨认」的另一个定义就是「存在 matrix representation」，这又可以再定义为「有某种 finitely generated K-submodule，它包含 $S \in K\langle\langle A\rangle\rangle$」（$S$ 是那个语言的形式级数）。

以上的内容部份摘自 Encyclopedia of mathematics《Noncommutative rational series with applications》(2011 剑桥大学出版)。

facts (propositions)	$\Leftrightarrow$	monoid words
KB	$\Leftrightarrow$	formal series
rules	$\Leftrightarrow$	non-linear operators acting on formal series

9 储存器

最后还不得不提，经典逻辑的结构实在太复杂了，我们的烦恼还未完...

在 Genifer 3 white paper 里，我解释过需要引入有「储存器记忆」的逻辑。那就是说需要加进一些逻辑「动作」，容许储存器的读和写。

memory register	$\Leftrightarrow$	vector space $K \times t$
actions	$\Leftrightarrow$	some meta-operations (?) over registers

实在太麻烦了，今次到此为止，下次再 update 吧！