关于 KRP 课程的内容梳理

赵一铮

*本文内容只代表作者的一些个人思考，并没有被纳入哪个官方文件或者哪本教材，如有不同观点，欢迎交流指正。我们希望把整门课程讲成一个逻辑严谨、通俗易懂的故事。

2022/4/11

本课程将围绕“定义”（definition）的两个不同类型展开讨论和思考：
（1）内涵式定义（intensional definition）：define a term by specifying the necessary and sufficient conditions of being it
（2）外延式定义（extensional definition）：define a term by listing everything that falls under that definition

比如 mother，使用内涵式定义：a female who has a child。female 是成为 mother 的一个必要条件，has a child 是成为 mother 的另一个必要条件，合二为一变成了 mother 的充分必要条件；而使用外延式定义会列出目前世界上所有 mother 的个体。这个个体的集合就是 mother 的外延式定义。

看待世界的方式：面向对象

学习这门课程不只是简单地对这个分类进行记忆，还要思考，思考更深入的事情（某个事物的底层哲学）。假如把当前的物理世界（physical world）看作是一个 universe，universe 内存在各种各样的 element。根据颗粒度（放大镜倍数）的不同，element 指代的事物也不同，它可以是原子、分子层面的事物，也可以是人、建筑层面的事物，还可以是比这些更为微观或者宏观的事物，只要“存在”即可。什么是“存在”？

existence: the ability to interact with the physical world

英文的解释是一个实体与物理世界的交互能力，有点哲学，不好理解。我自己给出这样一个定义：只要是能让这个物理世界或者其中 element 发生改变的任何事物都表明了该事物的“存在”，任何一点点小小的改变都可以。比如，我搬起石头，石头的位置变了，则我是存在的，石头也是存在的（位置改变了）；你的批评让另一个人委屈了，则这份批评是存在的，另一个人的委屈也是存在的。不仅仅是物理的改变，情绪、情感等都可以发生改变。

being 指的是任何存在的事物

举例：living being 生物；human being 人类；alien being 外星人；love being 爱；

becoming: 指的是 being 的变化

reality: 指的是一个充满 being 的世界

entity: 独立存在的 being

什么是“独立存在”？

比如，一块肉是独立存在的吧？这块肉就是 entity；但是这块肉在不切割的情况下，它的四分之三，就不是 entity，只是 being，因为它的四分之三并不独立存在。但如果用刀把这块肉切开，一份是四分之三，一份是四分之一，这两份肉又成了独立的 entities。

针对每一个 entity，我们首先应该为其取一个名字，这个名字是以“语言”来承载的，不管是什么类型的语言，总之是某种语言。这样这个 entity 就能有了除了它自身之外的第一种表示。这样做的好处是方便交流，当 A 提到 entity X 的时候，与 A 有同样认知的 B 明白 A 指代的是哪一个 entity。entity 只有名字就够用了么？那么 entity 之间不就完全没有任何联系，都是一个个 isolated 符号名称而已么？显然是不够的，接下来我们还得对每一个 entity 进行定义，这个 entity 是什么材质？做什么的？大小如何？形状如何？是人还是非生命类物品？一个 entity 展示给这个世界的所有信息都是形成这个 entity 的必要条件。同一个 entity，看待的角度不同，形成这个 entity 的必要条件也不同。比如 Lily 这个人，她是一个 mother，也是一个 teacher，成为 mother 的必要条件和成为 teacher 的必要条件显然是不一样。但所有的一切的必要条件形成了 Lily 这个独一无二的 entity。不同的 entity，它们可能会在某些特点一致，比如 Lily 和 Lucy 都是老师，而且都是母亲，这些 entities 会由于某些相似的特点，被归为同一种群体。但是我们上课讲过，群体往往是一个概念层面的事物，不是独立存在的。比如 Lily 是一位老师，她是老师这个群体的一个 instance，是一个 entity。但是老师这个群体是看不见摸不着的，没有哪个 entity 可以独立代表老师这个群体。所以我们把这种群体叫做 concept。Teacher 是一个 concept，Mother 也是。Concept 像是一种集合。

Entity 与 entity 之间可以交互、可以存在某种关联。作为集合概念，concept 与 concept 之间也可以，比如包含关系（Dog 和 Animal），互斥关系（Female 和 Male），相交但互不包含关系（Animal 和 Male），通过某些关系建立关联的关系（Human 和 Dog 之间可以通过 hasPet 联系起来）。思考一个问题：是区分人和海豚这两个群体容易？还是区分老师与律师这两个群体容易？还是区分南京大学学生与浙江大学学生这两个群体容易？还是区分 Lily 与 Lucy 这两个人容易？显然是难度越来越大。如果用一个包含特征的集合来区分两个群体，集合里面的特征共同构成了区分两个 concepts 的充分必要条件。人和海豚可以通过包含相对较少特征的集合区分，而 Lily 与 Lucy 就要用更多的特征，因为她们都属于 human，性别都为 female，有着相同的 occupation，区分她们需要更多更细节的特征，比如她们的面部特征、声音特征等等。换句话说，相似点越多的 entities 之间，concepts 之间需要更多更细节的特征区分。这也是为什么，对于机器学习来说，有效的数据越多，算法的结果越好。

*说到这里我们从内涵式定义和外延式定义角度区分下“知识表示与推理”和“机器学习”这两个 AI 子领域的底层哲学（个人一直觉得这个部分是课程中非常重要的讨论）：

知识表示与推理研究的重点是事物的内涵式定义，以及通过内涵式定义的“语义”（目前是

基于集合论和模型论）来进行智能计算行为。外延式定义集合中的每个元素可以看作是内涵式定义的一个 instance，比如 Lily 是 Mother 的内涵式定义的 instance。在物理世界里，这样的 instance 通过其它方式表示出来（data 或者一组 data）。这个世界上某个 concept 所有的 instances 合在一起反映了某个 concept 完整的内涵式定义，但遗憾的是这样的 instances 很可能是无限的。机器学习研究的重点是事物的外延式定义，以及通过外延式定义的“语义”来计算智能行为。通常是通过一组 training data 来尽可能学习到某个内涵式定义或者内涵式定义的片段（大部分情况下是片段，想学到完整的内涵式定义需要完整的数据集、完美的算法、强大的算力，理论上是不可能的）。既然外延式定义的 instances 可能是无限的，同样也受算法和算力所限，所以作为 training data 出现的就只是一部分，甚至是一小小部分的 instances，它天生无法完美地反映正确的完整的内涵式定义。所以机器学习有一个重要的课题是评估一个算法（比如分类算法）的好坏，它有多大能力去在 training data 上学习一个“内涵式定义”，再用这个定义在 test data 上展现其学习能力。

说到这里，我片面认为：只有内涵式定义才有机会称为知识。外延式定义，无论表示成什么形式，都只是知识的片面展现。最近几年，因为数据 + 知识，学习 + 推理的研究方向变得逐渐热门。一些工作尝试探索新的学习模型，并声称是将知识融入到了模型之中来提升 XX 性能。比如一种常见的方式是，引入一个知识图谱，或者其它方式表示的知识库，把知识图谱的内容映射到连续向量空间，比如 TransE，融入到模型之中。如果取得了好的实验结果则声称是知识带来的加成。这真的是知识带来的吗？首先，经典知识图谱是三元组组成的，类似于 ABox 之中的 role assertion（或者带有 nominal 的 TBox 的 concept inclusion），建立的是实体与实体之间的关系（或者 nominal 与 nominal 之间的关系）。这样的三元组不都是知识，比如“地球围绕太阳转”（$\{earth\} \sqsubseteq \exists orbits.\{sun\}$，这里我们用带有 nominal 的 TBox 来表示）是知识，可是“小明和小红是朋友”（$\{xiaoming\} \sqsubseteq \exists isFriendOf.\{xiaohong\}$）也是知识吗？俩人闹掰了怎么办？其次，知识转化为向量表示（相当于内涵式定义转化为外延式定义的 a set of instances），会失去原始的语义，转化过程涉及信息的损失和扭曲；最后，对于保留的信息，在模型训练过程中究竟用到了多少、怎么用的也不得而知。整个过程不就相当于“数据的扩充增广”吗？只不过新加入的数据的原始形态是知识图谱，给人一种加入了知识的假象。这样的工作只适合发 paper，糊弄不懂知识表示与推理的 reviewers，但对于知识 + 数据的 AI 范式的探索毫无推进作用，也没有底层哲学的启迪作用。因为人类使用知识的时候并不一定将其外延化，什么时候外延化的边界现在还不知道。

关于“知识”的定义和来源背后有一个宏大、旷日持久的争论。这是哲学里面关于认识论（epistemology）的研究范围。其中有很多不同的学派（school of thought）和代表人物。比如经验主义学派（empiricism），理性主义学派（rationalism），实用主义学派（pragmatism）、相对主义学派（relativism）等。这些学派之间的思想有的是互斥的，有的是相交但不重合，有的是包含（一种学派下面的子学派）。各个学派之间看待事物的角度不同，所以导致它们的思想并不一定都是互斥。我们简单提两个既互相对立、互相斗争，又互相影响、互相渗透的学派：经验主义学派和理性主义学派。在欧洲哲学史上，哲学家把这两种思想的冲突以及解决这两种冲突的不懈努力提到全部哲学的中心地位上来。

经验主义学派核心思想：知识仅来源于或主要来源于感官经验（knowledge comes only or primarily from sensory experience）
代表人物：弗兰西斯·培根（Francis Bacon）、托马斯·霍布斯（Thomas Hobbes）、乔治·贝克莱（George Berkeley）、约翰·洛克（John Locke）、大卫·休谟（David Hume）

还经常会听到联结主义（connectionism）这个词，它是实现经验主义的一种方法，核心思想是：智能行为通过人工神经网络实现（intellectual abilities via artificial neural networks）

理性主义学派核心思想：知识来源于理性和演绎，不依赖于感官经验（knowledge comes from reason and deduction）
代表人物：勒内·笛卡尔（René Descartes）、巴鲁赫·斯宾诺莎（Baruch Spinoza）、戈特弗里德·莱布尼茨（Gottfried Wilhelm Leibniz）

有意思是的，经验主义被广泛认为起源于欧洲大陆，但代表人物却都来自英伦；而理性主义被广泛认为起源于英伦，但却在欧洲大陆发扬光大。两个学派是认知内涵之争。一个外延的 instance 就是机器学习和知识表示与推理两个 AI 范式。这两种不同的哲学思想，体现在不同学科领域的研究方法的差异上，不仅仅是 AI 领域。以自然语言处理领域为例，带有经验主义色彩的工作包括早期的安德烈·马尔可夫（Andrey Markov）、克劳德·香农（Claude Shannon）；而理性主义的代表工作是诺姆·乔姆斯基（Noam Chomsky）的有限状态自动机对应语言模型。经验主义倾向使用概率和随机方法来研究语言，建立语言的概率模型，这种方法适合处理浅层次语言现象以及词语近距离依存关系。理性主义倾向使用符号表征方法来研究语言，适合处理深层次语言现象和词语长距离依存关系。现阶段，自然语言在经验主义方法的突破下取得了巨大的进步。不同的哲学思想甚至会外延到有神论和无神论之争。理性主义思想假设万物皆有“真理”，知识来源于真理和真理上的演绎，支持有神论，并相信神才是那个掌握真理的；而经验主义体现无神论，强调知识来源于人民群众的智慧。看待世界的不同方式（不同的哲学思想）衍生了不同的科学研究方法。英伦系科学家比如艾萨克·牛顿（Isaac Newton），他的科学思想带有明显的经验主义色彩。牛顿认为自然哲学只能从经验事实出发去解释世界事物，因而经验归纳法是最好的论证方法。他曾表达：“虽然用归纳法来从直言和观察中进行论证不能算是普遍的结论，但它是事物本性所许可的最好的论证方法，并随着归纳的愈为普遍，这种论证看起来也愈有力”。而理性主义学派代表人物莱布尼茨与经验主义学派代表人物牛顿之间关于谁先发明 calculus 的争论也成为了旷日持久的世纪争论（The Leibniz – Newton calculus controversy）。原始观点认为牛顿首先开启和完成了关于 calculus 的工作，但莱布尼茨发表得更早。现代观点认为二者各自独立开展了关于 calculus 的工作，是不是也意味着不同派别的思想其实有着更深层次的哲学来兼容所谓的理性主义学派和经验主义学派，从而衍生更先进，更接近人类本质的 AI 范式？其实我们使用两种不同哲学在 AI 领域的研究，以及在其它学科领域的研究，本质上都是在实践两种不同的哲学思想，我们都是哲学的外延式实践。

平时在学习和研究中发现，从事知识表示与推理研究的学者大多来自大陆系国家，比如德国、法国、意大利、奥地利、东欧国家；而学习领域的几位奠基人大多来自英美系国家，比如 Donald O. Hebb、Frank Rosenblatt、Geoffrey Hinton、Rich Sutton、Michael I. Jordan 等（从名字就可以看出，如果发现谁的国籍不是，也只是国籍不是，追溯一下其导师，或者导师的导师，发现最终会回归到英美系学者）。无聊的时候和身边人一起研究过各自的导师、导师的导师、导师的…的导师，发现最终都会追溯到莱布尼茨、高斯、拉格朗日。

知识表示与推理的内涵式定义（通常是一个符号表达式）的外延式语义通过 interpretation 来体现（所以通常这样的 interpretation 有无数个），但 interpretation 也是概念层面的（比如 interpretation 定义的 domain 是不是想象的？里面的 element 是不是想象的？），不是现实中真实的数据。机器学习接触的是真实的 data。机器学习做的是一种“表”；知识表示与推理

做的是一种“里”，也是一种“理”。机器学习更像是一种实验科学，一种实用的工程；知识表示与推理更像是一种理论科学，一种头脑的思维。机器学习算法的输入是数据，输出是一个与人类智能得到的结果“相似”的结果（有时候更糟、有时候更好、有时候差不多），中间的计算过程，很多算法与人类处理相同问题使用的方法并不一致。传统的机器算法基于统计和概率模型，而联结主义的神经网络目前不可解释。知识表示与推理的算法输入是一组逻辑表达式（现阶段主流用逻辑语言表示，理论上也可以不用逻辑语言，但得建立一种新的可计算的语义解释模型），输出是一个只可能比人类智能得到的结果“更好”或者至少一样的结果（因为是“理”的结果，是客观上正确的结果），中间的计算过程透明。

机器学习的缺点在于外延数据的天然不完备、数据质量难以保证、部分计算过程不透明、结果缺乏可解释性。知识表示与推理的缺点在于从来不与真实的数据打交道，不与真实世界交互，所有的研究进展都体现在对“理”、对“内涵”的探索上，使得该领域的研究与真实世界难以沟通，难以产生有实际应用价值的贡献；其次，知识的获取无法做到自动化（如果承认内涵式定义才算知识），这使得一个知识库的构建变得异常艰难（知识对不对？全不全？有没有冗余？）。还有，对于一些不确定性知识，不是非黑即白的知识难以很好地表示；最后，计算透明性的极致要求导致中间的推理过程计算复杂度极高，会受到算力的限制（所以要学习知识表示语言的可判定性和计算复杂度）。

在机器学习的科研中，当用算法 1 解决了一个问题 A，它可以平移到问题 B 去尝试效果，也可以有算法 2 来尝试解决问题 A。算法与问题之间是一个多对多的函数。面对同一个问题，人们根据现有的 SOTA 算法来开发更好的算法，可能是你有了更好的数据，可能是有了更 powerful、smart 的算法，可能是其它。所以一个问题一个算法可以有很多的 followers 去跟进研究，论文引用率往往高。知识表示与推理的科研中，问题与推理方法往往是一对一的函数关系，个别的是一对多，但这个“多”往往<3。机器学习的问题是外延式，同一类问题的 applications 往往很多，针对不同的 applications 都可以提出新的算法，哪怕只是准确率小小的提升，每一个新的算法都是一篇 paper。知识表示与推理问题是内涵式的，同一类问题只有一个内涵式问题本身，很少有外延式的 applications。问题解决就是解决了，解决的方式往往只有有限几种甚至是唯一方法。这导致客观上知识表示与推理的“理”的问题不如机器学习“表”的问题多，但其实本质上，它们研究的都是同一个“理”，底层哲学并没有区别。

从解决问题的类型上看，机器学习特别适合“学习”“不太需要可解释性”的任务，因为学习天生就是从物理世界的片段中总结“理”再去运用“理”的过程，计算重结果不重过程。知识表示与推理更适合“理性思考”“需要可解释性”的任务，比如医疗、法律等领域的部分问题，因为知识都是内涵式定义，过程的透明性才是第一位的。知识表示与推理最值得骄傲的是“theoretical soundness”，而机器学习展现了“empirical success”。

越来越多的 AI 学者开始意识到，仅仅追随某一种哲学，无论是机器学习还是知识表示与推理，都难以解决现实生活中的问题，或者难以模拟人类真实的智能行为。因为在面对现实问题的时候，人类的智能往往是“感知”与“理性”相结合，甚至二者不是串行发挥作用，而是有时串行、有时并行发挥作用。如何将二者有效地结合，科学地结合，需要我们对要解决的问题进行归类（哪些问题适合哪一种哲学，哪些不适合），需要我们对这两种哲学的边界有着更深层次的思考（哪一种哲学可以解决哪些问题），或许需要我们对内涵式定义（知识）和外延式定义（数据）找到统一的知识表示方式，才有可能完成知识与数据的

交互，才有可能找到学习+推理结合的契机和着陆点，才能继续推进 AI 这个领域的前进和发展。在这个过程中，我们也会更加懂得人类自己。

很多高校其实也开设了关于“知识表示与处理”的课程，但最终都是以“向量”等表示方式嵌入（one-hot encoding、bags of words、word embedding、knowledge graph），这样的知识表示方式适合深度学习模型，但计算过程依然不透明。少部分高校也有一些涉及逻辑语言作为知识表示语言的课程，但属于入门级的讲授，简单介绍一下，不涉及知识表示与推理的核心思想、理论、技术及证明。我们希望在这门课上，讲清楚这个领域的故事和它所承载的哲学思想。

话题：什么是人工智能？学习它为什么重要？

人工智能从狭义上讲，指的是让机器模拟人的智能行为（intelligent behavior）。就这样一句简单的解释，需要思考的问题却不少。第一个问题是，为什么要让“机器”，或者说得更狭义一点，要让“计算机”去模拟？为什么不是让猫、狗、或者昆虫去模拟？那一定是计算机本身有一些特别的性质，促使我们有兴趣去研究，这个特别的性质就是计算机的特性、优点、尤其是比起人类的优点。这个问题可以换一种问法：计算机比起人的优势在哪里？这个同学们应该去搜索很多资料，说法很多，但是普遍承认的有这样几个优势：
（1）计算机处理信息的速度要比人类快；
（2）计算机“记忆力”好，它们的大脑里可以装入很多信息而不遗忘，而人类的存储量就相对较少，忘性极大；
（3）计算机“体力”好，长时间工作也不需休息，是全天候工作的（round-the-clock）。不像人类，工作一段时间就需要进食，进食完就想睡觉，电能对比生物能的先天优势；
（4）计算机“可信度”高，他们计算的结果不犯错误，除非程序有错误（但程序是人写的，计算机只负责执行），人会受到情绪、状态、感受、感情等影响，犯错几率比计算机大。比如，面对你的亲人朋友，你受情感影响很难保持客观。
（5）计算机“无人性”，适合去做一些需要智能但是物理上危险的工作。即便除了安全问题也不会引起人类的情感变化。

计算机比人类出色的地方，一个词概括，就是“计算能力”。两个词概括，加上一个“非人性”。所以，学习 AI 的原因之一，可能也是最重要的原因，是希望通过计算机强大的计算能力模拟智能行为，从而解放生产力和提高生产效率，深层目的是 advance humanity。

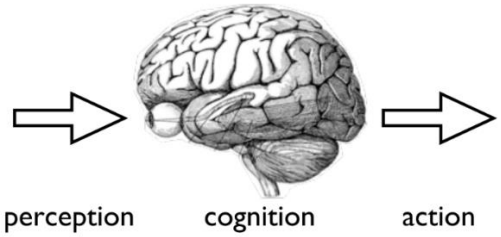
接下来一个问题：什么是人的智能行为？更广义地说，什么是智能行为？先尝试给出它的内涵式定义：

Artificial intelligence： using artifacts (often machines) to simulate intelligent behavior

然后是它的外延式定义：

Artificial intelligence： using artifacts (often machines) to simulate the abilities to see, hear, smell, feel, think, reason, calculate, communicate, read, understand, imagine…love

外延式定义的智能行为可以大致分为以下三类：



（1）perception（感知）：the abilities to see, hear, smell, feel, or become aware of something through the senses

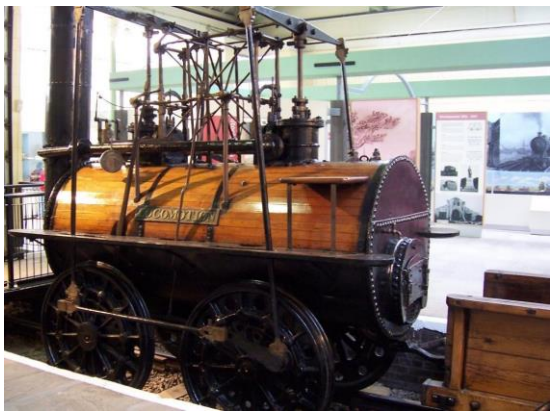
（2）cognition（认知）：the abilities to think, reason, calculate, remember, imagine, or other mental processes, and understand through thought, experience and the senses

（3）action（行为）：the abilities to move, act, speak…

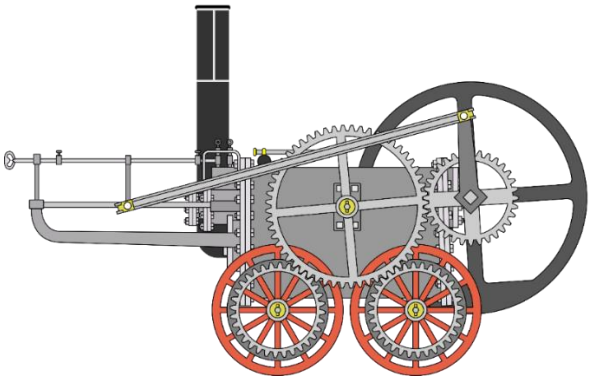
上面这个定义说智能行为是多种能力的组合，什么能力呢？从物理世界中感知，渐渐形成、补充、修正对世界的现有认知，再利用现有认知对周边环境做出行为反应的能力。人工智能旨在让机器完成上述智能行为或者其中一部分。其中一个重要的前提是这些智能行为是“可计算的”（computable）。只有可计算的“模型”，计算机才能够处理，人类处理信息的方式并不一定都是可以抽象为一个可计算的模型。

模型：对于世界上一个 entity 的抽象就是模型（model）。

比如现在要讲蒸汽机车的工作原理，有必要拿来真的蒸汽机车吗？



或许是不需要的…只需要展示给大家它的模型即可：



*注意区分“计算模型”和“可计算模型”：前者指各类计算机的模型，现存的计算模型包括：finite state machine，pushdown automata， Turing machine，Lambda calculus 等；后者指的是“被计算事物”的模型（数据模型），比如接下来要介绍的本体。

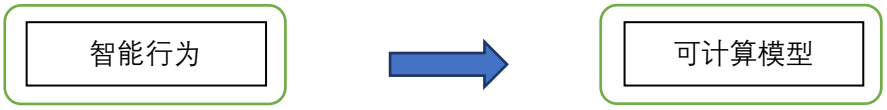
什么是“可计算的”？该怎么定义和理解“可计算的”？先理解“计算”：

compute: to execute a program (program = a set of instructions)

计算指的是执行程序的过程，程序解释为一组指令。连在一起就是按照一组指令，一步一步执行的过程就叫计算。一个计算的例子：要把大象关冰箱总共分几步？

（1）把冰箱门打开
（2）把大象装进去
（3）大冰箱们关上

按照这个步骤去执行的过程就叫做计算。可以完成计算这个行为的任何事物都叫做“计算机”。比如上述大象关冰箱步骤，人类可以完成吗？当然可以，如果大象不反对的话，所以人类本身就是一台计算机。这个就是广义的计算机的解释。随着时代的发展，计算机从机械计算机到现在的电子计算机、生物计算机。人工智能研究的是怎么把不同的智能行为变为可计算的模型，这个过程叫做“建模”：



想要一个东西可计算，一个通用方式就是转化为数学模型，这也是为什么我们要在大一大二学习那么多数学基础课程，比如高等代数、数学分析、概率论与数理统计、最优化方法、信息论、数理逻辑等。数理逻辑是知识表示与推理的数学基础。

一个数学建模的例子：m 元钱，投资 n 个项目。效益函数 $f_i(x)$，表示第 i 个项目产生 x 元的效益，i=1,2,⋯,n。求如何分配每个项目的钱数使得总效益最大？

实例：5 万元，投资给 4 个项目，效益函数 $f_i(x)$如下表：

x	$f_1(x)$	$f_2(x)$	$f_3(x)$	$f_4(x)$
0	0	0	0	0
1	11	0	2	2
2	12	5	10	21
3	13	10	30	22
4	14	15	32	23
5	15	20	40	24

建模过程：
输入：n, m, $f_i(x)$, 其中 i=1,2,⋯,n, x=1,2,⋯,m
解：n 维向量$<x_1, x_2,\cdots, x_n>$，xi 是投到第 i 个项目的钱数，使得满足下面条件：
目标函数：

$$max \sum_{i=1}^{n} f_i(x_i)$$

约束条件：

$$\sum_{i=1}^{n} x_i = m, x_i \in \mathbb{N}$$

根据这样的模型，我们可以设计不同的算法来完成，比如这里最直接的算法就是*蛮力算法* — 就是计算出所有的可行方案，然后找出受益最大的那个。显然这个算法的复杂度会很高，是对于 m 和 n 是指数增长的。

上述的建模过程相对简单，但是对于人工智能的建模不会那么直观。看下面一张照片：

从这张照片我们接收到的信息是一条狗，是一条幼年金毛，它坐在开满小黄花的草坪上，张着嘴卖萌。这个过程涉及到我们用眼睛去看（感知），再把图片中的物体与先验知识结合起来（认知）。我们知道这个样子的是狗，是金毛，是幼年金毛等等信息。现在把这张图片输入给计算机，让它也像人类一样通过“感知”“认知”处理这张照片，得到与我们一样的信息。这个“感知”“认知”的过程是通过“计算”的手段实现的。

这门课的内容主要涉及如何将“知识”表示为可计算的模型（一组逻辑表达式），并在模型上进行计算（推理）获得新的知识。这个计算基础是数理逻辑（数理逻辑包括集合论、模型论、证明论、递归论），具体说来，是基于集合论和模型论的知识表示方法和语义解释方法。基于这样的理论，我们可以清楚的看到计算机是如何对知识进行表示，如何对这种表示的知识进行理解，是如何基于这种表示和理解进行推理的。

人类思考的过程中，需不需要触碰现实中真实存在的 entity，还是只针对那个物体在我们大脑中的抽象进行思考？显然是后者。我们思考的对象不是现实中的 entity，而是这个 entity 在我们大脑中的一个概念映射（conceptual mapping）。这个概念映射是用语言来承载的。理想的知识表示语言是：

（1）有足够的表达力表示（物理世界中的）知识
（2）人类可以理解它的语义
（3）机器可以理解它的语义，且该语言可计算（是一种数学语言）

为此我们提出使用 formal language，它有三个特点：

（1）有确定的字母表（意味着语言中单词的组成只能使用字母表里面的元素）
（2）有确定的语法规则（意味着必须按照这个语法规则组成复杂短语或句子）
（3）有确定的语义解释（意味着必须按照这个方法去解释句子中的每个元素）

自然语言属不属于 formal language？虽然自然语言满足前两点，但是对于第三点自然语言是不满足的。同一词汇，不同的人看解释可以是不一样。比如“黄瓜”，大多数人的理解是 cucumber，但是有些地区 cucumber 被称为“青瓜”，而黄瓜指的是另一种蔬菜，见下图：

再比如，“饭”这个词，一些人理解为 meal，也会有人理解为 rice。“跳脚”这个词既可以指一种动作，也可以引申为“气急败坏”。英文也是如此，一词多义比比皆是。

典型的 formal language 有 programming language 和 logical language（二者合起来可以是 logic programming language，logic programming 也是一个很有意思的研究领域）。逻辑语言中，我们把一个逻辑表达式称为 a theorem 或 axiom，把 a set of axioms 称为 a logical theory。逻辑语言可以满足知识表示语言的三个条件，未来可能有某些方面更适合作知识表示的语言；同时也不能否认，formal language 比起自然语言，也有其局限性。比如自然语言的模糊性，有的时候事物并不是非黑即白的存在，而是有一定的“fuzziness”。“六点半左右”“黄色偏灰”

等等，这种 fuzziness 可以被 formal language 捕捉吗？这种 fuzziness 可以被计算吗？如果可以，这个数学模型该如何建立？

接下来开始本体（ontology）的讲解。首先，如何定义本体？因为我们学习了描述逻辑，是否可以直接定义本体为 description logic-based knowledge base；或者说，因为我们介绍了逻辑语言，是否可以直接定义本体为 logic-based knowledge base？答案是不行的。

哲学领域的本体研究的是 entities 以及 entities 是如何分类的（the classification of entities）。计算机领域本体该如何定义？Siri 之父 Tom Studer 给出的定义是：

ontology: a formal, explicit specification of a shared conceptualization

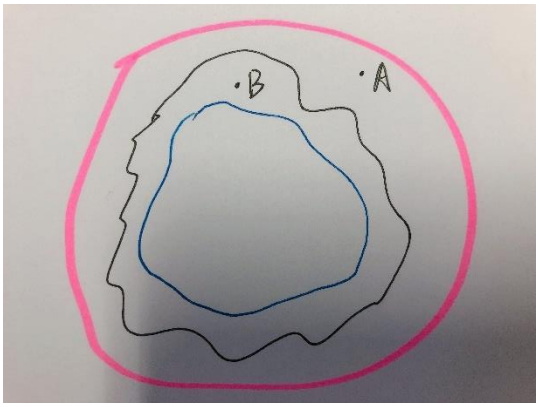
这个定义也是目前最为广泛接受的一个定义。分析一下这个定义：

conceptualization 是什么？指的是一个 physical world 的抽象。比如有这样一个 physical world，里面有实体小金毛，实体小绵羊，实体小老虎，实体小河马。那么这样一个 physical world 的抽象就是一个 conceptualization。Conceptualization 取决于个人（dependent on individuals）。因为对于同一个 physical world，不同的人形成的 conceptualization 可能不一样。有的人认为实体 a 属于概念 A，有的人认为属于概念 B。该听谁的？不可能对每一个 conceptualization 都做一个 ontology 吧？一定是最客观、最公认那个，所以定义中有 shared 这个词。

specification 是什么？词典解释为 a detailed description。detailed 好理解，description 是什么意思？表示对于一个人、事物、或事件的（口头或笔录）的描述。按照这个解释，原来的 ontology 定义就可以改写为：

ontology: a formal, explicit, detailed description of a shared conceptualization

explicit 是什么？词典解释为“described clearly and in detail, leaving no room for confusion or doubt”。就是描述得清楚、细致、不给留任何模糊的空间。看下面这张图：



假如此时我们想为图中的黑色圈出部分的概念下定义，最好就是描述符合成为这部分中元素的充分必要条件。假如我们用粉色勾出的部分来定义黑色部分，那么里面就混入了一些

本来不属于黑色部分的元素，比如 A。但反过来，如果我们用蓝色勾出的部分来定义黑色部分，那么有一些实际是黑色部分的元素被排除在外了，比如说 B。所以无论把一个定义做的太 strong（蓝色部分），还是太 weak（粉色部分），都不是一个好定义。都会留出模糊的部分，留出漏洞。所以一个 explicit definition 应该是如下的解释：

explicit definition: a definition which formally sets out the meaning of a concept or expression, as by specifying necessary and sufficient conditions for being it

明确地描述成为某个概念中的元素需要满足什么条件。比如要成为“人”必须是哺乳动物，必须有头，必须有腿，可是满足这些条件的动物有很多，不仅仅是人。那么这个定义就不够 explicit，不够 precise。

formal 是什么？定义中只有这个词没有解释了，但其实这个词我们之前解释过了，formal 可以理解为严格按照字母表，语法规则和语义解释规则来组成句子。

这样一个关于本体的定义就完整地呈现在面前了。

在这样一个定义下，要求本体是一个“可计算模型”，一些基于集合论、模型论的逻辑语言就进入了视野。我们将使用描述逻辑语言描述本体知识。这里需要注意：本体只是一个知识表示模型，是一个领域知识库，它用什么语言描述知识并没有任何规定。不是只能用描述逻辑，用一阶谓词逻辑、模态逻辑、时序逻辑、命题逻辑取决于具体的任务类型，它合适什么样的语言，甚至用自然语言都可以，只要能描述知识，都可以叫做本体。这门课，我们选择用描述逻辑描述本体知识，以描述逻辑为载体介绍各种推理任务，展示各种应用，覆盖知识表示与推理这个领域研究的核心内容，一方面是因为描述逻辑是目前最热门、最前沿的知识表示语言，另一方面它就是一个用于教学的“工具人”，换了别的语言，跟随的推理类型也不会改变，只不过语言的表达力和不同推理任务的复杂度会发生改变。

描述逻辑（description logic，后面简写为 DL）是一阶谓词逻辑（first-order predicate logic，后面简写为 FOPL）的子集，并且是它的可判定子集（decidable fragments），因为 FOPL 是不可判定的（undecidable）。现阶段通常使用网络本体语言（Web Ontology Language，以下简称 OWL）开发本体，那么 OWL 与 DL 之间是什么关系呢？OWL 是面向开发端的，因为本体开发出来，需要有一个可用的、可部署的文件格式，比如.txt、.pdf、.xml 等。本体的文件格式为.owl，大家可以用浏览器打开下面这个本体文件：

https://protege.stanford.edu/ontologies/pizza/pizza.owl

里面包含很多如下片段：

```
<owl:ObjectProperty rdf:about="http://www.co-ode.org/ontologies/pizza/pizza.owl#hasBase">
<rdfs:subPropertyOf rdf:resource="http://www.co-ode.org/ontologies/pizza/pizza.owl#hasIngredient"/>
<owl:inverseOf rdf:resource="http://www.co-ode.org/ontologies/pizza/pizza.owl#isBaseOf"/>
<rdf:type rdf:resource="http://www.w3.org/2002/07/owl#FunctionalProperty"/>
<rdf:type rdf:resource="http://www.w3.org/2002/07/owl#InverseFunctionalProperty"/>
<rdfs:domain rdf:resource="http://www.co-ode.org/ontologies/pizza/pizza.owl#Pizza"/>
<rdfs:range rdf:resource="http://www.co-ode.org/ontologies/pizza/pizza.owl#PizzaBase"/>
</owl:ObjectProperty>
```

可以看到，里面有三个关键字：owl、rdfs、rdf。它们之间的关系：rdfs 是 rdf 的扩展，owl

是 rdfs 的扩展。这些具体是什么，可以先不关注。现阶段，需要记住.owl 文件可以被浏览器处理，内容可以被浏览器展示。DL 是一种逻辑语言，我们学习它是要学习它的字符表、语法、语义、表达力、以及在上面的推理任务。所以：DL 是 OWL 的逻辑内核，OWL 是 DL 的外部封装。他们使用不同的术语来表达同样的事物：

DL	OWL	First Order Logic
individual	individual	constant
concept	class	unary predicate
role	object property data property	binary predicate

object property 的 range 是 individual 或者 class，比如{Tom} ⊑ ∃hasChild.Lawyer；data property 的 range 是 data types（a number or string），比如{Tom} ⊑ ∃hasAge.55。这里的 55 不是一个符号，而是阿拉伯数字，interpretation 不会对它进行解释。这门课，我们不涉及 data property，因为推理不会涉及到“自然语言”，尽管在工程上，它会起到表示作用。

接下来我们学习几种 DL 语言。无论是哪一种 DL，都会将知识分为两个层面。第一个层面是类的层面，包括类的性质以及类与类之间的相互关系。第二个层面是实体的层面，包括实体的属性和实体与实体之间的关系。

类的性质（也就是成为类中元素的必要条件），比如：
Father 的性质是∃hasGender.Male，是∃hasChild.Top
Medalist 的性质是∃hasWon.Medal
Car 的性质是∃hasComponent.Wheel

类与类之间的相互关系（上面类的性质，也可以看做是类与复杂类之间的关系），比如：
Student 是 Human 的子类
Human 是 Mammal 的子类

类层面的知识的集合叫做 Terminological Box，简称 TBox。TBox 里面都是类与类之间的包含关系，所以我们也把 TBox 叫做 concept hierarchy（relations between concepts）。只不过这里，包含关系不仅可以在 atomic concepts 之间，还可以在 complex concepts 之间。

讲完了类的性质和类之间的关系，接下来将目光放在实体上：

实体的属性（实体属于哪个类），比如：
Jack 是学生 Student(Jack)
Lily 是一个有孩子的人 ∃hasChild.X(Lily)

实体与实体之间的关系
Lily 是 Jack 的妈妈 isMotherOf(Lily, Jack)
Jack 养了一只小狗 dodo hasPet(Jack, dodo) Puppy(dodo)

实体的属性既可以是一个原子类（atomic concept），也可以是通过逻辑连接符组成的一个复

杂类（complex concept）。实体与实体的关系一定是由一个二元关系连接的两个实体。实体层面的知识的集合叫做 Assertional TBox，简称 ABox。

我们想一下 TBox 和 ABox 的名字是不是很贴切。我们看一下 terminology 的词义：

Terms: words and compound words that in specific contexts are given specific meanings

上面定义中的 words 对应 DL 的 concept names，compound words 对应 complex concepts，也叫 compound concepts。

Assert：这个词有很多自然语言语义，其中一个意思是将某个 object 归类

这里需要统一部分术语。首先一个 TBox 和 ABox 共同组成一个 Knowledge Base，简称 KB。一个本体就应该等同于一个 KB。但是在很多文献中，会发现有两种定义：一种是将本体定义为 TBox，不包括 ABox；还有一种是将本体等同于 KB。我们的规矩是在口语、写作中使用任何一种方式都可以，只需要定义清楚即可。

除了 TBox 和 ABox，本体有时还会包含一个 Role Box（简称 RBox）。Role Box 顾名思义，是关于 role 的性质。比如说，role 之间也可以有 inclusion 关系：hasFather 是 hasAncester 的子关系。意味着，a 如果是 b 的父亲，a 就一定是 b 的祖先。role 之间的包含关系称为 role inclusion，用字母 H 表示。EL 语言中如果加入 role inclusion，则称为 ELH；同样地，ALC 称为 ALCH。除此之外，RBox 还可以加入 role equivalence，表示两个 role 等价（同样地，a role equivalence 可以表示成两个 role inclusions）。一些 role 拥有其它的性质，比如 role 的传递性（transitivity），对称性（symmetry），自反性（reflexivity），函数性（functional）等等。这里，传递性是我们在这门课上介绍的一个代表性性质，其字母表示规则很特殊：ALC + transitivity = S，ALCHI + transitivity = SHI。一个 role r 是 transitive 的，其语义为：对于 domain 任意元素 x, y, z 来说，$\forall x \forall y \forall z((r(x, y) \wedge r(y, z)) \rightarrow r(x, z))$。比如，hasAncester 就是一个 transitive role；但 hasFriend 和 hasParent 不是。

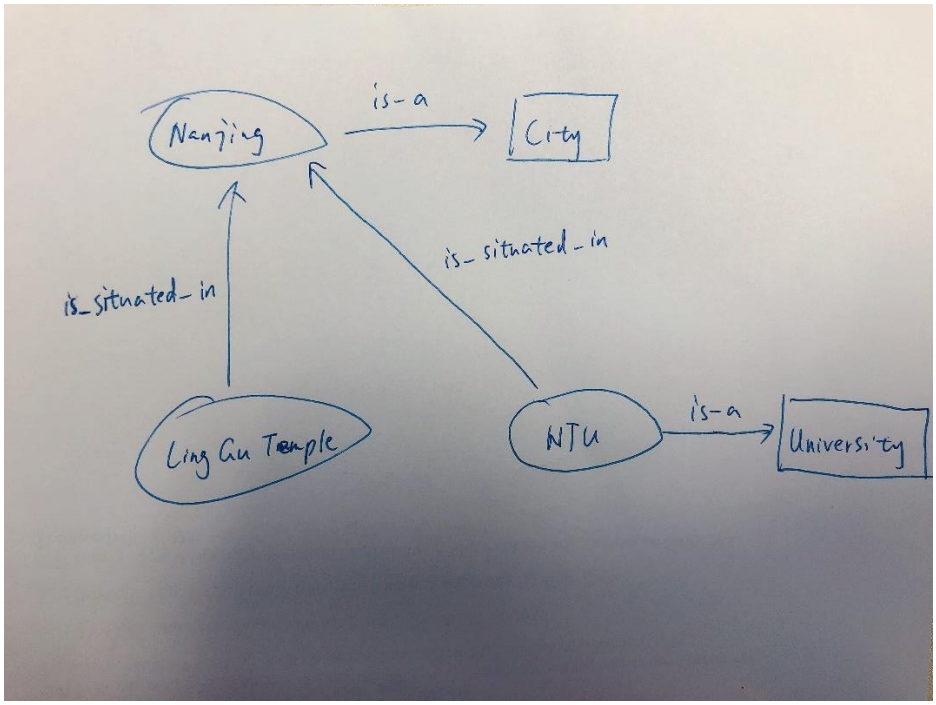
如果有 RBox，则 KB = TBox + ABox + RBox
如果没有，则 KB = TBox + ABox

TBox 表达式称为 TBox axioms 或者 TBox inclusions（因为 TBox 里面都是 GCI），ABox 表达式称为 ABox assertions，而 RBox 表达式称为 RBox axioms。有的文献不会刻意区分 axioms 与 assertions，所以也会称 ABox assertions 为 ABox axioms。文献中也会看到 TBox statements，ABox statements 这样的表达，不要困惑。这门课把 KB 中所有表达式统称为 axioms。

一个 DL 的知识类型确定（TBox、ABox、RBox 确定），它的表达力取决于 concept 和 role 的表达力。更确切地说，取决于哪些逻辑运算符可以被使用来构建 complex concept 和 complex role。比如 EL 就只允许使用 top concept、and、exists 来构建 complex concept；而 ALC 进一步允许使用 bottom concept、negation、or、forall 来构建 complex concept；ALCI 进一步允许使用 inverse 来构建 complex role，等等。

为了说明 DL 作为当前知识表示语言的先进性，我们对比它与其它一些知识表示语言。详

细介绍一个：Semantic Network。



上图为一个 semantic network。semantic network 是一个有向图（directed graph），由 nodes 和 edges 组成。其中，nodes 既可以是一个实体，比如上图中的 Nanjing、Ling Gu Temple、NJU；也可以是一个类，比如 City、University。Nodes 对于实体和类不做区分（上图中我们将实体的 node 用椭圆表示，类的 node 用矩形表示，是为了方便读者区分）。edges 用箭头标识，表示两个 nodes 之间的关系。每个 edge 上会有一个 label，代表关系的名称。

看下 semantic network 的缺点。首先，最大的缺点是没有 formal semantics。所有的 nodes 名称和 edges 名称都保持自然语言语义，没有像本体类似的基于集合论的语义。University 不会被解释为一个集合，NJU 不会被解释为集合中的一个元素，is-a 不会被解释为 domain 上的双元关系。这样一来，“计算”的基础就没有了。计算机是无法在“原始的自然语义”上进行计算的，这是它先天性的缺陷。其次，nodes 对实体和类是不做区分的，这种粗糙的处理方式对于知识表示来说不够理想，埋下了隐患。最后，像“非 University”“City 或 University”这种“复杂类”是无法表示的，说明表达力不够。其实，后两点缺点并不存在，它们只是第一个缺点衍生出来的产品。没有语义解释，意味着 semantic network 就仅仅是一个数据模型，和传统的关系数据库没有任何区别，都是有固定的语法，没有可支撑计算的语义，作用就是存储数据用的物理框架。

上面讲了 DL 的词汇表和语法规则（怎样组成复杂的类），接下来讲 DL 的语义，也就是如何解释 DL 表达式。解释方法是基于集合论（set theory）和模型论（model theory）。两个理论都属于数理逻辑的范畴。具体做法是基于面向对象的世界观，用一个叫做 interpretation

的东西来解释 DL 表达式。Interpretation 包含两个部分：第一个是确定 domain，告诉这个故事发生在哪个世界/宇宙（world/universe）里面，以及里面会有哪些 elements（entities）；第二个是将 DL 表达式中的符号对应到这个 domain 上去：将 individual name 对应到 domain 中的一个 element；将 concept name 对应到 domain 上的一个子集，这个子集代表一个类，类中元素有共同的性质（所以才将它们分为一类）；将 role name 对应到 domain 上的一对实体之间的二元关系（可能不止一对，而是多对）。比如 hasFather，满足父子/女关系的实体对儿可能有多个。逻辑连接符的解释与 interpretation 没关系，它有逻辑上的预定义解释。

能够解释同一个或者同一组 DL 表达式（a DL axiom or a set of DL axioms）的 interpretation 不只有一个，这个合理吗？答案是合理的。比如 domain 是南京大学，里面的 element 是南京大学世界里的各种人，包括学生和老师。那么存在一个关于老师的属性是：Teacher $\sqsubseteq$ ∃teaches.Student。这个 axiom 只在南京大学这个 domain 才成立吗？不是，在其它大学的 domains 里依然成立。所以越是通用的知识满足它的 interpretations 越多，当然也有一些知识只在少数的 interpretations 解释下才成立，甚至在任何 interpretations 解释下都不成立。注：（无限集合之间也可以比较大小，比如整数集和正整数集）。

规定了语义的解释方法后，接下来可以根据这个方法建立起一种“可计算模型”。通过这个模型可以进一步定义几种推理运算，这些推理运算会在本体被用作各种 applications 的 KB 的时候发挥作用。比如 query answering：问某个 KB，is Lily a Teacher？换成逻辑表达式：

$$KB \models Teacher(Lily)$$

常见的推理运算包括：

（1）验证两个 concepts 之间是否存在 inclusion 关系（equivalence 关系验证可以转化成两个 inclusions 关系验证，下面涉及 equivalence 的地方也是一样）

（2）验证一个 individual 是否属于某个 concept

（3）验证两个 individual 之间是否存在某二元关系

（4）计算哪些 concepts 之间存在 inclusion 关系

（5）计算每一个 individual 都属于哪些 concept

（6）计算哪些对儿 individuals 存在二元关系

还能想到其它的推理类型吗？

（1）（2）（3）和（4）（5）（6）的推理类型不一样，前者问的是“是否”（yes or no）的问题，称为 Boolean questions，后者 Non-Boolean questions。区别在于，Boolean questions 是事先假设一个情况，然后去验证这个情况是否成立，除了问题本身从头至尾没有任何“未知因素”。但是 Non-Boolean questions 是利用已有知识去计算未知的知识。直觉上讲，难度要比 Boolean questions 大。能不能将（4）（5）（6）问题转化为 Boolean questions？比如（4）问题，是否可以使用枚举方法，先找出逻辑表达式中所有的 concepts，再去验证每个组合是否满足 inclusion 关系？答案是不可以。因为 concepts 不仅是 concept name，还可以是 complex concepts，而 complex concepts 的数量可以是无限的：用 concept name A 和 role name r 可以造出∃r.A、∃r.∃r.A、∃r.∃r.∃r.A…等无数个 concepts。这样的方式是行不通的。Non-Boolean questions 不是都能转化为 Boolean questions 进行处理的。刚才列举了一些推理类型，是否已经将课上讲的推理类型完全覆盖？并不是，还有两种推理类型：

（7）验证一个 ontology 是否 consistent（部分文献也叫 satisfiable）

（8）验证一个 concept 是否 satisfiable

目前为止我们只关心推理类型的（1）（2）（3）（7）（8）这五个 Boolean questions，并且会在接下来展示：（1）（2）（3）（7）都可以转化为（8）问题。部分文献也会写（1）（2）（3）（8）都可以转化为（7）问题；也对，因为（7）（8）可以互相转化，看从哪个角度看待这个问题。

除了（7），其它问题又需要考虑两种情况：是否有一个背景知识？比如我说：食堂的饭菜是很难吃的。这句话正确吗？不一定正确，因为有的食堂的饭菜好吃有的不好吃。但是如果我说：XX 大学食堂的饭菜是很难吃的。一下子就变得很肯定了。这说明什么？说明增加了背景知识之后，原来不正确的可能就正确了。

我们按照（8）（7）（1）（2）（3）的顺序进行逐步讲解：

问题（8）：一个 concept C 是 satisfiable，则一定存在一个 interpretation I 使得 C^I 非空。一个 concept 是否必须是 satisfiable？从直觉角度，并不一定。物理世界里有没有一类群体里面是没有任何元素的？答案是有。比如“吃肉的素食主义者”、“没有生物学父母的哺乳动物”、“好吃不贵的南大食堂”等等。一个 ontology 里面，某些 concept 是 unsatisfiable 的，完全没有问题。看下面的例子：

$$\text{EukaryoticCell} \sqsubseteq \text{Cell}$$
$$\text{EukaryoticCell} \sqsubseteq \exists \text{hasPart.Nucleus}$$

$$\text{RedBloodCell} \sqsubseteq \text{EukaryoticCell}$$
$$\text{RedBloodCell} \sqsubseteq \forall \text{hasPart.}\neg\text{Nucleus}$$

$$\text{Blood} \sqsubseteq \exists \text{hasPart.RedBloodCell}$$

这个例子中：真核细胞一定有细胞核，血红细胞是真核细胞但是其成分中没有细胞核，很明显的 contradiction。所以血红细胞这个类里面不应该存在任何元素（任何 Interpretation I 都不可能让 RedBloodCell 非空）。如果 RedBloodCell 任何时候都为空，则 Blood 也如此（无论与常识是否符合，目前逻辑表达式展示的信息就是如此。事实上，血红细胞是真核细胞，在其成熟期的时候没有细胞核，但是成熟期之前有细胞核，它是特殊的真核细胞）。

那么我们现在如何判断一个 concept 是 satisfiable 的呢？分为两种情况：

（a）一种是没有背景知识（without TBox）
（b）一种是有背景知识（with TBox）

Concept satisfiability 的问题只和 TBox 有关，和 ABox 没关系。无论添加任何 ABox axioms 到背景知识里面，都不影响一个 concept 的 satisfiability。这句话非常重要，想一想为什么？

对于上述（a）的情况，一个 concept C 是 unsatisfiable 的，则 C is born to be unsatisfiable，它天然就是永远不可能非空的，非常类似于命题逻辑中的永假式。比如 Person ⊓ ¬Person 和 ∃hasChild.Male ⊓ ∀hasChild.¬Male。这时候想证明 C 是 satisfiable 的，最简单的方法是构建一个 interpretation I 使得 C^I 非空。但是对于 unsatisfiable 的情况，就无法这么做了，因为不可能找到世界上所有的 interpretations，并验证每一个是否都让 C 为空（这时候可以用反证法

假设存在这样一个 interpretation I 使得 C 是 satisfiable 的，并试图寻找 contradiction）。于是就有了 Tableau 算法：该算法尝试构建一个 interpretation I 使得 C^I 非空，并且保证如果 C 是 satisfiable 的，就一定能够通过它构建这样一个 interpretation（这是算法的完备性保证的）。如果 C 是 unsatisfiable 的，Tableau 算法会在中间的某个步骤达到 contradiction（也叫 clash）。

contradiction：也就是⊥。在这里，我们把⊥外延解释为：一个元素属于 A 同时属于¬A，这里 A 指的是 atomic concept。

这个算法从何处开始？假设这个 concept C 是 satisfiable 的，则必定存在一个 domain element x 使得：$x \in C^I$。再根据这个假设，结合 C 的语义去构建 interpretation I 或者寻找 clash。

概念 C 有哪几种类型？换句话说，C 的根节点可以是哪几种？取决于 DL 是什么？如果是 ALC，则 C 可以是一个 atomic concept（concept name），还可以是 negation，and，or，exists，forall。是不是针对该语言中的每一个逻辑运算符，都设计一个 interpretation 的构造规则？

为了避免∃hasChild.Male ⊓ ¬∃hasChild.Male 这种客观上是 clash 的情况无法找到上面定义的 clash（因为∃hasChild.Male 并不是一个 atomic concept）我们引入 negation normal form（NNF）的概念。可能有人会问：为什么不在定义 clash 的时候直接将外延解释为：一个元素属于 C 同时属于¬C，其中 C 为任意 concept。这样 clash 的概念不更有普适性么？看这样一个例子：∃r.¬∀s.A ⊓ ¬∀r.∀s.A 确实是 clash，但是却找不到 C 和¬C 这种明显互斥的表示。这是因为原式之中¬符号出现的位置不够“对称”。NNF 要求所有的¬都只能出现在 atomic concept 之前。这样一来，¬后面只有 concept name，则是否存在 clash 就非常直观了。不满足这个语法的式子在使用 Tableau 算法之前，都需要使用如下规则转为 NNF：

$$
\begin{array}{rcll}
\neg\top & \equiv & \bot & \\
\neg\bot & \equiv & \top & \\
\neg\neg C & \equiv & C & \\
\neg(C \sqcap D) & \equiv & \neg C \sqcup \neg D & \text{(De Morgan's law)} \\
\neg(C \sqcup D) & \equiv & \neg C \sqcap \neg D & \text{(De Morgan's law)} \\
\neg\forall r.C & \equiv & \exists r.\neg C & \\
\neg\exists r.C & \equiv & \forall r.\neg C &
\end{array}
$$

图中的规则保证我们可以将 ALC 的任意 concept 转为 NNF，分别处理¬出现在 top，bottom，negation，and，or，exists，forall 这六个非 atomic concept 的情况。上述规则可能会递归使用。每当出现一个算法，或者一组规则，“原则上”我们需要证明这个算法或者这组规则的正确性、完备性、可终止性。对于一些非常简单、直观的算法和规则，我们通常成为该算法或者规则为“standard”，意味着不需要证明也公认成立。上述这组规则就是这样一个例子。

转为 NNF 之后，我们就可以在 Tableau 算法中不再针对¬设计 interpretation I 的构造规则。另外，top 和 bottom 已经没有任何叶节点，所以就只剩下其它几种类型的根节点。如下图

所示，到达哪个根节点，就使用对应的 rule 构造 interpretation I。并且 rule 的可使用性（applicability）除了根据根节点判断以外，还要检测是否满足 if 条件，满足才能使用。

$$S \rightarrow_{\sqcap} S \cup \{ x : C,\ x : D \}$$

if (a) $x : C \sqcap D$ is in S
(b) $x : C$ and $x : D$ are not both in S

$$S \rightarrow_{\sqcup} S \cup \{ x : E \}$$

if (a) $x : C \sqcup D$ is in S
(b) neither $x : C$ nor $x : D$ is in S
(c) $E = C$ or $E = D$ **(branching!)**

$$S \rightarrow_{\forall} S \cup \{ y : C \}$$

if (a) $x : \forall r.C$ is in S
(b) $(x, y) : r$ is in S
(c) $y : C$ is not in S

olicable if role successors can be found

$$S \rightarrow_{\exists} S \cup \{ (x, y) : r,\ y : C \}$$

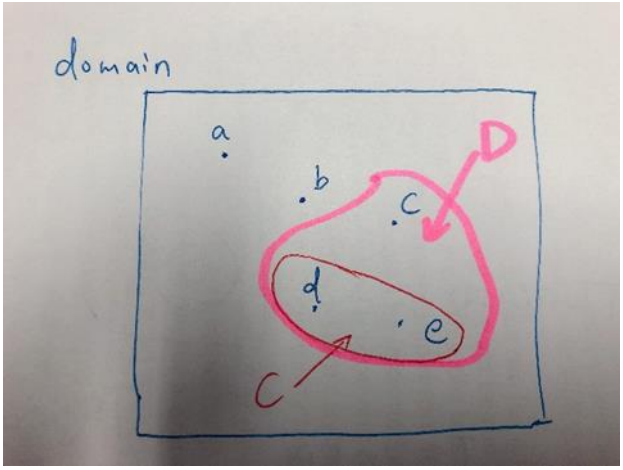
if (a) $x : \exists r.C$ is in S
(b) y is a fresh individual
(c) there is no z such that
both $(x, z) : r$ and $z : C$ are in S

Tableau 算法的本质是 decompose 或者叫 break down 整个 concept 的语义。一个不能被满足的 concept 一定会在 decompose 的过程中遇到语义上的矛盾（这是 Tableau 算法的完备性保证的，需要证明）。按照 Tableau 算法一直 decompose 到叶节点，也就是 atomic concept 和 negated atomic concept 层面，算法才会终止。如果最后出现了某个元素同时在 A 和¬A 的情况，则到达 clash。这说明原始的 concept C 不可能存在一个 interpretation I 使得 $x \in C^I$。也就是说 C 是 unsatisfiable 的。反过来，如果直到已经无 rule 可用了，依然没找到 clash，则 C 是 satisfiable 的。这里面唯一特殊的规则是 or 这个 rule。它的每个分支都要探索到位才可以

保证结果的正确性：如果在某一个 branch 上得到了已经无 rule 可用，且并没有找到 clash 的效果，则可以直接宣判 C 是 satisfiable；但是，如果一个 branch 中得到了 clash，不能着急下结论说 C 是 unsatisfiable，因为还需要探索其它的 branches 的情况。当全部 branches 都检查完了且都找到了 clash，C 才是 unsatisfiable。

对于上述（b）情况，也就是有 TBox 的情况，我们不能在全部可能的 interpretations 里面寻找让 C 不为空的那个，来证明 C 的 satisfiability。相反，我们需要在 TBox 的 models 里面寻找让 C 是 satisfiable 的。也就是说：判断一个 concept C 是否 satisfiable w.r.t. a TBox T，我们需要去看 T∪{x: C}是否有一个 model。满足 x: C 上面已经看过了怎么做。那么怎么体现是 T 的 model?

对于 T 中每一个 axiom（GCI）$C \sqsubseteq D$，如果某个 interpretation I 是 $C \sqsubseteq D$ 的 model：$I \vDash C \sqsubseteq D$，则 $I \vDash \top \sqsubseteq \neg C \sqcup D$。意思是：C 是 D 的子集这件事如果在 I 上为真，则¬C 与 D 的并集一定是整个 domain 这件事在 I 也一定为真。看下图：



因为任何元素都一定在 domain 里，则 $x: (\neg C \sqcup D)^I$。如果 TBox 之中包含多个 GCI，则 x 都会出现在每个用同样方式表示的 domain 之中：

Check whether C is satisfiable w.r.t. a TBox = $\{A \sqsubseteq B, D \sqsubseteq E\}$。则存在一个 Interpretation I 使得 domain 中存在一个元素 x:

$x \in C^I$,

$x \in (\neg A \sqcup B)^I$,

$x \in (\neg D \sqcup E)^I$

这时候，我们需要在 Tableau 算法中添加一个新的规则：

$$S \rightarrow_U S \cup \{x: D\}$$

if (a) $\top \sqsubseteq D$ is in S

(b) x occurs in S

(c) $x: D$ is not in S

当发现 $C \sqsubseteq D$ 出现在 T 中的时候，等价地转化为 $\top \sqsubseteq \neg C \sqcup D$。如果发现 $x: \neg C \sqcup D$ 没有在现有的状态系统 S 里，则将此事实添加进去。所以，多了背景知识（一个 TBox，也就是一组 GCIs）其实是帮助我们添加一些 interpretation 构建过程中的限制条件。比如 $C \sqsubseteq D$ 意味着所有出现在 S 状态系统里面的 individual 都要出现在 $\neg C \sqcup D$ 之中，这就是一个限制。

至此，我们已经学会如何解决 concept satisfiability 的问题，无论有无 TBox。接下来看如何将其它推理问题转化为 concept satisfiability 的问题。首先看问题（7）。

问题（7）：check 一个 ontology 是否是 consistent 的

对于 ontology O 的 consistency 定义为：存在一个 Interpretation I 使得 I 可以成为 O 的 model。也就是存在一个 interpretation I 使得 I 是 O 中每个 axiom 的 model。至于 I 满足一个 axiom 是从集合论和模型论角度定义：I 满足 $C \sqsubseteq D$，则存在一个 interpretation I 满足 $C^I \subseteq D^I$。比如，如果 I 对于 C 的解释为{1, 2}，对于 D 的解释为{1, 2, 3}；如果 I 对一个 individual name a 的解释为 1，则 I 满足 C(a)和 D(a)这两个 axioms。

为什么要 check ontology consistency?

如果 O 是 inconsistent 的，意味着没有任何 interpretation 可以使得 O“自洽”。一个不自洽的本体是无法从中得到任何正确信息的。看一个例子，$O \vDash$ alpha（alpha 是任意一个 axiom）。如果此时 O 是不自洽的，意味着没有任何 interpretation 可以满足 O，成为 O 的 model。而 $O \vDash$ alpha 成立的要求是所有 O 的 model 都是 alpha 的 model，写作 for any interpretation I, if I is a model of O, then it is a model of alpha。因为 O 没有 model，所有这个式子是 vacuously true（理解为虽然为 true，但 true 得毫无意义）。我们看这样一个自然语言的例子，$\vDash$符号左边是：萨摩耶拥有律师证；$\vDash$符号右边是：它是律师。让左边这句话成立的 model 一定是右边这句话的 model，这没问题。但是萨摩耶是不可能拥有律师证的，这让后续的任务推理结论都毫无意义。

一个 O 是 inconsistent 的例子：A(a)和¬A(a)同时出现在 O 中。另一个例子，RedBloodCell 的那个例子，我们已经得到结论 RedBloodCell 和 Blood 必然是 unsatisfiable 的。这种情况下，如果向 O 中添加一个 ABox axiom：RedBloodCell(a)或者 Blood(b)。则 RedBloodCell 和 Blood 中必然有元素，与上面事实相悖，则整个 ontology 是 inconsistent 的。这里注意，当 O 是空集时，它是自洽的。空集意味着任何 interpretation 都是它的 model。

了解了 check ontology consistency 的重要性，如何验证一个 ontology O 是否是 consistent 的呢？思路很简单，就是想办法为 O 构造一个 model，找到了它，则 O 就是 consistent 的。问题（7）可以以如下方式转化为问题（8）：

例子：Check $O = \{C \sqsubseteq D\}$是否是 consistent。如果是，则存在一个 interpretation I 满足 $C^I \subseteq D^I$，则$(\neg C \sqcup D)^I$一定是 I 的 domain，则 I 定义的 domain 中一定存在一个 element x 使得 domain 非空：$x \in (\neg C \sqcup D)^I$。所以 check：$O = \{C \sqsubseteq D\}$是否是 consistent 的问题就成功转化为 check $\neg C \sqcup D$ 是否是 satisfiable 的问题。使用 Tableau 算法将原始的表达式转化为 $\top \sqsubseteq \neg C \sqcup D$，并使用 U 规则进行 S 状态系统的升级。

一个更复杂的例子（O 中包含多个 axioms）：Check O = {C ⊑ D, D ⊑ E}是否是 consistent。如果是，则存在一个 interpretation I ⊨ C ⊑ D，D ⊑ E，则¬C ⊔ D 和¬D ⊔ E 都一定是 I 的 domain，则 domain 中一定存在一个元素 x 使得 $x \in (\neg C \sqcup D)^I$ 且 $x \in (\neg D \sqcup E)^I$。所以 check O = {C ⊑ D, D ⊑ E}是否是 consistent 的问题就成功转化为 check ¬C ⊔ D 和¬D ⊔ E 是否是 satisfiable 的问题。使用 Tableau 算法将原始的表达式转化为⊤ ⊑ ¬C ⊔ D 和⊤ ⊑ ¬D ⊔ E，并使用 U 规则进行 S 状态系统的升级。

如果 O 中包含 ABox axioms 呢：Check O = {C ⊑ D, D(a)}是否是 consistent？此时不需要再创造一个虚拟的元素 x 来进行判断，因为已经有了一个现成的元素：a。此时只需要将 a: D 添加到 S 状态系统，并使用 U 规则将 a: ¬C ⊔ D 添加到 S 状态系统里面即可继续 Tableau 算法接下来的操作。直到全部叶节点为 clash，则 O 是 inconsistent；或者某一个叶节点在饱和状态下（无 rule 可用）依然没出现 clash，则 O 是 consistent。

接下来看如何将问题（1）转化为（8）：

问题（1）：验证两个 concepts 之间是否存在 inclusion 关系。分为两种情况，一种是没有背景知识（without TBox），另一种是有背景知识（with TBox）。

没有 TBox 的情况下，如果两个 concepts C 和 D 之间存在 inclusion 关系，说明 C ⊑ D 这件事永真。说明任意一个 interpretation I，都可以满足 C ⊑ D。显然，我们不可能找出所有的 interpretation 来验证是否都成立。一个可行的方法，或者叫可计算的方法，假设存在一个 interpretation I 使得 C ⊑ D 不成立：I \not\models C ⊑ D。说明⊤ ⊑ ¬C ⊔ D 这件事在 I 中不成立，说明¬C ⊔ D 不再是 I 的 domain，说明 domain 中一定存在一个元素 x，x 在¬C ⊔ D 的补集之中，也就是在¬(¬C ⊔ D)之中，也就是在 C ⊓¬D 之中。所以问题变成了 C ⊓¬D 的可满足性问题。将 x: C ⊓¬D 加入到 S 状态系统，使用 Tableau 算法进行验证。如果 C ⊓¬D 可满足，说明确实存在这样一个 interpretation I 使得 C ⊑ D 不成立，则说明 C 和 D 之间不存在 inclusion 关系；反过来，如果 C ⊓¬D 不可满足，说明不存在这样一个 interpretation I 使得 C ⊑ D 不成立，说明 C 和 D 之间存在 inclusion 关系。

有 TBox 的情况下，如果两个 concepts C 和 D 之间存在 inclusion 关系，说明 C ⊑ D 这件事在所有满足 TBox 的 models 中永真。问题就变为了验证 a TBox T：T ⊨ C ⊑ D 是否成立。这个式子的语义是：所有 T 的 model 都是 C ⊑ D 的 model。依然是通过反证法来证明。假设存在一个 T 的 model I，它不是 C ⊑ D 的 model。则⊤ ⊑ ¬C ⊔ D 这件事在 I 中不成立，说明¬C ⊔ D 不再是 I 的 domain，说明 domain 中一定存在一个元素 x，x 在¬C ⊔ D 的补集之中，也就是在¬(¬C ⊔ D)之中，也就是在 C ⊓¬D 之中。所以问题变成了 C ⊓¬D 针对 TBox T 的可满足性问题。如果 C ⊓¬D 针对 T 是可满足的，说明原始的 T ⊨ C ⊑ D 不成立；如果 C ⊓¬D 针对 T 是不可满足的，说明原始的 T ⊨ C ⊑ D 成立。

比如让 T = {C ⊑ A, A ⊑ D}，check T ⊨ C ⊑ D。我们首先假设有一个 T 的 model I 不能让 C ⊑ D 成立，则 x: C ⊓¬D 就可以加入到 S 状态系统里面了。接下来，将原来 T 中的两个 GCIs 转化为⊤ ⊑ ¬C ⊔ A 和⊤ ⊑ ¬A ⊔ D。并使用 U 规则将 x: ¬C ⊔ A 和 x: ¬A ⊔ D 添加入 S 状态系统，继续执行 Tableau 算法的其它步骤。

这里注意，concept inclusion 关系的验证与 ABox 存在与否依然没有任何关系。

接下来看如何将问题（2）转化为（8）：

问题（2）：验证一个 individual 是否属于某个 concept。依然分为两种情况，一种是没有背景知识（without ontology），另一种是有背景知识（with ontology）。这里注意，背景知识不再局限于 TBox，而是整个 ontology（TBox + ABox）。

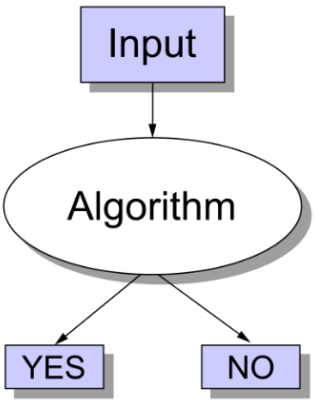
没有 ontology 的情况下，如果一个 individual a 属于一个 concept C，说明 C(a)这件事永真。说明任意一个 interpretation I，都可以满足 C(a)。只有一种可能，对于任意 interpretation I，C^I都是 I 的 domain。也就是说对于任意 interpretation I, $(\neg C)^I$都是空集，则¬C 是 unsatisfiable 的。问题就从验证 C 是 valid 的问题（C 是 tautology）变成验证¬C 的 satisfiability 问题（要验证 C(a)是否永真，理论上不可能找出所有 interpretation，每个都验证一遍。但可以假设存在一个 interpretation I 使得 C(a)不成立，则使得¬C(a)成立）。因为原始的式子中已经出现了一个 individual a，我们直接用它来当做 domain 的 element 即可。将 a: ¬C 加入到 S 状态系统中，并继续执行 Tableau 算法其它步骤。如果结果是 clash，则说明这样的 interpretation I 不存在，说明 C(a)永真；反过来，如果是 no rule is applicable，则说明 C(a)不永真。

在有 ontology 的情况下，如果一个 individual a 属于一个 concept C，说明 C(a)这件事在所有 ontology 的 models 上永真。问题就变为了验证 an ontology O：$O \models C(a)$是否成立，也就是要验证所有的 O 的 model 都是 C(a)的 model。反证法假设存在一个 O 的 model I 使得 C(a)不成立，则 I 要满足¬C(a)。所以需要将 a: ¬C 加入到当前的 S 状态系统，同时，将 O 中所有的 axioms 转化为一定形式后加入 S。比如 O 中存在 GCI $C \sqsubseteq D$，则按照 U 规则添加 a: $\neg C \sqcup D$ 进入 S；O 中存在 ABox axiom D(b)，则添加 b: D 进入 S；O 中存在 r(a,b)，则添加(a,b): r 进入 S。所以问题变成了¬C(a)针对 Ontology O 的可满足性问题。如果¬C(a)针对 Ontology O 是可满足的，说明原始的 $O \models C(a)$不成立；¬C(a)针对 Ontology O 是不可满足的，说明原始的 $O \models C(a)$成立。

如何将问题（3）转化为（8）是不是可以不用具体讲了？Tableau 算法过程讲解到此为止（其计算性质和性质的证明将在后续讲解）。

至止，是不是可以看到 concept satisfiability 问题有多么重要？我们把这类问题叫做 decision problem（Boolean problem），翻译为“决定性问题”。

decision problem: a yes-or-no question on specified sets of inputs



简单地说，decision problem 就是给定一组输入，输出一定是 yes 或者 no 的问题。比如，给定一个自然数，判断其是不是偶数？比如，给定两个自然数 x 和 y，判断 x 是否可以被 y 整除？用上图表示（维基百科）：

Tableau 就是这样一个算法。当一个 concept 是 satisfiable 的时候，算法会返回 yes，否则返回 no。是不是对于所有的 decision problem 都存在一个算法，使得当“客观事实”为 yes 的时候，该算法都能返回 yes 的结果，当“客观事实”为 no 的时候，该算法都能返回 no 的结果？答案当然是否定的。一个最典型的例子就是一阶谓词逻辑的 validity checking 问题（可等价转化为 satisfiability 问题）。客观上不存在一个 algorithm 使得当一组一阶谓词逻辑表达式为 invalid 的时候，一定返回一个 no 的答案。如果从 satisfiability 的角度看待，那就是，客观上不存在一个 algorithm 使得当一组一阶谓词逻辑表达式为 satisfiable 的时候，一定返回一个 yes 的答案（随着课程进行后续展示细节）。

对于一个 decision problem，如果存在一个 algorithm 使得在客观事实是 yes 的时候返回 yes，或者在客观事实是 no 的时候返回 no，则说明该问题是可决定的（decidable）。

接下来再想一个问题，当我们确定一个 decision problem 是 decidable 的，意味着世界上存在一个 algorithm 可以解决这个 decision problem。但是不是随便开发一个关于这个问题的算法都能确保解决这个问题？答案当然不是的。一个 decision problem 是 decidable 的是这个问题本身的性质，相当于“客观条件很好”。但是能不能开发出解决这个问题的算法是主观问题。比如计算 1+1 这个问题当然是可解决的，但是你开发了一个算法永不终止，或者返回了 3，也是有可能的，所以算法也要有一些性质来保证它的“质量”。

推理算法需要满足的三个重要性质：终止性（termination）、正确性（soundness）、完备性（completeness）。

终止性：给定任何输入，该算法都会在有限步骤内终止（不存在算法无限运行下去）

正确性：对于一个 concept C 进行 satisfiability checking，如果 Tableau 返回 yes，则客观事实上 C 一定是 consistent（不存在算法返回了 yes，但是客观事实 C 是 unsatisfiable 的；或者返回了 no，客观事实是 satisfiable 的）

完备性：对于客观事实是 satisfiable 的 concept C，Tableau 都能返回 yes（不存在一种情况，只针对某些 satisfiable 的 concept，算法才返回 yes；而对于另一些，算法计算不了）；对于客观事实是 unsatisfiable 的 concept C，Tableau 都能返回 no

当一个 decision problem 的算法满足上述三个性质的时候，我们将这个算法称为 decision procedure。这是知识表示与推理领域非常重要的一个概念。KR 学者的一个重要目标就是为各种逻辑语言寻找 decision procedure（前提是该语言本身是 decidable 的）。

Description Logic（DL）是一个语言家族，根据所允许使用的逻辑连接符和 basic building blocks 的不同，其提供的表达力，相应的计算性质也不同。我们学习了家族中的 EL, ALC, SHOIQ，其中重点学习了 EL 和 ALC。学习 EL 是因为 EL 有着特别好的计算性质，以及在现实中非常广泛的应用。学习 ALC 是因为它是目前公认的 DL 语言的核心，从 ALC 开始

（包括 ALC）扩展到 SHOIQ 和 SROIQ，这中间的所有语言都被认为是 expressive DL（表达力很强的 DL），计算性质都相对较差。大量的学者在研究 DL 家族不同成员的计算性质，其中的一些结果可以在：http://www.cs.man.ac.uk/~ezolin/dl/查询到。设计和实现表达力强、计算性质好的 KR 语言（不一定是现有的 DL，可能是新的扩展，可能完全跳出 DL 框架，甚至跳出一阶谓词逻辑框架的语言）是 KR 学者孜孜不倦追求的目标。