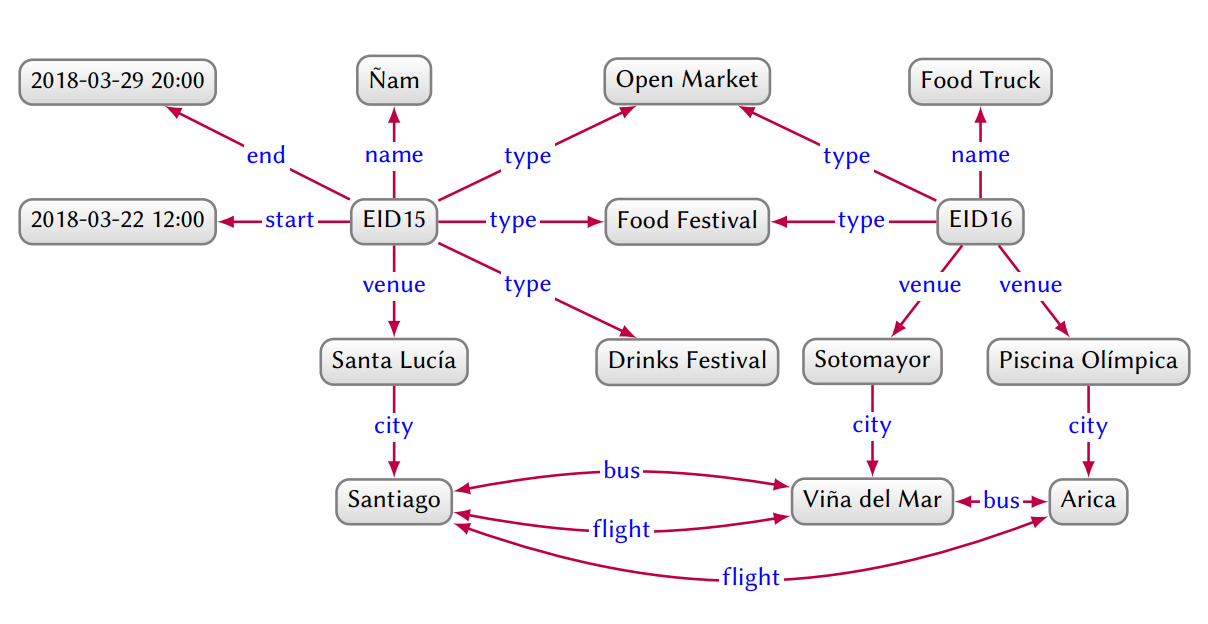
**4 演绎式知识**

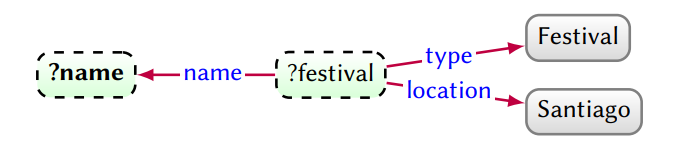
作为人类，我们可以基于图 1 推断出一些新的信息，例如 EID15 的举办地点是 Santiago、有航班相连的城市必定存在机场等。在这些情况下，给定图中的数据作为**「前提」**（premise），加上一些关于世界的通用规则作为**「先验」**（priori），我们就可以进行演绎来推导出新的数据，了解多比给定数据更多的信息。这些前提和先验一般被多人共享，构成了所谓的**「常识知识」**（commonsense knowledge）；与之相反，某些信息只在一定范围内被一些专家共享，构成了所谓的**「领域知识」**（domain knowledge），也可以理解为只有部分人掌握的专业性知识。



而对于机器来说，其并没有执行上述演绎的能力，我们需要为机器给出形式化的说明，包括前提数据以及**「蕴涵制度」**（entailment regimes），以使其具备类似于人类的演绎能力。蕴涵制度简单来说就是对基于逻辑得出的一个前提数据集合的形式化，机器可以基于该制度来执行高效而精准的推断。这种演绎可以应用到多个领域，包括改善查询结果、分类等。下面给出一个关于查询的例子，假定我们想了解在 Santiago 举办的节日，如下图中的图模式所示。如果直接在图 1 中进行查询，是没有结构的，因为图中没有 Festival 节点，也没有边 location。不过，如果我们给出如下的蕴涵制度：

* *x* 是一个 Food Festival 蕴涵了其是一个 Festival
* *x* 在城市 *z* 有一个 venue *y* 蕴涵了 *x* 有一个地点 *z*

那么机器就会自动推断出 Ñam 这个答案。那么如何捕捉到这种蕴涵呢？对于第一条我们可以通过之前介绍的子类关系来实现；而第二条则需要表达性更强的方法来实现。



本章节将介绍如何对更加复杂的蕴涵进行表达与自动化的方法。虽然存在着很多的逻辑框架可以达到该目的，但是我们将专注于**「本体」**（ontologies），其构建了一个关于知识的形式化表示，同时可以表达为图的形式。我们将介绍本体如何被形式化定义，其与现存逻辑框架的关系，以及如何对本体执行推理。

### 4.1 本体

为了实现蕴涵，我们必须明确所使用到的术语的含义。以图 1 为例，我们可能想了解诸如 EID16 这样的事件（event）的精确定义，其实发生在连续时间间隔内的还是可以多次发生？其是否可以发生在多个地点？对于这些问题并没有所谓的正确答案，我们要做的就是定义**「约束」**（convention），来明确这些问题的答案。

在计算机领域，本体就是用来制定这样的约束的。本体就是对术语在某个范围内（如一个给定的领域）的含义的具体而形式化的表示。例如，在一个事件本体中，我们定义如果一个实体为一个”事件“，那么其必须（仅）包含一个地点和一个起始时间。而在另一个事件本体中，我们则可以定义一个事件可以包含多个地点与多个起始时间。每个这样的本体都形式化地定义了一个特定的**「约束」**。我们可以使用这样的约束来自动化蕴涵。

与其他约束一样，一个本体的有用性依赖于其所定义约束的约定等级、细节程度、适用范围及一致性。在一个知识图谱中使用本体可以保证该知识图谱中术语使用和建模的一致性，而在多个知识图谱达成一致（使用本体）则可以增强这些知识图谱的互操作性。

在实践中，最常用的本体语言是 W3C 推荐的**「网络本体语言」**（OWL），其与 RDF 图兼容；以及生物医学领域常用的 **「OBOF」**。两者有许多类似的特征，本文将专注于 OWL 语言。在介绍 OWL 的特征之前，首先需要了解图是如何被**「解释」**（interpreted）的。

### 4.1.1 解释

对于一张图（例如图 1），作为人类我们可以基于图中的节点与边来解释得到现实世界中的信息，如将 Santiago —flight→ Arica 解释为有从城市 Santiago 飞到城市 Arica 的航班。这种**「解释」**的过程将数据图解释为另一张包含通过现实世界关系连接的现实世界实体的图，我们称之为**「领域图」**（domain graph），其具体操作为将数据图中的节点和边**「映射」**至领域图中的节点和边。

我们将一张数据图的**「解释」**（interpretation）定义为包含两个元素：一张领域图，以及数据图中的**「项」**（terms）到领域图中的项的映射。领域图遵循与数据图相同的模型。为了简洁性，我们只讨论有向标记图，将领域图中的节点称为**「实体」**（entities），将领域图中的边称为**「关系」**（relations）。在这种抽象化定义下，我们并不要求领域图中的实体与关系来自现实世界，只需要将其理解为数据图的映射即可。

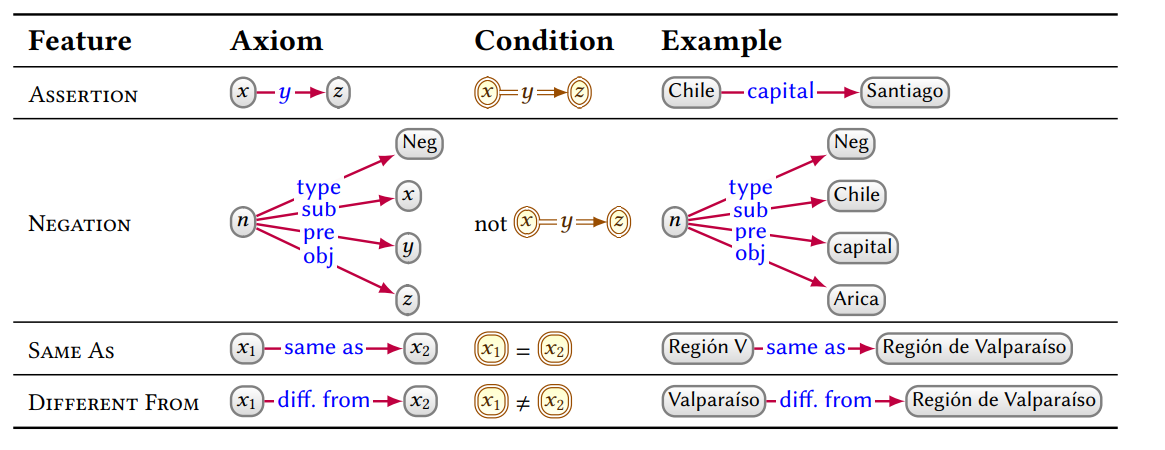
对解释进行抽象化定义的作用在于，当我们定义本体的特征及蕴涵时，节点/边与实体/关系之间的差别就会显现出来。下面举例来说明这种区别，在图 1 所表示的数据图中，如果我们询问在 Arica 和 Viña del Mar 之间是否存在一条标签为 flight 的边，那么答案是**「否定」**的；然而，如果我们询问在对应的领域图中，实体 Arica 和实体 Viña del Mar 之间是否存在关系 flight，那么答案则取决于我们在解释图时采取了何种假设，在**「封闭世界假设」**（CWA）下，如果我们没有额外的知识，那么答案是否定的（未知的即是不存在的），而在**「开放世界假设」**（OWA）下，答案则是不确定的。类似地，在唯一命名假设（UNA）下，数据图中每一个不同名称的节点都对应不同的实体，但在非唯一命名假设（NUNA）下，数据图中不同名称的节点所对应的实体可能是同一个。

这些假设定义了哪些解释是合法的，以及特定的解释满足哪些数据图。UNA 假设不允许解释将两个不同名称的节点映射为同一个实体，NUNA 则允许这种解释；在 CWA 假设下，一个包含边 x —flight→ y 的领域图只能满足包含对应边 x —p→ y 的数据图，而在 OWA 解释下，同样的领域图则可以满足不包含对应边的数据图，只要其没有相矛盾的边。在 OWL 中，采用了**「非唯一命名」**和**「开放世界」**假设，即数据图中的多个节点/边可能指向相同的实体/关系，以及任何不在数据图中的事实并不假定为 false。

在这些基础假设之上，我们可以将数据图中的某些模式通过**「语义条件」**（semantic conditions）和领域图关联起来，语义条件的作用是定义哪些解释是满足对应的模式的。例如，我们可以定义一个语义条件来声明如果数据图中包含边 p —subp. of→ q ，那么对于领域图中的任意边 x —p→ y ，都必须有一条对应的边 x —q→ y 来满足数据图。这些语义条件即构成了一个本体语言的特征。下面我们将通过抽象化的图形表示来介绍 OWL 的主要特征。

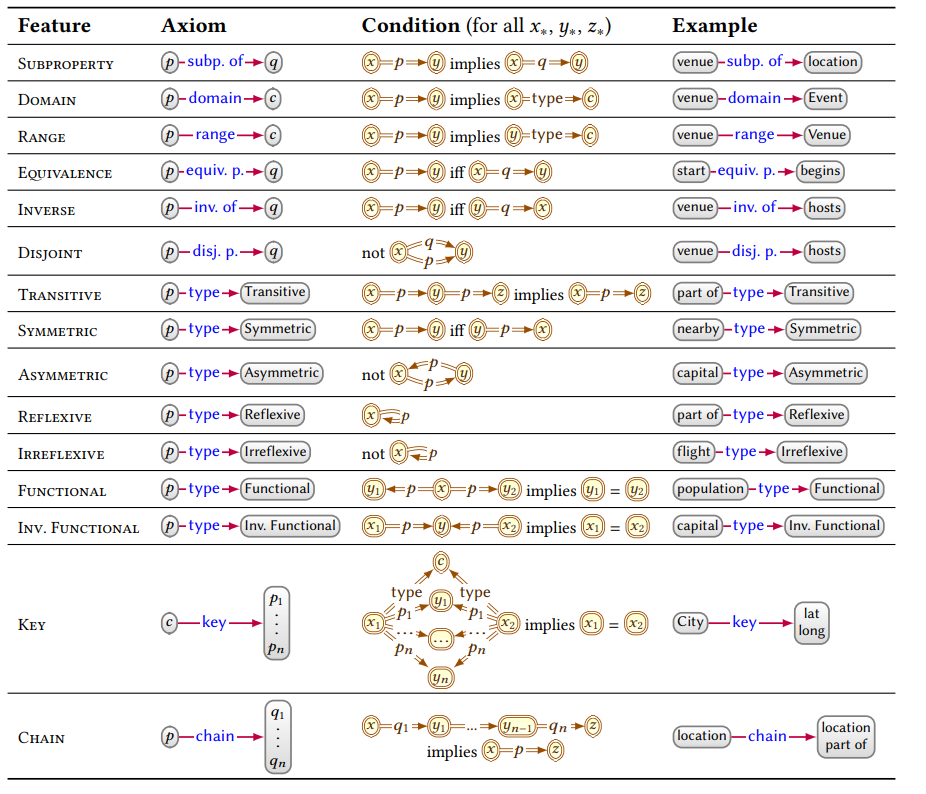
### 4.1.2 个体

下表列出了 OWL 用于描述**「个体」**（individuals）所支持的主要特征。个体指类的具体实例（如 Santiago、EID16），通常与类和属性相区分。首先我们可以通过边来**「断言」**（assert）两个实体间的关系，在 condition 一列中给出的是给定公理成立时对应的解释。除了断言特征外，OWL 还支持**「否定」**（针对非个体之间的边，例如 type 属性和 RDF 实化）、**「相等」**（两个节点是否对应相同的实体）以及**「不等」**特征。



### 4.1.3 属性

在 3.1.1 节中，我们已经讨论了属性的**「子属性」**（subproperties）、**「领域」**（domains）以及**「范围」**（ranges）要如何定义。OWL 不仅支持上述特征，还支持了其他的特征，如下表所示。我们可以定义一对属性的**「相等」**（equivalent）、**「反转」**（inverses）以及**「分离」**（disjoint）；可以定义一个属性的**「传递」**（transitive）、**「对称」**（symmetric）、**「不对称」**（asymmetric）、**「反身」**（reflexive）以及**「非反身」**（irreflexive）关系；可以定义属性关系的多重性，包括**「功能性」**（functional，多对一）、**「反功能性」**（inverse-functional，一对多）。我们还可以定义一个类的**「键」**（key），其由一个属性的集合组成，其值可以唯一地识别该类的实体（注意上述三个特征都是在 UNA 假设下才能将不同名称的节点识别为相同实体的）。最后，我们可以将一个属性与一条**「链」**（chain）相关联，链指的是一个只允许属性连接的路径表达式，通过该链所连接的实体对也可以通过给定的属性关联。针对最后两个特征中包括的属性列表，可以通过不同的方式实现，OWL 使用的是 RDF 列表。

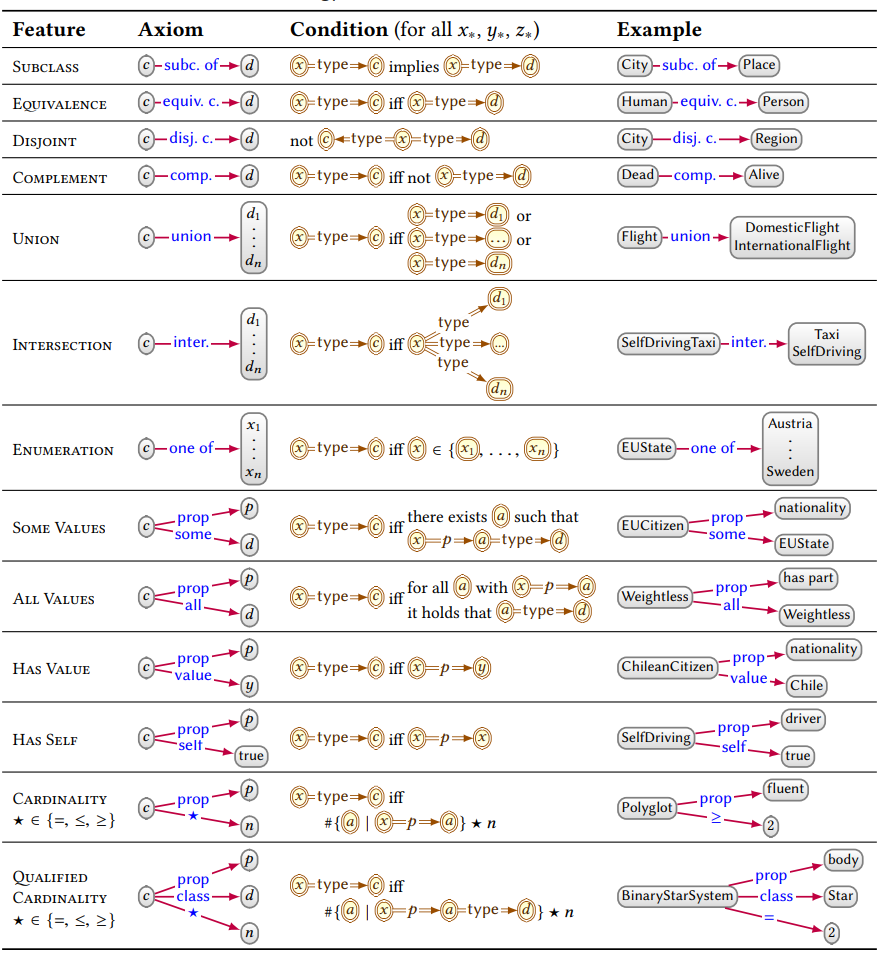


### 4.1.4 类

在 3.1.1 节中，我们讨论了类的层级结构可以通过**「子类」**（sub-class）关系来构建。OWL 支持包含子类在内的很多特征，用来定义与类相关的公理，如下表所示。给定一对类，OWL 允许定义它们是**「相等」**（equivalent）或**「分离」**（disjoint）的；此外，OWL 提供了很多特征来定义一个新的类，通过使用集合操作符或是基于类的实例的属性所满足的条件。首先，使用集合操作符，我们可以定义一个新的类作为另一个类的**「补集」**（complement），或是其他类的**「并集」**（union）或**「交集」**（intersection），或是作为类中所有实例的**「枚举」**（enumeration）。其次，通过对特定属性 p 进行限制，我们可以定义类中的所有实例满足如下公理：

* *some value*：通过 p 连接到一个指定类的实例（存在即可，至少一个）
* *all values*：通过p  连接到一个指定类的实例（只允许该类，可以不存在）
* *has value*：通过 p 连接到特定的个体
* *has self*：通过 p 连接至自身
* *has cardinality*：包含指定数量的 p
* *qualified cardinality*：包含连接到指定类的指定数量的 p

关于上述的属性限制，可以作为类的必要条件（原始类），也可以作为类的充要条件（定义类）。由于 *all values* 公理对于不存在关系的时候也成立，有时我们会将 *some value*（存在性限制）和 *all value*（普遍性限制）结合起来定义更强的约束。



### 4.1.5 其他特征

除了上述特征之外，OWL 还支持一些其他的特征，包括：

* **「注解属性」**（annotation properties）：提供本体的元数据，例如版本信息
* **「数据类型属性和对象属性」**（datatypevs. object properties）：将指向数据类型值的属性与指向个体的属性区分开来
* **「数据类型限制」**（datatype facets）：对数据类型值添加限制，如具体的类型与大小范围

关于更多 OWL 的细节，可以参考 OWL2 **官方文档**[1]。

### 4.2 语义与蕴涵

在上面的表格中列出的**「条件」**（condition）声明了每个特征应该如何被解释，这即引发了**「蕴涵」**（entailments）。下面我们将对条件如何产生蕴涵进行详细的描述。

### 4.2.1 模型论语义

对于上述表格中描述的每条公理，当将其添加到一张图中，即会触发某些满足该图的解释的条件，我们将这些满足图的解释称为图的**「模型」**（model）。在 OWA 假设下，任何图的模型的数量都是无限的；在 NUNA 假设下，只要边没有非反身特征的限制，则 a —a→ a 可以是任何图的一个模型，只要图中存在边 x —y→ z （将所有节点指向同一实体）。

### 4.2.2 蕴涵

我们称一张图**「蕴涵」**（entails）了另一张图，当且仅当前者的任意模型也是后者的一个模型。直观来看这就意味着前一张图包含了后一张图中的所有信息，即后一张图为前一张图的逻辑结果。举例来说，对于图 Santiago —type→ City —subc. of→ Place 和图 Santiago —type→ Place，后者中的所有模型必须满足对应实体的类均为 Place，而前者不仅需要满足这一条件，还需要进一步满足子类关系。因此前一张图的任意模型都是后一张图的模型（注意不要弄反了，前者要求的信息多于后者），即前一张图蕴涵了后一张图。

### 4.2.3 if-then 与 if-and-only-if 语义

考虑两张图 nearby —type→ Symmetric 和 nearby —inv. of→ nearby，其会得出相同的语义条件，那么这两张图是否相互蕴涵？实际上这取决于领域图所应用的语义。在 if-then 语义下，如果**「公理」**（Axiom）匹配数据图那么**「条件」**（Condition）在领域图中成立，此时两张图并不相互蕴涵；而在 if-and-only-if 语义下，公理匹配数据图当且仅当条件在领域图中成立，此时两张图相互蕴涵。两张图都指向相同的条件，该条件又会转换回描述它的所有可能公理。因此 if-and-only-if 语义允许在本体语言中蕴涵更多的公理，OWL 一般采用 if-and-only-if 语义。

### 4.3 推理

实际上，给定两张图，根据蕴涵的定义以及上述所有本体特征，我们是**「无法」**决定其中一张图是否蕴涵另一张图的——没有（有限）算法可以保证基于输入的所有蕴涵判断都是正确的。不过我们可以为本体提供推理算法来帮助判断蕴涵，具体来说有三种选择：

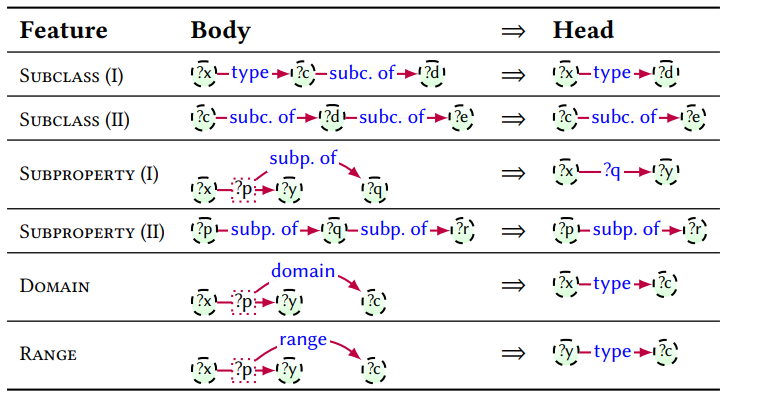
* 对于任意输入本体都可以完成判断（不会无限循环），但是可能会遗漏一些蕴涵，停止在错误的判断
* 总是可以停止于正确的判断，但是对输入本体的特征有所限制
* 对于任意输入本体都只返回正确的判断，但是对于某些特定的输入可能会无限循环

在实践中，选项 1 和 2 的使用更多，通常基于**「规则」**和（或）**「描述逻辑」**来实现。选项 1 通常允许更高效与可扩展的推理算法且在数据不完整的时候作用更大；选项 2 则在某些领域更加适用，例如医学本体，错误的蕴涵可能会引发不好的结果。

### 4.3.1 规则

为演绎式知识提供自动化推理的最直接的方法就是使用**「推理规则」**（inference rules）来编码 IF-THEN 风格的结果。一条规则由一个 **「body」**（IF) 和一个 **「head」**（THEN）组成，两者都通过图模式的形式给出。一条规则表明如果我们将 body 中的变量替换为数据图中的术语，形成一张给定数据图的子图，那么在 head 中使用相同的变量替换会得出一个合法的**「蕴涵」**。Head 一般来说需要使用出现在 body 中的变量的子集，以确保结果中没有未替换的变量。

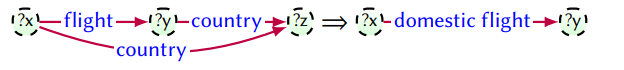
规则可以用来捕捉本体条件下的蕴涵。下表列举了部分用于子类、子属性、领域和范围特征的示例规则。这些规则可能是不完整的，例如其无法捕捉到每个类都是其自身的子类，每个属性都是其自身的子属性。针对之前表格中的 OWL 特征的更加全面的规则集合被定义为 **「OWL 2 RL/RDF」**。不过这些规则同样无法捕捉一些特征，如否定、存在性限制、普遍性限制等。



规则可以通过多种方式来进行推理。**「物化」**（Materialisation）指将规则递归地应用于图，将生成的结论添加回图中，直到到达一个固定的点，不能再添加其他东西为止。物化后的图可以当做普通的图来看待。虽然可以通过优化（如 Rete 网络）或分布式框架（如 MapReduce）来提升物化的效率与可扩展性，但是基于规则和数据，物化图可能会变得过大以致于难以管理。**「查询重写」**（query rewriting）指给定一个查询，可以自动化地扩展该查询以找出规则集合所得出的蕴涵。下图给出了一个查询重写的例子，重写的查询是本章最开始给出的图查询。**「OWL2 QL」** 是针对这种形式的查询重写而特别设计的一个 OWL 子集。



除了将规则用于捕捉本体蕴涵，其还可以独立于本体语言进行定义，用于捕捉给定领域的蕴涵。下图给出了一个示例，其捕捉了航空领域内的一个蕴涵。此外，由于可计算性的原因，这些规则并不支持从循环图模式中推理关系。在实践中，常用的独立于本体语言的规则语言有：**「Notation3」**（N3）、**「Rule Interchange Format」**（RIF）、**「Semantic Web Rule Language」**（SWRL）以及 **「SPARQL Inferencing Notation」**（SPIN）。



### 4.3.2 描述逻辑

**「描述逻辑」**（DL）最初是作为形式化**「框架」**（frames）与**「语义网络」**（semantic networks）的方式而引入的，由于语义网络可以看做是知识图谱的早期版本，且 DL 很大程度上影响了 OWL 的形成，因此 DL 在知识图谱的逻辑形式化中占有重要地位。DL 形成的并非一条特定逻辑，而是一系列的逻辑。DL 最开始是作为**「一阶逻辑」**（FOL）的受限片段出现的，可以执行可确定的推理任务，例如蕴涵检查。不同的 DL 在表达能力和推理的计算复杂度之间取得了不同的平衡。之后 DL 引入了其他的特征进行扩展，在建模图数据上非常有用。

描述逻辑基于三种类型的元素：**「个体」**（individuals）、**「类」**（classes）以及**「属性」**（properties）。DL 可以关于这些元素进行声明，也被称为**「公理」**（axioms）。**「断言公理」**（assertional axioms）可以是个体的一元类关系，如 City(Santiago)，也可以是个体的二元属性关系，如 flight(Santiago, Arica)。这些公理会形成**「断言盒」**（A-Box）、DL 又进一步地引入了逻辑符号，来定义**「类公理」**（class axioms）以及**「属性公理」**（property axioms），类公理会组成**「术语盒」**（T-Box），属性公理则会组成**「角色盒」**（B-Box）。举例来看，类公理 City⊑Place 表明前一个类是后一个类的子类，属性公理 flight⊑connectsTo 则表明前一个属性是后一个属性的子属性。DL 还引入了逻辑符号的富集，不仅用于定义类与属性公理，还可以基于现有术语定义新的类。例如我们可以定义一个类 ∃flight.⊤⊑∃nearby.Airport 来表示有航班的个体都在其位置附近有机场存在（类似于存在性限制），DL 中的 ⊤ 用来表示类中的所有个体。

可以看到，上述 DL 特征与之前介绍的 OWL 特征存在很多相似之处，这并不是巧合，因为 OWL 标准的制定受了 DL 很大的影响。**「OWL2 DL」** 语言是一种受限的 OWL 语言，能够保证蕴涵是可决定的。例如通过约束 DomesticAirport⊑=1destination◦country.⊤，我们可以定义国内的航班只能飞往一个国家（p◦q 表示属性链）。然而，在 DL 中为了确保可判定性，对属性链进行计数是不被允许的。

表达性的 DL 支持复杂的蕴涵，包括存在性、普遍性、计数等。一种确定这些蕴涵的常用策略是将蕴涵减少至**「可满足性」**（satisfiability），其决定了一个本体是否是一致的。我们可以使用诸如 **「tableau」** 的方法来检查可满足性，通过使用类似之前提到的物化策略的方式构建模型，但需要额外地进行一些操作，如在涉及到**「分离」**（disjunction）时需要额外对模型进行分支；在涉及到**「存在性」**（existentials）时引入新元素等。如果模型构建完成，该过程会总结出原始的定义是否可以满足。由于计算复杂度较高，虽然这种推理策略在建模复杂领域时很有用，但通常不会在大规模的数据中使用。