Графы Знаний (часть 1/3)

План лекции:

1. Модель данных и язык запросов к графу знаний
2. Схема, идентичность и контекст в графах знаний
3. Дедуктивные знания

---

1. Модель данных и язык запросов к графу знаний

Модели

Предположим, что Совет по туризму из нашего примера еще не решил, как моделировать релевантные данные о достопримечательностях, событиях, услугах и т. д. Совет директоров сначала рассматривает возможность использования табличной структуры – в частности, реляционных баз данных – для представления требуемых данных, и хотя они не знают точно, какие данные им нужно будет захватить, они начинают разрабатывать первоначальную реляционную схему. Они начинаются с таблицы событий с пятью столбцами:

Событие(название, Место проведения, тип, начало, конец)

где имя и начало вместе образуют первичный ключ таблицы, чтобы однозначно идентифицировать повторяющиеся события. Но когда они начинают заполнять данные, они сталкиваются с различными проблемами: события могут иметь несколько названий (например, на разных языках), события могут иметь несколько мест проведения, они могут еще не знать даты начала и окончания будущих событий, события могут иметь несколько типов и т. д. Постепенно решая эти проблемы моделирования по мере того, как данные становятся более разнообразными, они генерируют внутренние идентификаторы событий и адаптируют свою реляционную схему до тех пор, пока они не будут:

EventName(id, name), EventStart(id, start), EventEnd(id, end), (1)

EventVenue(id, место проведения), EventType(id, тип)

С помощью приведенной выше схемы организация теперь может моделировать события с именами, местами проведения и типами, а также датами начала и окончания. По пути правлению приходится несколько раз постепенно менять схему, чтобы поддерживать новые источники данных. Каждое такое изменение требует дорогостоящей перестройки, перезагрузки и переиндексации данных; здесь мы рассматривали только одну таблицу. Совет по туризму борется с реляционной моделью, потому что они априори не знают, какие данные должны быть смоделированы или какие источники они будут использовать. Но как только они достигают последней реляционной схемы, правление обнаруживает, что они могут интегрировать дальнейшие источники без дополнительных изменений: с минимальными допущениями о кратности связей (1-1 , 1–n и т. д.) эта схема предлагает большую гибкость для интеграции неполных и разнообразных данных.

На самом деле утонченная, гибкая схема, которую получает Совет, показана в (1), моделирует набор бинарных отношений между сущностями, которые действительно можно рассматривать как моделирование графа. Вместо этого, приняв с самого начала графовую модель данных, правление могло бы отказаться от необходимости в предварительной схеме и могло бы определить любое (бинарное) отношение между любой парой сущностей в любое время. Теперь мы представляем графовые модели данных, обычно используемые на практике.

Направленные графы с именованными ребрами. Направленный граф с именованными ребрами (также известный как мультиреляционный граф) определяется как набор узлов – подобных Santiago, Arica, EID16, 2018-03-22 12:00 – и набор направленных именованных ребер между этими узлами, подобных Santa Lucía —city→ Santiago. В случае графов знаний узлы используются для представления сущностей, а ребра-для представления (бинарных) отношений между этими сущностями. На рис. 1 приведен пример того, как Совет по туризму может смоделировать некоторые релевантные данные о событиях в виде ориентированного графа с маркировкой ребер. Граф включает в себя данные о названиях, типах, дате начала и окончания, а также местах проведения мероприятий. 2. добавление информации, такой граф обычно включает в себя добавление новых узлов и ребер (с некоторых исключениях поговорим чуть позже). Моделирование данных в виде графа обеспечивает большую гибкость для интеграции новых источников данных, по сравнению со стандартной реляционной моделью, где схемы должны быть определены заранее и необходимо следовать им на каждом этапе. В то время как другие структурированные модели данных, такие как деревья (XML, JSON и т. д.), предлагают аналогичную гибкость, графы не требуют иерархической организации данных (например, должны ли они быть родительскими, дочерними или родственными по типу?). Они также позволяют представлять циклы и запрашивать их (например, обратите внимание на направленный цикл на маршрутах между Сантьяго, Арикой и Винья-дель-Мар). А стандартизированной моделью данных, основанной на направленных графах с маркировкой ребер, является структура описания ресурсов (RDF - Resource Description Framework), которая была рекомендована W3C. RDF модель определяет различные типы узлов, в том числе многоязычных идентификаторов ресурсов (IRI), которые подходят для глобальной идентификации лиц на веб-сайте; литералы, которые подходят для представления строк (с или без тегов языка) и других данных значений (целые числа, даты и т. п.); и пустые узлы, которые являются анонимными узлы, на которые не назначены идентификатор (например, вместо того, чтобы создавать внутренние идентификаторы, как EID15, EID16, PCO, у нас есть возможность использовать пустые узлы).

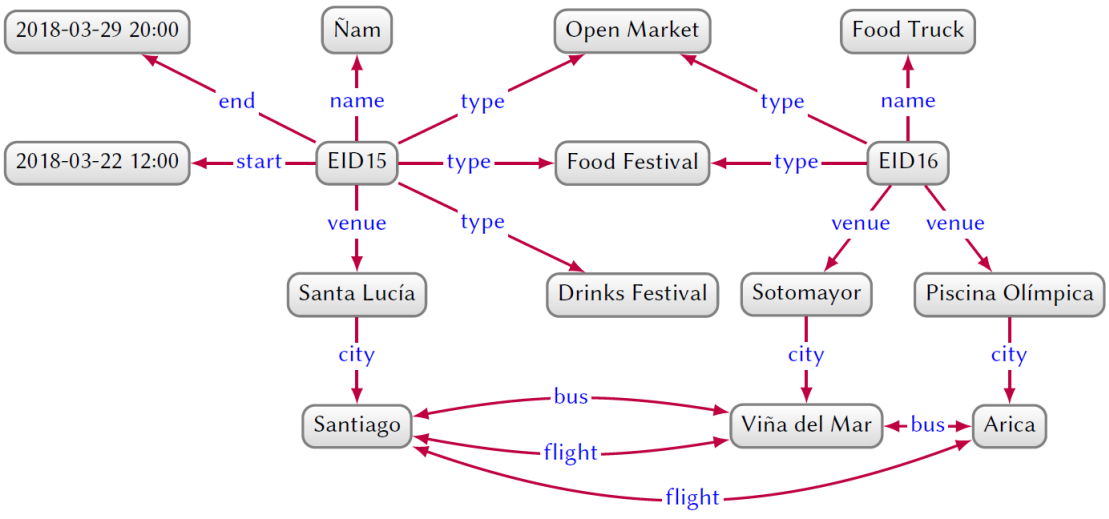


Рис. 1. Направленный граф с именованными ребрами, описывающий события и их места проведения

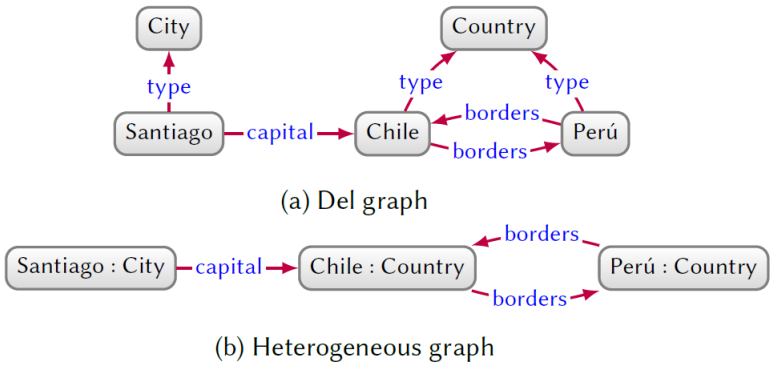


Рис. 2. Данные о столицах и странах в ориентированном графе с маркировкой ребер и гетерогенном графе

Гетерогенные графы. Гетерогенный граф (или гетерогенная информационная сеть) - это граф, в котором каждому узлу и ребру присваивается один тип. Таким образом, гетерогенные графы сродни графам с метками ребер, соответствующими типам ребер, – но где тип узла является частью самой Графовой модели, а не выражается в виде специального отношения, как показано на рис. 2. Ребро называется однородным, если оно находится между двумя узлами одного типа (например, границами); в противном случае оно называется гетерогенным (например, капиталом). Преимущество гетерогенных графов состоит в том, что они позволяют разбивать узлы в соответствии с их типом, например, для целей задач машинного обучения. И наоборот, они обычно поддерживают только взаимно однозначное отношение между узлами и типами, чего нельзя сказать о графах с именованными ребрами (см., например, узел Santiago с нулевыми типами и EID15 с несколькими типами на Рис. 1).

Графы свойств. Графы свойств были введены для обеспечения дополнительной гибкости при моделировании более сложных отношений. Рассмотрите возможность интеграции входящих данных, которые предоставляют информацию о том, какие компании предлагают тарифы на какие рейсы, что позволяет совету директоров лучше понимать доступные маршруты между городами (например, на национальных авиалиниях). В случае направленных графов с маркировкой ребер мы не можем напрямую аннотировать ребро, как Santiago — flight → Arica в случае с компанией (или компаниями), предлагающими этот маршрут. Но мы могли бы добавить новый узел, обозначающий рейс, связать его с источником, пунктом назначения, компаниями и режимом, как показано на рис. 3. Применение этого моделирования ко всем маршрутам на рис. 1, повлечет за собой значительное изменение графа. Другой вариант может заключаться в том, чтобы поместить рейсы разных компаний в разные именованные графы, но если именованные графы уже используются для отслеживания источника графов (например), это может стать громоздким. Таким образом, модель графа свойств была предложена для обеспечения дополнительной гибкости при моделировании данных в виде графа. Граф свойств позволяет связать набор пар свойств–значений и метку как с узлами, так и с ребрами. На рис. 4 показан краткий пример графа свойств с данными, аналогичными Рис. 3. На этот раз мы используем пары свойства–значение на ребрах для моделирования компаний. Тип отношения фиксируется меткой flight. Далее мы используем метки узлов, чтобы указать типы этих двух узлов, и используем пары свойств–значений, чтобы указать их широту и долготу. Графы свойств наиболее широко используются в популярных графовых базах данных, таких как Neo4j. При выборе между графовыми моделями важно отметить, что графы свойств могут быть переведены в/из ориентированных графов с маркировкой ребер без потери информации (см., например, рис.4). Таким образом, направленный граф с помеченными ребрами предлагает более компактную модель, в то время как графы свойств предлагают более гибкую. Часто выбор модели

вторичен по отношению к другим практическим факторам, таким как реализации, доступные для различных моделей, и т. д.

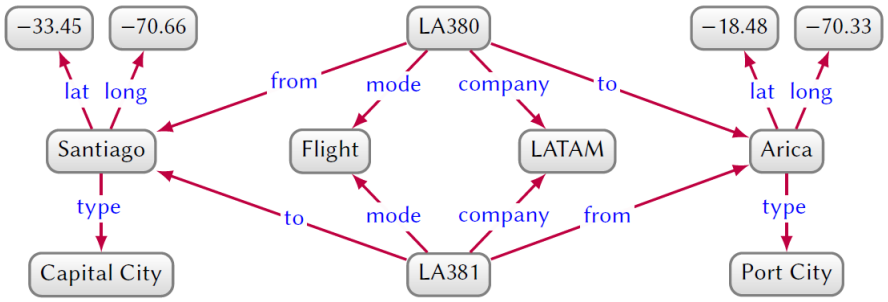


Рис. 3. Направленный граф с компаниями, предлагающими рейсы между Сантьяго и Арикой

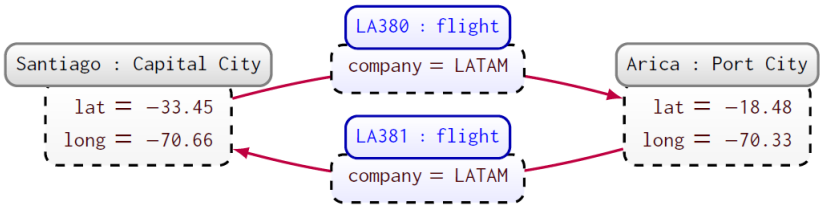


Рис. 4. Граф свойств с компаниями, предлагающими рейсы между Сантьяго и Арикой

Набор данных графа. Хотя несколько направленных графов могут быть объединены, часто желательно управлять несколькими графами, а не одним монолитным графом; например, может быть полезно управлять несколькими графами из разных источников, что позволяет обновлять или уточнять данные из одного источника, отличать ненадежные источники от более надежных и т. д. Набор данных графа состоит из набора именованных графов и графа по умолчанию. Каждый именованный граф представляет собой пару идентификатора графа и графа.

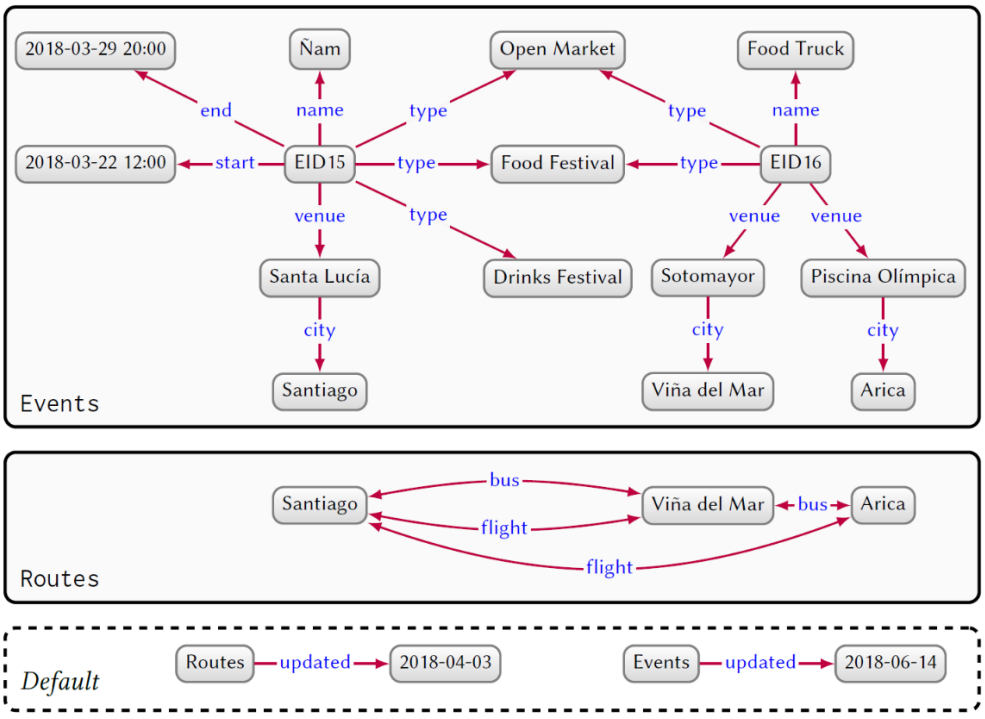


Рис. 5. Набор графовых данных с двумя именованными графами и графом по умолчанию, описывающим события и маршруты

Граф по умолчанию-это граф без идентификатора, на который ссылаются “по умолчанию”, если идентификатор графа не указан. На рис.5 приведен пример, в котором события и маршруты хранятся в двух именованных графах, а граф по умолчанию управляет метаданными о именованных графах. Имена графов также могут использоваться в качестве узлов графа. Кроме того, узлы и ребра могут повторяться в графах, где один и тот же узел в разных графах ссылается на одну и ту же сущность, что позволяет интегрировать данные об этой сущности при объединении графов. Хотя пример иллюстрирует набор данных ориентированных графов, понятия обобщаются прямо на другие типы графов. Явным примером использования графовых наборов данных является управление и запросы к связанным данным, состоящих из взаимосвязанных документов графов RDF, охватывающих WEB. При работе с веб-данными отслеживание источника данных приобретает ключевое значение.

Другие графовые модели данных. Предыдущие модели являются популярными примерами графовых представлений. Другие графовые модели данных существуют, например со сложными узлами, которые могут содержать отдельные ребра или вложенные графы (иногда называемые гипернодами). Точно так же математическое понятие гиперграфа определяет сложные ребра, которые соединяют множества, а не пары узлов. На наш взгляд, граф знаний может принять любую такую графовую модель данных на основе узлов и ребер: часто данные могут быть преобразованы из одной модели в другую (см. рис. 3 против Рис. 4). Далее мы предпочитаем обсуждать направленные графы с именованными ребрами, учитывая их относительную лаконичность, но большая часть дискуссий естественным образом распространяется и на другие модели.

Графовые хранилища данных. Были предложены различные методы хранения и индексирования графов, облегчающие эффективную оценку запросов (о чем речь пойдет далее). Направленные графы с именованными ребрами могут храниться в реляционных базах данных либо как единое отношение трех сущностей (таблица триплетов), либо как бинарное отношение для каждого свойства (вертикальное разбиение), либо как n-арные отношения для сущностей данного типа (таблицы свойств). Были также разработаны специальные методы хранения данных для различных графовых моделей, обеспечивающие эффективный доступ для поиска узлов, ребер и смежных с ними элементов. Ряд систем также позволяют распределять графы по нескольким серверам на основе популярных хранилищ NoSQL или пользовательских схем секционирования.

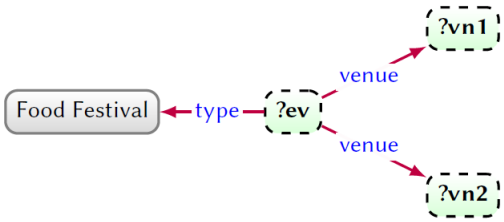
Механизм запросов

Был предложен ряд практических языков для запросов к графам, включая язык запросов SPARQL для графов RDF; и Cypher, Gremlin и G-CORE для запросов к графам свойств. В основе этих языков запросов лежат некоторые общие примитивы, включая (базовые) графовые шаблоны, реляционные операторы, выражения путей и многое другое. Теперь мы опишем эти основные функции для запросов к графам, начиная с шаблонов графов.

Графовые паттерны. В основе каждого структурированного языка запросов для графов лежат (базовые) графовые шаблоны, которые следуют той же модели, что и запрашиваемый граф данных, дополнительно позволяя переменные в качестве терминов. Термины в графовых шаблонах, таким образом, делятся на константы, такие как Arica или место, и переменные, которые мы префиксируем вопросительными знаками, такими как ?событие или ?rel. Затем шаблон графа вычисляется по графу данных путем создания отображений из переменных шаблона графа на константы в графе данных. На рис. 6 мы приводим пример графового шаблона, для поиска места проведения фестивалей еды, вместе с возможными отображениями, генерируемыми графовым шаблоном из графа данных из Рис.1. В некоторых из представленных отображений (последних двух перечисленных) несколько переменных сопоставляются с одним и тем же термином, который может быть или не быть желательным в зависимости от приложения. Поэтому ряд семантик был предложен для оценки графовой модели, среди которых наиболее важными являются:

гомоморфная-семантика, которая позволяет использовать несколько переменных, чтобы сопоставить один и тот же термин как показано на рис. 6.; и изоморфизм на основе семантики, которая требует переменных в узлах и/или ребрах, чтобы быть сопоставленным с уникальными условиями, при этом не считая последних трех сопоставлений на рис. 6. Различные практические языки запросов принимают различную семантику для оценки графовых шаблонов, где, например, SPARQL принимает семантику, основанную на гомоморфизме, в то время как Cypher принимает семантику, основанную на изоморфизме ребер. Как мы увидим в последующих примерах (в частности, на рис. 8), графовые шаблоны также могут формировать циклы (будь то направленные или неориентированные) и могут заменять метки ребер переменными. Графовые шаблоны в контексте других моделей, таких как графы свойств, могут быть определены аналогично, позволяя переменным заменять термины в любой позиции модели.

Сложные графовые шаблоны. Шаблон графа преобразует входной граф в таблицу результатов (как показано на рис.6). Затем мы можем рассмотреть возможность использования реляционной алгебры для объединения и/или преобразования таких таблиц, формируя таким образом более сложные запросы из одного или нескольких шаблонов графов. Напомним, что реляционная алгебра состоит из унарных операторов, которые принимают одну входную таблицу, и бинарных операторов , которые принимают две входные таблицы. Унарные операторы включают проекцию (π) для вывода подмножества столбцов, выделение (σ) для вывода подмножества строк, соответствующих заданному условию, и переименование столбцов (ρ).



?ev	?vn1	?vn2
EID16	Piscina Olímpica	Sotomayor
EID16	Sotomayor	Piscina Olímpica
EID16	Piscina Olímpica	Piscina Olímpica
EID16	Sotomayor	Sotomayor
EID15	Santa Lucía	Santa Lucía

Рис. 6. Графовая модель (слева) с отображениями, сгенерированными на графе Рис. 1 (справа)

Бинарные операторы включают union (∪) для объединения строк двух таблиц в одну таблицу, difference (−) для удаления строк из первой таблицы, присутствующей во второй таблице, и joins (⋈) для расширения строк одной таблицы строками из другой таблицы, удовлетворяющими условию соединения. Условия выбора и соединения обычно включают равенства (=), неравенства (≤), отрицание (), дизъюнкцию (∨) и т. д. Из этих операторов, можно дополнительно определить другие (синтаксических) операторы, такие как пересечение (∩) для вывода строки в обеих таблицах, анти-присоединиться (▷ака не существует) для вывода строки из первой таблицы, для которых нет совместимо для соединения строк во второй таблице, слева-соединение (⟕ ака необязательно) чтобы выполнить соединение, но сохраняя строки из первой таблицы без совместимого использования строки во второй таблице, и т. д. Затем графовые паттерны могут быть выражены в подмножестве реляционной алгебры (а именно π, σ, ρ, ⋈). Если, например, одного триплета $G(s, p, o)$, представляющих граф, – то есть, таблица G с тремя столбцами s, p, o – и запрос на рис. 6 может быть выражен в реляционной алгебре:

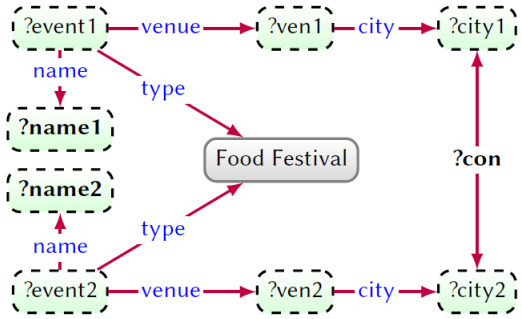
$$\pi_{ev,vn1,vn2}(\sigma_{p=\texttt{type}\wedge o=\texttt{Food Festival}\wedge p_1=p_2=\texttt{venue}}(\rho_{s/ev}(G \bowtie \rho_{p/p_1,o/vn1}(G) \bowtie \rho_{p/p_2,o/vn2}(G))))$$

где ⋈ обозначает естественное соединение, означающее, что равенство проверяется по парам столбцов с одинаковым именем в обеих таблицах (здесь соединение, таким образом, выполняется по предметному столбцу s). Результатом этого запроса является таблица со столбцом для каждой переменной: ev, $vn1$, $vn2$. Однако не все запросы, использующие π, σ, ρ и ⋈ на G, могут быть

выражены в виде графовых шаблонов; например, мы не можем выбрать, какие переменные проецировать в графовом шаблоне, а скорее должны проецировать все переменные, не фиксированные к константе.

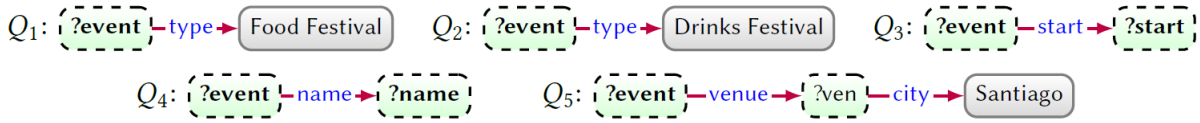
Графовые языки запросов, такие как SPARQL и Cypher, позволяют в полной мере использовать реляционные операторы над результатами графовых шаблонов, что приводит к возникновению сложных графовых шаблонов. На рис. 7 представлен пример сложного графового шаблона с проецируемыми переменными, выделенными жирным шрифтом, с выбором конкретных переменных для отображения в окончательных результатах. С точки зрения выразительности графовые паттерны с (неограниченной) проекцией этой формы приравниваются к конъюнктивным запросам на графах. На рис. 8 мы приводим еще один пример сложного графового паттерна, ищущего фестивали еды или напитков, которые не проводятся в Сантьяго, при необходимости возвращая их название и дату начала (если таковые имеются). Такие запросы, позволяющие в полной мере использовать реляционные операторы поверх графовых шаблонов, приравниваются к запросам первого порядка на графах. Сложные графовые паттерны могут привести к дублированию результатов; например, первый результат на рисунке 7 появляется дважды, так как ?city1 соответствует Arica, а ?city2 соответствует Viña del Mar в одном результате и наоборот в другом. Затем языки запросов предлагают две семантики: семантика bag сохраняет дубликаты в соответствии с множественностью базовых отображений, в то время как семантика set (обычно вызываемая с помощью отдельного ключевого слова) удаляет дубликаты из результатов.

Навигационные графовые паттерны. Ключевой особенностью, отличающей языки запросов графов, является возможность включать выражения путей в запросы. Путь выражения r это регулярное выражение, которое позволяет задать соответствия произвольной длины пути между двумя узлами, которое выражается как обычный путь запроса (x, r,y), где x и y могут быть переменными или константами (или даже тот же срок). Базовое выражение пути-это где r - константа (метка ребра). Кроме того, если r – выражение пути, то r-(обратное) 6 и $r*$ (звезда Клина: ноль или больше) также являются выражениями пути. Наконец, если $r1$ и $r2$-это путь выражения, тогда $r1 \mid r2$ (дизъюнкция) и $r1 \cdot r2$ (объединения) также являются путем выражения. Регулярные запросы пути затем могут быть оценены в соответствии с рядом различных семантик. Например, (Arica, bus*, ?city), вычисленные по графу на рис. 1, могут совпадать с путями на рис. 9. На самом деле, поскольку цикл присутствует, бесконечное число путей потенциально совпадают. По этой причине часто применяется ограниченная семантика, возвращающая только самые короткие пути или пути без повторяющихся узлов или ребер (как в случае Cypher). Вместо того, чтобы возвратить пути, другой вариант-вместо того, чтобы вернуть (конечного) множества пар узлов, соединенных в соответствии с путем (как и в случае с SPARQL 1.1). Регулярный путь запросов, может быть использован в графовой модели, чтобы выразить навигационные диаграммы моделей, как показано на рис. 10, который иллюстрирует запросе для поиска фестивалей еды в городах, куда можно добраться (рекурсивно) из Арика на автобусе или самолете. Кроме того, когда регулярные запросы путей и графовые шаблоны объединяются с такими операторами, как проекция, выделение, объединение, разность и необязательность, результат называется сложными навигационными графовыми шаблонами.



?name1	?con	?name2
Food Truck	bus	Food Truck
Food Truck	bus	Food Truck
Food Truck	bus	Ñam
Food Truck	flight	Ñam
Food Truck	flight	Ñam
Ñam	bus	Food Truck
Ñam	flight	Food Truck
Ñam	flight	Food Truck

Рис. 7. Конъюнктивный запрос (слева) с отображениями, генерируемыми над графом Рис. 1 (справа)



?event	?name	?start
EID16	Food Truck	

Рис. 8. Сложный графовый паттерн (Q) с отображениями, сгенерированными ($Q(G)$) над графом Рис. 1 (G)

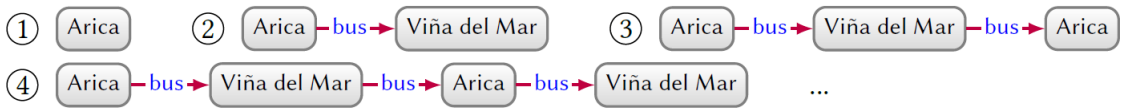
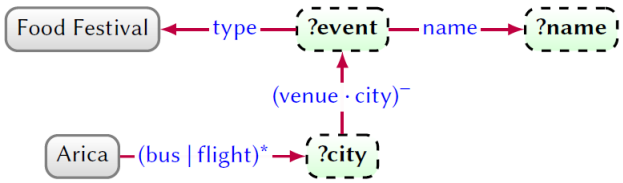


Рис. 9. Некоторые возможные пути сопоставления (Arica, bus*, ?city) на графике Рис. 1

Другие особенности. До сих пор мы обсуждали особенности, которые формируют практическую и теоретическую основу любого языка запросов для графов. Однако, конкретные языки запросов для графов могут поддерживать другие практические функции, такие как агрегирование (группировка, подсчет и т. д.), более сложные фильтры и тип данных операторов (например, извлечение диапазона запросов из даты), объединения для выполнения запросов удаленных графов через web, языки для обновления графов, поддержка семантического следования режимов и т. д.



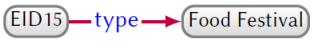
?event	?name	?city
EID15	Ñam	Santiago
EID16	Food Truck	Arica
EID16	Food Truck	Viña del Mar

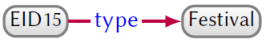
Рис. 10. Шаблон навигационного графа (слева) с отображениями, генерируемыми по графу Рис. 1 (справа)

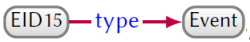
2. Схема, идентичность и контекст в графах знаний

Схема

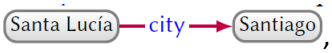
Одним из преимуществ моделирования данных в виде графа – по сравнению, например, с реляционной моделью – является возможность отказаться или отложить определение схемы. Однако при моделировании данных в виде графа, схемы могут использоваться для определения высокоуровневой структуры и/или семантики, которой следует или должен следовать граф. Мы обсуждаем три типа графовых схем: семантические, валидирующие и эмерджентные.

Семантическая схема позволяет определить значение высокоуровневых терминов (то есть словаря или терминологии), используемых в графе, что облегчает рассуждения о графах, использующих эти термины. Рассматривая, например, рисунок 1, мы можем заметить некоторые естественные группировки узлов, основанные на типах сущностей, к которым они относятся. Таким образом, мы можем решить определить классы для обозначения этих группировок, такие как событие, город и т. д. Фактически, на рис. 1 уже показаны три низкоуровневых класса – открытый рынок, продовольственный рынок, фестиваль напитков, – группирующие подобные объекты с маркированным типом ребром. Впоследствии мы можем наблюдать некоторые естественные отношения между некоторыми из этих классов, которые мы хотели бы уловить. На рис. 11 мы представляем иерархию классов для событий, в которых дети определяются как подклассы своих родителей, так что если мы найдем ребро 

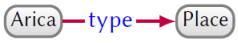
в нашем графе, мы также можем сделать вывод, что 

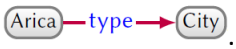
и 

.

Помимо классов, мы можем также захотеть определить семантику меток ребер, то есть свойств. Возвращаясь к Рис.1, мы можем считать, что свойства city и venue являются подсвойствами более общего свойства location , так что 

, , например, учитывая ребро, мы также можем сделать вывод, что 

мы также можем считать, например, что bus и flight являются подсвойствами более общего свойства Connections. Таким образом, свойства могут также образовывать иерархию. Мы можем далее определить область свойств, указывая класс(классы) сущностей для узлов, от которых простираются ребра с этим свойством; например, мы можем определить, что область соединений C является классом Place, таким образом, что, учитывая предыдущие отношения суб-свойств, мы могли бы заключить, что 

. И наоборот, мы можем определить диапазон свойств, указав класс (ы) сущностей для узлов, до которых простираются ребра с этим свойством; например, мы можем определить, что диапазон city - это класс City, предполагая, что 

.

Известным стандартом для определения семантической схемы для графов RDF является стандарт RDF Schema (RDFS), который позволяет определять подклассы, подсвойства, домены и диапазоны среди классов и свойств, используемых в графе RDF, где такие определения могут быть сериализованы в виде графа. Мы проиллюстрируем семантику этих признаков в таблице 2 и приведем конкретный пример определений на рис.12 для выборки терминов, используемых в текущем примере. Затем эти определения могут быть встроены в граф данных. В более общем плане семантика терминов, используемых в графе, может быть определена гораздо глубже, чем показано здесь, что поддерживается стандартом языка веб-онтологий (OWL) для графов RDF.

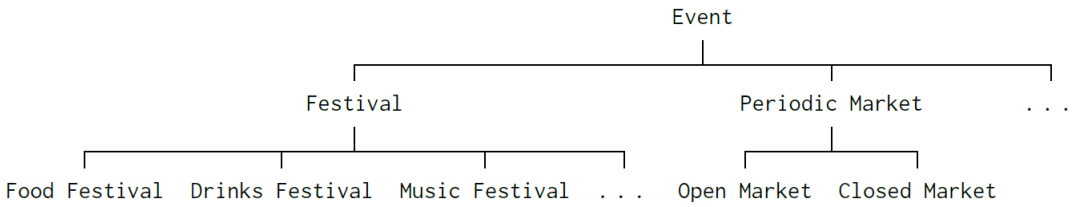


Рис. 11. Пример иерархии классов для события

Таблица 2. Определение признаков подкласса, подсвойства, предметной области и диапазона в семантических схемах

Feature	Definition	Condition	Example
SUBCLASS	(c) –subc. of→ (d)	(x) –type→ (c) implies (x) –type→ (d)	City –subc. of→ Place
SUBPROPERTY	(p) –subp. of→ (q)	(x) –p→ (y) implies (x) –q→ (y)	venue –subp. of→ location
DOMAIN	(p) –domain→ (c)	(x) –p→ (y) implies (x) –type→ (c)	venue –domain→ Event
RANGE	(p) –range→ (c)	(x) –p→ (y) implies (y) –type→ (c)	venue –range→ Venue

Семантические схемы обычно определяются для неполных графовых данных, где отсутствие ребра между двумя узлами, например Arica –type→ City, не означает, что отношение не выполняется в реальном мире. Поэтому из графа на Рис. 1 мы не можем предположить, что между Винья-дель-Мар и Арикой нет рейса. В противоположность этому, если бы было принято допущение о замкнутом мире (CWA) – как это имеет место во многих классических системах баз данных, – то предполагалось бы, что граф данных является полным описанием мира, что позволяет с уверенностью утверждать, что никакого полета между двумя городами не существует. Системы, которые не принимают CWA, как говорят, принимают допущение открытого мира (OWA). Следствием CWA является то, что добавление ребра к графу данных может противоречить тому, что ранее считалось ложным (из-за отсутствия информации), тогда как в случае OWA утверждение, которое доказано ложным, продолжает оставаться ложным с добавлением большего количества ребер. Учитывая наш текущий пример, было бы неразумно предполагать, что туристская организация обладает полным знанием всего, что может быть описано в ее графе знаний. Однако неудобно, если система не может определенно ответить “да” или “нет” на такие вопросы, как “есть ли рейс между Арикой и Винья-дель-Мар?”, особенно когда организация уверена, что она полностью знает рейсы. Компромиссом между OWA и CWA является локальное предположение о замкнутом мире (LCWA), где части графа данных считаются полными.

Валидирующая схема. Когда графы используются для представления разнообразных, неполных данных в крупном масштабе, OWA является наиболее подходящим выбором для семантики по умолчанию. Но в некоторых сценариях мы можем гарантировать, что наш граф данных – или его отдельные части – в некотором смысле “полны”.

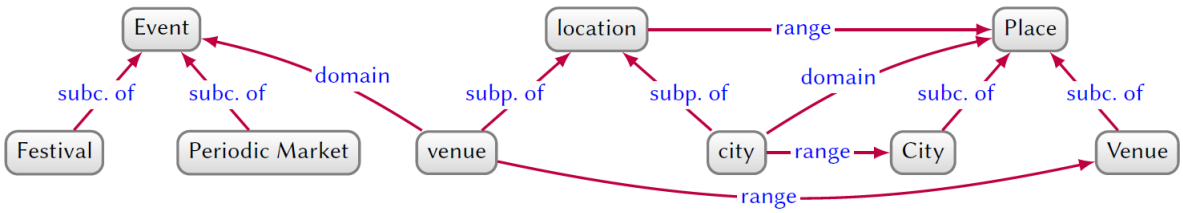


Рис. 12. Пример графа схемы, описывающего подклассы, подсвойства, домены и диапазоны

Возвращаясь к Рис. 1, например, мы можем пожелать убедиться, что все события имеют по крайней мере название, Место проведения, дату начала и дату окончания, чтобы приложения, использующие эти данные – например, те, которые отправляют уведомления о событиях пользователям – могли обеспечить себе минимальную требуемую информацию. Кроме того, мы можем пожелать убедиться, что город события заявлен как город (вместо того, чтобы делать вывод, что это город). Мы можем определить такие ограничения в проверяющей схеме и проверить граф данных по отношению к результирующей схеме, перечисляя нарушения ограничений (если таковые имеются). Таким образом, в то время как семантические схемы позволяют выводить новые графовые данные, проверяющие схемы позволяют проверять существующие графовые данные. Стандартный способ определения схемы проверки для графов-это использование схем. Схема нацелена на набор узлов в графе данных и задает ограничения на эти узлы. Цель схемы может быть определена многими способами, такими как таргетирование всех экземпляров класса, домена или диапазона свойств, результата запроса, узлов, связанных с целью другой схемы заданным свойством, и т. д. Затем ограничения могут быть определены на целевых узлах, например, чтобы ограничить количество или типы значений, принятых для данного свойства. Граф схемы формируется из набора взаимосвязанных схем. Графы схем могут быть изображены в виде UML-подобных диаграмм классов, рисунок 13 иллюстрирует пример графа схем на основе рисунка 1, определяющего ограничения для четырех взаимосвязанных схем. Каждая схема – обозначенная рамкой, как Place, Event, и т. д. – связана с набором ограничений. Узлы соответствуют схеме тогда, когда они удовлетворяют всем ограничениям, определенным на схеме. Внутри каждого блока формы помещаются ограничения на число (например, [1..*] обозначает один-ко-многим, [1..1] обозначает ровно один, и т. д.) и типы (например, строка, DateTime ит. д.) узлов, соответствующих узлам может относиться к константам (например, имя, старт и др.). Другой вариант состоит в установлении ограничений на количество узлов, соответствующей конкретной форме, что соответствующий узел может относиться к константе (и, следовательно, создания соединений между формами), например, Event —venue 1..*→ Venue обозначает, что соответствующие узлы Event должны относиться как минимум один узел со свойством место, которое соответствует Venue форме. Схемы могут наследовать ограничения родительских схем, обозначаемых соединителем △ connector – , как в случае City и Venue чьи соответствующие узлы также должны соответствовать Place форме. Учитывая форму и целевой узел, это можно проверить, если узел соответствует этой форме или нет, для чего может потребоваться проверка соответствия от других узлов, например, узел EID15 соответствует Event форме не только исходя из его локальных свойств, а также исходя соответствия В Santa Lucía К Venue и Santiago в City соответствие зависимостей также может быть рекурсивными, где соответствие Santiago в City требует, чтобы она соответствовала Place , которая требует Viña del Mar и Arica соответствовать Place , и так далее. И наоборот, EID16 не соответствует Event , так как не имеет начальных и конечных свойств, требуемых графом схемы примера. При объявлении схем разработчик данных может не знать заранее весь набор свойств , которые могут иметь некоторые узлы. Открытая форма позволяет узлу иметь дополнительные свойства , не заданные формой, в то время как закрытая форма этого не делает. Например, если мы добавим ребро Santiago -founder→ Pedro de Valdivia к графу, представленному на рис. 1, то Santiago оно будет соответствовать the City схеме только в том случае, если эта схема определена как открытая (поскольку схема не упоминает об ounder).

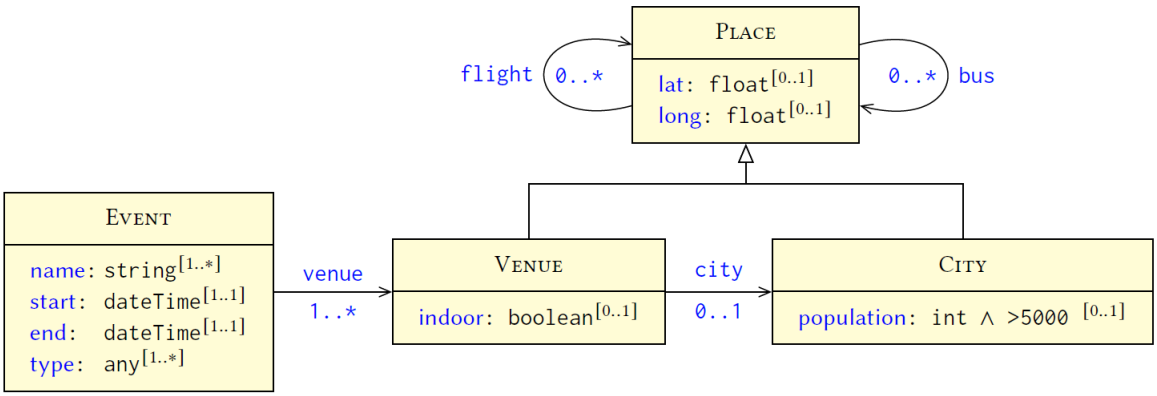


Рис. 13. Пример графа схемы, изображенной в виде UML- диаграммы классов

Практические языки для схем часто поддерживают дополнительные логические функции, такие как конъюнкция (и), дизъюнкция (или) и отрицание (не) схем; например, мы можем сказать, что все значения мест должны соответствовать форме Venue *and* (*not* City), делая явным, что места в графе данных не должны быть непосредственно заданы как города. Однако языки форм, которые свободно сочетают рекурсию и отрицание, могут привести к семантическим проблемам, в зависимости от того, как определяется их семантика. Для иллюстрации рассмотрим следующий случай, вдохновленный парадоксом Барбера, включающий форму , соответствующие узлы которой бреют по крайней мере один узел, соответствующий Barber которому соответствующие узлы бреют по крайней мере один узел, соответствующий Person *and* (*not* Barber) сейчас, заданный (только) Bob –shave→ Bob с Bob conforming to Person, does Bob соответствием соответствует ? Barber Если да – если Bob соответствует Barber – то Bob нарушает ограничение, не Брея по крайней мере один узел , соответствующий Person *and* (*not* Barber), если нет – если Bob не соответствует Barber – то Bob удовлетворяет the Barber ограничению, Брея такой узел. Семантика, чтобы избежать таких парадоксальных ситуаций , была предложена на основе стратификации, частичных назначений и устойчивых моделей. Хотя валидирующие и семантические схемы служат различным целям, они могут дополнять друг друга. В частности, валидирующая схема может учитывать семантическую схему, такую, что, например, валидация применяется к графу данных, включая выводы. Взяв , например, иерархию классов на рис. 11 и граф схем на рис. 13, мы можем определить цель Event схемы как узлы, имеющие тип Event (класс). Если мы сначала применим вывод относительно иерархии классов семантической схемы, Event форма теперь будет нацелена на EID15 и EID16. Наличие семантической схемы может, однако, потребовать адаптации проверяющей схемы. принимая во внимание, например, вышеупомянутую иерархию классов, потребуется определить ослабленную мощность на свойстве type. Открытые формы также может быть предпочтительным в таких случаях, а не перечисление ограничения на возможные свойства, которые могут быть установлены на узле. Две формы языков для схем недавно вышли для RDF графов: Shape Expressions (ShEx), публикуется на правах консорциума W3C Community Group Report; и SHACL (Shapes Constraint Language), опубликованных как в W3C Recommendations. Эти языки поддерживают обсуждаемые функции (и многое другое) и были приняты для проверки графов в ряде областей, связанных с здравоохранением, научной литературой, пространственными данными и другими.

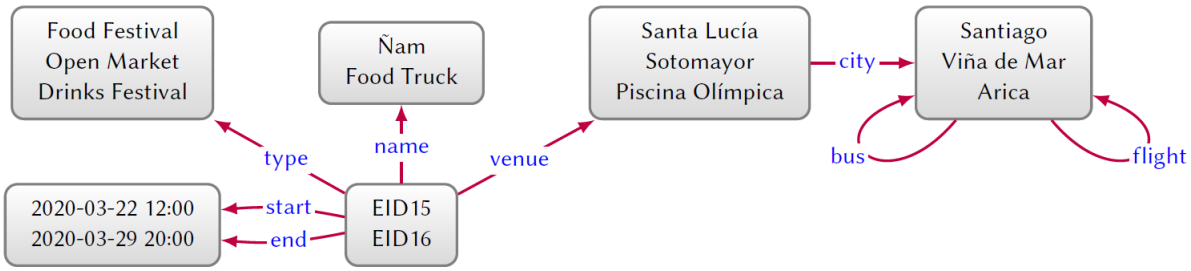


Рис. 14. Пример факторного графа, имитирующего граф данных на рис. 1

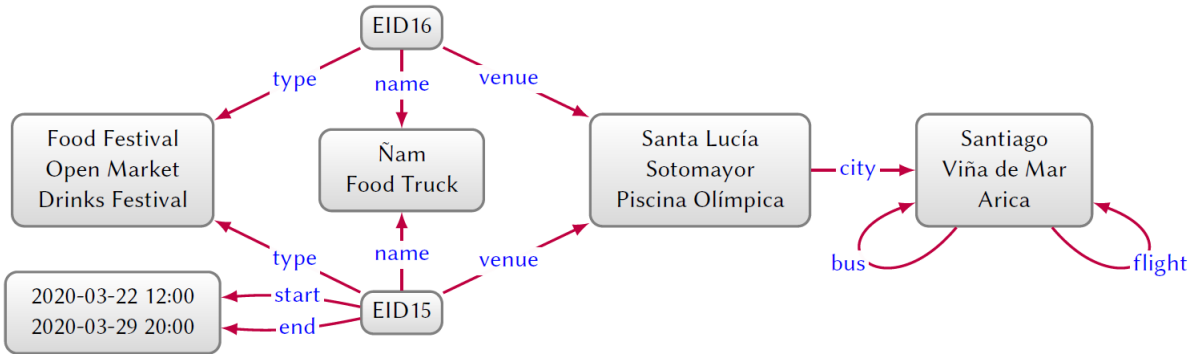


Рис. 15. Пример факторного графа с двойным сходством, имитирующего граф данных на Рис. 1

Производная схема. Как семантическая, так и проверяющая схемы требуют, чтобы эксперт в предметной области явно указывал определения и ограничения. Однако граф данных часто демонстрирует скрытые структуры, которые могут быть автоматически извлечены как производная схема. Структура, часто используемая для определения производной схемы, - это фактор-графы, которые разделяют группы узлов в графе данных в соответствии с некоторым отношением эквивалентности, сохраняя при этом некоторые структурные свойства графа. Взяв рисунок 1, мы можем интуитивно разделить различные типы узлов в зависимости от их контекста, такие как узлы событий, которые связаны с узлами места проведения, которые, в свою очередь, связаны с узлами города и т. д. Чтобы описать структуру графа, мы могли бы рассмотреть шесть разделов узлов: событие, имя, место проведения, класс, дата-время, город. На практике эти разделы могут быть вычислены на основе класса или формы узла. Объединение узлов каждого раздела в один узел с сохранением ребер приводит к фактор-графу, показанному на рисунке 14: узлы этого фактор-графа являются разделами узлов из графа данных и ребра. $(X) - y \rightarrow (Z)$ находится в фактор- графе тогда и только тогда, когда существуют $x \in X$ и $z \in Z$ такие, что $(x) - y \rightarrow (z)$ S в графе данных. Существует много способов определения фактор-графов, зависящих не только от того, как разбиты узлы, но и от того, как определены ребра. Различные факторные графы могут предоставлять различные гарантии в отношении структуры, которую они сохраняют. Формально, можно сказать, что каждый фвктор-граф моделирует свой входной граф (на основе имитационного моделирования отношений между исходными узлами и фактор-узлами), что означает, что для всех $x \in X$ с x входной узел и X фактор-узел, если $(x) - y \rightarrow (z)$ ребро в графе данных, тогда там должно существовать ребро $(X) - y \rightarrow (Z)$ в фактор- графе, что $z \in Z$; например, фактор-граф на рис. 14 моделирует данные, графа на Рис. 1. Однако этот фактор-граф, по-видимому, предполагает (например), что (EID16) мог бы содержать в графе данных дату начала и окончания, когда как это не так. Более сильное понятие структурной сохранности дается двойным сходством, которое в этом случае дополнительно требует, чтобы, если $(X) - y \rightarrow (Z)$ это ребро в фактор-графе, то для всех $x \in X$, должен существовать $z \in Z$ такой, что $(x) - y \rightarrow (z)$ в графе данных; не удовлетворенный (EID16) в фактор-графе на рис. 14, которые не имеют исходящего ребра обозначены как начало и конец исходных данных диаграммы. На рис. 15

показана версия фактор-графа с двойным сходством, с разделенной секцией событий на два узла, отражающих их различные исходящие ребра. Интересным свойством двойного сходства является то, что оно сохраняет прямые пути: учитывая выражение пути r без инверсий и два графа с двойнвм сходством, r будет соответствовать пути в одном графе тогда и только тогда, когда он соответствует соответствующему пути в другом графе с двойным сходством. Можно проверить, например, что путь совпадает $x \xrightarrow{\text{city}\cdot(\text{flight}|\text{bus})^*} z$ с рисунком 1 тогда и только тогда, когда есть путь , совпадающий $X \xrightarrow{\text{city}\cdot(\text{flight}|\text{bus})^*} Z$ с рисунком 15, такой, что $x \in X$ и$z \in Z$.

Существует много способов определения фактор-графов в зависимости от отношения эквивалентности, разделения узлов. Кроме того, есть много способов, с помощью которых можно определить другие подобные или графы с двойным сходством, в зависимости от отношения моделирования, которое сохраняет структуру графа данных . Такие методы направлены на обобщение графа данных в топологию более высокого уровня . Чтобы уменьшить объем памяти для факторного графа, на практике узлы могут быть помечены меткой раздела и/или меткой высокого уровня (например, событие, город) для раздела, а не хранить метки всех узлов в разделе. Также были предложены различные другие формы производной схемы, не основанной на структуре фактор-графа ; примеры включают производные схемы, основанные на реляционных таблицах, формальном анализе концепций и т. д. производные схемы могут использоваться для обеспечения понятного человеку обзора графа данных, для помощи в определении семантической или проверяющей схемы, для оптимизации индексирования и запроса графа, для руководства интеграцией графов данных и т. д.

Идентичность

На рис. 1 мы используем узлы типа , но к какому из них относится этот узел? Santiago Имеем ли мы в виду Сантьяго-де-Чили, Сантьяго-де-Куба, Сантьяго-де-Компостела, или, возможно, мы имеем в виду инди -рок-группу Сантьяго? Основываясь на таких ребрах , как Santa Lucía —city→ Santiago , мы можем сделать вывод, что это один из трех упомянутых городов (не рок-группа), и основываясь на том, что график описывает туристические достопримечательности в Чили, мы можем далее сделать вывод, что он относится к Сантьяго-де-Чили. Без дальнейших подробностей, однако, расплывчатые узлы этой формы могут полагаться на эвристику, склонные к ошибкам в более сложных случаях. Чтобы избежать такой неоднозначности, во-первых, мы можем использовать глобально уникальные идентификаторы , чтобы избежать конфликтов имен, когда граф знаний расширяется внешними данными, а во-вторых, мы можем добавить внешние идентификационные ссылки, чтобы устранить неоднозначность узла по отношению к внешнему источнику.

Постоянные идентификаторы. Предположим, мы хотели сравнить туризм в Чили и на Кубе, и мы получили соответствующий граф знаний для Кубы. Одним из преимуществ использования графов для моделирования данных является то, что мы можем объединить два графа, взяв их объединение. Однако, как показано на рис. 16, использование неоднозначного узла like Santiago может привести к конфликту имен: узел ссылается на два разных реальных города на обоих графах, где объединенный граф указывает, что Сантьяго-это город как в Чили, так и на Кубе (а не два разных города).8 чтобы избежать таких столкновений, можно создать долговременные постоянные идентификаторы (PID) для уникальной идентификации объекта. Известные примеры PID-схем включают цифровые идентификаторы объектов (doi) для статей, идентификаторы ORCID для авторов, международные стандартные книжные номера (ISBNs) для книг, коды Альфа-2 для округов и многое другое. В контексте семантической сети модель данных RDF идет еще дальше и рекомендует использовать глобальные веб-идентификаторы для узлов и меток ребер. Однако вместо принятия единых локаторов ресурсов (url), используемых для определения местоположения информационных ресурсов, таких как веб-страницы, RDF 1.1 предлагает

использовать Интернационализированные идентификаторы ресурсов (IRIs) для идентификации неинформационных ресурсов, таких как города или события.

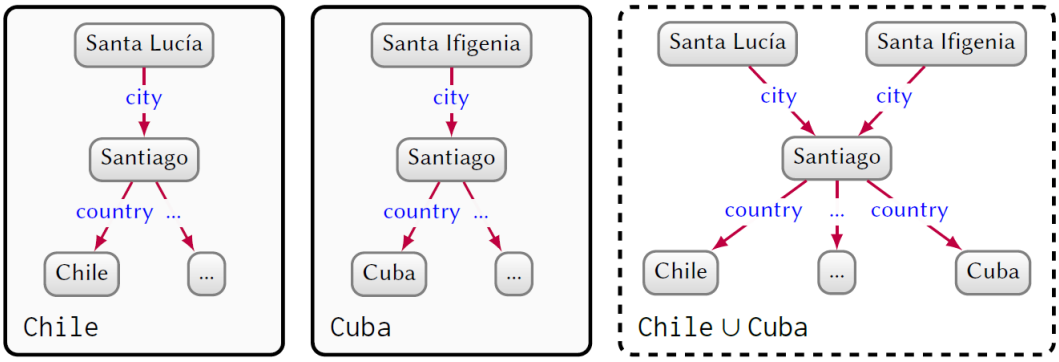
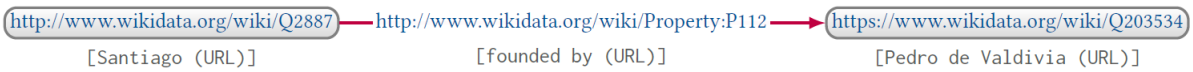
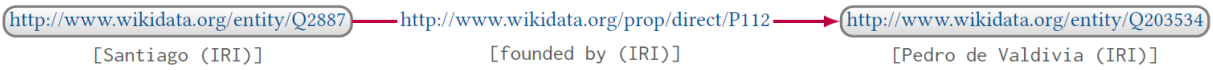


Рис. 16. Результат слияния двух графов с неоднозначными локальными идентификаторами

Так, например, в RDF – представлении Викиданных – графе знаний, предложенном в дополнение к Википедии, в то время как URL https://www.wikidata.org/wiki/Q2887 -адрес относится к веб-странице, которая может быть загружена в браузер, предоставляющий читаемые человеком метаданные о Сантьяго, IRI http://www.wikidata.org/entity/Q2887 относится к самому городу. Различение идентификаторов для обоих ресурсов (веб-страницы и самого города) позволяет избежать конфликтов именования; например, если мы используем URL-адрес для идентификации как веб-страницы, так и города, мы можем получить ребро в нашем графике, например (с читаемыми метками ниже ребра):



Такая грань оставляет неясность: был ли Педро де Вальдивия основателем веб-страницы или города? Использование IRIs для сущностей, отличных от URL-адресов веб-страниц, которые их описывают, позволяет избежать таких неоднозначных случаев, когда Викиданные, таким образом, скорее определяют предыдущее ребро следующим образом:



использование IRIs для города, человека и основателя, отличного от веб-страниц, описывающих их. Эти Wikidata идентификаторы Викиданных используют префикс http://www.wikidata.org/entity/ для сущностей и префикса http://www.wikidata.org/prop/direct/ для отношений. Такие префиксы известны как пространства имен и часто сокращаются префиксными строками, такими как wd: или wdt:, где последняя тройка может быть записана более сжато, используя такие аббревиатуры, как wd:Q2887 wdt:P112 wd:Q203534.

Если HTTP IRI используются для идентификации сущностей графа, при поиске IRI (через HTTP) веб-сервер может вернуть (или перенаправить) описание этой сущности в таких форматах, как RDF. Это также позволяет графам RDF связываться со связанными сущностями, описанными во внешних графах RDF через Интернет, что приводит к появлению связанных данных. Хотя HTTP IRI предлагают гибкий и мощный механизм для выдачи глобальных идентификаторов в Интернете, они не обязательно являются постоянными: веб-сайты могут отключаться, ресурсы, описанные в данном месте, могут изменяться и т. Д. Для повышения устойчивости таких идентификаторов, Службы постоянного URL (PURL) предлагают перенаправления с центрального сервера в определенное место, где PURL может быть перенаправлен в новое место, если необходимо, изменяя адрес документа без изменения его идентификатора. Затем устойчивость HTTP IRI может быть улучшена за счет использования пространств имен, определенных через службы PURL

Внешние идентификационные ссылки. Предположим, что Совет по туризму решает определить пространство имен chile: с помощью IRI, такого как http://turismo.cl/entity/ на веб-сервере, которым они управляют, позволяя таким узлам, как chile:Santiago – ярлык для IRI http://turismo.cl/entity/Santiago – просматриваться через сеть. Хотя использование такой схемы именования помогает избежать конфликтов именования, использование IRIs не обязательно помогает обосновать идентичность ресурса. Например, внешний граф географических знаний может присвоить одному и тому же городу IRI geo:SantiagoDeChile в своем собственном пространстве имен, где у нас нет прямого способа узнать, что эти два идентификатора относятся к одному и тому же городу. Если мы объединим два графа знаний, то получим два разных узла для одного и того же города.

Существует несколько способов обосновать идентичность сущности. Первый-связать объект с однозначно идентифицирующей информацией в графе, такой как его географические координаты, почтовый индекс, год его основания и т. д. Каждый дополнительный фрагмент информации устраняет двусмысленность в отношении того, о каком городе идет речь, предоставляя (например) больше возможностей для сопоставления города с его аналогом во внешних источниках. Второй вариант-использовать ссылки на идентичность, чтобы указать, что локальная сущность имеет ту же идентичность, что и другая внешняя сущность, найденная во внешнем источнике; экземпляр этой концепции можно найти в стандарте OWL, который определяет свойство owl:sameAs, связывающее идентичные сущности. Используя это свойство, мы могли бы указать ребро chile:Santiago -owl:sameAs→ geo:SantiagoDeChile в нашем графе RDF, установив таким образом тождественную связь между соответствующими узлами в обоих графах. Семантика owl:sameAs, определенная стандартом OWL, затем позволяет нам объединить данные для обоих узлов.

Типы данных. Рассмотрим две даты-времени слева от рисунка 1: как мы должны назначить этим узлам постоянные/глобальные идентификаторы? Интуитивно было бы нецелесообразно, например, назначать IRIs этим узлам, поскольку их синтаксическая форма говорит нам, к чему они относятся: конкретные даты и время в марте 2020 года. Эта синтаксическая форма далее распознается машиной, а это означает, что с помощью соответствующего программного обеспечения мы можем упорядочить такие значения в порядке возрастания или убывания, извлечь год и т. д. большинство практических моделей данных для графов позволяют определить узлы, которые являются значениями типа данных. RDF использует, среди прочего, типы данных XML-схемы (XSD), где узел типа данных задается в виде пары (l,d), где l - лексическая строка, например "2020-03-29T20:00:00", а d - IRI, обозначающий тип данных, например xsd:dateTime. Затем узел обозначается "2020-03-29T20:00:00"^^xsd:dateTime типом данных узлы в RDF называются литералами и не имеют исходящих ребер. Другие типы данных , обычно используемые в данных RDF , включают xsd:string, xsd:integer, xsd:decimal, xsd:booleanи т. д. В случае, если тип данных опущен, предполагается, что значение имеет тип xsd:string. Приложения , построенные поверх RDF, могут затем распознавать эти типы данных, анализировать их в объекты типов данных и применять проверки равенства , нормализацию, упорядочение, преобразования, приведение в соответствие со своим стандартным определением. В контексте графов свойств Neo4j также определяет набор внутренних типов данных для значений свойств, который включает числа, строки, логические значения , пространственные точки и временные значения.

Лексикализация. *Глобальные* идентификаторы сущностей иногда имеют интерпретируемую человеком форму, например chile:Santiago , но сами строки идентификаторов не несут никакого формального семантического значения. В других случаях используемые идентификаторы могут быть не интерпретируемы человеком по замыслу. В Викиданных, например, Сантьяго-де-Чили обозначается как wd:Q2887, где такая схема имеет то преимущество, что обеспечивает лучшую сохраняемость и не предвзятость к определенному человеческому языку. Например,

идентификатор Викиданных для Eswatini (wd:Q1050) не был затронут, когда страна сменила свое название со Свазиленда, и не требует выбора между языками для создания (более удобочитаемых) IRIs, таких как wd:Eswatini (английский), wd:eSwatini (Свазиленд), wd:Esuatini (испанский) и т. д.

Поскольку идентификаторы могут быть произвольными, обычно добавляют ребра, которые обеспечивают интерпретируемую человеком метку для узлов , например wd:Q2887 -rdfs:label→ "Santiago", указывающую, как люди могут относиться к предметному узлу лингвистически. Лингвистическая информация этой формы играет важную роль в обосновании знаний таким образом, что пользователи могут более четко определить, на какую сущность реального мира конкретный узел в графе знаний фактически ссылается; это также позволяет перекрестным ссылкам на метки сущностей с текстовыми данными находить, например, документы, которые потенциально говорят о данной сущности.

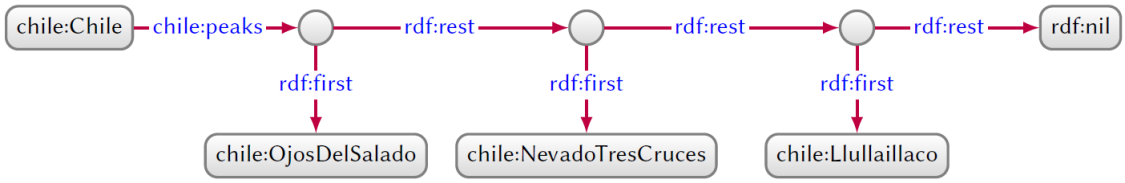
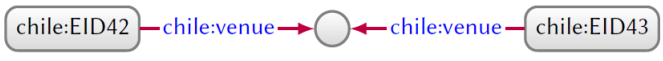


Рис. 17. Список RDF, представляющий три самые большие вершины Чили, в порядке убывания

Метки могут быть дополнены псевдонимами (например, wd:Q2887 -skos:altLabel→ "Santiago de Chile") или комментариями (например wd:Q2887 -rdfs:comment→ "Santiago is the capital of Chile"), чтобы еще больше помочь обосновать идентичность узла.

Такие узлы "Santiago" обозначают строковые литералы, а не идентификатор. В зависимости от конкретной Графовой модели такие литеральные узлы также могут быть определены как пара (s,l), где s обозначает строку и l код языка; в RDF , например, мы можем указать chile:City -rdfs:label→ "City"@en и chile:City -rdfs:label→ "Ciudad"@es т. д. , указывая метки для узла на разных языках. В других моделях соответствующий язык может быть скорее задан, например, с помощью метаданных на ребре. Графы знаний с человеко-интерпретируемыми метками, псевдонимами, комментариями и т. д. (на разных языках) иногда называют (многоязычными) лексикализованными графами знаний.

Экзистенциальные узлы. При моделировании неполной информации мы можем в некоторых случаях знать, что в графе должен существовать определенный узел с определенными отношениями к другим узлам, но не имея возможности идентифицировать соответствующий узел. Например, у нас может быть два совместных мероприятия chile:EID42 , chile:EID43 Место проведения которых еще не объявлено. Один из вариантов заключается в том, чтобы просто опустить рebра места проведения, и в этом случае мы теряем информацию о том, что эти события имеют место и что оба события имеют одно и то же место. Другим вариантом может быть создание нового IRI, представляющего место проведения, но семантически это становится неотличимым от известного места проведения. Поэтому некоторые графовые модели позволяют использовать экзистенциальные узлы, представленные здесь в виде пустого круга:



Это отображение означает, что существует общая площадка для chile:EID42 и chile:EID43 без идентификации. Экзистенциальные узлы поддерживаются в RDF как пустые узлы, которые также обычно используются для поддержки моделирования сложных элементов в графах, таких как

списки RDF. На рис. 17 приведен пример списка RDF, который использует пустые узлы в структуре связанного списка для кодирования порядка. Хотя экзистенциальные узлы могут быть удобными, их наличие может усложнить операции над графами, например, решить, имеют ли два графа данных одинаковую структуру по модулю экзистенциальных узлов. Поэтому были предложены методы сколемизации экзистенциальных узлов в графах – замены их каноническими метками. Другие авторы скорее призывают минимизировать использование таких узлов в графовых данных.

Контекст

Многие (а возможно, и все) факты, представленные на графе данных на рис. 1, могут считаться истинными в определенном контексте. Что касается временного контекста, Santiago то город существовал только с 1541 года, полеты из Arica в Santiago начались в 1956 году и т. д. Что касается географического контекста, то граф описывает события в Чили. Что касается происхождения, то данные, относящиеся к EID15 , были взяты с веб –страницы Nam 4 января 2020 года – и, таким образом, считаются правдивыми в отношении нее. Можно использовать и другие формы контекста. Мы можем далее комбинировать контексты, например указать, что Arica это чилийский город (географический) с 1883 года (временный) в соответствии с Анконским договором (происхождение).

Под контекстом мы здесь подразумеваем сферу истинности и, таким образом, говорим о контексте, в котором некоторые данные считаются истинными. Граф на рис. 1 оставляет большую часть своего контекста неявным. Однако представленный контекст может позволить интерпретировать данные с разных точек зрения, например, понять, что было правдой в 2016 году, что было правдой, исключая веб-страницы, позже обнаружившие ложные данные, и т. д. Как видно из предыдущих примеров, контекст графовых данных может рассматриваться на разных уровнях: на отдельных узлах, отдельных ребрах или множествах ребер (подграфах). Теперь мы обсудим различные репрезентации, посредством которых контекст может быть определен на различных уровнях.

Прямое представительство. Первый способ представления контекста-рассматривать его как данные, ничем не отличающиеся от других. Например, даты события EID15 на Рис. 1 можно рассматривать как форму временного контекста, указывающего временную область, в пределах которой такие ребра считаются EID15 —venue→ Santa Lucía истинными. Другой вариант-изменить отношение, представленное ребром , например Santiago —flight→ Arica, в узел, как показано на рис. 3, что позволяет присвоить этому отношению дополнительный контекст. В то время как в этих примерах контекст представлен специальным образом, был предложен ряд спецификаций для представления контекста в виде данных более стандартным способом. Было предложено несколько спецификаций для представления контекста в виде данных более стандартным способом. Одним из примеров является онтология времени, которая определяет, как временные объекты, интервалы, время – и отношения между ними такие, как раньше, перекрытия и т. д.– может быть описан в графах RDF в интероперабельном виде. Другой пример - модели данных, которые указывают, как происхождение данных может быть описано в RDF графе, где объекты (например, графы, узлы, физической документа) являются производными от других объектов, не создается и/или используется (например, добыча, авторство), и относятся к агентам (например, люди, программного обеспечения, организации).

Овеществление. Часто мы можем захотеть непосредственно определить контекст самих ребер; например, мы можем захотеть заявить, что ребро Santiago —flight→ Arica действительно с 1956 года. Хотя мы могли бы использовать паттерн превращения ребра в узел – как показано на рис. 3 – для непосредственного представления такого контекста, другой вариант-использовать овеществление, которое позволяет делать утверждения об утверждениях общим способом (или, в случае графа, для

определения ребер о ребрах). На рис. 18 мы представляем три формы овеществления, которые могут быть использованы для моделирования временного контекста на вышеупомянутом ребре внутри ориентированного графа с именованными ребрами используем e для обозначения произвольного идентификатора, представляющего само ребро, с которым может быть связана контекстуальная информация. В отличие от прямого представления, e представляет собой узел, а не ребро. Овеществление RDF (рис. 18а) определяет новый узел e для представления ребра и соединяет его с исходным узлом (через субъект), целевым узлом (через объект) и меткой ребра (через предикат) ребра. Напротив, n-арные отношения (рис. 18б) соединяют исходный узел ребра непосредственно с реберным узлом e с меткой ребра; целевой узел ребра затем соединяется с e (через значение). Наконец, свойства синглтона (рис.18С) скорее используют e в качестве метки ребра, соединяя его с узлом, указывающим исходную метку ребра (через синглтон). В литературе были предложены и другие формы овеществления. В общем, овеществленное ребро не утверждает то ребро, которое оно овеществляет; например, мы можем овеществить ребро, чтобы заявить, что оно больше не является действительным.

Представление с увеличением числа аргументов. В качестве альтернативы овеществлению мы можем скорее использовать представления более высокой точности для моделирования контекста. Снова беря ребро Santiago —flight→ Arica , рисунок 19 иллюстрирует три высших представления временного контекста. Сначала мы можем использовать именованный граф (рис. 19а), чтобы содержать ребро, а затем определить временной контекст для имени графа. Во-вторых, мы можем использовать граф свойств (рис. 19б), где временной контекст определяется как атрибут на ребре. В-третьих, мы можем использовать RDF (рис. 19С): расширение RDF, которое позволяет определять ребра как узлы. Среди этих вариантов наиболее гибким является представление именованного графа, где мы можем назначить контекст сразу нескольким ребрам, поместив их в один именованный граф; например, мы можем добавить больше ребер к именованному графу на рис. 19а, которые также действительны с 1956 года. Наименее гибкий вариант-RDF, который при отсутствии идентификатора ребра не позволяет назначать ребру различные группы контекстных значений; например, учитывая ребро Chile —president→ M. Bachelet если мы добавим четыре контекстных значения к этому ребру, чтобы заявить, что оно было действительным с 2006 по 2010 год и действительным с 2014 по 2018 год, мы не сможем объединить значения в пары, но, возможно, придется создать узел для представления разных представлений (в других моделях, мы могли бы использовать два именованных графа или идентификаторы ребер).

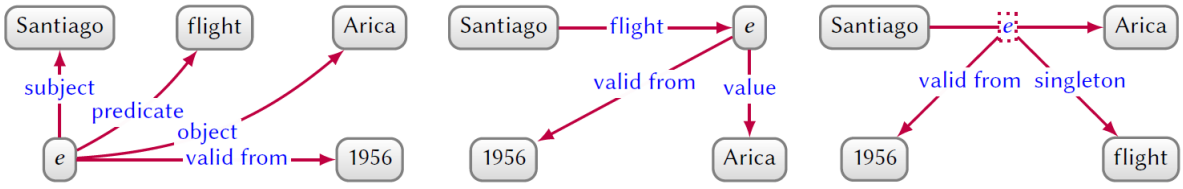


Рис. 18. Три представления временного контекста на ребре в направленном графе с именованными ребрами

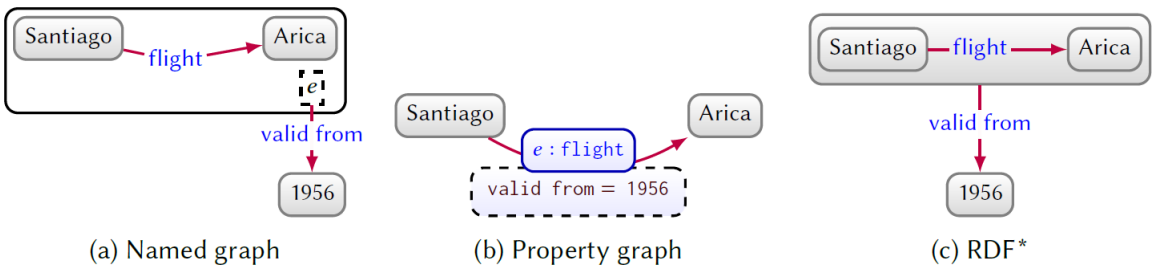


Рис. 19. Три представления с увеличением числа аргументов временного контекста для ребра

Аннотации. До сих пор мы обсуждали представление контекста в графе, но мы не говорили об автоматизированных механизмах рассуждения о контексте; например, если существуют только сезонные летние рейсы из Santiago в Arica, мы можем пожелать найти другие маршруты из Сантьяго для зимних событий, происходящих внутри Arica. В то время как даты автобусов, рейсов и т. д. могут быть представлены непосредственно на графе или с помощью овеществления, написание запроса для ручного пересечения соответствующих временных контекстов будет утомительным – или даже вообще невозможным. Другой альтернативой является рассмотрение аннотаций, которые предоставляют математические определения контекстной области и ключевые операции, возможные в этой области, которые затем могут быть применены автоматически. Некоторые аннотации моделируют определенную контекстуальную область; например, временную RDF позволяет аннотировать ребра с временными интервалами, например Chile —president [2006, 2010]→ M. Bachelet, в то время как нечеткий RDF позволяет аннотировать ребра со степенью истинности такой как Santiago —climate 0.8→ Semi-Arid, указывая, что это более или менее верно-со степенью 0,8-что Сантьяго имеет полузасушливый климат. Другие формы аннотации являются доменно-независимыми; например, аннотированный RDF позволяет представлять различные формы контекста, моделируемые как полукольца: алгебраические структуры , состоящие из значений области (например, временные интервалы, нечеткие значения и т. д.) и два основных оператора для объединения значений области: meet и join. Мы приводим пример на рис. 20, где G аннотируется значениями из упрощенной временной области, состоящей из наборов целых чисел (1-365), представляющих дни года. Запрос Q затем запрашивает рейсы из Сантьяго в города с событиями; этот запрос будет проверять и возвращать аннотацию, отражающую временную действительность каждого ответа.

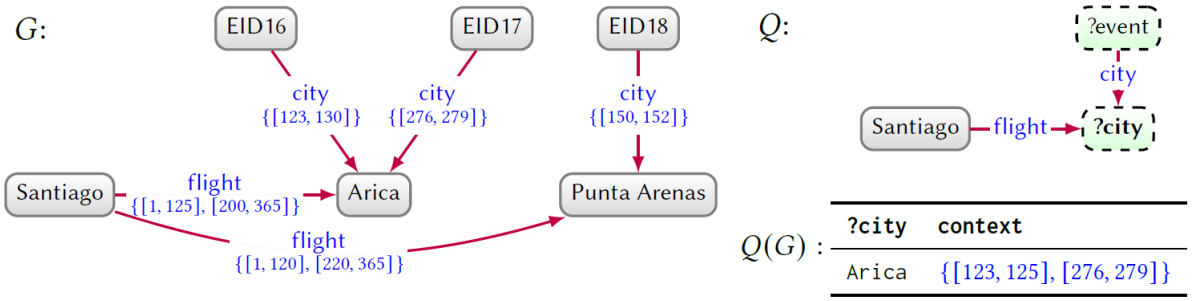


Рис. 20. Пример запроса на временном аннотированном графе

Чтобы получить эти ответы, мы сначала требуем применения конъюнкции аннотаций на совместимых ребрах полета и города, применяя оператор meet для вычисления аннотации, для которой удерживаются оба ребра. Самый естественный способ определения в нашем случае это, пересечение множеств дней, где, например, пересечение аннотации мероприятия {[150, 152]} и перелета {[1, 120], [220, 365]} для Punta Arenas приводит к пустому временному интервалу {}. Однако для Arica мы находим два разных непустых пересечения: {[123, 125]} Для EID16 и {[276, 279]} для EID17. Учитывая, что нас интересует город (проецируемая переменная), а не событие, мы можем таким образом объединить эти две аннотации для Arica используя оператора join, возвращая аннотацию, в которой любой результат имеет значение true. В нашем сценарии естественный способ определить join-это объединение множеств дней, дающих {[123, 125], [276, 279]}.

Другие контекстуальные рамки. Другие рамки были предложены для моделирования и рассуждения о контексте в графах. Заметным примером является создание контекстных хранилищ знаний, которые позволяют назначать отдельные (под-)графы в свой собственный контекст. В отличие от именованных графов, контекст явно моделируется вдоль одного или нескольких измерений, где каждый (под-)граф должен принимать значение для каждого измерения. Каждое измерение дополнительно связано с частичным порядком над его значениями – например, 2020-03-22 ≤ 2020-03 ≤ 2020 позволяя выбирать и объединять подграфы, которые допустимы в контекстах с различными уровнями детализации. Аналогичным образом может быть предложена форма контекстной онлайн-аналитической обработки (OLAP), основанной на Кубе данных, образованном измерениями, где отдельные ячейки содержат графы знаний. Такие операции, как ”срез“ (выбор знаний в соответствии с заданными измерениями), а также ”свертка" (агрегирование знаний на более высоком уровне) могут быть поддержаны.

3. Дедуктивные знания

Как люди, мы можем вынести больше информации из графа данных рисунка 1, чем то, на что явно указывают ребра. Мы можем сделать вывод, например, что фестиваль (EID15) будет расположен в Сантьяго, даже если граф не содержит ребра EID15 —location→ Santiago , мы можем далее сделать вывод , что города, связанные рейсами, должны иметь какой-то аэропорт поблизости, даже если граф не содержит узлов, относящихся к этим аэропортам. В этих случаях, учитывая данные как предпосылки и некоторые общие правила о мире, которые мы можем знать априори, мы можем использовать дедуктивный процесс для получения новых данных, позволяя нам знать больше, чем то, что явно указано данными. Эти типы общих предпосылок и правил, когда их разделяют многие люди, составляют часть «здравого смысла»; и наоборот, когда их скорее разделяют несколько экспертов в какой-либо области, они образуют часть «знаний предметной области», где, например, специалист в области биологии может знать, что гемоцианин - это белок, содержащий медь, который переносит кислород в крови некоторых видов животных. Машины, напротив, не имеют априорного доступа к таким дедуктивным способностям; скорее, им нужно дать формальные инструкции с точки зрения предпосылок и режимов следования, чтобы делать выводы, аналогичные тому, что может сделать человек. Эти режимы следования формализуют выводы, которые логически следуют как следствие данного набора предпосылок. Получив такие инструкции, машины могут (часто) применять выводы с точностью, эффективностью и масштабом, превосходящими возможности человека. Эти выводы могут служить ряду приложений, таких как улучшение ответов на запросы, (дедуктивная) классификация, поиск несоответствий и т. д. В качестве конкретного примера, включающего ответы на запросы, предположим, что мы заинтересованы в знании фестивалей, расположенных в Сантьяго; мы можем просто выразить такой запрос в соответствии с шаблоном графа, показанным на рисунке 21. Этот запрос не возвращает результатов для графа на рисунке 1: нет узла с именем Festival , и места Santiago. Однако, ответ (Ñam) мог бы быть автоматически извлечен, если бы мы заявили, что быть фестивалем еды означает, что является фестивалем, или что наличие места в городе z влечет за собой наличие места. Как же тогда следует улавливать такие следствия? В этом разделе мы обсуждаем способы выражения и автоматизации более сложных следствий. Хотя мы могли бы использовать для этих целей ряд логических структур, таких как логика первого порядка, журнал данных, программирование набора ответов и т. д., Мы фокусируемся на онтологиях, которые представляют собой формальное представление знаний, которые, что важно для нас, могут быть представленным в виде графа. Затем мы обсуждаем, как эти онтологии могут быть формально определены, как они соотносятся с существующими логическими структурами и как можно проводить рассуждения в отношении таких онтологий.

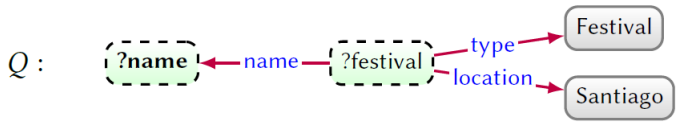
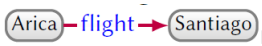


Рис. 21. Графовая модель запроса названий фестивалей в Сантьяго

Онтологии

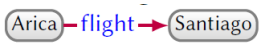
Для того чтобы сделать возможным извлечение знаний, мы должны быть точны в том, что означают термины, которые мы используем. Возвращаясь, например, к рисунку 1 и рассматривая узел EID16 более внимательно, мы можем начать задаваться вопросом, как он моделируется, особенно в сравнении с EID15. Оба узла – в соответствии с иерархией классов на рис. 11 – считаются событиями. Но что, если, например, мы хотим определить две пары дат начала и окончания для EID16 для соответствующих разных мест? Должны ли мы скорее рассматривать то, что происходит в каждом месте, как отдельное событие? Что же тогда делать, если событие имеет различные даты начала и окончания в одном месте проведения: будут ли они также рассматриваться как одно (повторяющееся) событие или как множество событий? Эти вопросы являются гранями более общего вопроса: что именно мы подразумеваем под “событием”? Происходит ли это за один непрерывный промежуток времени или может происходить много раз? Происходит ли это в одном месте или в нескольких? На такие вопросы нет “правильных” ответов – мы можем понимать термин “событие” по-разному, и поэтому ответы являются вопросом условности. В контексте вычислений онтология-это конкретное формальное представление того, что означают термины в той области, в которой они используются (например, в данной области). Например, онтология одного события может формально определить, что если сущность является “событием”, то она имеет точно одно место и точно один момент времени, в котором она начинается. И наоборот, другая онтология событий может определить , что “событие” может иметь несколько мест проведения и несколько времен начала и т. д. Каждая такая онтология формально фиксирует определенную перспективу – определенную условность. В рамках первой онтологии, например, мы не могли бы назвать Олимпиаду “событием”, в то время как во второй онтологии мы могли бы. Точно так же онтологии могут определять, как моделируются графовые данные. В рамках первой онтологии мы можем разделить EID16 на два события. Во втором случае мы можем выбрать EID16 как одно мероприятие с двумя площадками. В конечном счете, учитывая, что онтологии являются формальными представлениями, их можно использовать для автоматизации извлечения информации. Как и все соглашения, полезность онтологии зависит от уровня согласования относительно того, что эта онтология определяет, насколько она детализирована и насколько широко и последовательно она принимается. Принятие онтологии сторонами, участвующими в одном графе знаний, может привести к последовательному использованию терминов и последовательному моделированию в этом графе знаний. Согласование нескольких графов знаний, в свою очередь, повысит интероперабельность этих графов знаний. Среди наиболее популярных языков онтологий, используемых на практике, - Web Ontology Language (OWL), рекомендованный W3C и совместимый с графами RDF; и Open Biomedical Ontologies Format (OBOF), используемый в основном в биомедицинской области. Поскольку OWL более широко распространена, мы сосредоточимся на ее особенностях, хотя многие сходные черты встречаются и у той, и у другой. Однако прежде чем вводить такие особенности, мы должны обсудить, как следует интерпретировать графы.

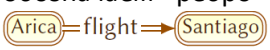
Интерпретации. Мы, как люди, можем интерпретировать узел 

в графике данных на рисунке 1 как относящийся к реальному городу, который является столицей Чили. Далее мы можем интерпретировать ребро 

как утверждение, что из города Арика в этот город есть рейсы. Таким образом, мы интерпретируем граф данных как другой граф – то, что мы здесь называем графом предметной области, – состоящий из сущностей реального мира, связанных реальными отношениями. Процесс интерпретации здесь включает отображение узлов и ребер в графе данных в узлы и ребра графа домена. Таким образом, мы можем абстрактно определить интерпретацию графа данных как состоящего из двух элементов: графа домена и отображения терминов (узлов и меток ребер) графа данных на термины графа домена. Граф предметной области следует той же модели, что и граф данных; например, если граф данных представляет собой ориентированный граф с именованными ребрами, то таким же будет и граф предметной области. Для простоты мы будем говорить об ориентированных графах с именованными ребрами и называть узлы графа предметной области сущностями, а ребра графа предметной области отношениями. Учитывая граф данных и интерпретацию, в то время как мы обозначаем узлы в графе данных 

, мы будем обозначать сущность, на которую он ссылается в графе предметной области 

(в соответствии с отображением данной интерпретации). Аналогично, в то время как мы обозначаем ребро через 

, Мы будем обозначать отношение через 

(опять же, в соответствии с отображением данной интерпретации). В этом абстрактном понятии интерпретации мы не требуем, чтобы 

н 

города реального мира, ни даже чтобы граф предметной области содержал реальные сущности и отношения: интерпретация может иметь любой граф предметной области и отображение. Почему такое абстрактное понятие интерпретации полезно? Различие между узлами/ребрами и сущностями/отношениями становится важным, когда мы определяем значение признаков и следствий онтологии. Чтобы проиллюстрировать это различие, если мы спросим, существует ли ребро, обозначенное как полет между 

and 

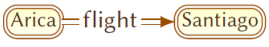
графом данных на рис. 1 и для него, ответ будет отрицательным. Однако если мы спросим, связаны ли сущности 

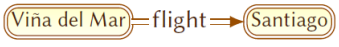
и 

связаны ли отношением полета, то ответ зависит от того, какие предположения мы делаем при интерпретации графа. Согласно предположению о замкнутом мире (CWA), если у нас нет дополнительных знаний, то ответ будет однозначным "нет", поскольку то, что неизвестно , считается ложным. И наоборот, в предположении открытого мира (OWA) мы не можем быть уверены , что это отношение не существует, поскольку оно может быть частью некоторого знания, которое (еще) не описано графом. Аналогично в предположении об уникальном имени (UNA) граф данных описывает по крайней мере два перелета до 

(поскольку 

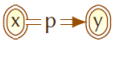
and 

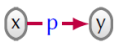
предполагается, что и являются разными сущностями и, следовательно 

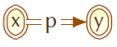
, и 

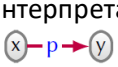
должны быть разными ребрами). И наоборот, не предполагая никакого уникального имени, мы можем только сказать, что существует по крайней мере один такой полет с тех пор 

и 

может быть одной и той же сущностью с двумя “именами”. Эти допущения (или их отсутствие) определяют, какие интерпретации верны и какие интерпретации удовлетворяют тем или иным графам данных. NUNA запрещает интерпретации, которые сопоставляют два термина данных с одним и тем же термином предметной области. NUNA допускает такие толкования. Согласно CWA, интерпретация , содержащая ребро 

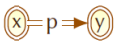
в своем графе предметной области, может удовлетворять только графу данных, из которого мы можем 

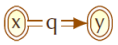
сделать вывод . Согласно OWA, интерпретация, содержащая ребро 

, может удовлетворять графу данных, не влекущему 

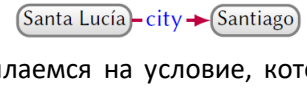
до тех пор, пока это не противоречит этому ребру.13 В случае OWL приняты NUNA и OWA, что представляет собой наиболее общий случай,

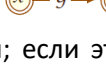
когда несколько узлов / меток ребер в графе могут относиться к одному и тому же объекту / типу отношения (NUNA), и где что-либо, не связанное с графом данных, как следствие не считается ложным (OWA). Помимо наших базовых предположений, мы можем связать определенные шаблоны в графе данных с семантическими условиями, которые определяют, какие интерпретации ему удовлетворяют; например, мы можем добавить семантическое условие, чтобы обеспечить выполнение этого условия, если наш граф данных содержит ребро 

, тогда любое отношение 

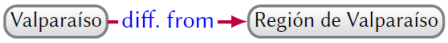
в доменом графовой интерпретации также соответствует отношению 

в удовлетворении данных графа. Эти семантические условия затем формируют особенности языка онтологии. В дальнейшем, чтобы облегчить читаемость, мы представим особенности OWL, используя абстрактную графическую нотацию с сокращенными терминами. Для подробностей конкретных синтаксисов мы скорее ссылаемся на стандарты OWL и OBOF. Точно так же мы представляем семантические условия для интерпретаций, связанных с каждым элементом, в одном и том же графическом формате.

Индивидуумы. В таблице 3 мы перечислим основные признаки, поддерживаемые OWL для описания индивидов (например, Santiago, EID16), иногда отличающиеся от классов и свойств. Во-первых, мы можем утверждать (бинарные) отношения между индивидами, используя такие ребра, как 

. Например, в колонке "условие", когда мы пишем 

, мы ссылаемся на условие, которое данное отношение имеет место в интерпретации; если это так, интерпретация удовлетворяет аксиоме. OWL также позволяет для определения отношений явно указать, что два термина относятся к одному и тому же объекту, где, например, 

говорится, что оба они относятся к одному и тому же региону; или что два термина относятся к разным объектам, где, например, 

отличается город от одноименного региона. Мы можем также утверждать, что отношение не выполняется с помощью отрицания, которое может быть сериализовано в виде графа с использованием формы овеществления (см.

Свойства. OWL допускает такие определения и дополнительно включает другие функции, перечисленные в таблице 4. Мы можем определить пару свойств как эквивалентные, обратные или непересекающиеся. Далее мы можем определить конкретное свойство для обозначения транзитивного, симметричного, асимметричного, рефлексивного или нерефлексивного отношения. Мы также можем определить множественность отношения, обозначаемого свойствами, исходя из того, что оно функционально (многие-к-одному) или обратно функционально (один-ко-многим). Далее мы можем определить ключ для класса, обозначающий набор свойств, значения которых однозначно идентифицируют сущности этого класса. Не принимая допущения об уникальности имени (UNA), из этих последних трех признаков мы можем заключить, что два или более термина относятся к одной и той же сущности. Наконец, мы можем связать свойство с цепочкой (выражение пути, допускающее только конкатенацию свойств) так, чтобы пары сущностей, связанных цепочкой, также были связаны данным свойством. Заметим, что для последних двух признаков в таблице 4 мы требуем представления списка, обозначенного вертикальной нотацией 

, хотя такой список может быть сериализован в виде графа несколькими конкретными способами, OWL использует списки RDF (см. рис. 17).

Классы. OWL поддерживает подклассы и множество дополнительных функций для определения и утверждения классов; эти дополнительные функции суммированы в таблице 5. Учитывая пару классов, OWL позволяет определить, что они эквивалентны или не пересекаются. После этого OWL предоставляет множество возможностей для определения новых классов, применяя операторы множества к другим классам или основываясь на условиях, которым удовлетворяют свойства его

экземпляров. Во-первых, используя операторы множества, можно определить новый класс как дополнение к другому классу, объединение или пересечение списка (произвольной длины) других классов или как перечисление всех его экземпляров. Во-вторых, путем наложения ограничений на конкретное имущество p, можно определить классы, чьи экземпляры будут все объекты, что есть: какое-то значение из заданного класса на p; все значения из заданного класса на p; у конкретного физического лица в качестве значения p (не имеет значения); сами как рефлексивное значение p (самостоятельно); иметь, по крайней мере, или точно какой - количество значений p (мощности); и, по крайней мере, по большей мере или точно какой-количество значений p из данного класса (квалифицированных количества элементов). Для последних двух случаев в таблице 5 мы используем обозначение "$\#\{a \mid \phi\}$" для подсчета различных сущностей, удовлетворяющих ϕ интерпретации ϕ. Затем эти характеристики могут быть объединены для создания более сложных классов, где объединение примеров для Пересечения и Has Self в таблице 5 дает определение: самоуправляемые такси-это такси, имеющие себя в качестве водителя.

Таблица 3. Особенности онтологии для людей

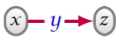
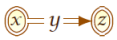
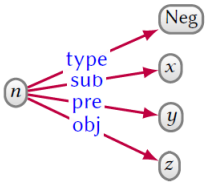
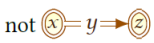
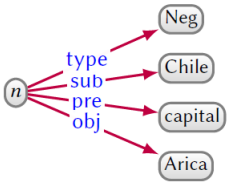
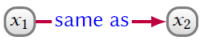
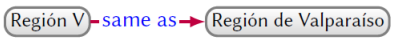
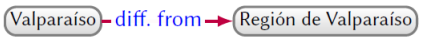
Feature	Axiom	Condition	Example
Assertion			
Negation			
Same As			
Different From			

Таблица 4. Особенности онтологии для аксиом свойств

Feature	Axiom	Condition (for all x_*, y_*, z_*)	Example
Subproperty	p — subp. of → q	$x \xrightarrow{p} y$ implies $x \xrightarrow{q} y$	venue — subp. of → location
Domain	p — domain → c	$x \xrightarrow{p} y$ implies x — type → c	venue — domain → Event
Range	p — range → c	$x \xrightarrow{p} y$ implies y — type → c	venue — range → Venue
Equivalence	p — equiv. p. → q	$x \xrightarrow{p} y$ iff $x \xrightarrow{q} y$	start — equiv. p. → begins
Inverse	p — inv. of → q	$x \xrightarrow{p} y$ iff $y \xrightarrow{q} x$	venue — inv. of → hosts
Disjoint	p — disj. p. → q	not $x \xrightarrow{q,p} y$	venue — disj. p. → hosts
Transitive	p — type → Transitive	$x \xrightarrow{p} y \xrightarrow{p} z$ implies $x \xrightarrow{p} z$	part of — type → Transitive
Symmetric	p — type → Symmetric	$x \xrightarrow{p} y$ iff $y \xrightarrow{p} x$	nearby — type → Symmetric
Asymmetric	p — type → Asymmetric	not $x \xleftrightarrow{p,p} y$	capital — type → Asymmetric
Reflexive	p — type → Reflexive	$x \xrightarrow{p}$	part of — type → Reflexive
Irreflexive	p — type → Irreflexive	not $x \xrightarrow{p}$	flight — type → Irreflexive
Functional	p — type → Functional	$y_1 \xleftarrow{p} x \xrightarrow{p} y_2$ implies $y_1 = y_2$	population — type → Functional
Inv. Functional	p — type → Inv. Functional	$x_1 \xrightarrow{p} y \xleftarrow{p} x_2$ implies $x_1 = x_2$	capital — type → Inv. Functional
Key	c — key → $p_1 \dots p_n$	$x_1 \xrightarrow{p_1 \dots p_n} y_1$, $x_2 \xrightarrow{p_1 \dots p_n} y_1$, y_1 — type → c implies $x_1 = x_2$	City — key → lat long
Chain	p — chain → $q_1 \dots q_n$	$x \xrightarrow{q_1} y_1 \dots y_{n-1} \xrightarrow{q_n} z$ implies $x \xrightarrow{p} z$	location — chain → location part of

Другие особенности. OWL поддерживает другие языковые функции, которые ранее не обсуждались, в том числе: свойства аннотации, которые предоставляют метаданные об онтологиях, например информацию о версиях; тип данных и свойства объекта, которые позволяют отличать свойства, принимающие значения типа данных, от свойств, не принимающих их; и фасеты типов данных, которые позволяют определять новые типы данных путем применения ограничений к существующим типам данных, например, для определения того, что места в Чили должны иметь значение с плавающей запятой между -66,0 и -110,0 в качестве значения для свойства (datatype) latitude. Для более подробной информации обратитесь к стандарту OWL.

Семантика и последствия

Условия, перечисленные в предыдущих таблицах, указывают, как следует интерпретировать каждый признак. Эти условия порождают извлечения данных, где, например, в отношении симметричной функции таблицы 4 определение nearby — type → Symmetric и ребро Santiago — nearby → Santiago Airport влекут за собой ребро Santiago Airport — nearby → Santiago в соответствии с условием, данным для этой функции. Теперь мы опишем, как эти условия приводят к результатам.

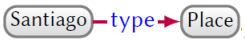
Теоретико-модельная семантика. Каждая аксиома, описанная в предыдущих таблицах, будучи добавленной к графу, накладывает некоторые условия на интерпретации, удовлетворяющие графу. Интерпретации , удовлетворяющие графу, называются моделями графа. Если бы мы рассматривали , например, только базовое условие функции утверждения в таблице 3, то модели графа были бы

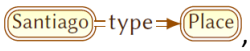
любой интерпретацией , такой, что для каждого ребра x — y → z графа существует отношение x = y ⇒ z в модели. учитывая , что в модели могут быть и другие отношения (в соответствии с OWA), число моделей любого такого графа бесконечно. Кроме того, учитывая, что мы можем сопоставить несколько узлов графа с одной сущностью в модели (под NUNA), любая интерпретация с (например) отношением a = a ⇒ a является моделью любого графа до тех пор, пока для каждого ребра x — y → z графа она выполняется x = y = z = a в интерпретации (другими словами, интерпретация сопоставляет все a). Добавляя к графу аксиомы со связанными с ними условиями, мы ограничиваем модели для графа; например, рассматривая граф с двумя ребрами - x — y → z и y — type → Irreflexive – интерпретация с a = a ⇒ a, x = y = ... = a ним больше не является моделью, поскольку она нарушает условие нерефлексивной аксиомы.

Таблица 5. Особенности онтологии для аксиом и определений классов

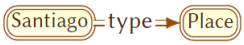
Feature	**Axiom**	**Condition** (for all x_*, y_*, z_*)	**Example**
Subclass	c — subc. of → d	x = type ⇒ c implies x = type ⇒ d	City — subc. of → Place
Equivalence	c — equiv. c. → d	x = type ⇒ c iff x = type ⇒ d	Human — equiv. c. → Person
Disjoint	c — disj. c. → d	not c ← type = x = type ⇒ d	City — disj. c. → Region
Complement	c — comp. → d	x = type ⇒ c iff not x = type ⇒ d	Dead — comp. → Alive
Union	c — union → d_1 … d_n	x = type ⇒ c iff x = type ⇒ d_1 or … or x = type ⇒ d_n	Flight — union → DomesticFlight, InternationalFlight
Intersection	c — inter. → d_1 … d_n	x = type ⇒ c iff x = type ⇒ d_1, …, d_n	SelfDrivingTaxi — inter. → Taxi, SelfDriving
Enumeration	c — one of → x_1 … x_n	x = type ⇒ c iff $x \in \{x_1, \ldots, x_n\}$	EUState — one of → Austria … Sweden
Some Values	c — prop → p, some → d	x = type ⇒ c iff there exists a such that $x = p \Rightarrow a$ = type ⇒ d	EUCitizen — prop → nationality, some → EUState
All Values	c — prop → p, all → d	x = type ⇒ c iff for all a with $x = p \Rightarrow a$ it holds that a = type ⇒ d	Weightless — prop → has part, all → Weightless
Has Value	c — prop → p, value → y	x = type ⇒ c iff $x = p \Rightarrow y$	ChileanCitizen — prop → nationality, value → Chile
Has Self	c — prop → p, self → true	x = type ⇒ c iff $x = p \Rightarrow x$	SelfDriving — prop → driver, self → true
Cardinality $\star \in \{=, \leq, \geq\}$	c — prop → p, $\star$ → n	x = type ⇒ c iff $\#\{a \mid x = p \Rightarrow a\} \star n$	Polyglot — prop → fluent, ≥ → 2
Qualified Cardinality $\star \in \{=, \leq, \geq\}$	c — prop → p, class → d, $\star$ → n	x = type ⇒ c iff $\#\{a \mid x = p \Rightarrow a$ = type ⇒ $d\} \star n$	BinaryStarSystem — prop → body, class → Star, = → 2

Логическое следствие. Мы говорим, что один граф влечет за собой другой тогда и только тогда, когда любая модель первого графа является также моделью второго графа. Интуитивно это означает, что последний граф не говорит ничего нового по сравнению с первым графом и, таким образом, является логическим следствием первого графа. Например, рассмотрим граф 

и граф 

. Все модели последнего должны иметь отношение 

, но также должны иметь его и все модели первого, которые должны иметь 

и далее должны удовлетворять условию подкласса, которое требует, чтобы 

также выполнялось. Отсюда мы заключаем, что любая модель последнего графа должна быть моделью первого графа, или, другими словами, Первый граф влечет за собой второй граф.

Рассуждения (Reasoning)

К сожалению, учитывая два графа, решение, влечет ли первый за собой второй - в соответствии с понятием следования, которое мы определили, и для всех онтологических характеристик, перечисленных в таблицах 3–5 - неразрешимо: не может существовать (конечный) алгоритм для такого следования, который останавливается на всех входах с правильным ответом "правда / ложь". Однако мы можем предоставить практические алгоритмы рассуждений для онтологий, которые (1) останавливаются на любой входной онтологии, но могут пропускать следствия, возвращая ложь вместо истины, (2) всегда останавливаются с правильным ответом, но принимают только входные онтологии с ограниченными функциями или (3) возвращать только правильные ответы для любой входной онтологии, но никогда не может останавливаться на определенных входах. Хотя вариант (3) исследовался с использованием, например, средств доказательства теорем для логики первого порядка, варианты (1) и (2) чаще всего рассматриваются с использованием правил и / или логики описания. Вариант (1), как правило, позволяет использовать более эффективные и масштабируемые алгоритмы рассуждений и полезен, когда данные являются неполными и некоторые следствия важны. Вариант (2) может быть лучшим выбором в таких областях, как медицинские онтологии, где отсутствующие следствия могут иметь нежелательные результаты.

Правила. Один из самых простых способов обеспечить автоматический доступ к дедуктивным знаниям - это правила вывода (или просто правила), кодирующие последствия в стиле «если-то». Правило состоит из тела (если) и головы (тогда). И тело, и голова представлены в виде графовых образов. Правило указывает, что если мы можем заменить переменные тела терминами из графа данных и сформировать подграф данного графа данных, то использование той же замены переменных в заголовке даст действительное следствие. Голова обычно должна использовать подмножество переменных, появляющихся в теле, чтобы в заключении ни одна переменная не оставалась незамещенной. Правила этой формы соответствуют (положительному) протоколу данных в базах данных, предложениям Хорна в логическом программировании и т. д. Правила могут использоваться для фиксации следствий в онтологических условиях. В Таблице 6 мы перечисляем некоторые примеры правил для функций подкласса, подсвойства, домена и диапазона; эти правила могут рассматриваться как неполные, не отражающие, например, того, что каждый класс является подклассом самого себя, что каждое свойство является подклассом самого себя и т. д. Более полный набор правил для функций OWL в Таблицах 3 –5 были определены как OWL 2 RL / RDF; эти правила также неполны, поскольку такие правила не могут полностью охватить отрицание (например, дополнение), существование (например, некоторые значения), универсалии (например, все значения) или подсчет (например, количество элементов и определенное количество элементов). Однако были предложены другие языки правил для поддержки таких дополнительных функций, включая экзистенциальные (см., Например, Datalog ±), дизъюнкцию (см., Например, Disjunctive Datalog) и т. д. Правила можно использовать для рассуждений несколькими способами. Материализация относится к идее рекурсивного применения правил к графу,

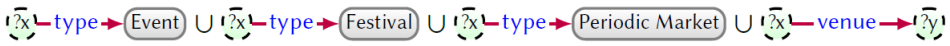
добавления сгенерированных выводов обратно к графу до тех пор, пока не будет достигнута фиксированная точка, и больше ничего нельзя будет добавить. Затем материализованный граф можно рассматривать как любой другой граф. Хотя эффективность и масштабируемость материализации можно повысить за счет таких оптимизаций, как сети Rete, или использования распределенных фреймворков, таких как MapReduce, в зависимости от правил и данных материализованный граф может стать невероятно большим для управления.

Рис. 22. Пример перезаписи запроса для запроса Q на рис. 21

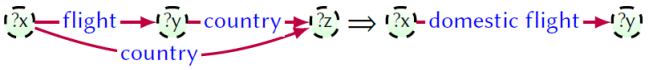
Таблица 6. примеры правил для объектов подкласса, подсвойства, домена и диапазона

Feature	Body	⇒	Head
Subclass (I)	?x type ?c subc. of ?d	⇒	?x type ?d
Subclass (II)	?c subc. of ?d subc. of ?e	⇒	?c subc. of ?e
Subproperty (I)	?x ?p ?y (?p subp. of ?q)	⇒	?x ?q ?y
Subproperty (II)	?p subp. of ?q subp. of ?r	⇒	?p subp. of ?r
Domain	?x ?p ?y (?p domain ?c)	⇒	?x type ?c
Range	?x ?p ?y (?p range ?c)	⇒	?y type ?c

Другая стратегия заключается в использовании правил перезаписи запросов, которые при заданном запросе автоматически расширяют запрос для поиска решений, связанных с набором правил; например, взяв граф схемы на рис. 12 и правила в таблице 6, (суб)шаблон ?x type Event в данном входном запросе будет переписан в следующий дизъюнктивный шаблон, вычисленный на исходном графе:



На рис. 22 представлен более полный пример онтологии, которая используется для переписывания запроса на рис. 21; если вычислить его по графу на рис. 1, Nam то он будет возвращен в виде решения. Однако не все вышеупомянутые функции OWL могут быть поддержаны таким образом. Профиль OWL QL - это подмножество OWL, разработанное специально для переписывания запросов этой формы. в то время как правила могут быть использованы для (частичного) захвата онтологических следствий, они также могут быть определены независимо от языка онтологий, захватывая следствия для данной области. Фактически, некоторые правила, такие как следующие, не могут быть захвачены ранее рассмотренными функциями онтологии, поскольку они не поддерживают способы вывода отношений из циклических графовых паттернов (по соображениям вычислимости):



Различные языки позволяют выражать правила над графами – независимо или вместе с языком онтологий – включая нотацию, формат обмена правилами (RIF), язык правил семантической сети (SWRL) и систему вывода SPARQL (SPIN).

Описательные Логики. Логики описания (DL) были первоначально введены как способ формализации значения фреймов и семантических сетей. Учитывая, что семантические сети являются ранней версией графов знаний, и тот факт, что DL сильно повлияли на язык веб-онтологий, DL, таким образом, занимают важное место в логической формализации графов знаний. DL образуют скорее семейство логик, чем конкретную логику. Первоначально DL были ограниченными фрагментами логики первого порядка (FOL), которые разрешали решаемые задачи рассуждения, такие как проверка следования. Различные DL обеспечивают разный баланс между выразительной силой и вычислительной сложностью рассуждений. Позднее DL будут расширены функциями, выходящими за рамки FOL, но полезными в контексте моделирования данных графа. Описательная Логика основана на трех типах элементов: отдельные лица, такие как Сантьяго; классы (также известные как понятия), такие как Город; и свойства (также известные как роли), такие как полет. Затем DL позволяют делать заявления, известные как аксиомы, об этих элементах. Аксиомы утверждений могут быть либо унарными классовыми отношениями для индивидов, такими как Город (Сантьяго), либо бинарными отношениями для индивидов, такими как перелет (Сантьяго, Арика). Такие аксиомы образуют бокс утверждений (A-Box). DL далее вводят логические символы, позволяющие определять аксиомы классов (формирующие Терминологический бокс или T-Box для краткости) и аксиомы свойств (формирующие Role Box, R-Box); например, аксиома класса City $\sqsubseteq$ Place утверждает, что первый класс является подклассом второго, в то время как аксиома свойства flight $\sqsubseteq$ ConnectsTo утверждает, что первое свойство является подклассом второго. Затем DL могут вводить богатый набор логических символов не только для определения аксиом классов и свойств, но также для определения новых классов на основе существующих терминов; в качестве примера последнего мы можем определить класс $\exists$nearby.Airport как класс субъектов, у которых есть аэропорт поблизости. Отметив, что символ $\top$ используется в DL для обозначения класса всех субъектов, мы можем добавить аксиому класса $\exists$flight.$\top$ $\sqsubseteq$ nearby.Airport, чтобы заявить, что субъекты, выполняющие исходящий рейс, должны иметь аэропорт поблизости. Отметив, что символ $\sqcup$ может использоваться в DL для обозначения того, что класс является объединением других классов, мы можем дополнительно определить, что аэропорт $\sqsubseteq$ внутренний аэропорт $\sqcup$ международный аэропорт, т. е. что аэропорт является либо внутренним аэропортом, либо международным аэропортом (или обоими). Сходства между этими функциями DL и функциями OWL, ранее описанными в таблицах 3-5, не случайны: стандарт OWL находился под сильным влиянием DL, где, например, язык OWL DL является фрагментом OWL, ограниченным так, что следствие становится разрешимым. В качестве примера ограничения, когда DomesticAirport $\sqsubseteq$ = 1 пункт назначения - страна.$\top$, мы можем определить в синтаксисе DL, что внутренние аэропорты имеют рейсы, предназначенные точно в одну страну (где p ◦ q обозначает цепочку свойств). Однако подсчет цепочек часто запрещен в DL для обеспечения разрешимости. Выразительные DL поддерживают сложные следствия, включающие экзистенциалы, универсалии, подсчет и т. д. Общая стратегия для принятия решения о таких следствиях - свести следствие к выполнимости, которая определяет, является ли онтология непротиворечивой или нет. После этого для проверки выполнимости можно использовать такие методы, как tableau, осторожное построение моделей путем их завершения по аналогии с ранее описанной стратегией материализации, но дополнительно модели ветвления в случае дизъюнкции, введение новых элементов для представления экзистенциальных элементов и т. д., успешно «завершенный» процесс приходит к выводу, что исходные определения приемлемы. Из-за непомерно высокой вычислительной

сложности - например, когда дизъюнкция может привести к экспоненциальному количеству возможностей ветвления - такие стратегии рассуждений обычно не применяются в случае крупномасштабных данных, хотя они могут быть полезны при моделировании сложных областей.

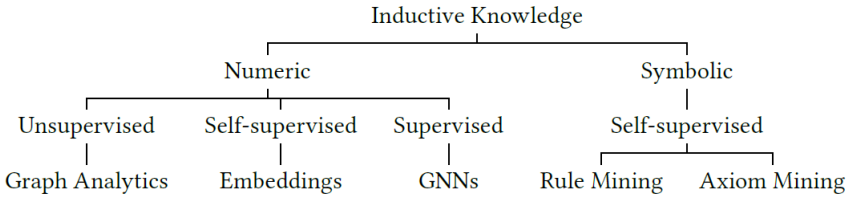


Рис. 23. Концептуальный обзор индуктивных методов построения графов знаний с точки зрения типа генерируемого представления (числовое/символьное) и типа используемой парадигмы (бесконтрольное/Самоконтролируемое/контролируемое).

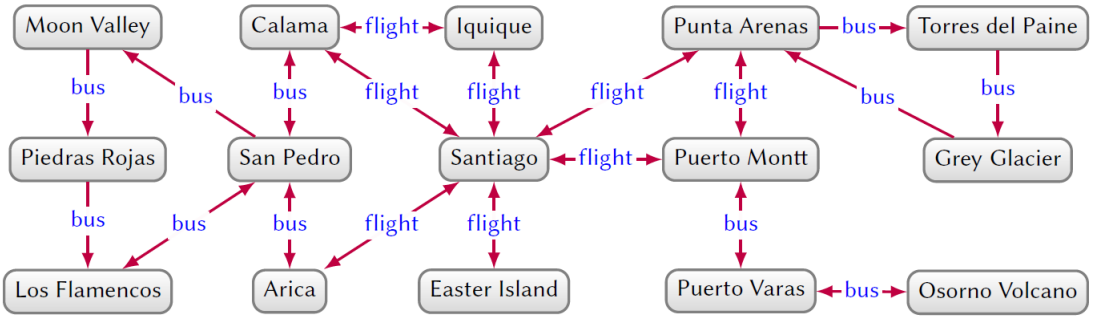


Рис. 24. Граф данных, представляющий транспортные маршруты в Чили