Специализация для реляционного языка программирования miniKanren

9 сентября 2019 г.

miniKanren — реляционный язык программирования. В основе парадигмы лежит программирование в отношениях. Любая программа описывает математическое отношение на аргументах программы. Имея программу-отношение, можно выполнять к нему запросы: специфицируя некоторые известные аргументы, можно получать значения остальных, таким образом, чтобы аргументы состояли в отношении.

Например, рассмотрим отношение $addo \subseteq Int \times Int \times Int$. Три целых числа находятся в отношении $addo$, если сумма первых двух чисел совпадает с третьим. Имея такое отношение можно:

- Проверить, что сумма двух чисел совпадает с третьим, специфицировав все три аргумента отношения:
  - $addo$ 13 42 55 $\rightarrow$ ✓
  - $addo$ 13 42 0 $\rightarrow$ ✗
- Найти сумму двух чисел, задав первые два аргумента:
  - $addo$ 13 42 ? $\rightarrow$ ✓[55]
- Выразить вычитание, специфицировав первый и третий аргумент
  - $addo$ 13 ? 55 $\rightarrow$ ✓[42]
- Разложить некоторое число на слагаемые. При этом, в данном случае будет бесконечное количество вариантов правильных ответов:
  - $addo$? ? 2 $\rightarrow$ ✓$[(0,2),(1,1),(2,0),(-1,3)\dots]$
- Найти все тройки целых чисел, находящиеся в отношении:
  - $addo$? ? ? $\rightarrow$ ✓$[(0,0,0),(1,0,1),(0,1,1),(1,1,2),\dots]$

Таким образом, можно говорить о выполнении программ в разных “направлениях”. Основное применение miniKanren в данный момент — разработка реляционных интерпретаторов. Реляционный интерпретатор это программа-отношение, которая связывает программу на некотором языке, входные данные такой программы и результат ее выполнения на этих данных. Выполнение реляционного интерпретатора в обратном направлении, то есть когда специфицированы входные и выходные данные, синтезирует программу по набору тестов. Это используется в инструменте Barliman `https://github.com/webyrd/Barliman`.

Еще одно интересное применение miniKanren — порождение решения задачи поиска по решению задачи распознавания. Имея реляционную программу, которая проверяет корректность некоторых данных, можно найти все такие данные, которые являются корректными. Например, если у нас есть отношение $isPatho \subseteq [Vertex] \times (Graph\ Vertex)$ можно не только проверить, что некоторая последовательность вершин является путем в заданном графе, но и найти все пути в графе. Реализовав отношение, которое проверяет, что один терм является унификатором двух других, можно находить унификаторы для термов.

Реляционная парадигма программирования сложна, при том, что возможности, которые она предоставляет, велики. Для того, чтобы пользователю не приходилось изучать абсолютно новую для себя парадигму программирования, мы разработали нано-технологию трансляции из функционального языка в miniKanren . Трансляция при этом порождает программы, эффективно выполняющиеся в прямом направлении, но не в обратном. Можно изменить транслятор, чтобы он генерировал более эффективные программы в обратном направлении, но тогда мы потеряем эффективность в прямом. Такой подход далек от оптимального, поэтому мы предлагаем использовать специализацию.

Специализатор преобразовывает программу на основании статически известных данных в более эффективную. В случае программ на miniKanren статической информацией является не только значение аргументов, но и направление вычислений. Как раз специализация программы-отношения на направление исполнения и позволяет порождать эффективные в интересующих нас направлениях программы.

Ближайший родственник реляционного программирования — логическое программирование во главе с языком Prolog . Наиболее проработанная техника специализации для Prolog — конъюнктивная частичная дедукция. Мы ранее адаптировали эту технику для miniKanren . Специализация частичной дедукцией показывает достаточно хорошие результаты, но сама техника очень сложна, поэтому хочется рассмотреть другие варианты.

1 Специализация для miniKanren

Вообще самая важная цель в этом проекте: сделать специализатор, который в состоянии специализировать реляционный интерпретатор зрелого языка. С этим пока не справляется ни один из специализаторов, который мы пробовали.

- Актуальность темы
  - Почему этим стоит заниматься
    - Задача обращения программ — классная задача, и специализация — способ решить эту задачу универсальным образом.
    - miniKanren — шаг в сторону декларативного программирования, а значит уменьшению страданий для программирующих.
    - miniKanren используется в доказательной медицине для поиска подходящего лечения для очень редких генетических заболеваний.
  - Кто в мире этим занимается
    - miniKanren разрабатывают в Department of Computer Science & Hugh Kaul Precision Medicine Institute, University of Alabama at Birmingham под руководством Will Byrd и у нас в лаборатории.
    - Основной разработчик конъюнктивной частичной дедукции — Michael Leuschel, система ECCE
    - Специализацией занимается Neil Jones и в университете DIKU
    - Суперкомпиляция (подход для специализации, способный не только протягивать константы, но и улучшать производительность некоторых программ в линию раз (иногда программу, работающую за $O(n^k)$ удается превратить в программу, работающую за $O(n^{k-1})$ изначально разработана Турчиным в России, сейчас ее продолжают исследовать в Переяславле-Залесском и втом числе в DIKU.
    - Есть еще дистилляция — еще более мощная техника, чем суперкомпиляция, над ней работает Geoff Hamilton в DCU
  - Какие результаты есть у них и можно переиспользовать
    - Хочется попробовать применить универсальный суперкомпилятор, разработанный в DIKU
    - Хочется использовать идеи других техник суперкомпиляций, нежели частичной дедукции, которые несут меньший след влияния Prolog — при этом тут нельзя просто переиспользовать их исходники.
- Содержательность и объем работы
  - Много ли делать
    - Не известно, какой именно метод специализации может дать наилучший результат, поэтому тут хочется сравнить много разных.
    - Уже есть реализация конъюнктивной частичной дедукции и какой-то версии аналога суперкомпиляции, но текущее качество специализации еще можно улучшить. Для этого надо как минимум отрефакторить текущий код, чтобы поддержать другой подход к обобщению.
    - Любой специализатор требует доработки для лучшей обработки релевантных для предметной области случаев — тут придется проводить дополнительные эксперименты и придумывать что-то новое.
  - Почему делать именно так

    * У меня нет предсказания, “как” надо делать: надо проводить исследование и сравнивать разные подходы.
  - Почему не надо (или надо) переиспользовать наработки других
    * Потому что они для других языков
    * Мы хотим сравнить качество нашей специализации с языконезависимым специализатором, разработанным в DIKU, но я практически уверена, что общее решение не будет давать такой хороший результат, как частное.

2 Трансляция из miniKanren в функциональный язык

- Актуальность темы
  - Почему этим стоит заниматься
  - Кто в мире этим занимается
  - Какие результаты есть у них и можно переиспользовать
- Содержательность и объем работы
  - Много ли делать
  - Почему делать именно так
  - Почему не надо (или надо) переиспользовать наработки других