$L(G) = \{w(p)\}$, $w(p) = w(v_0 l_0 v_1, v_1 l_1 v_2, ...)$, $v_i, l_j, v_k \in E$, то можно рассмотреть следующие задачи:

1. Поиск паттерна. Найти все пути в G, содержащие слова из L': $L'(G) \subseteq L(G) : \{P_G^{patterns}\} = \{P_G | w(P_G) \in L'(G)\}$.

2. Проверка на анти-паттерн: пусто ли пересечение языка графа с «языком анти-паттернов» $L''(G)$: $\{P_G \cap L''(G)\}? \equiv \emptyset$.

3. Классическая задача достижимости – найти все пары вершин (состояний программы, точек останова и т.д.), таких, что между ними существует нужный путь?

4. Подзадача классической задачи достижимости – существует ли нужный путь из точки A в B в программе?

Возможна и постановка последовательной проверки на КС-достижимость: сначала выделяется множество путей по (1), а далее проверяется (2). Такие паттерны (2) для (1) назовём «ограничивающими».

5.4 Восходящий разбор:LR

5.4.1 LR(0)

5.4.2 SLR

Автомат – такой же, как в LR(0). Таблица отличается только тем, что reduce выполняется только там, где это имеет смысл.

5.4.3 (C)LR(1)

Канонический LR.

5.4.4 LALR

Наиболее часто реализуемый на практике подход.

https://github.com/meyerd/flex-bison-example

Пусть есть грамматика, не разбираемая из-за конфликтов сдвиг-свертка или свертка-свертка по алгоритму SLR.

В этом случае грамматика преобразуется следующим образом:

- ищется нетерминал, на котором возникла вызвавшая конфликт свертка. Обозначим его А.

- вводятся новые нетерминалы $A_1, A_2, \ldots, A_n$, по одному на каждое появление А в правых частях правил.

- везде в правых частях правил А заменяется на соответствующее A_k.

- набор правил с А в левой части повторяется n раз по разу для каждого A_k.

- правила с A в левой части удаляются, тем самым полностью удаляя A из грамматики. Для преобразованной грамматики (она порождает такой же язык, что и исходная) повторяется попытка построения SLR(1) таблицы разбора.

Действие основано на том, что Follow(A) есть объединение всех Follow(A_k). В каждом конкретном состоянии новая грамматика имеет уже не A, а одно из A_k, то есть множество Follow для данного состояния имеет меньше элементов, чем для A в исходной грамматике.

Это приводит к тому, что для LALR(1) совершается меньше попыток поставить «приведение» в клеточку таблицы разбора, что уменьшает риск возникновения конфликтов с приведениями, иногда вовсе избавляет от них и делает грамматику, не разбираемую по SLR(1), разбираемой после преобразования.

Множество Follow(Ak) называется lookahead set для A и к-той встречи в правилах, отсюда название алгоритма.

6 Приложение

6.1 Необходимые определения из близких областей

6.1.1 Графы

В данном курсе мы будем рассматривать только конечные ориентированные помеченные графы, подразумевая под «графами» именно такие графы, если не указано противное.

Опр. 6.1 *Граф $G = (V, E, L)$, где V — конечное множество вершин, E — конечное множество рёбер, L*

Опр. 6.2 *Отношением достижимости на графе в смысле нашего определения называется двухместное,*

Опр. 6.3 *Транзитивным замыканием графа называется транзитивное замыкание отношения достижимости по всему графу.*

6.1.2 Матрицы

Список литературы

[1] `http://neerc.ifmo.ru/wiki/index.php?title=Минимизация_ДКА,_алгоритм_за_O(n%5E2)_с_построением_пар_различимых_состояний`

[2] `http://neerc.ifmo.ru/wiki/index.php?title=Минимизация_ДКА,_алгоритм_Хопкрофта_(сложность_O(n_log_n))`

[3] `http://neerc.ifmo.ru/wiki/index.php?title=Алгоритм_Бржозовского`