5. Задания практикума

Задания практикума были разработаны в поддержку теоретического изучения языка Лисп и предполагают составление лисп-программ решения типичных задач анализа и обработки символьных данных.

Отождествление рефал-выражений

Реализовать алгоритм отождествления выражений языка Рефал в виде лисповской функции `(Match P L)`, где `L` – произвольный список, `P` – список-образец, в состав которого кроме обычных атомов входят атомы-переменные вида `SX`, `WX`, `EX` и `VX`. Буква `E`, `W`, `V` или `S` обозначает тип переменной (множество ее допустимых значений). Буква `X` играет роль имени переменной; в качестве имени может быть взята любая латинская буква или десятичная цифра.

Функция `Match` производит *сопоставление* (*отождествление*) списка `L` с образцом `P` по правилам языка Рефал, рассматривая лисповские скобки как структурные рефальские скобки. Функция вырабатывает значение `T` или `NIL` в зависимости от успешности сопоставления. В случае удачного сопоставления переменные образца получают соответствующие значения: значением переменной вида `SX` может быть лисповский атом, а значением переменной вида `WX` – лисповское выражение (атом или список). Переменная вида `EX` или `VX` может быть сопоставлена соответственно произвольной или непустой последовательности лисп-выражений, при этом её значением является список из этих выражений.

Пример успешного сопоставления:

`(Match '(WA (C1 SB) EX) '(DO (C1 2) 5 ON))` ⇒ `T`,

при этом: `WA => DO`, `SB => 2`, `EX => (5 ON)`.

Набор тестов для функции `Match` должен включать тесты, демонстрирующие её применение для распознавания структуры алгебраических выражений.

Преобразование алгебраического выражения

Преобразовать в приведённый полином заданное алгебраическое выражение, в котором используются операции сложения, вычитания, умножения, возведения в степень, целые числа, однобуквенные переменные, а также круглые скобки, задающие нужный порядок вычисления операций.

Полином представляет собой сумму или разность нескольких одночленов. Одночленом может быть целое число, а также произведение целого числа и нескольких множителей – целых положительных степеней переменных (степень, равная единице, не записывается). Знак произведения в записи полинома опускается. Например, полиномом является запись `X+XY-15Y↑3+2`.

Полином называется *приведённым*, если в нём нет подобных одночленов, а в каждом его одночлене любая из переменных не встречается более одного раза. Приведённым является полином `X+XY-15Y↑3+2`, полиномы же `46ZYZ` и `X+X-7+3XY+2` не являются приведёнными.

В упрощённом решении этой задачи при записи исходного алгебраического выражения знаки арифметических операций, числа и переменные разделяются пробелами, а само выражение заключается в круглые скобки (чтобы сделать его лисповским списком и тем самым облегчить его ввод и обработку). В полном решении задачи на вход программы подаётся текст алгебраического выражения, которое обычно не содержит объемлющих скобок и в котором знаки операций, имена переменных и числа могут быть записаны слитно.

Суммирование рациональных выражений

Реализовать символьное вычисление суммы двух заданных рациональных выражений. Полученное рациональное выражение должно состоять только из приведённых многочленов.

Рациональное выражение представляет собой дробь, в числителе и знаменателе которой стоят полиномы от одной однобуквенной переменной. Полиномы являются в свою очередь суммой или разностью нескольких одночленов. Одночленом может быть целое число, а также произведение целого числа и целой положительной степени переменной (степень, равная единице, не записывается). Знак операции умножения в записи одночлена опускается. Например, полиномом является запись `X-15X↑3+2`. Указанный полином не содержит подобных одночленов, такие полиномы называются *приведёнными*.

В упрощённом решении задачи при записи исходных рациональных выражений можно заключить их в скобки, а знаки арифметических операций, числа и переменные разделять пробелами (чтобы упростить ввод и обработку выражений). Например, выражения можно задать так: `((7 + 15 - 3 X) / X)+(X /(5 X ↑ 8))`, и в результате будет вычислено выражение
`(110 X ↑ 8 - 15 X ↑ 9 + X ↑ 2) / (5 X ↑ 9)` , или с учётом упрощения: `(110 X ↑ 6 - 15 X ↑ 7 + 1) / (5 X ↑ 7)`

В полном решении задачи на вход программы поступают тексты рациональных выражений без лишних скобок, а числа, буквы переменных и знаки операций могут не разделяться пробелом, например: `X/5X↑8`.

Преобразование формулы алгебры логики в ДНФ

Преобразовать в *сокращенную дизъюнктивную нормальную форму* произвольную формулу алгебры логики, в которой используются логические константы, однобуквенные переменные и операции логического отрицания, конъюнкции, дизъюнкции и импликации. Результирующая дизъюнктивная нормальная форма должна быть *правильной*, т.е. каждая её элементарная конъюнкция не должна содержать повторных вхождений переменных, и все конъюнкции должны быть различными. Полученная формула не должна содержать избыточных скобок. Например, формула `(¬A∨D∨FALSE)⊃¬(B&¬C∨D)` преобразуется к виду `A&¬D∨¬B&¬D∨C&¬D`.

В упрощённом решении задачи при записи исходной логической формулы знаки логических операций, константы и переменные разделяются пробелами (если только между ними не стоит другой разделитель – круглая скобка), а сама формула заключается в объемлющие скобки (чтобы сделать её лисповским списком и упростить её ввод и преобразование), например: `(A ⊃ (B & ¬ C ∨ D))`

В полном решении задачи на вход программы поступает текст формулы, который обычно не содержит объемлющих скобок и в котором все символы могут быть записаны слитно.

Построение по ДНФ равносильной ей КНФ

По заданной дизъюнктивной нормальной форме некоторой булевской функции построить ее *совершенную конъюнктивную нормальную форму*, а также *сокращенную конъюнктивную нормальную форму*. Как исходная дизъюнктивная нормальная форма, так и результирующие конъюнктивные нормальные формы предполагаются *правильными*, т.е. все переменные, встречающиеся в каждой элементарной конъюнкции/дизъюнкции, различны, а также нет одинаковых элементарных конъюнкций/дизъюнкций. В *совершенной* КНФ каждая элементарная дизъюнкция содержит вхождения всех переменных булевской функции, в *сокращённой* форме это не требуется.

В исходной ДНФ используются только однобуквенные переменные. Построенные КНФ не должны содержать избыточных скобок. Например, для ДНФ `A∨B&¬C` получается сокращённая КНФ `(A∨B)&(A∨¬C)` и совершенная КНФ `(A∨B∨C)&(A∨B∨¬C)&(A∨¬B∨¬C)`.

В упрощённом решении задачи при записи исходной ДНФ знаки логических операций и переменные разделяются пробелами, а сама ДНФ заключается в объемлющие скобки (чтобы сделать её списком и упростить её ввод и обработку). В полном решении задачи на вход программы подаётся запись ДНФ без объемлющих скобок, в которой символы переменных и знаки операций могут быть записаны слитно.

Сколемизация формулы исчисления предикатов

Преобразовать в *предварённую нормальную форму* правильную замкнутую формулу исчисления предикатов первого порядка, в которой используются переменные, предикаты, кванторы общности и существования, а также логические операции (конъюнкция, дизъюнкция, импликация, отрицание). Переменные и предикаты исходной формулы предикатов имеют однобуквенные имена.

Преобразование включает следующие этапы:

1) исключение операции импликации;
2) уменьшение области действия операции отрицания;
3) исключение кванторов существования путем *сколемизации*, т.е. введения функций Сколема;
4) вынесение кванторов общности в начало формулы.

Получаемая в результате этих преобразований формула включает *префикс* – цепочку кванторов общности и *основу* (матрицу), не содержащую кванторы. В этой формуле должны быть удалены все избыточные (не влияющие на порядок выполнения операций) скобки.

Например, формула `∀x(P(x)⊃¬(∃y(Q(x,y)⊃P(y))))` преобразуется к виду: `∀x(¬P(x)∨Q(x,g(x)) & ¬P(g(x)))`, где `g(x)` – функция Сколема.

Предполагается, что в исходной формуле исчисления предикатов каждый квантор общности и существования связывает переменную с уникальным именем – поэтому для преобразования в предварённую нормальную форму нет необходимости производить переименование переменных с одним именем, но связываемых разными кванторами. Учесть также, что в формуле могут быть опущены незначащие скобки, т.е. скобки, не изменяющие порядка выполнения операций.

В упрощённом решении задачи при записи исходной формулы предикатов знаки логических операций, кванторов, имена переменных и предикатов разделяются пробелами, а сама формула заключается в объемлющие скобки (чтобы сделать её лисповским выражением и тем самым облегчить её ввод и преобразование). В полном решении задачи на вход программы подаётся запись формулы, в которой нет объемлющих скобок, а все символы имён переменных, предикатов, кванторов и операций могут быть записаны слитно.

Построение конечного автомата по грамматике

По заданному тексту *регулярной* (леволинейной или праволинейной) формальной грамматики построить соответствующий конечный автомат для распознавания предложений языка, порождаемого этой грамматикой.

Грамматика задаётся как конечный набор правил вида `T = aN|b`, альтернативы в правых частях правил не могут быть пустыми. Нетерминальные символы грамматики записываются заглавными (большими) латинскими буквами, а терминальные – строчными (маленькими). Начальный символ грамматики обозначается буквой `H` (для праволинейной) или `S` (для леволинейной грамматики).

Построенный автомат представляет собой ориентированный и помеченный граф. Вершины графа соответствуют состояниям автомата и помечены нетерминальными символами грамматики; в множество вершин входит начальное состояние `H` и заключительное состояние `S`. Рёбра графа соответствуют переходам между состояниями автомата и помечены терминальными символами грамматики. Граф записывается в виде списка входящих в него рёбер, каждое ребро представлено трехэлементным списком вида (*метка_вершины метка_ребра метка_вершины*).

В упрощённом решении задачи при записи исходной грамматики можно заключить в круглые скобки каждое её правило и составить из них лисповский список, в котором все символы грамматики разделены пробелами, а знак `|` и строчные буквы записаны как особые символы языка Лисп. Например:

```
((H = \a N \| \b N) (N = \c N \| \d)).
```

В полном решении задачи на вход программы подаётся текст грамматики без разделяющих пробелов, при этом правила грамматики разделены точкой с запятой, к примеру: `H=aN|bN;N=cN|d`. В случае недетерминированности полученного по грамматике автомата необходимо построить эквивалентный ему детерминированный автомат (детерминированный автомат, распознающий тот же самый язык).

Построение регулярной грамматики по конечному автомату

По заданному конечному автомату восстановить соответствующую *регулярную* леволинейную (или праволинейную) формальную грамматику, включающую алфавиты (множества) терминальных и нетерминальных символов и набор правил грамматики, а также начальный символ грамматики: `H` (для праволинейной) или `S` (для леволинейной) .

Конечный автомат задан как ориентированный граф, вершины которого соответствуют состояниям автомата и помечены нетерминальными символами грамматики. В множество вершин входит начальное состояние H и заключительное S. Рёбра графа соответствуют переходам между состояниями автомата и помечены терминальными символами грамматики. Граф записан как лисповский список входящих в него рёбер, каждое ребро представлено трёхэлементным списком вида (*метка_вершины метка_ребра метка_вершины*).

В записи восстановленной по конечному автомату грамматики нетерминальные символы грамматики записываются заглавными (большими) латинскими буквами, а терминальные – строчными (маленькими). Правая и левая часть каждого правила разделяются знаком =, а сами правила – знаком ; , в полученном тексте нет пробелов, например: `B=aN|bN;N=cN|d`.

Исходный конечный автомат может быть как детерминированным, так и недетерминированным. В полном решении задачи в случае недетерминированности автомата необходимо построить, кроме грамматики, эквивалентный ему детерминированный автомат (детерминированный автомат, распознающий тот же самый язык).

Исследование свойств графа

Для заданного графа провести исследование трёх его свойств, выбрав по одному свойству из трёх следующих групп:

1) – Связность графа: в случае связности графа необходимо найти его каркас, в ином случае – все его компоненты связности;

   – Древесность графа: является ли граф деревом или лесом, в последнем случае найти составляющие его деревья;

2) – Двудольность графа; если граф двудольный, то необходимо определить, является ли он полным двудольным графом;

   – Существование в графе мостов и точек сочленения, в случае существования найти их количество;

3) – Является ли граф эйлеровым, и если да, то найти эйлеров цикл;
   – Является ли он гамильтоновым, и если да, то найти гамильтонов цикл.

Каркас графа есть наименьшее по числу рёбер входящее в него дерево, сохраняющее связность между всеми его вершинами. Следует учесть, что в общем случае каркас в графе находится неоднозначно.

Дерево есть связный граф, не имеющий циклов. Совокупность несвязанных деревьев есть *лес*.

Граф называется *двудольным* (*биграфом*), если множество его вершин допускает разбиение на два непересекающихся подмножества (доли), причём каждое ребро графа соединяет вершины из разных долей. Двудольный граф есть *полный двудольный* граф, если каждая вершина одной доли соединена ребром с каждой вершиной другой доли. Граф двудолен тогда и только тогда, когда все простые циклы в нём имеют чётную длину.

Мостом называется ребро, удаление которого увеличивает число компонентов связности графа. *Точка сочленения* – вершина, удаление которой ведёт к увеличению количества компонентов связности;

Замкнутый путь (цикл) в графе называется *эйлеровым*, если он проходит через каждое ребро графа; при этом граф с таким циклом называется *эйлеровым*. Цикл в графе называется *гамильтоновым*, если он содержит все вершины графа ровно по одному разу; граф с таким циклом называется *гамильтоновым*.

Исследуемый граф записывается как лисповский список входящих в него рёбер, а каждое ребро – как двухэлементный список из меток его вершин. В качестве меток вершин используются буквенные идентификаторы или целые числа.

Поиск путей по карте дорог

Задана карта железнодорожных и шоссейных дорог между несколькими городами, все дороги между городами являются двусторонними. Не для всех возможных пар рассматриваемых городов существует железнодорожная или шоссейная дорога между ними, в то же время между некоторыми городами возможно наличие обоих типов дорог. Известно, что дорожное сообщение позволяет добраться из произвольного города в любой другой город.

Для двух заданных городов найти все связующие их пути без циклов, длина которых не более чем в полтора раза превышает минимальный по длине путь между указанными городами, и из этих путей выбрать путь с минимальным количеством пересадок с одного вида транспорта на другой.

В качестве результата вывести найденный минимальный по длине путь и путь с минимальным числом пересадок, указав в нём пересадки. Если таких путей несколько, то последовательно показать их все, упорядочив по длине пути. Если такого пути не существует, предложить любой другой приемлемый путь из одного города в другой.

Карта дорог задана в виде помеченного связного *мультиграфа*, вершины которого соответствуют городам и помечены названиями городов, а рёбра графа соответствуют дорогам между городами. Каждое ребро графа помечено меткой типа дороги (железнодорожная, шоссейная), а также целым числом – длиной дороги между соответствующими городами. Граф записан как список входящих в него рёбер, причём каждое ребро представлено четырёхэлементным списком, который включает названия городов, соединенных этим ребром (дорогой), а также метку типа дороги и её длину (расстояние между городами).