1 Введение

2 Описания языков программирования

2.1 C-подобная грамматика

2.2 Pascal-подобная грамматика

3 Оптимизации

3.1 В пределах базового блока

3.1.1 Def-Use информация о переменных

3.1.2 Устранение мертвого кода

3.1.3 Свертка констант и алгебраические тождества

3.1.4 Оптимизация общих подвыражений

3.1.5 Протяжка констант

3.1.6 Генерация 3-х адресного кода

3.1.7 Выделение базовых блоков

3.2 Построение графа потока управления

3.3 Итерационные алгоритмы

3.3.1 Достигающие определения

3.3.1.1 Вычисление множеств gen и kill

3.3.1.2 Реализация передаточной функции

3.3.1.2.1 Суперозиция передаточных функций команд

3.3.1.2.2 Общая формула

3.3.1.3 Реализация алгоритма

3.3.1.4 Тесты

3.3.2 Активные переменные

3.3.2.1 Анализ

3.3.2.2 Набор тестов

3.3.2.3 Оптимизации

3.3.3 Доступные выражения

3.3.3.1 Анализ

3.3.3.2 Набор тестов

3.3.3.3 Оптимизации

3.3.4 Распространение констант

3.3.5 Объединение итерационных алгоритмов

3.3.6 Отношение доминирования

3.3.6.1 Итерационный алгоритм

3.3.6.2 Построения графа доминатора

3.3.7 Изменение итерационного алгоритма

3.4 Анализ графа для алгоритма выделения областей

3.4.1 Обход в глубину с нумерацией

3.4.2 Построение остовного дерева

3.4.3 Классификация рёбер

3.4.4 Определить обратные рёбра в CFG

3.4.5 Определить приводимость графа

3.4.6 Определение всех естественных циклов

Постановка задачи

Определение всех естественных циклов

От каких задач зависит(на каких задачах основывается), входные данные

На вход алгоритму подаются:

- Список базовых блоков

```
IEnumerable<T> blocks
```

- Граф потока управления

```
IGraph<T> graph
```

- Список обратных дуг

```
List<DomGraph.ValPair<T>> reverseEdges
```

- Дерево доминатора

```
IDominatorRelation<T>
```

Данный алгоритм основывается на задачах, которые занимаются построением соотвествующих структур данных, за исключением IDominatorRelation, который предоставляется как интерфейс:

```
/// <summary>
///
/// </summary>
/// <typeparam name="T">      </typeparam>
public interface IDominatorRelation<T>
{
    /// <summary>
    ///
    /// </summary>
    /// <param name="a">        </param>
    /// <param name="b">        </param>
    /// <returns></returns>
    bool FirstDomSeccond(T a, T b);

    /// <summary>
    ///       ,
    /// </summary>
    /// <param name="a">    </param>
    /// <returns></returns>
    IEnumerable<T> UpperDominators(T a);
}
```

Для каких задач нужна(для каких задач является основой), выходные данные

Выходные данные алгоритма - список циклов:

```
List<Cycle>
```

Данная задача является основой для задачи “Определение вложенности естественных циклов”

Реализация(интерфейс т.е. классы, методы)

Шаблон типа необходим для проведения тестирования алгоритма с помощью типа int, но алгоритм предназначен для использования с типом Block.

Данный алгоритм реализуется классами:

1. AllCycles< T > - находит все циклы типа CycleUsual (цикл типа CycleSpecialCase распознается как 2 цикла типа CycleUsual)

```
public class AllCycles<T> where T : IComparable<T>
{
     public List<Cycle<T>> cycles { get; protected set; }
     public AllCycles(IEnumerable<T> blocks, IGraph<T> graph,
```

```
        List<DomGraph.ValPair<T>> reverseEdges,
               IDominatorRelation<T> domTree);
}
```

2. AllCyclesSpecialCase< T > - находит все циклы типа CycleUsual и CycleSpecialCase

```
public class AllCyclesWithSpecialCase<T> : AllCycles<T>
where T: IComparable<T>
{
     public AllCyclesWithSpecialCase(IEnumerable<T> blocks,
                IGraph<T> graph,
     List<DomGraph.ValPair<T>> reverseEdges,
            IDominatorRelation<T> domTree);
}
```

Тип Cycle< T > является абстрактным классом цикла, который реализуется 2 классами:

```
/// <summary>
///
/// </summary>
/// <typeparam name="T">      </typeparam>
public abstract class Cycle<T>
{
    /// <summary>
    ///
    /// </summary>
    public T N { get; set; }

    /// <summary>
    ///        ,
    /// </summary>
    public List<T> DATA { get; set; }

    /// <summary>
    ///     -
    /// </summary>
    public List<DomGraph.ValPair<T>> OUTS { get; set; }
}

/// <summary>
///
```

```
    /// </summary>
    /// <typeparam name="T">      </typeparam>
    public class CycleUsual<T>: Cycle<T>
    {
        public CycleUsual(T n, List<T> data,
                List<DomGraph.ValPair<T>> outs, T d)
        {
            this.N = n;
            this.DATA = data;
            this.OUTS = outs;
            this.D = d;
        }

        /// <summary>
        ///
        /// </summary>
        public T D { get; set; }
    }

    /// <summary>
    ///
    /// </summary>
    /// <typeparam name="T">      </typeparam>
    public class CycleSpecialCase<T> : Cycle<T>
    {
        public CycleSpecialCase(T n, List<T> data,
        List<DomGraph.ValPair<T>> outs, T d1, T d2)
        {
            this.N = n;
            this.DATA = data;
            this.OUTS = outs;
            this.D1 = d1;
            this.D2 = d2;
        }

        /// <summary>
        ///
        /// </summary>
        public T D1 { get; set; }

        /// <summary>
        ///
        /// </summary>
        public T D2 { get; set; }
    }
```

Других видов циклов не бывает.
List< Cycle< T > > - полиморфный контейнер, который хранит в себе 2 вида циклов.

Пример использования:

```
//
List<Block> block;
ControlFlowGraph graph;
List<DomGraph.ValPair<Block>> reverseEdges;
DomTree domTree;
//...
//
AllCycles<Block> allCyclesSpec =
    new AllCyclesWithSpecialCase<Block>(blocks, graph,
                reverseEdges, domTree);
AllCycles<Block> allCycles =
    new AllCycles<Block>(code.blocks, code.graph,
            reverseEdges, domTree);
//                "           "
List<Cycle<Block>> cyclesSpec = allCycles.cycles;
//                 "           "
List<Cycle<Block>> cycles = allCycles.cycles;
```

Описание(что делает и как реализовано)

- Алгоритм ищет все циклы в графе потока управления.
- Считает граф потока управления приводимым.
- Ищет все циклы в графе вне зависимости от их вложенности.
- Алгоритм раелизован с помощью рекурсивного обхода вглубину по напрвлению дуг, обратных графу потока управления. Обход начинается с каждой точки выхода из цикла, и заканчивается обходом всего цикла.

Тесты(если есть и если не являются другой задачей)

Тестовые структуры данных, реализующие соответствующие интерфейсы, а также классы примеров вы можете видеть на dgml диаграмме (см. рис. 1)

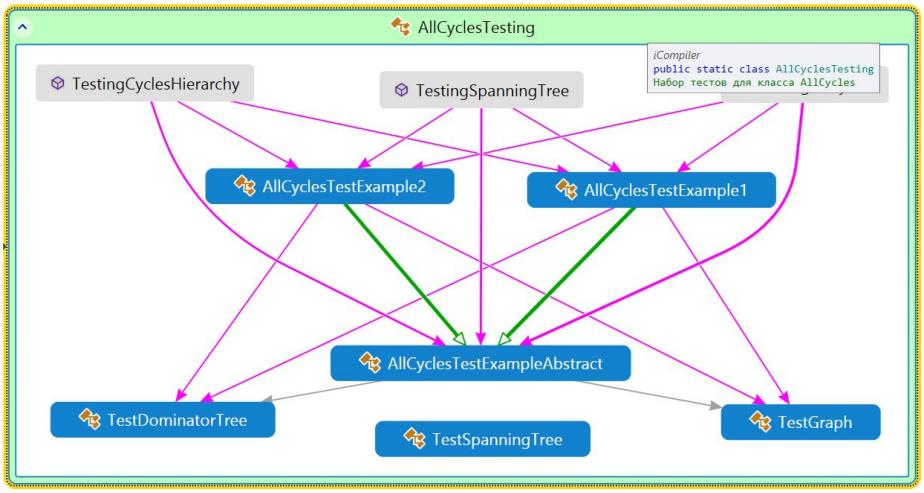

Рис. 1

Для прогона тестов необходимо вызвать

```
    AllCyclesTesting.TestingAllCycles();
```

Команда Альфа

3.4.7 Вложенность естественных циклов

3.4.8 Глубина CFG

3.5 Алгоритм выделения областей

3.5.1 Выделение областей

3.5.2 Реализация алгоритма на основе областей

4 Сопутствующие задачи

4.1 GUI

4.2 Генерация IL-кода

4.3 Документация

4.4 Подсчёт статистики

5 Заключение