А.В. Макаров, С.Ю. Скоробогатов, А.М. Чеповский

Учебный курс “CIL и системное программирование в Microsoft .NET”

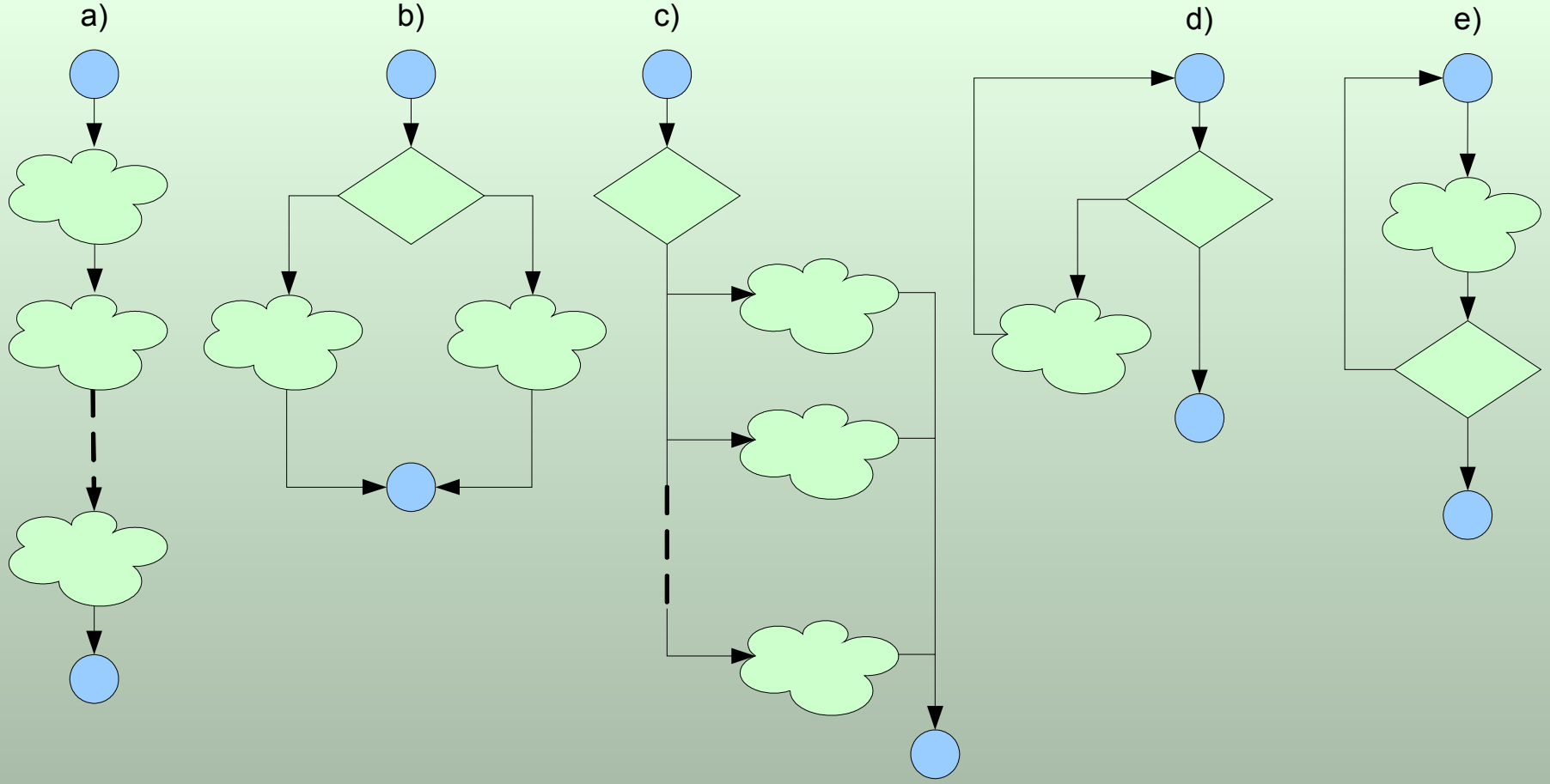
Лекция 18.

Генерация развилок

Введение

- Генерация кода, содержащего инструкции переходов, представляет некоторую сложность по сравнению с генерацией линейного кода:
  - Появляются переходы вперед по коду, то есть переходы на инструкции, которые еще не были сгенерированы
- Генерация развилок существенно упрощается, если в процессе генерации придерживаться определенной дисциплины, известной в программировании как структурный подход

Структурные конструкции



- Структурные конструкции удобны тем, что имеют ровно один вход и ровно один выход. Этот факт в сочетании с тем, что они вкладываются друг в друга, позволяет использовать для их порождения рекурсивные алгоритмы

18.1. Генерация кода для логических выражений

- Логические выражения отличаются от рассмотренных в прошлой лекции арифметических выражений тем, что могут вычисляться не полностью. Например, в выражении `(a = 10) and (sin(x) = 0.5)` второе равенство имеет смысл вычислять, только если первое равенство истинно
- Это означает, что в коде, вычисляющем логические выражения, должны активно использоваться условные переходы

18.1.1. Абстрактный синтаксис логических выражений

- Будем рассматривать логические выражения, содержащие арифметические выражения в качестве подвыражений
- Пусть также логические выражения содержат операции сравнения логические операции (логическое И, логическое ИЛИ, логическое НЕ)

```
LogExpr ::= Expr
          | LogExpr ComparisonOp LogExpr
          | LogExpr and LogExpr
          | LogExpr or LogExpr
          | not LogExpr.
ComparisonOp ::= equal | less | greater.
```

18.1.2. Отображение абстрактного синтаксиса логических выражений в CIL (слайд 1)

```
GenLogExpr[Expr] = GenExpr[Expr];

GenLogExpr[LogExpr1 ComparisonOp LogExpr2] =
    GenLogExpr[LogExpr1],
    GenLogExpr[LogExpr2],
    GenComparisonOp[ComparisonOp];

GenLogExpr[LogExpr1 and LogExpr2] =
    GenLogExpr[LogExpr1],
    dup,
    brfalse LABEL,
    GenLogExpr[LogExpr2],
    and,
    LABEL:  ;
```

Отображение абстрактного синтаксиса логических выражений в CIL (слайд 2)

```
GenLogExpr[LogExpr1 or LogExpr2] =
    GenLogExpr[LogExpr1],
    dup,
    brtrue LABEL,
    GenLogExpr[LogExpr2],
    or,
    LABEL:  ;
GenLogExpr[not LogExpr] =
    GenLogExpr[LogExpr],
    not;

ComparisonOp[equal] = ceq;
ComparisonOp[less] = нужный вариант clt;
ComparisonOp[greater] = нужный вариант cgt;
```

18.2. Генерация кода для управляющих конструкций

- Воспользовавшись уже отработанной схемой генерации кода, перейдем на уровень выше и рассмотрим генерацию основных структурных управляющих конструкций

18.2.1. Абстрактный синтаксис управляющих конструкций

- Рассмотрим абстрактный синтаксис для последовательности, выбора и циклов с предусловием и постусловием
- При записи абстрактного синтаксиса используется определенный ранее нетерминал LogExpr для представления условий выбора и циклов

```
Statement ::= Expr
            | if LogExpr StatementList
                 else StatementList
            | while LogExpr StatementList
            | do StatementList while LogExpr
StatementList ::= Statement StatementList
                | пусто
```

18.2.2. Отображение абстрактного синтаксиса управляющих конструкций в CIL (слайд 1)

```
GenStatement[Expr] =
    GenExpr[Expr],
    pop;

GenStatement[if LogExpr StatementList1
                  else StatementList2] =
    GenLogExpr[LogExpr],
    brfalse LABEL1,
    GenStatementList[StatementList1],
    br LABEL2,
    LABEL1: GenStatementList[StatementList2],
    LABEL2: ;
```

Отображение абстрактного синтаксиса управляющих конструкций в CIL (слайд 2)

```
GenStatement[while LogExpr StatementList] =
    LABEL1: GenLogExpr[LogExpr],
    brfalse LABEL2,
    GenStatementList[StatementList],
    br LABEL1,
    LABEL2: ;

GenStatement[do StatementList while LogExpr] =
    LABEL: GenStatementList[StatementList],
    GenLogExpr[LogExpr],
    brtrue LABEL;
```

Отображение абстрактного синтаксиса управляющих конструкций в CIL (слайд 3)

```
GenStatementList[Statement StatementList] =
    GenStatement[Statement],
    GenStatementList[StatementList];

GenStatementList[пусто] = ;
```

18.3. Оптимизация кода, содержащего развилки

- Рассмотрим несколько простых методов оптимизации кода, содержащего развилки, а именно:
  - удаление избыточных инструкций сохранения значений в переменных
  - удаление псевдонимов переменных
  - воспроизведение констант
  - удаление неиспользуемых переменных
- Хороших результатов можно достичь, если применять эти методы в совокупности с peephole-оптимизацией
- Получаемая цепочка оптимизирующих преобразований должна выполняться над одним и тем же кодом многократно до тех пор, пока не будет достигнута неподвижная точка

18.3.1. Удаление избыточных инструкций сохранения значений в переменных

- Это преобразование уменьшает количество присваиваний
- Оно осуществляется только для переменных, адреса которых не используются. Под переменными мы будем понимать как локальные переменные, так и параметры методов
- Для обнаружения избыточных инструкций сохранения значений выполняется анализ использования переменной, состоящий из двух фаз:
  - Построение графа использования переменной.
  - Анализ графа использования переменной

Инструкции использования переменных

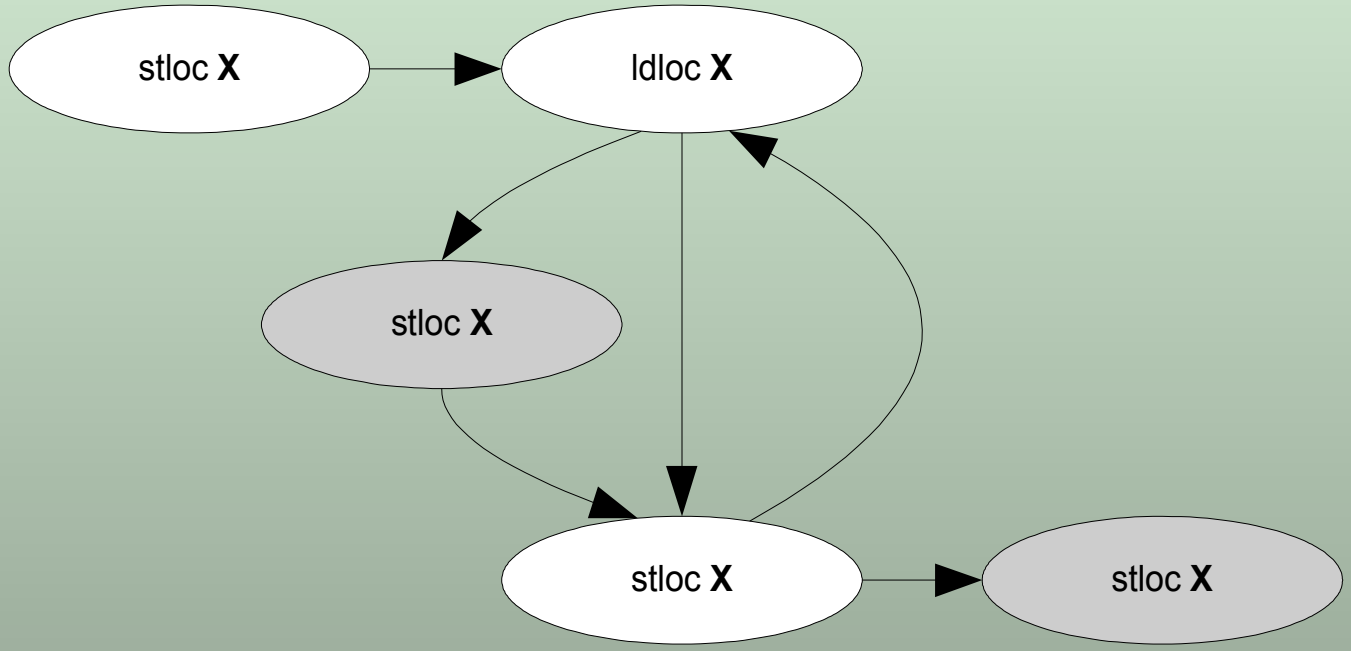
- Инструкции **ldloc**(**ldarg**) X и **stloc**(**starg**) X будем называть инструкциями использования переменной X
- Мы будем говорить, что в графе потока управления инструкция использования B следует за инструкцией использования A на пути w, если:
  - Инструкции A и B используют одну и ту же переменную X
  - Путь w соединяет A и B
  - Путь w не содержит ни одной инструкции использования переменной X, кроме инструкций A и B

Граф использования переменной

- Граф использования переменной X – это ориентированный граф, в узлах которого находятся инструкции использования переменной X, а дуги задают отношение следования для этих инструкций
- То есть если инструкция B следует за инструкцией A на каком-либо пути в графе потока управления, то в графе использования переменной X имеется дуга от инструкции A к инструкции B

Анализ графа использования переменной

- Анализ графа использования переменной заключается в нахождении таких инструкций **stloc**(**starg**), за которыми не следует ни одной инструкции **ldloc**(**ldarg**)
- Эти инструкции являются избыточными и заменяются инструкциями **pop**

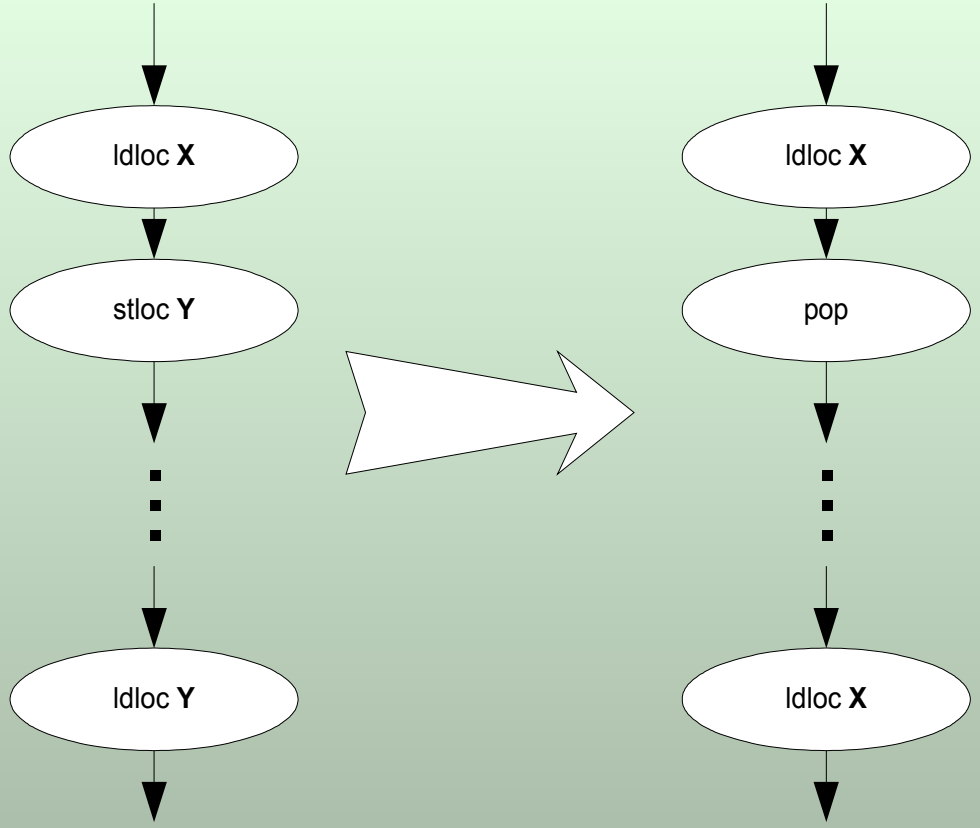


18.3.2. Удаление псевдонимов переменных (слайд 1)

- Это преобразование позволяет избавиться от лишних присваиваний и уменьшает количество локальных переменных
- Переменная Y является псевдонимом переменной X тогда и только тогда, когда:
  - Переменная X используется в теле метода только один раз, причем в инструкции **ldloc**(**ldarg**) X
  - За инструкцией **ldloc**(**ldarg**) X непосредственно следует инструкция **stloc**(**starg**) Y (впрочем, допускается наличие между ними любого количества инструкций **dup**). Причем инструкция **stloc**(**starg**) Y является первым использованием переменной Y (назовем ее инструкцией инициализации переменной Y)

Удаление псевдонимов переменных (слайд 2)

- При удалении осуществляются два действия:
  - Инструкция инициализации переменной Y заменяется инструкцией **pop**
  - Все использования переменной Y заменяются использованиями переменной X

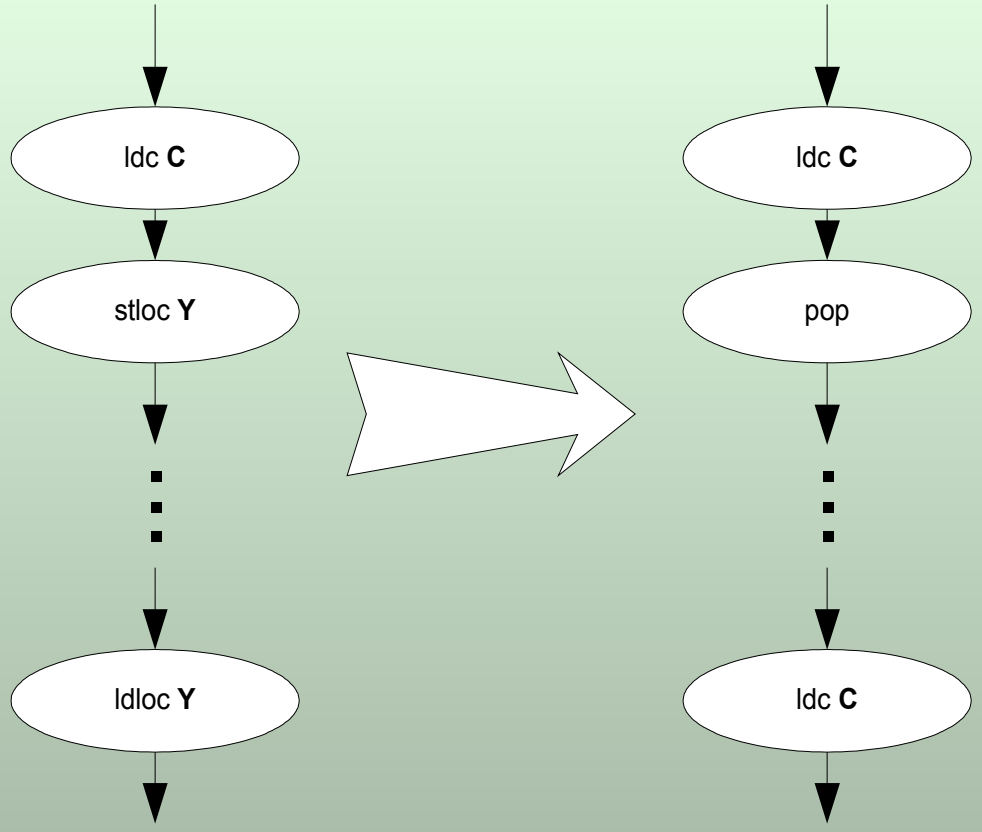


18.3.3. Воспроизведение констант (слайд 1)

- Это преобразование позволяет избавиться от переменных, имеющих константное значение
- Переменная Y имеет константное значение C тогда и только тогда, когда:
  - Первым использованием переменной Y является инструкция **stloc**(**starg**) Y (назовем ее инструкцией инициализации переменной Y)
  - Инструкция инициализации переменной Y непосредственно следует за инструкцией **ldc** C (впрочем, допускается наличие между ними любого количества инструкций **dup**)
  - За исключением инструкции инициализации, переменная Y используется только в инструкциях **ldloc**(**ldarg**) Y

Воспроизведение констант (слайд 2)

- При воспроизведении осуществляются два действия:
  - Инструкция инициализации переменной Y заменяется инструкцией **pop**.
  - Инструкции **ldloc** (**ldarg**) Y заменяются инструкциями **ldc** C



18.3.4. Удаление неиспользуемых переменных

- Если некоторая переменная не используется в графе метода или встречается только в инструкциях **stloc** (**starg**), то она удаляется
- При этом все инструкции **stloc**(**starg**), использующие эту переменную, заменяются инструкциями **pop**