О ПОЛИВАРИАНТНОМ АНАЛИЗЕ ВРЕМЕН СВЯЗЫВАНИЯ В СПЕЦИАЛИЗАТОРЕ ОБЪЕКТНО-ОРИЕНТИРОВАННОГО ЯЗЫКА

Ю.А. Климов

Введение

Рассмотрим программу f от двух аргументов x и y и значение одного из ее аргументов $x=a$. Результатом специализации программы f по известному аргументу $x=a$ называется новая программа g со следующим свойством: для любого y $f(a,y)=g(y)$.

Одним из методов специализации является метод частичных вычислений [3, 4], состоящий из двух фаз. На первой фазе, анализе времен связывания (binding-time analysis, BT-анализ), фрагменты программы классифицируются и размечаются как статические и динамические. На второй фазе, генерации остаточной программы (residual program generator), происходят "частичные вычисления": статические части программы вычисляются, а динамические переходят в остаточную программу.

Метод частичных вычислений хорошо развит для функциональных языков. Однако в реальных приложениях широко используются объектно-ориентированные языки, методы специализации для которых менее исследованы [1, 5, 6, 7]. В данной работе представлен поливариантный BT-анализ для объектно-ориентированных языков и описаны используемые в нем конфигурации. Этот анализ используется в специализаторе CILPE объектно-ориентированного языка CIL (Common Intermediate Language) платформы Microsoft.NET [2].

Анализ времен связывания

В процессе BT-анализа строится программа, в которой инструкции помечаются как статические и динамические. Для того чтобы генератор остаточной программы смог выполнить все статические инструкции, разметка должна быть корректной: данные, потребляемые статическими инструкциями, должны порождаться только статическими инструкциями. Поэтому для проведения анализа и проверки результата необходимо разметить переменные (аргументы и результаты инструкций). При этом статические инструкции должны потреблять только статические аргументы и возвращать только статические результаты.

Во многих программах одна и та же переменная используется в разных контекстах: в одном случае она хранит статическое значение, в другом – динамическое. Для того чтобы BT-анализ смог адекватно разметить такую программу, необходимо, чтобы разметка переменной зависела от точки программы. То есть BT-анализ должен к каждой точке программы приписать разметку всех переменных. Такая разметка называется *поливариантной* по переменным.

Если разметка переменных *моновариантна* по переменным (разметка переменной одинакова во всех точках программы), то результат разметки невозможно улучшить путем преобразования исходной программы (развертки цикла, дублирования кода в каждой ветви условия). А при использовании поливариантной разметки BT-анализ имеет возможность получать более качественную разметку, выполняя преобразования программы.

Рассмотрим немного измененный классический пример, показанный на рисунке 1. Функция *MaxToPower*(x,y,n) возводит в степень n наибольшее число из x и y. Специализируемая функция *Test* вызывает *MaxToPower*(2.0,y,5). В этом примере BT-анализ построит разметку отдельно для случая $x<y$, и отдельно для случая $x>=y$. В первом случае в цикле возведения в степень переменная x будет размечена как динамическая, а во втором случае переменная x будет размечена как статическая. Поэтому генератор остаточной программы в первом случае не сможет выполнить операции над x и они перейдут в остаточную программу, а во втором случае он сможет выполнить все операции и породить в остаточной программе только результат "32.0". В результате получится программа, в которой сначала сравнивается 2.0 и y, если 2.0 меньше y, то вычисляется y в пятой степени, иначе сразу возвращается 32.0.

```
[Inline]
double MaxToPower (double x,
double y, int n) {
  double res = 1.0;
  if (x < y)
    x = y;
  while (n != 0)
    if (n%2 == 1) {
      n--;
      res *= x;
    } else {
      n /= 2;
      x *= x;
    }
  return res;
}

[Specialize]
double Test (double y) {
  return MaxToPower(2.0, y, 5);
}
```

```
double Test (double y) {
  if (2.0 < y) {
    double v = y*y;
    return y*v*v;
  } else {
    return 32.0;
  }
}
```

Рис. 1. Исходная программа и результат специализации.

Если в языке присутствуют только примитивные типы, то BT-анализу достаточно в каждой точке программы разметить переменные как статические и динамические. Когда в языке присутствуют сложные типы данных (классы или структуры), то такой разметки недостаточно. Такая разметка не описывает случаи, когда объект статический (создается и используется во время генерации остаточной программы), а его поле динамическое (хранит данные, не известные во время генерации остаточной программы). Для описания подобной разметки вводится понятие *BT-конфигурации*.

BT-конфигурация

Рассмотрим упрощенный объектно-ориентированный язык, в котором есть только классы, переменные и стандартный набор операций над объектами (создание объекта, сравнение объектов по ссылке, чтение/запись поля и т.п.). Назовем конфигурацией K пятерку (VS,NS,BS,$V2N$,$N2B$), где VS – множество переменных, NS – множество имен BT-объектов, BS – множество BT-объектов, $V2N$:VS->NS – отображение переменных VS в имена BT-объектов NS, $N2B$:NS->BS – отображение имен NS в BT-объекты BS. Каждый BT-объект B – это тройка (C,T,$F2N$), где C – имя класса, T – BT-тип (статический S или динамический D), $F2N$:FS->NS – отображение полей FS класса C в имена BT-объектов NS. Пример конфигурации показан на рисунке 2.

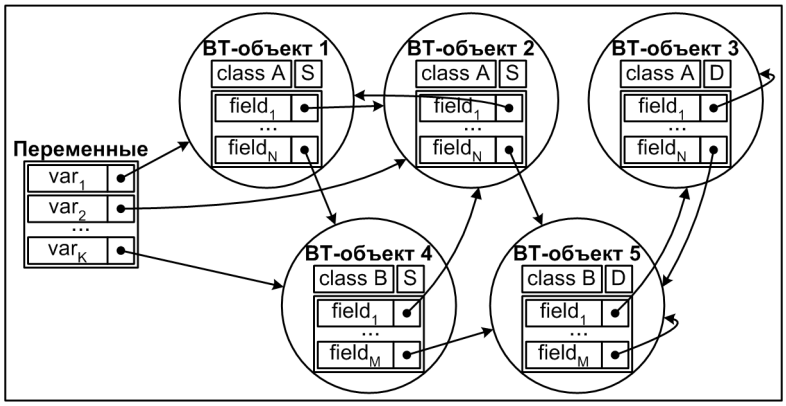

Рис. 2. BT-конфигурация.

Типы переменных и полей накладывают ограничения соответствия типов на отображения $V2N$ и $F2N$: если переменная V имеет тип C, то BT-объект O=$N2B$($V2N$(V)), соответствующий переменной, должен описываться классом C или его наследником; если поле F класса C BT-объекта O=(C,T,$F2N$) имеет тип C', то и BT-объект O'=$N2B$($F2N$(F)), соответствующий полю, также должен описываться классом C' или его наследником. BT-типы накладывают ограничение монотонности на отображения $F2N$: если BT-объект O=(C,T,$F2N$) имеет динамический BT-тип T=D, то для всех полей F класса C BT-объект B'=$N2B$($F2N$(F)) должен также иметь динамический BT-тип D.

В результате работы BT-анализа к каждой точке программы приписывается конфигурация, множество переменных которой совпадает с множеством переменных, доступных из данной точки программы. Опишем, как должны быть связаны две конфигурации, приписанные в точках программы, разделенных одной инструкцией. Для упрощения потребуем, чтобы BT-объекты BS, их имена NS и отображения имен в BT-объекты $N2B$ в этих конфигурациях были одинаковы: разрешается изменить только отображение $V2N$ переменных в имена BT-объектов и, если необходимо, список переменных.

Для описания отличий в отображении $V2N$ переменных в имена BT-объектов в разных конфигурациях, рассмотрим в качестве примера инструкцию чтения поля объекта: *varK=var1.field1*. Пусть

конфигурация перед инструкцией изображена на рисунке 2: переменной *var1* соответствует ВТ-объект с именем "1", а полю *field1* ВТ-объекта 1 соответствует ВТ-объект с именем "2". Тогда после выполнения инструкции переменной *varK* должен также соответствовать ВТ-объект с именем "2". Аналогичным образом описываются ограничения для всех остальных инструкций.

Несколько разных ВТ-объектов в конфигурации могут соответствовать одному классу (например, классу A соответствуют ВТ-объекты 1, 2 и 3). Это позволяет разные переменные одного типа разметить по-разному в зависимости от контекста их использования. А приписывание конфигурации к точкам программы позволяет одну переменную разметить по-разному в различных точках программы.

Проведение ВТ-анализа в терминах конфигурации позволяет избежать дополнительного "анализа на идентичность" (alias analysis): если переменным соответствуют разные ВТ-объекты, то во время исполнения в них будут находиться разные объекты.

Заключение

Используя конфигурации и поливариантный анализ времен связывания, был простроен специализатор CILPE для объектно-ориентированного языка CIL (Common Intermediate Language) платформы Microsoft.NET. В нем конфигурации были расширены на случай примитивных типов и массивов. В настоящий момент CILPE поддерживает лишь подмножество верифицируемого CIL: поддерживаются примитивные типы, объекты и массивы и инструкции по работе с ними; не поддерживаются структуры и указатели, исключения и блоки обработки исключений. В дальнейшем планируется полностью поддержать CIL в специализаторе CILPE.

ЛИТЕРАТУРА:

1. A.M. Chepovsky, A.V. Klimov, A.V. Klimov, Y.A. Klimov, A.S. Mishchenko, S.A. Romanenko, S.Y. Skorobogatov "Partial Evaluation for Common Intermediate Language" // Manfred Broy and Alexandre V. Zamulin (eds.), Perspectives of Systems Informatics, 5th International Andrei Ershov Memorial Conference, PSI 2003, Akademgorodok, Novosibirsk, Russia, July 9-12, 2003. Lecture Notes in Computer Science, Vol. 2890, Springer-Verlag, December 2003, pp.171-177
2. ECMA International: "Common Language Infrastructure (CLI), Standard ECMA-335", 2nd edition (December 2002) // http://www.ecma-international.org/publications/standards/ecma-335.htm
3. А.П. Ершов "О сущности трансляции" // Программирование №5, Москва, 1977, с.21-39
4. N.D. Jones C.K. Gomard, P. Sestoft "Partial Evaluation and Automatic Program Generation" // New York: Prentice-Hall, 1993.
5. Ю.А. Климов "Поливариантный анализ времен связывания в специализаторе CILPE для Common Intermediate Language платформы Microsoft.NET" // Труды Всероссийской конференции "Технологии Microsoft в теории и практике программирования", Москва: Издательство МГТУ им. Н.Э. Баймана, 2005, с.128
6. U.P. Schultz "Object-Oriented Software Engineering using Partial Evaluation" // PhD. Thesis. University of Rennes I, Rennes, France, 2000
7. V.F. Turchin "The concept of a supercompiler" // ACM Transactions on Programming Languages and Systems, 8, 1986, pp.292-325