ГЕНЕРАТОР ОСТАТОЧНОЙ ПРОГРАММЫ И КОРРЕКТНОСТЬ СПЕЦИАЛИЗАТОРА 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: $g(y)=f(a,y)$ для любого y.

Одним из методов специализации является метод частичных вычислений [6], состоящий их 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, статические части программы вычисляются, а динамические переходят в остаточную программу.

Метод частичных вычислений, как и смешанные вычисления [5] и суперкомпиляция [10], хорошо развит для функциональных языков [6]. Однако в реальных приложениях широко используются объектно-ориентированные языки, методы специализации для которых менее исследованы [1,2,3,7,8,9].

В данной работе представлен генератор остаточной программы специализатора объектно-ориентированного языка и дана схема доказательства корректности специализатора.

Генератор остаточной программы

По размеченной программе, построенной анализом времен связывания [6,7,8], и значениям статической части аргументов, генератор остаточной программы строит остаточную программу, зависящую только от динамической части аргументов.

Генератор остаточной программы производит "частичные вычисления": как и интерпретатор, он переходит с одной инструкции на другую и преобразовывает состояние. Но в отличие от интерпретатора, генератор не может выполнить все инструкции из-за неизвестности части аргументов. Поэтому генератор одновременно с вычислением строит остаточную программу.

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 объекта и т.п.).

Состояние интерпретатора такого языка *Si* это пятерка (*Vi*,*Ni*,*Oi*,*VNi*,*NOi*), где *Vi* – множество имен переменных, *Ni* – множество имен объектов, *Oi* – множество объектов, *VNi*:*Vi*->*Ni*+{*NULL*} – отображение имен переменных в имена объектов или в значение *NULL*, задающее значение переменных, *NOi* – отображение имен объектов в объекты (рис. 1). Каждый объект это пара (*C*,*FNi*), где *C* – имя класса, *FNi*:*F*->*Ni*+{*NULL*} – отображение имен полей класса *C* в имена объектов или в значение *NULL*, задающее значение полей объекта.

Состояние генератора остаточной программы *Sg*=(*Vg*,*Ng*,*Og*,*VNg*,*NOg*,*Wg*) почти повторяет состояние интерпретатора за двумя исключениями (рис. 1). Во-первых, оно дополнено *Wg* – множеством имен переменных остаточной программы. Во-вторых, значениями переменных и полей объектов кроме имен объектов *Ng* и значения *NULL* могут быть имена переменных остаточной программы *Wg*: *VNg*:*Vg*->*Ng*+*Wg*+{*NULL*} и *FNg*:*F*->*Ng*+*Wg*+{*NULL*} для каждого объекта (*C*,*FNg*).

В состоянии генератора вместо неизвестных значений записаны имена переменных остаточной программы. Если генератору остаточной программы необходимо выполнить действие с неизвестным значением, то он генерирует инструкцию, выполняющее то же действие с переменной остаточной программы, записанной вместо этого неизвестного значения.

Рассмотрим один шаг генератора остаточной программы на примере инструкции чтения поля объекта: *var2*=*var1*.*field1*. Возможны следующие варианты разметки переменных и состояний генератора:

1. Переменным *var1* и *var2* соответствуют BT-классы с BT-типом D. Значит, значения этих переменных – переменные остаточной программы *varA* и *varB* соответственно. В этом случае генератор остаточной программы не изменит своего состояния, а сгенерирует инструкцию чтения поля объекта для переменных остаточной программы: *varB*=*varA*.*field1*.

2. Переменной *var1* соответствует BT-класс с BT-типом S, а полю *var1*.*field1* этого BT-класса и переменной *var2* соответствует BT-класс с BT-типом D. Значит, значение переменной *var1* – объект, значение поля *var1*.*field1* – переменная остаточной программы *varA*, значение переменной *var2* – переменная *varB*. В этом случае генератор остаточной программы не изменит своего состояния и сгенерирует инструкцию чтения переменной: *varB*=*varA*.

3. Переменным *var1* и *var2* и полю *var1*.*field1* соответствуют BT-классы с BT-типом S. Значит, переменная и поле имеют значения имя объекта или *NULL*. В этом случае генератор только выполнит инструкцию чтения поля.

Действуя аналогичным образом на каждом шаге, генератор остаточной программы частично вычислит всю исходную программу и сгенерирует остаточную программу.

Доказательство корректности специализатора

Необходимо показать, что генератор остаточной программы построил правильную остаточную программу: результат ее выполнения совпадает с результатом выполнения исходной программы.

Для этого проведем параллельную интерпретацию исходной программы и генерацию и интерпретации остаточной программы. В этом процессе на каждом шаге с одной стороны будет состояние интерпретатора исходной программы. С другой – состояние генератора остаточной программы *Sg*=(*Vg*,*Ng*,*Og*,*VNg*,*NOg*,*Wg*) и состояние интерпретатора остаточной программы *Si*=(*Vi*,*Ni*,*Oi*,*VNi*,*NOi*), *Wg*=*Vi*.

Состояния генератора и интерпретатора остаточной программы можно объединить в одно общее состояние (рис. 1), объединив объекты и заменив значения переменных и полей объектов с имен переменных на значение этих переменных: *Sg*+*Si*=(*Vg*,*Ng*+*Ni*,*Og'*+*Oi*,*VNg'*,*NOg*+*NOi*), где *VNg'*:*Vg*->*Ng*+*Ni*+{*NULL*} – отображение, определенное по следующему правилу: пусть *var* – переменная, если *VNg*(*var*) – имя объекта или *NULL*, то *VNg'*(*var*)=*VNg*(*var*), если *VNg*(*var*) – имя переменной остаточной программы, то *VNg'*(*var*)=*VNi*(*VNg*(*var*)). Множество *Og'* определяется по множеству *Og* с заменой для каждого объекта отображения полей объекта *FNg* на отображение *FNg'*:*F*->*Ng*+*Ni*+{*NULL*} аналогично *VNg'*.

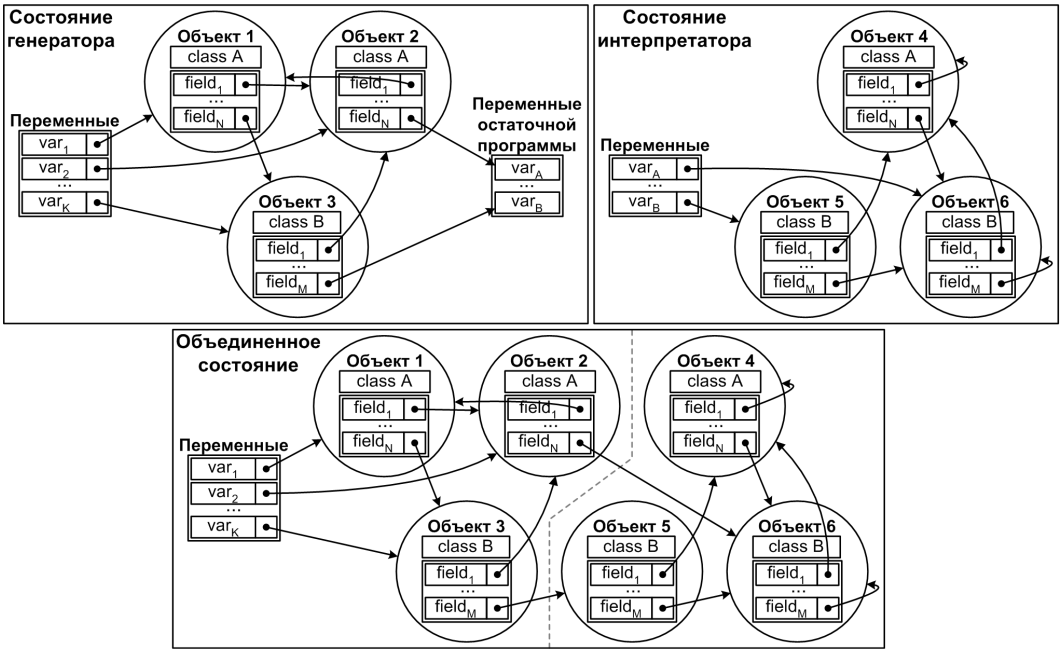

Рис. 1. Состояния генератора остаточной программы и интерпретатора остаточной программы. Объединенное состояние генератора и интерпретатора остаточной программы – состояние интерпретатора исходной программы.

Покажем индукцией по количеству шагов, что объединенное состояние генератора и интерпретатора остаточной программы совпадает с состоянием интерпретатора исходной программы на каждом шаге. В начальный момент они совпадают. Покажем на примере инструкции чтения поля объекта *var2=var1.field1*, что после каждого шага состояния будут совпадать. Возможны три описанные выше случая.

В первом случае состояние генератора не изменится, а состояния интерпретаторов исходной и остаточной программ изменятся одинаково: они выполнят инструкцию чтения поля объекта.

Во втором случае состояние генератора также не изменится, а состояния интерпретаторов изменятся следующим образом: интерпретатор исходной программы выполнит инструкцию *var2=var1.field1*, а интерпретатор остаточной программы – инструкцию *varB=varA*. Но так как в состоянии генератора остаточной программы полю *var1.field1* соответствует переменная *varA*, а переменной *var2* – переменная *varB*, то объединенное состояние генератора и интерпретатора остаточной программы изменится так же, как и состояние интерпретатора исходной программы.

В третьем случае состояние генератора остаточной программы и интерпретатора исходной программы изменятся одинаково: они выполнят инструкцию чтения поля объекта, а состояние интерпретатора остаточной программы не изменится.

Аналогично можно показать, что состояния изменяются одинаково на каждом шаге. Значит, состояния всегда будут совпадать при параллельной интерпретации исходной программы и генерации и интерпретации остаточной программы.

Заключение

В работе представлен генератор остаточной программы и продемонстрирована схема доказательства корректности генератора остаточной программы и специализатора. Описанный генератор

остаточной программы используется в экспериментальном специализаторе CILPE объектно-ориентированного языка CIL (Common Intermediate Language) платформы Microsoft.NET [4].

Работа выполнена при поддержке гранта РФФИ № 06-01-00574.

ЛИТЕРАТУРА:

1. K. Asai "Binding-Time Analysis for Both Static and Dynamic Expressions" // In A. Cortesi, G. File (eds.) Static Analysis, 6th International Symposium, SAS'99, Venice, Italy, September 1999. LNCS 1694, Springer-Verlag, 1999, pp.117-133
2. P. Bertelsen "Binding-time analysis for a JVM core language" // April 1999, ftp://ftp.dina.kvl.dk/pub/Staff/Peter.Bertelsen/bta-core-jvm.ps.gz
3. 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. LNCS 2890, Springer-Verlag, December 2003, pp.171-177
4. ECMA International "Common Language Infrastructure (CLI), Standard ECMA-335" // 3rd edition, June 2005, http://www.ecma-international.org/publications/standards/ecma-335.htm
5. А.П. Ершов "О сущности трансляции" // Программирование №5, Москва, 1977, с.21-39
6. N.D. Jones, C.K. Gomard, P. Sestoft "Partial Evaluation and Automatic Program Generation" // New York, Prentice-Hall, 1993
7. Ю.А. Климов "О поливариантном анализе времен связывания в специализаторе объектно-ориентированного языка" // Труды Всероссийской научной конференции "Научный сервис в сети ИНТЕРНЕТ: технологии распределенных вычислений", Новороссийск, 19-24 сентября 2005 г., Изд-во МГУ, с.89-91
8. 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, 17-18 февраля 2005 г., Изд-во МГТУ им. Н.Э. Баумана, 2005, с.128
9. U.P. Schultz "Object-Oriented Software Engineering using Partial Evaluation" // PhD Thesis, University of Rennes I, Rennes, France, 2000
10. V.F. Turchin "The concept of a supercompiler" // ACM Transactions on Programming Languages and Systems, 8, 1986, pp.292-325