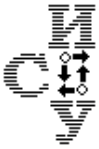

Заключение

В работе были рассмотрены модули онтологии и модели онтологии предметной области *Химия,* в которых определены термины, используемые при описании свойств физико-химических процессов. Данные термины используются при определении входных данных и результатов решения прикладных задач в интеллектуальной системе по химии [5].

ЛИТЕРАТУРА

1. *Клещев А.С., Артемьева И.Л.* Математические модели онтологий предметных областей. Ч.2. Компоненты модели // НТИ. Сер. 2. – 2001. – № 3. – С.19-29.
2. *Артемьева И.Л., Решетаненко Н.В*. Модульная модель онтологии органической химии // Информатика и системы управления. – 2004. – № 2. – С.98-108.
3. *Клещев А.С., Артемьева И.Л.* Необогащенные системы логических соотношений. В 2-х частях // НТИ. Сер. 2. – 2000. – № 7-8.
4. *Артемьева И.Л., Решетаненко Н.В., Цветников В.А*. Многоуровневая онтология химии // Тр. Всероссийской конф. с межд. участием "Знания-Онтологии-Теория" / Институт математики. Новосибирск, 2007. – Т.1. – С. 138-146.
5. *Артемьева И.Л., Решетаненко Н.В*. Разработка интеллектуальных Интернет-систем на основе многоуровневых онтологий // Системный анализ и информационные технологии: Тр. конф. – В 2 т. – М.: Изд-во ЛКИ, 2007. – Т.1. – С.96-99.

Статья представлена к публикации членом редколлегии А.С. Клещевым.

УДК 004.89

© 2008 г. **М.С. Маевский**
(Институт автоматики и процессов управления ДВО РАН, Владивосток)

ЯЗЫК ОПИСАНИЯ ПРЕОБРАЗОВАНИЙ ПРОГРАММ[1]

В статье представлен язык описания преобразований программ. Данный язык предназначен для формализации формул контекстных условий и формул трансформаций с целью их последующего хранения и автоматической обработки.

Введение

Методы преобразований программ широко используются при создании инструментальных средств поддержки программирования.

К преобразованиям программ относятся трансляция программ, их оптимизация и распараллеливание, реинжениринг программ при их переносе в новое

[1] Работа выполнена при финансовой поддержке ДВО РАН (инициативный научный проект "Интернет-система управления информацией о преобразованиях программ").

операционное окружение, модификация программ при их сопровождении и др.

Основная область применения преобразования программ – улучшение их качества.

Для проведения исследований в области компиляции в большинстве случаев, наиболее рациональным выходом для исследователей и разработчиков оказываются использование и модификация открытого компилятора. Разработка собственного компилятора требует больших затрат времени и ресурсов, а также знаний, которых у разработчиков может не быть. Покупка коммерческого компилятора приведет к значительным материальным затратам, что не всегда приемлемо.

Исследование преобразований в оптимизирующих компиляторах является непростой задачей. Во-первых, на работу скомпилированной программы всегда влияет много факторов, связанных с работой операционной системы, стандартных библиотек, драйверов – все это уменьшает эффект от оптимизации программы. Во-вторых, современные компиляторы имеют большое количество оптимизирующих преобразований. Основной вопрос, связанный с влиянием одних оптимизирующих преобразований на другие, становится очень сложным для исследования ввиду огромного количества возможных комбинаций оптимизирующий преобразований [1, 2].

В современных компиляторах и СПТ порядок выполнения оптимизирующих преобразований определяется фиксированной стратегией, набор методов потокового анализа и преобразований также является фиксированным. Анализ проблемной области показал, что нет систем, в которых бы можно было менять стратегии их преобразований, которые зависели бы от потокового анализа программ и от трансформаций.

В работе [3] для решения научных, практических и образовательных проблем в области преобразования программ была предложена концепция управления информацией о преобразованиях программ в рамках Специализированного банка знаний о преобразованиях программ (СБкЗ_ПП), основанная на результатах работы [4].

В данной статье предложена модель языка описания преобразований программ. Язык позволяет описывать формулы контекстных условий и формулы трансформации. Модель представлена фрагментом операционной семантики формулы контекстного условия. В качестве общей концепции, в рамках которой ведется реализация подсистемы проверки контекстных условий и поиска участков экономии управляемого знаниями [5], принята концепция многоцелевого компьютерного банка знаний [4] (http://www.iacp.dvo.ru/es/mpkbank).

Возможности языка описания оптимизирующих преобразований

Модель структурной программы, определенная в [5], является единым внутренним представлением программ в системе СБкЗ_ПП. Она представляется в виде графа. Расширенная МСП – это управляющий и информационный графы программы [6]. Расширение МСП –добавление к представлению программы специальных дуг управления и введение новых фрагментов программы в МСП в результате потокового анализа программ. Расширенная МСП является основой для

преобразования программ.

Знания о преобразованиях программ представлены в виде формул на языке описания преобразований программ (ЯОПП). Это логический язык программирования, позволяющий записывать формулы контекстного условия и формулы трансформации в виде формул.

Так как семантика поиска участков экономии и построение преобразованной МСП существенно различаются, хотя и используют одни и т.е. же термины онтологии, то описывать ЯОПП для контекстных условий и трансформации следует отдельно.

ЯОПП контекстных условий описывает *формулу КУ* – предикат, определяющий, является ли пара, составленная из кортежа адресов фрагментов и множества кортежей адресов фрагментов, участком экономии для данного преобразования.

ЯОПП трансформации описывает *формулу трансформации*, – предикат, сопоставляющий двум участкам экономии истину, если второй участок получен в результате выполнения оптимизирующего преобразования над первым.

База преобразований программ формируется экспертами предметной области. В нее могут входить как преобразования, взятые из различных источников, так и методы, разработанные непосредственно экспертом. Таким образом, формируется база знаний о преобразованиях программ в терминах ЯОПП, понятных для людей и удобных для машинной обработки.

Синтаксис и операционная семантика языка описания оптимизирующих преобразований

Синтаксис языка описания оптимизирующих преобразований приведен в нотации расширенной БНФ. Терминальные символы заключены в парные кавычки.

1. <Контекстное условие>::=
   "!<"<Вектор переменных>**">("**<Формулы>**[","**<Дополнительные условия>**]")"**

Семантика:

проц Контекстное_условие()
текущая формула КУ = первая формула КУ()
пока (следующая вершина КУ ≠ ложь) выполнять
зафиксировать значения переменных()
вычислить текущую формулу()
если (текущая формула = ложь) то
текущая формула КУ = первая зависимая формула КУ()
обновить значения переменных()

2. <Вектор переменных>::=
   <ПЕРЕМЕННАЯ>[","<ПЕРЕМЕННАЯ>]*
3. <Дополнительные условия>::=
   <Множественная формула> |
   <Отрицательная множественная формула>

4. <Множественная формула>::=
"!{<"<Вектор переменных>**">}("**<Условия>**")"**
5. <Отрицательная множественная формула>::=
"*{<"<Вектор переменных>**">}("**<Формулы>**")"**
6. <Формулы>::=
<Конъюнкция формул> |
<Дизъюнкция формул> |
<Отрицание формулы> |
<Формула в скобках>
7. <Конъюнкция формул>::=
<Формула>**"&"**<Формулы>

Семантика:

Если левая и правая формулы = истина, вернуть истину, иначе вернуть ложь.

8. <Дизъюнкция формул>::=
<Формула>**"||"**<Формулы>

Семантика:

Если левая и правая формулы = истина, вернуть истину, иначе вернуть ложь.

9. <Отрицание формулы>::=
"not"<Формула в скобках>

Семантика:

Вернуть противоположное логическое значение <формулы в скобках>.

10. <Формула в скобках> ::=
"("<Формулы>**")"**
11. <Формула>::=
<Атомарная формула>
12. <Атомарная формула>::=
<Формула над фрагментами> |
<Формула над множествами>
13. <Формула над фрагментами>::=
<Определение класса фрагмента> |
<Отношение над фрагментами> |
<Операция над фрагментами> |
<Функциональная зависимость>
14. <Определение класса фрагмента>::=
"FragClass("<Переменная>**") = "**<Класс фрагмента> |
Семантика:
Присвоить <Переменной> адрес первого непомеченного фрагмента МСП типа <Класс фрагмента>
"FragClass("<Переменная>**") in {"**<Список классов фрагментов>**"}"**
Семантика:
Присвоить <Переменной> адрес первого непомеченного фрагмента МСП любого типа входящего в <Список классов фрагментов>
15. <Список классов фрагментов>::=
<Класс фрагмента>[**","**<Класс фрагмента>]*

16. <Класс фрагмента>::=
"DSch" | **"DExpr"** | **"DBody"** | **"DIf"** | **"DFor"** | **"DWhile"** | **"DRepeat"** | **"DCall"** | **"DDispose"** | **"DAss"** | **"DInput"** | **"DOutput"** | **"DVdef"** | **"DFdef"** | **"DPdef"** | **"DVdefBody"** | **"DFdefBody"** | **"DPdefBody"** | **"DFr**
17. <Отношение над фрагментами>::=
<Вид отношения>**"("**<Список фрагментов>**")"**
18. <Вид отношения>::= **"TSimilarity"** |

Семантика:

Отношение подобия; связывает два фрагмента одного класса, структура которых совпадает с точностью до адресов фрагментов. Отношение выполняется в следующих случаях:

если у данного отношения 2 аргумента, все они фрагменты и их классы совпадают

либо оба аргумента являются пустыми фрагментами

либо оба аргумента не пустые и тогда:

оба аргумента являются операторами присваивания и их левые и правые части подобны

либо оба аргумента являются операторами ввода и их вводимые выражения подобны

либо оба аргумента являются операторами вывода и их выводимые выражения подобны

либо оба аргумента являются условными операторами и условные выражения, а также тела Then и Else подобны

либо оба аргумента являются операторами цикла For и их начальные границы, конечные границы, шаги циклов и тела подобны, а переменные цикла совпадают

либо оба аргумента являются операторами циклов While и Repeat и их условия и тела подобны

либо оба аргумента являются операторами вызова процедуры и идентификаторы вызываемых функций совпадают, а фактические параметры подобны

либо оба аргумента являются операторами описания переменной, параметра или функции и идентификаторы описываемых объектов совпадают

либо оба аргумента являются выражениями и тогда

либо в этих выражениях знаки операций не пусты и совпадают, а левые и правые части выражений подобны

либо знаки операций в этих выражениях пусты и значения атрибутов Left и LeftExpr совпадают

либо оба аргумента являются последовательностями, программными блоками, блоками описаний и их длины совпадают, а все соответствующие элементы подобны

19. **"TDirPreced"** |

Семантика:

Отношение непосредственного предшествования; связывает любые два оператора, которые стоят непосредственно друг за другом. Отношение выполняется в следующих случаях:
если у данного отношения 2 аргумента, оба являются фрагментами и
либо хотя бы один из аргументов является пустым фрагментом
либо они оба не пустые и тогда:

первый операнд связан со вторым дугой Next

либо первый операнд – оператор, а второй – непрерывная последовательность, и начало второго операнда непосредственно следует за первым операндом

либо левый операнд – непрерывная последовательность, а второй – оператор, и конец первого операнда непосредственно предшествует второму операнду

либо оба операнда – непрерывные последовательности, и начало второго операнда непосредственно следует за концом первого

20. "**TPrecedence**" |

Семантика:

Отношение предшествования; связывает любые два оператора, которые стоят друг за другом, необязательно непосредственно; между ними могут вклиниваться и другие операторы. Отношение выполняется в следующих случаях:
если у данного отношения 2 аргумента, все они фрагменты и
первый операнд непосредственно предшествует второму
либо есть транзитивное замыкание: существует фрагмент, следующий за первым операндом и предшествующий второму

21. "**TIsPart**" |

Семантика:

Отношение "являться частью"; связывает любые два фрагмента, причем второй входит в состав первого на верхнем уровне вложенности. Отношение выполняется в следующих случаях:
если у данного отношения 2 аргумента, все они фрагменты и
оба аргумента являются пустыми фрагментами
либо второй аргумент – пустой, а первый – нет
либо оба аргумента не пустые и тогда:

оба аргумента не являются последовательностями, блоками описаний переменных, функций и параметров, а также программными блоками и связаны некоторой дугой управления

либо первый операнд – последовательность, блок описаний переменных, функций и параметров или программный блок, а второй – фрагмент, и второй аргумент является элементом первого

либо первый операнд – последовательность, блок описаний переменных, функций и параметров или программный блок, а второй – последовательность и каждый элемент второго операнда является частью первого

либо оба операнда являются выражениями, причем у первого знак операции не пуст, и второй операнд является левым или правым подвыражением пер-

вого.

22. "**TIsSubModel**" |

Семантика:

Отношение "быть подмоделью"; связывает любые два фрагмента, причем второй входит в состав первого на любом уровне вложенности. Отношение выполняется в следующих случаях:

если у данного отношения 2 аргумента, все они фрагменты и

либо второй операнд “является частью" первого

либо есть транзитивное замыкание: существует фрагмент, который является подмоделью первого операнда и для которого второй операнд является подмоделью

23. "**TPrecedenceSubM**" |

Семантика:

Отношение "предшествование подмодели"; связывает любые два фрагмента, для которых фрагменты, чьими подмоделями они являются, находятся в состоянии предшествования. Отношение выполняется в следующих случаях:

если у данного отношения 2 аргумента, все они фрагменты и

либо оба аргумента являются пустыми фрагментами

либо оба аргумента не пусты, и тогда существует пара таких фрагментов, для которых операнды являются подмоделями, а сами они находятся в отношении предшествования

24. "**TCons**" |

Семантика:

Связывает 3 фрагмента, если третий фрагмент является последовательностью, а оба остальных фрагмента являются ее частями. По сути, это отношение является операцией конкатенации фрагментов. Отношение имеет место:

если у данного отношения 3 аргумента, все они фрагменты и класс третьего – последовательность и

либо все аргументы являются пустыми

либо первый аргумент – пуст, а остальные нет и при этом если второй аргумент – последовательность, то третий совпадает со вторым, иначе третий содержит второй как единственный элемент последовательности

либо второй аргумент – пуст, а остальные нет и при этом если первый аргумент – последовательность, то третий совпадает со первым иначе третий содержит первый как единственный элемент последовательности

либо все аргументы не пусты и тогда:

либо класс первых двух аргументов не принадлежит к последовательностям, блокам описаний функций, переменных, параметров и программным блокам, а третий аргумент является последовательностью из этих элементов

либо первый аргумент является последовательностью, блоком описаний функций, переменных, параметров или программным блоком, а второй – нет, а третий аргумент является результатом добавления второго в конец первого

либо второй аргумент является последовательностью, блоком описаний

функций, переменных, параметров или программным блоком, а первый – нет, а третий аргумент является результатом добавления первого в начало второго

либо если первый и второй аргументы являются последовательностями, блоками описаний функций, переменных, параметров или программными блоками, а третий аргумент является последовательностью – результатом их объединения.

25. "**TPred**" |

Семантика:

Связывает 3 фрагмента, причем третий фрагмент обязательно является последовательностью. Отношение имеет место, если второй аргумент является частью первого, а третий представляет собой последовательность фрагментов, лежащих между началами первого и второго аргументов.

если у данного отношения 3 аргумента, все они фрагменты, класс третьего – последовательность, второй и третий аргументы являются частью первого и:

либо все аргументы являются пустыми

либо второй аргумент пуст, а остальные – не пусты и тогда:

либо первый и третий аргументы совпадают друг с другом

либо если первый аргумент является блоком, то третий представляет собой последовательность из всех его элементов

либо третий аргумент пуст, а остальные не пусты и тогда:

либо первый и второй аргументы совпадают друг с другом

либо второй аргумент является первым элементом первого

либо все аргументы не пусты и тогда:

либо первый аргумент – блок или непрерывная последовательность, второй аргумент – простой оператор или непрерывная последовательность, а третий аргумент – непрерывная последовательность и ее начало совпадает с началом первого аргумента, а конец - непосредственно предшествует второму аргументу

либо первый аргумент – последовательность, второй аргумент – простой оператор, а третий аргумент – последовательность, состоящая из всех элементов первого аргумента, предшествующих второму аргументу

либо первый и второй аргументы – выражения, а третий аргумент – не непрерывная последовательность выражений, являющаяся результатом разложения первого аргумента на последовательность подвыражений, вычисляемых до выражения-второго аргумента и расположенных в порядке вычисления

26. "**TPosl**" |

Семантика:

Связывает 3 фрагмента, причем третий фрагмент обязательно является последовательностью. Отношение имеет место, если второй аргумент является частью первого, а третий представляет собой последовательность фрагментов, лежащих между концами второго и первого аргументов.

если у данной дуги 3 аргумента, все они фрагменты, класс третьего – последовательность, второй и третий аргументы являются частью первого и:

либо все аргументы являются пустыми

либо второй аргумент пуст, а остальные – не пусты и тогда:

либо первый и третий аргументы совпадают друг с другом

либо если первый аргумент является блоком, то третий представляет собой последовательность из всех его элементов

либо третий аргумент пуст, а остальные не пусты и тогда:

либо первый и второй аргументы совпадают друг с другом

либо второй аргумент является последним элементом первого

либо все аргументы не пусты и тогда:

у данной дуги 3 аргумента, все они фрагменты, класс третьего – последовательность, второй и третий аргументы являются частью первого и:

либо первый аргумент – блок или непрерывная последовательность, второй аргумент – простой оператор или непрерывная последовательность, а третий аргумент – непрерывная последовательность, ее конец совпадает с концом первого аргумента а начало непосредственно следует за окончанием второго аргумента

либо первый аргумент – последовательность, второй аргумент – простой оператор, а третий аргумент – последовательность, состоящая из всех элементов первого аргумента, последующих за вторым аргументом

либо первый и второй аргументы – выражения, а третий аргумент – не непрерывная последовательность выражений, являющаяся результатом разложения первого аргумента на последовательность подвыражений, вычисляемых до выражения-второго аргумента и расположенных в порядке вычисления

27. "**TBetween**" |

Семантика:

Отношение связывает 4 фрагмента. Это отношение имеет место, если второй и третий аргументы являются частью первого и второй предшествует третьему, а четвертый представляет собой последовательность фрагментов, находящихся между вторым и третьим аргументами.

если у данной дуги 4 аргумента, все они фрагменты, первый аргумент - блок или последовательность, второй, третий и четвертый аргументы являются частью первого, второй предшествует третьему и:

либо все аргументы являются пустыми

либо четвертый аргумент пуст, а остальные – не пусты и тогда:

либо второй и третий аргументы совпадают друг с другом

либо второй аргумент непосредственно предшествует третьему

либо все аргументы не пусты и четвертый аргумент является последовательностью, состоящей из всех элементов первого аргумента, лежащих между вторым и третьим аргументом

28. <Список фрагментов>::= <Фрагмент>[**","** <Фрагмент>]*

Пример представления контекстного условия на ЯОПП

В качестве примера контекстного условия, записанного на этом языке, служит оптимизирующее преобразование "Нормализация циклов".

Неформальное контекстное условие:
участком экономии в данном случае является цикл For, у которого шаг цикла вычислим и положителен, заголовок цикла не зависит от тела цикла и результатное множество заголовка цикла пусто.

Формальное контекстное условие на ЯОПП:
∃<Y, E1, E2, E3,B>(FragClass(Y)=DFor & FragClass(E1)=DExpr & FragClass(E2)=DExpr & FragClass(E3)=DExpr & FragClass(B)=DExpr & E1=For(Y) & E2=Step(Y) & E3=Until(Y) & B=Body(Y) & Eval(E2, Null, Null)<>Null & Eval(E2, >, 0)=True & [R(E1)∪R(E2)∪R(E3)=∅] & [(A(E1)∪A(E2) ∪A(E3))∩R(B)= ∅],

∃{<K>}(FragClass(K)=Dexpr & Left(K)=Var & LeftExpr(K)=Par(Y))
)

Нетерминальные математические кванторы сохранены для удобства пользователя, работающего через редактор.

Пусть Y – цикл, E1, E2, E3 – его границы и шаг приращения, B – тело цикла, K – множество фрагментов – подвыражений в теле цикла, ссылающихся на переменную цикла. При этом выражение E2 вычислимо до выполнения и >0.

Простые параметры: Y, E1, E2, E3, B.

Множественные параметры: K.

Введем псевдонимы и определим типы: Y – первый простой фрагмент; E1 – второй простой фрагмент; E2 – третий простой фрагмент; E3 – четвертый простой фрагмент; B – пятый простой фрагмент;
FragClass(Y)=DFor & FragClass(E1)=DExpr & FragClass(E2)=DExpr & FragClass(E3)=DExpr & FragClass(B)=DExpr & E1=For(Y) & E2=Step(Y) & E3=Until(Y) & B=Body(Y) & Eval(E2, Null, Null)<>Null & Eval(E2, >, 0)=True & [R(E1)∪R(E2) ∪R(E3)=∅] & [(A(E1)∪A(E2) ∪A(E3))∩R(B)= ∅].

Y является оператором цикла For, E1, E2, E3 – выражения из заголовка цикла, B – тело цикла, E2 – вычислимо и положительно, заголовок цикла не имеет побочных эффектов и тело цикла на него не влияет.

Введем псевдонимы: K – первый множественный параметр, является выражением, причем его левая часть является переменной – параметром цикла.

FragClass(K)=Dexpr & Left(K)=Var & LeftExpr(K)=Par(Y)

Заключение

В данной работе представлен фрагмент формальной модели ЯОПП для контекстных условий. На примере было продемонстрировано, каким образом при помощи описанного языка можно определять контекстные условия. В настоящее время разработан преобразователь программ, управляемый знаниями на ЯОПП, в рамках системы преобразований программ в СБкЗ_ПП.

ЛИТЕРАТУРА

1. GNU Compilers Collection 3.3.2. http://gcc.gnu.org/onlinedocs/gcc-3.3.2/gcc/.
2. *Bacon D.F., Graham S.L., Sharp O.J.* Compiler transformations for high-performance computing //ACM Computing Surveys. –1994. – V.26, № 4. – P.345-420.
3. *Клещев А.С., Князева М.А.* Управление информацией о преобразованиях программ. I. Анализ проблем и пути их решения на основе методов искусственного интеллекта //Изв. РАН. ТиСУ. – 2005. – № 5. – С.120-129.
4. *Орлов В.А., Клещев А.С.* Компьютерные банки знаний. Многоцелевой банк знаний // Информационные технологии. – 2006. – № 2. – С.2-8.
5. *Князева М.А., Купневич О.А.* Модель онтологии предметной области "Оптимизация последовательных программ". Определение языка модели структурных программ // НТИ. Сер. 2. – 2005. – № 2. – С.17-21.
6. *Князева М.А., Купневич О.А.* Модель онтологии предметной области "Оптимизация последовательных программ". Определение расширения языка модели структурных программ терминами потокового анализа // НТИ. Сер. 2. – 2005. – № 4.

Статья представлена к публикации членом редколлегии А.С. Клещевым.