Интеллектуальные системы
2007. №1(13)

УДК 004.89

© 2007 г. **Д.А. Волков**
(Институт автоматики и процессов управления ДВО РАН, Владивосток)

ЯЗЫК ОПИСАНИЯ МЕТОДОВ ПОТОКОВОГО АНАЛИЗА[1]

В статье представлен язык описания методов потокового анализа. Данный язык предназначен для формализации методов потокового анализа с целью их последующего хранения и автоматической обработки.

Введение

Основной проблемой в сфере науки о преобразованиях программ является невозможность своевременно выполнять компьютерные эксперименты. Цель таких экспериментов – определить, насколько часто в реальных программах применимы те или иные преобразования, какой эффект дает их применение и какая стратегия оптимизирующих преобразований является наилучшей для заданного набора оптимизирующих преобразований. В настоящее время единственное средство проведения подобных экспериментов – оптимизирующие компиляторы [1, 2]. Однако период времени, который обычно проходит от момента публикации описания нового преобразования до момента окончания реализации оптимизирующего компилятора, содержащего в своем наборе данное преобразование (если такой компилятор разрабатывается), настолько велик, что результаты компьютерных экспериментов с этим преобразованием оказываются мало актуальными. Кроме того, оптимизирующий компилятор обычно содержит большой набор преобразований и встроенную стратегию их применения. Поэтому получить достоверные результаты компьютерных экспериментов, относящихся к конкретному преобразованию (а не ко всему набору) или иной стратегии, невозможно.

Отсутствие средств для проведения экспериментов приводит к тому, что в оптимизирующие компиляторы включаются преобразования и стратегии применения преобразований с не полностью известными свойствами, что отрицательно сказывается на их производстве. Поэтому актуаль-

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

ным является поиск подходов к созданию системы для экспериментов в области преобразований программ, предназначенный для решения вышеперечисленных выше проблем. Основная идея, которая в работе положена в основу такой схемы, – применение методов искусственного интеллекта в области преобразований программ.

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 управления информацией о потоковом анализе программ управляемым знаниями, являющемся средством получения достоверной информации о поведении программы без ее исполнения в системе преобразований программ в СБкЗ_ПП.

В качестве общей концепции, в рамках которой ведется реализация системы преобразований программ с потоковым анализом программ управляемым знаниями, является концепция многоцелевого компьютерного банка знаний [4] (http://www.iacp.dvo.ru/es/mpkbank).

Возможности языка методов потокового анализа программ

Модель структурной программы, определенная в [5], является единым внутренним представлением, на котором проходит потоковый анализ программ. 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, которые появляются в результате потокового анализа программ. Расширенная МСП является основой для преобразования программ. На множестве фрагментов и идентификаторов программы определены функции и отношения, которые позволяют задать некоторые свойства программы. Функции с одним аргументом называются атрибутами.

Язык описания методов потокового анализа программ (язык МПА) разработан с целью формализации процесса расширения МСП. При разработке языка были проанализированы основные формы записи методов потокового анализа, описанные в литературе[7,8].

Язык позволяет описывать конструкции, которые являются результатом потокового анализа программы: атрибуты вершин графа; отношения и функции над вершинами графа; вершины и дуги графа.

Язык описания методов потокового анализа содержит переменные, которые в качестве значений могут принимать ссылки на различные элементы модели программы; основные конструкции алгоритмических языков программирования (такие как цикл, выбор, присваивание); операции с множествами, поскольку в процессе накопления информации о программе

часто оперируют со множествами переменных и идентификаторов; операции обходов деревьев.

База методов потокового анализа формируется экспертом предметной области. В нее могут входить как методы, взятые из различных источников (описанные в литературе, в сети Интернета и т.д.), так и методы, разработанные непосредственно экспертом.

Синтаксис и операционная семантика языка методов потокового анализа программ

Синтаксис языка методов потокового анализа описан в нотации расширенной БНФ. Терминальные символы заключены в парные кавычки.

1. <Метод потокового анализа> ::= “Метод_потокового_анализа” “(“ < Название метода > “)” <Блок объявления переменных> <Последовательность конструкций>

Семантика:

проц Метод_потокового_анализа()
 первая вершина дерева МСП
 пока (следующая вершина дерева МСП ≠ ложь) выполнять
  если вершина=«*Конструкция*» то вызов Конструкция()
  если вершина=«*Фрагмент по дуге*» то вызов Фрагмент_по_дуге()
  если вершина=«*Атрибут фрагмента*» то вызов Атрибут_фрагмента()
  если вершина=«*Получить класс*» то вызов Получить_класс()
  если вершина=«*Получить переменную выражения*» то вызов Получить_ переменную_выражения()
  если вершина=«*Первый фрагмент по дуге из последовательности*» то вызов Первый_фрагмент_по_дуге_из_последовательности()
  если вершина=«*Следующий фрагмент по дуге из последовательности*» то вызов Следующий фрагмент по дуге из последовательности()
  если вершина=«*Пересечение множеств*» то вызов Пересечение множеств()
  если вершина=«*Объединение множеств*» то вызов Объединение множеств()
  если вершина=«*Равенство множеств*» то вызов Равенство_множеств()
  если вершина=«*Логическая формула*» то вызов Логическая_формула()
  если вершина=«*Составная логическая формула*» то вызов Составная_логическая_формула()
  если вершина=«*Терм логической формулы*» то вызов Терм_логической_формулы()
  если вершина=«*Знак логической операции*» то вызов Знак_логической_операции()
  если вершина=«*Обход дерева программы*» то вызов Обход_дерева_ программ()
  если вершина=«*Обход дерева выражения*» то вызов Обход_дерева_ вы-

ражения()
если вершина=«*Выбор*» то вызов Выбор()
если вершина=«*Цикл*» то вызов Цикл()
если вершина=«*Создание фрагмента*» то вызов Создание_фрагмента()
если вершина=«*Создание атрибута*» то вызов Создание_атрибута()
если вершина=«*Изменение атрибута*» то вызов Изменение_атрибута()
если вершина=«*Создание дуги*» то вызов Создание_дуги()
если вершина=«*Создание отношения*» то вызов Создание_отношения()
если вершина=«*Создание переменной*» то вызов
Создание_переменной()
если вершина=«*Тип переменной*» то вызов Тип_переменной()
если вершина=«*Значение переменной*» то вызов Значение_переменной()
если вершина=«*Присваивание*» то вызов Присваивание()
если вершина=«*Выражение*» то вызов Выражение()
если вершина=«*Выражение в скобках*» то вызов Выражение_в_скобках()
следующая вершина дерева МСП

2. <Название метода> ::= <Строка>
3. <Строка> ::= <Буква> | <Строка> <Буква> | <Строка> <Цифра>
4. <Буква> ::= А | ... | Я | а | ... | я| - | _ |
5. <Цифра> ::= 0 | 1 … | 9
6. <Последовательность конструкций> ::= “{” [<Конструкция>] “}”
7. <Блок объявления переменных> ::= [<Объявление переменной>]
8. <Объявление переменной> ::= <Тип переменной> “:” (<Переменная-фрагмент> | <Переменная-атрибут> | <Переменная-дуга> | <Переменная-отношение> | <Переменная> [“,”]) “;”
9. <Тип переменной> ::= “Переменная-фрагмент” | “Переменная-атрибут” | “ Переменная-дуга ” | “ Переменная-отношение ” | “Целое” | “Вещественное”
10. <Конструкция> ::= <Формула> | <Обход> | <Выбор> | <Цикл> | <Присваивание> | <Модификация программы>
11. <Формула> ::= <Формула над фрагментом> | <Формула над множеством> | <Логическая формула>
12. <Обход> ::= <Обход дерева программы> | <Обход дерева выражения>
13. <Выбор> ::= “Если” “(” <Логическая формула> ”)” “То” < Последовательность конструкций > [“Иначе” <Последовательность конструкций>]

Семантика: Если значение Условия есть истина, то выполнить последовательность Конструкций Если истина, в противном случае выполнить последовательность Конструкций Если ложь.

14. <Цикл> ::= “Пока” <Условие> <Последовательность конструкций>

Семантика:

Выполнение Цикла есть выполнение следующих шагов:
Шаг1: Проверить Условие. Если значение Условия есть истина, выполнить Шаг2.
Шаг2: Выполнить последовательность Конструкций, вернуться к Шаг1.

15. <Присваивание> ::= <Левая часть присваивания> “=” <Правая часть присваивания>
16. <Модификация программы> ::= <Создание фрагмента> | <Создание атрибута> | <Изменение атрибута> | <Создание дуги> | <Создание отношения> | <Создание переменной>
17. <Формула над фрагментом> ::= <Фрагмент по дуге> | <Атрибут фрагмента> | <Получить класс> | <Получить переменную выражения> | <Первый фрагмент по дуге из последовательности> | <Следующий фрагмент по дуге из последовательности>
18. <Фрагмент по дуге> ::= “Фрагмент_по_дуге” “(” <Переменная-фрагмент>, <Имя дуги>, <Переменная-фрагмент> “)”

Семантика: Присвоить переменной Результат-фрагмент фрагмент, к которому от Аргумент-фрагмента ведет дуга Имя дуги.

19. <Атрибут фрагмента> ::= “Атрибут_фрагмента” “(” <Переменная-фрагмент>, <Имя атрибута>, <Переменная-атрибут> “)”

Семантика: Присвоить переменной Результат-атрибут атрибут фрагмента Аргумент-фрагмент с именем Имя атрибута.

20. <Получить класс> ::= “Получить_класс” “(” <Переменная-фрагмент>, <Класс фрагмента>“)”

Семантика: Присвоить Результату-классу фрагмента класс фрагмента Аргумент-фрагмент.

21. <Получить переменную выражения> ::= “Получить_переменную_выражения” “(” <Переменная-фрагмент>, <Переменная> “)”

Семантика: Если Аргумент-фрагмента является терминальной вершиной бинарного дерева выражения МСП, то присвоить Результату-переменной переменную, которая является левой частью Выражения, в противном случае присвоить Результату-переменной пустое множество.

22. <Первый фрагмент по дуге из последовательности> ::= “Первый_фрагмент_по_дуге_из_последовательности” “(” <Переменная-фрагмент>, <Переменная-фрагмент > “)”

Семантика: Присвоить Результат-фрагменту фрагмент к которому от фрагмента Аргумент-фрагмент ведет первая дуга Оператор.

23. <Следующий фрагмент по дуге из последовательности> ::= “Следующий_фрагмент_по_дуге_из_последовательности” “(” < Переменная-фрагмент >, < Переменная-фрагмент >, < Переменная-фрагмент > “)”

Семантика: Присвоить Результат-фрагменту фрагмент к которому от фрагмента Аргумент-фрагмент ведет дуга Оператор следующая за дугой,

которая ведет к Аргумент-фрагменту2.

24. <Формула над множеством> ::= <Пересечение множеств> | <Объединение множеств> | <Равенство множеств>
25. <Пересечение множеств> ::= “Пересечение_множеств” “(” <Переменная-множество> <Переменная-множество> <Переменная-множество> “)”

Семантика: Пересечь множества Аргумент-множество и Аргумент-множество2 и присвоить полученное множество переменной Результат-множество.

26. <Объединение множеств> ::= “Объединение_множеств” “(” <Переменная-множество> <Переменная-множество> <Переменная-множество> “)”

Семантика: Объединить множества Аргумент-множество и Аргумент-множество2 и присвоить полученное множество переменной Результат-множество.

27. <Равенство множеств> ::= “Равенство_множеств” “(” <Аргумент-множество> < Переменная-множество> <Булево множество> “)”

Семантика: Если множества Аргумент-множество и Аргумент-множество2 пусты, либо содержат одинаковые элементы присвоить Результату равенства значение истина, в противном случае присвоить Результату равенства значение ложь.

28. <Логическая формула> ::= <Терм логической формулы>

Семантика: Присвоить аргументу Булево множество истину или ложь в зависимости от результата вычисления логической формулы, которая задается Термом логической формулы

29. <Составная логическая формула> ::= <Терм логической формулы> <Знак логической операции> <Терм логической формулы>
30. <Терм логической формулы> ::= <Составная логическая формула> | <Булево множество> | <Равенство множеств> | <Класс фрагмента> | <Имя дуги> | <Переменная-фрагмент> | <Переменная-атрибут> | <Переменная-дуга> | <Переменная-отношение> | <Имя атрибута> | <Имя отношения> | <Переменная>

Семантика: Если Терм логической формулы является терминальной вершиной логической формулы, то его значение является либо Булево множество, либо Класс фрагмента, либо Имя дуги, либо Переменная, либо Переменная-фрагмент, либо Переменная-атрибут, либо Переменная-дуга, либо Переменная-отношение. Если Терм логической формулы является логической формулой, то его значение является результатом вычисления этой логической формулы.

31. <Знак логической операции> ::= “>” | “<” | “>=” | “<=” | “<>” | “==” | “И” | “ИЛИ” | “НЕ”

Семантика: Необходимо выполнить действие в зависимости от вида знака.

32. <Обход дерева программы> ::= “Обход_дерева_программы” “(“

<Переменная-фрагмент>, <Переменная-фрагмент>, <Логическая формула> “)” <Последовательность конструкций>

Семантика:

Выполнение Обход дерева программы представляет собой рекурсию:

Шаг1(база рекурсии):

Вызвать рекурсивную функцию Шаг2 Текущий фрагмент равен Начальный фрагмент.

Шаг2(рекурсивная функция с параметром Текущий фрагмент):

Если класс переменной-фрагмента Текущий фрагмент равен Описание_одной_функции, то: выполнить Шаг2 (Текущий фрагмент равен фрагменту Блок_описаний_функций, к которому от текущего фрагмента ведет дуга Описание_функций); выполнить Шаг2 (Текущий фрагмент равен фрагменту Блок_описаний_параметров, к которому от текущего фрагмента ведет дуга Описание_параметров); выполнить Шаг2 (Текущий фрагмент равен фрагменту Блок_описаний_типов, к которому от текущего фрагмента ведет дуга Описание_типов); выполнить Шаг2 (Текущий фрагмент равен фрагменту Блок_описаний_переменных, к которому от текущего фрагмента ведет дуга Описание_переменных); выполнить Шаг2 (Текущий фрагмент равен фрагменту Программный_блок, к которому от текущего фрагмента ведет дуга Тело).

Если класс текущего фрагмента Программный_блок, то последовательно для каждого фрагмента, к которому от Текущего фрагмента ведет дуга Оператор выполнить Шаг2 (Текущий фрагмент равен фрагменту к которому ведет дуга Оператор).

Если класс текущего фрагмента Условный_оператор, то: выполнить Шаг2 (Текущий фрагмент равен фрагменту Программный_блок, к которому от текущего фрагмента ведет дуга То); выполнить Шаг2 (Текущий фрагмент равен фрагменту Программный_блок, к которому от текущего фрагмента ведет дуга Иначе).

Если класс текущего фрагмента Цикл_for, то: выполнить Шаг2 (Текущий фрагмент равен фрагменту Программный_блок, к которому от текущего фрагмента ведет дуга Тело).

Если класс текущего фрагмента Цикл_while, то: выполнить Шаг2 (Текущий фрагмент равен фрагменту Программный_блок, к которому от текущего фрагмента ведет дуга Тело).

Если класс текущего фрагмента Цикл_repeat, то: выполнить Шаг2 (Текущий фрагмент равен фрагменту Программный_блок, к которому от текущего фрагмента ведет дуга Тело).

Если Условие обхода истинно, то поочередно выполнить Конструкции из Тела обхода.

33. <Обход дерева выражения> ::= “Обход_дерева_выражения” “(“ <Переменная-фрагмент>, <Переменная-фрагмент> “)” <Последовательность конструкций>

Семантика:
Выполнение Обход дерева выражения представляет собой рекурсию:
Шаг1(база рекурсии): Вызвать рекурсивную функцию
Шаг2 Текущий фрагмент равен Начальный фрагмент.
Шаг2(рекурсивная функция с параметром Текущий фрагмент):
Если у фрагмента Текущий фрагмент по дуге Выражение_справа находится фрагмент класса Выражение, то выполнить Шаг2 (Текущий фрагмент равен фрагменту, к которому от текущего фрагмента ведет дуга Выражение_справа).
Если у фрагмента Текущий фрагмент по дуге Выражение_слева находится фрагмент класса “Выражение”, то выполнить Шаг2 (Текущий фрагмент равен фрагменту, к которому от текущего фрагмента ведет дуга Выражение_слева).
Поочередно выполнить Конструкции из Тела обхода.

34. <Модификация программы> ::= <Создание фрагмента> | <Создание атрибута> | <Создание дуги> | <Создание отношения> | <Создание переменной>
35. <Создание фрагмента> ::= “Создать_фрагмент” “(“ <Переменная-фрагмент>, <Класс фрагмента> “;” <Переменная-фрагмент> “)”

Семантика: Добавить к МСП новый фрагмент Результат-фрагмент. Класс нового фрагмента – Класс фрагмента, а предок – Аргумент-фрагмент.

36. <Создание атрибута> ::= “Создать_атрибут” “(“ <Переменная-фрагмент>,<Имя атрибута>, <Переменная-атрибут> “)”

Семантика: Добавить к фрагменту Аргумент-фрагмент новый атрибут с именем Имя атрибута и значением Значение атрибута.

37. <Создание дуги> ::= “Создать_дугу” “(“ <Переменная-фрагмент>, <Переменная-фрагмент>, <Имя дуги>, <Переменная-дуга> “)”

Семантика: Добавить к МСП новую дугу Переменная-дуга от фрагмента Аргумент-фрагмент к фрагменту Аргумент-фрагмент2 с именем Имя дуги.

38. <Создание отношения> ::= “Создать_отношение” “(“ <Переменная-фрагмент>, <Переменная-фрагмент>[, < Переменная-фрагмент> <Переменная-отношение> “)”

Семантика: Добавить к МСП новое отношение Переменная-отношение с именем Имя отношения, которое бы связывало фрагменты Аргумент-фрагмент, Аргумент-фрагмент2, Аргумент-фрагмент3, Аргумент-фрагмент4.

39. <Значение> ::= <Целое> <Вещественное> <булево множество>
40. <Целое> ::= (<Цифра>)
41. <Вещественное> ::= (<Цифра>)[,(<Цифра>)]
42. <Присваивание> ::= <Левая часть присваивания> = < Правая часть присваивания >
43. <Левая часть присваивания> ::= <Переменная-фрагмент> | <Переменная-атрибут> | <Переменная-дуга> | < Переменная-отношение> |

<Переменная>

44. <Правая часть присваивания> ::= <Переменная-фрагмент> | < Переменная-атрибут> | <Переменная-дуга> | < Переменная-отношение> | <Переменная> | <Значение> | <Арифметическое выражение>
45. <Арифметическое выражение> ::= <Терм арифметического выражения> <Знак арифметической операции> <Терм арифметического выражения>

Семантика: Для того, чтобы вычислить значение Выражения необходимо вычислить значение Термов выражения и применить к ним Знак арифметической операции.

46. <Знак арифметической операции> ::= “+” | “- ” | “* ” | “/ ” | “^”
47. <Класс фрагмента> ::= “Описание_переменной” | “Описание_функции” | “Описание_параметра” | “Описание_переменных” | “Описание_функций” | “Описание_параметров” | “Присваивание” | “Ввод” | “Вывод” | “Программный_блок” | “Условный_оператор” | “Цикл_с_шагом” | “Цикл_с_предусловием” | “Цикл_с_постусловием” | “Вызов_процедуры” | “Уничтожение_динамической_переменной” | “Выражение” | “Последовательность_операторов”
48. <Имя атрибута> ::= “Обратная_польская_запись” | “Результат_массив” | “Указатель” | “Функция_рекурсивна” | “Побочный_эффект” | “Ссылка_на_область_памяти” | “Уровень_вложенности” | “Приоритет” | “Тип” | “Параметры_по_ссылке” | “Параметры_по_значению” | “Фактические_параметры_по_ссылке” | “Изменяемые_фактические_параметры_по_ссылке” | “Фактические_параметры_по_значению” | “Аргументное_множество” | “Результатное_множество” | “Сильнорезультатное_множество” | “Оператор_описания_функции” | “Непрерывная_последовательность_фрагментов” | “Классы_фрагментов_последовательностей” | “Количество_фрагментов” | “Идентификатор_результата” | “Псевдопеременная” | “Конструирование_новых_типов” | “Леводопустимое выражение”
49. <Имя дуги> ::= “Если” | “То” | “Иначе” | “Условие_цикла” | “Для” | “До” | “Шаг” | “Тело_оператора” | “ Блок_параметров” | “Блок_локальных_параметров” | “Блок_вложенных _функций” | “Выражение_справа” | “Выражение_слева” | “Первый_элемент_последовательности” | “Последний_элемент_последовательности” | “Дуга_последовательность_операторов” | “Сопоставляет_фрагменты” | “Следующий_фрагмент” | “Список_параметров”
50. <Имя отношения> ::= “Непосредственное_предшествование” | “Предшествование” | “Подобие” | “Являться_частью” | “Являться_подмоделью” | “Предшествование_подмоделей” | “Объединен-

ная_последовательность” | “Промежуточная_последовательность” | “Предшествующая_последовательность” | “Следующая_последовательность”

51.<Булево множество> ::= “истина” | “ложь”
52.<Переменная> ::= <Строка>
53.<Переменная-множество> ::= <Строка>
54.<Переменная-фрагмент> ::= <Строка>
55.<Переменная-атрибут> ::= <Строка>
56.<Переменная-дуга> ::= <Строка>
57.<Переменная-отношение> ::= <Строка>

Пример представления метода потокового анализа на языке МПА

Метод копирования аргументного множества каждого оператора программы в результатное множество программы на языке МПА представлен следующим образом:

```
Метод_потокового_анализа(Копирование_множеств)
    Тип-фрагмент: Текущий_фрагмент;
    Тип-атрибут: Временный_атрибут;
{
    Обход_дерева_программы(Функция_Главная; Текущий_фрагмент;
    Класс_фрагмента(Текущий_фрагмент) == Присваивание){
        Атрибут_фрагмента(Текущий_фрагмент; Аргументное_множество; Временный_атрибут);
        Создание_атрибута(Текущий_фрагмент; Результатное_множество; Временный_атрибут);
        }
}
```

Описание метода на языке методов МПА начинается с ключевого слова *Метод_потокового_анализа*, за которым в круглых скобках указывается название метода:

Метод_потокового_анализа(Копирование_множеств)

Далее следует раздел описания переменных:

Тип-фрагмент: Текущий_фрагмент;
Тип-атрибут: Временный_атрибут;

Непосредственно за разделом описания переменных следует тело метода, которое состоит из последовательности конструкций:

```
{
    Обход_дерева_программы(…)
    {
        Атрибут_фрагмента(…);
        Создание_атрибута(…);
    }
}
```

Тело метода заключено в фигурные скобки, а конструкции внутри разделяются символом точка с запятой. В примере тело метода состоит из одной конструкции *Обход_дерева_программы*, которая в свою очередь состоит из двух конструкций: *Атрибут_фрагмента* и *Создание_атрибута*.

Конструкция *Обход_дерева_программы* представляет собой функцию с тремя аргументами, которые следуют за ключевым словом, заключены в круглые скобки и разделены символом точка с запятой, и телом, заключенным в фигурные скобки:

Обход_дерева_программы(Функция_Главная; Текущий_фрагмент;
Класс_фрагмента(Текущий_фрагмент) == Присваивание)
{
…
}

Эта функция реализует обход дерева фрагментов МСП. Первый аргумент задает фрагмент МСП, являющийся корнем поддерева, которое предстоит обойти. Второй аргумент это переменная, которая принимает значение фрагмента на очередном шаге обхода. Третий аргумент – логическая формула, в случае истинности которой выполняется последовательность конструкций из тела. В данном примере первый аргумент – константа *Функция_Главная*, значение которой – ссылка на корневой фрагмент дерева МСП. Второй аргумент – переменная *Текущий_фрагмент*, которая будет принимать значения очередного фрагмента на каждом шаге обхода. Третий аргумент – логическая формула, принимает значение истина, если фрагмент МСП, на который ссылается *Текущий_фрагмент*, имеет класс *Присваивание*.

Конструкция Атрибут_фрагмента называется формулой языка МПА:

Атрибут_фрагмента(Текущий_фрагмент, Аргументное_множество, Временный_атрибут);

По сути Атрибут_фрагмента есть функция с тремя аргументами. Данная формула для заданного фрагмента возвращает атрибут с заданным именем. Первый аргумент – это фрагмент МСП, на который ссылается переменная *Текущий_фрагмент*. Второй аргумент – название атрибута МСП, который необходимо получить. Третий аргумент – переменная типа Типатрибут которая является результатом и принимает значение ссылки на заданный атрибут фрагмента *Текущий_фрагмент*.

Создание_атрибута, как и Атрибут_фрагмента, является формулой:

Создание_атрибута(Текущий_фрагмент, Результатное_множество, Временный_атрибут);

Создание_атрибута имеет три аргумента. Данная функция для заданного фрагмента создает атрибут с заданным названием и значением. Первый аргумент – это фрагмент МСП, на который ссылается переменная *Текущий_фрагмент*. Второй аргумент – название атрибута МСП, который

необходимо создать, в данном случае это *Результатное_множество*. Третий аргумент – переменная типа Тип-атрибут, значение которой будет скопировано вновь создаваемому атрибуту.

Заключение

В данной работе была представлена концепция потокового анализа, управляемого знаниями. На примере было продемонстрировано, каким образом при помощи описанного языка можно определять различные методы потокового анализа. В настоящее время на основе концепции потокового анализа, управляемого знаниями, разработана подсистема потокового анализа 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.
7. Касьянов В.Н. Оптимизирующие преобразования программ. – М.: Наука, 1988.
8. Воеводин В.В., Воеводин Вл.В. Параллельные вычисления. – СПб.: БХВ Петербург, 2002.

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