УДК 681.3

А.С. Мохница, В.А. Иовчев, Е.А. Андрющенко

ОСОБЕННОСТИ РЕАЛИЗАЦИИ СРЕДСТВ ТРАНСФОРМАЦИОННОГО СИНТЕЗА ПАРАЛЛЕЛЬНЫХ АЛГОРИТМОВ

Рассмотрены особенности реализации средства трансформационного синтеза параллельных алгоритмов с применением последовательной рекурсивной декомпозиции на основе таблиц иерархии. Представлена архитектура соответствующего инструментария (Трансформатор).

Введение

К числу актуальных областей современной компьютерной науки относится алгебраическая алгоритмика (АА) [1]. В последние годы за рубежом была предложена концепция ментального программирования (Intentional Programming (IP)), которая неразрывно связана с формализацией программирования [2]. Данная концепция основана на моделировании программных систем и их реализации. В Украине исследования в данной области восходят к фундаментальным работам В.М. Глушкова по теории систем алгоритмических алгебр (САА) [3]. Дальнейшее развитие теории САА нашло свое воплощение в алгебре алгоритмики (<АА>) – в новом, интенсивно развивающемся направлении украинской алгебро-кибернетической школы [4].

Цель данной работы – осветить особенности реализации средств трансформационного синтеза параллельных алгоритмов.

1. Средства автоматизации проектирования параллельных алгоритмов

Рассмотрим концепции IP и САА в их сравнительном анализе [1, 2, 5]. Основные принципы, на которых базируется IP, можно сформулировать таким образом:

− общие и предметно-ориентированные абстракции;

− биология программирования;

− экология программирования.

Приведенные ранее принципы соответствуют базовым составляющим <АА>. Абстракции представляются в виде взаимосвязанных форм проектирования алгоритмов:

− аналитическая – позволяет описывать алгоритмы в виде формул соответствующих алгебр; является удобной для трансформации, а именно для их улучшения по выбранным критериям (памяти, скорости и т. д.);

− текстовая (естественно-лингвистическая) – позволяет проектировать алгоритмы на различных входных языках;

− визуальная – граф-схемная.

Биологическая компонента в <АА> – теория клонов, используется для описания, построения и исследования семейств алгебр [6]. При использовании теории клонов можно описать различные языки, принадлежащие выбранному семейству алгебр. Примером служит язык САА и его модификации.

Экологическая компонента в <АА> представима в виде интегрированных инструментальных средств разработки и синтеза алгоритмов и программ в объектных средах: МУЛЬТИПРОЦЕССИСТ, САА-машина, ДСП-конструктор, Трансформатор. Метод многоуровневого структурного проектирования программ (МСПП) объединяет алгебраический и грамматический подходы по конструированию программ.

Следует отметить, что основное назначение языков проектирования схем алгоритмов – представление алгоритмов в форме, ориентированной на человека и

облегчающей процесс производства программного обеспечения (ПО). Для достижения цели возможно объединение разных технических средств преобразования и синтеза программ [7, 8].

Инструментальные средства МСПП появились еще в 80-х годах при разработке автоматизированной системы МУЛЬТИПРОЦЕССИСТ [7]. Данная система проектировалась путем «самораскрутки». Таким образом, компоненты системы, не относящиеся к синтезированным, были получены с помощью разработанных компонент системы. К числу средств инструментальной поддержки МСПП относятся языки проектирования в терминах САА-схем многоуровневых формализованных проектов классов алгоритмов и программ, структур данных и памяти. Схемы допускают лингвистические, аналитические и визуальные представления в рамках интегрированной технологии проектирования классов алгоритмов и программ. Сама технология базируется на соответствующих алгебрах алгоритмов. Математическим базисом данного языка является аппарат САА и его модификации [3]. Метод МСПП реализован в виде САА-машины, которая является интерпретатором САА-схем, предназначенных для отладки и верификации [8]. ДСП-конструктор представляет собой диалоговый конструктор синтаксически правильных САА-схем [9–11]. Трансформатор – инструмент для преобразования аналитических спецификаций алгоритмов с использованием соотношений и тождеств соответствующей алгебры [11].

Разрабатываемый в настоящее время инструментарий предназначен для проектирования параллельных программ для современных сред выполнения, в которых алгеброалгоритмические спецификации и преобразования алгоритмов объединены с мощными средствами автоматизации и синтеза программ на основе Web 2.0. Отметим, что под современными средами выполнения понимаются мультиядерные, мультипотоковые, кластерные, Грид-системы, облачные системы и т. д. [12]. Данный инструментарий (в отличие от системы МУЛЬТИПРОЦЕССИСТ) состоит в интеграции всех трех представлений алгоритма [13]: аналитического (формула в избранной алгебре), естественно-лингвистического (САА-схемы) и графового (граф-схемы Калужнина) при его конструировании. Как следствие, использование в инструментарии всех трех форм представления дает более полную картину о конструируемом алгоритме, чем любое из них по отдельности. Изменение любого из этих представлений в инструментарии автоматически приведет к соответствующему преобразованию остальных.

2. Архитектура инструментальных средств трансформации

Необходимо отметить, что задача трансформации аналитических спецификаций алгоритмов и программ – не нова. Тождественным преобразованиям алгоритмов символьной мультиобработки посвящен ряд работ Г.Е. Цейтлина [13-15]. Не менее важна проблема автоматизации применения тождественных преобразований в САА. Работа [16] посвящена тождественным преобразованиям недетерминированных схем алгоритмов и программ. Доказательство тождеств в аксиоматизированных САА рассматривается в [17]. Проблема автоматизации оптимизирующих преобразований – в [18]. Решение проблемы тупиков в параллельных регулярных схемах программ с помощью тождественных преобразований приведено в [19].

Решение этих и ряда других важных задач алгоритмики связано с разработкой интегрированного инструментария <АА>. В рамках работы над инструментальными средствами синтеза алгоритмов и программ в объектных средах с помощью аппарата <АА> был спроектирован и реализован на языке программирования Delphi инструментарий трансформации схем алгоритмов и программ (Трансформатор) [20]. Внешний вид Трансформатора показан на рис. 1.

Интерфейс приложения состоит из следующих элементов:

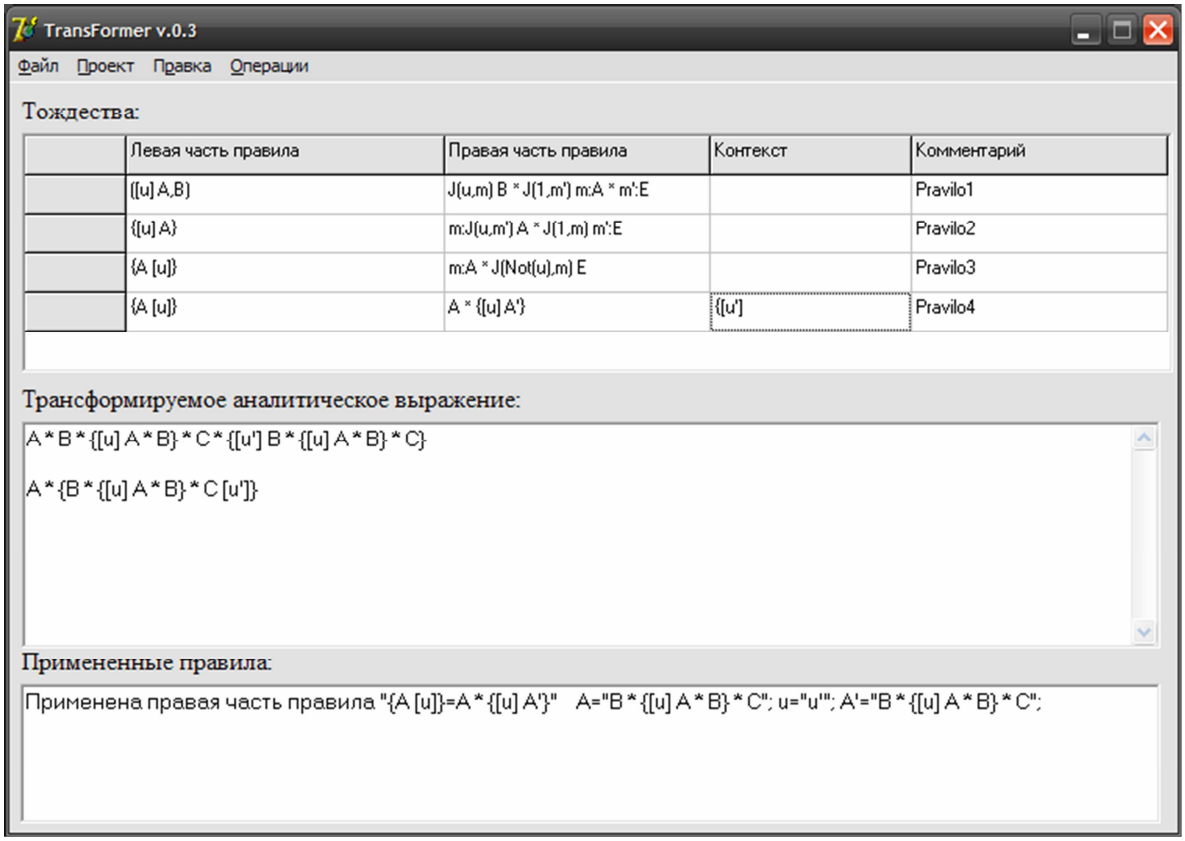

Рис. 1. Главное окно инструментария трансформации схем алгоритмов и программ

− трансформации и хранящихся в базе данных Трансформатора [20];

− поля редактирования, содержащего трансформируемое аналитическое выражение на данном этапе преобразования, а также на все предыдущих этапах;

− поля редактирования, отображающего информацию о правилах (соотношениях), примененных к трансформируемому выражению для каждого этапа процесса преобразования.

Кроме того, программа предлагает отдельное окно для ввода новых или редактирования существующих тождеств с сохранением в базе тождеств и соотношений.

Выше упоминалось, что первоначально Трансформатор базировался на алгоритме последовательной статической декомпозиции ДЕК/С [21], осуществляющем посимвольный анализ входной цепочки и накладываемой на нее формы с последующей переинтерпретацией выходной цепочки с помощью получивших значение переменных формы.

В настоящее время Трансформатор использует более эффективный алгоритм последовательной рекурсивной декомпозиции на основе таблиц иерархии, предложенный авторами.

Рассмотрим архитектуру инструментальных средств трансформации, показанную на рис. 2. Трансформатор состоит из следующих функциональных блоков:

ядро – базовый блок Трансформатора, управляет и обеспечивает взаимодействие остальных блоков;

ИП – интерфейс пользователя, отвечает за взаимодействие Трансформатора с пользователем;

БСИ – функциональный блок Трансформатора, отвечающий за связь с инструментальными средствами <AA>;

редактор правил трансформации – блок, предназначенный для редактирования правил (грамматики) и контекстов, применяемых во время трансформации;

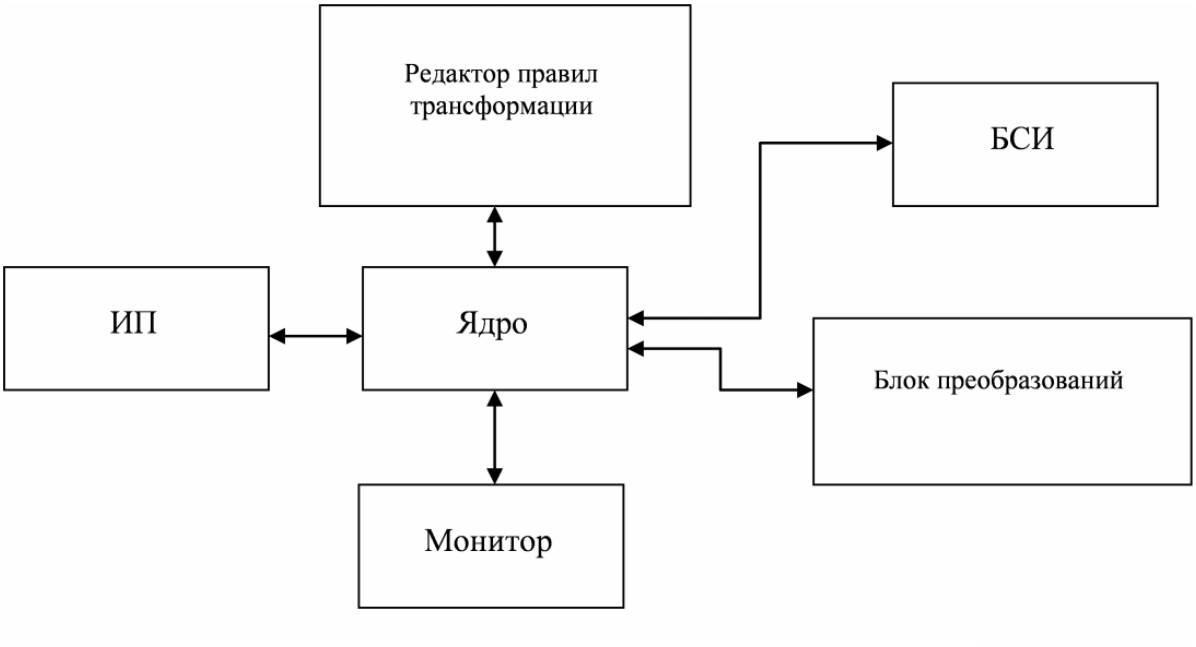

Рис. 2. Архитектура Трансформатора

монитор отслеживает и протоколирует изменения на каждом шаге трансформации, благодаря чему можно отменить текущее действие и вернуться на предыдущий этап;

блок преобразований – производит все действия по трансформации входной цепочки с использованием правил.

3. Особенности реализации инструментальных средств трансформации

Пусть $G = (V_т, V_н, n_0, P)$ – бесконтекстная грамматика и $P(u) = H\ u\ K$ – разметка цепочки u, подлежащей анализу, $L(G) \in V^*_т$ – язык, порожденный грамматикой G, где n из $V_н$ – нетерминал, а u из $V_т$ – терминал.

ТРАНСФОРМАТОР :: == ИНИЦ *
*ПОКА НЕ[КОНЕЦ ТРАНСФОРМАЦИИ]
ЦИКЛ
СЖАТИЕ.ТФ($Ф_0$) * СЖАТИЕ.ТПл($П.Л_0$) *
*СЖАТИЕ.ТПп($П.П_0$) *
* ПРИМЕНЕНИЕ(u') * РАСЖАТИЕ($Ф_0$)
КЦ.

Где ТФ (таблица иерархий формулы), ТПл (таблица иерархий левой части правила), ТПп (таблица иерархий правой части правила) – очереди;

Ф – цепочка формулы, $Ф_i$ – подцепочка, где $i = 0, 1, 2, ..., m$;

П.Л – цепочка, левая часть правила, $П.Л_i$ – подцепочка, где $i = 0, 1, 2, ..., m$;

П.П – цепочка, правая часть правила, $П.П_i$ – подцепочка, где $i = 0, 1, 2, ..., m$;

ИНИЦ – оператор выполняющий разметку входящей цепочки формулы и цепочки правила;

u' – уникальный терминал;

[КОНЕЦ ТРАНСФОРМАЦИИ] – предикат, истинный в случае успешного завершения трансформации, или обнаружения в процессе работы ошибок.

СЖАТИЕ($Ц_n$) :: =
= ЕСЛИ [$Ф(n)$ – форма-суперпозиция]
ТО ДЕК/С(u/$Ф(n)$) * ЗАП.Т[(n, u)] *
*СЖАТИЕ ($Ц_{n+1}$) КЕ.

/* В результате выполнения данного оператора осуществляется декомпозиция подцепочки $Ц_n$ по форме $Ф(n)$ с фиксацией пар (n_1, u_1), (n_2, u_2), … (n_m, u_m), сформировавшихся в процессе декомпозиции; генерируется соответствующее количество дуг, исходящих из обрабатываемого узла (n, u) к узлам, помеченным указанными парами и расположенными на следующем уровне дерева формируемой таблицы иерархий. При этом помечающие пары в указанном

порядке записываются оператором ЗАП.Т [(*n*, *u*)] в соответствующую Т, где Т ∈ {Ф, П.Л, П.П }. */;

Рекурсивные операторы СЖАТИЕ.ТФ($Ф_0$) и СЖАТИЕ.ТПл($П.Л_0$), СЖАТИЕ.ТПп($П.П_0$) по сути одинаковы, отличаясь только тем, что СЖАТИЕ.ТФ($Ф_0$) декомпозирует формулу и записывает узлы в ТФ; СЖАТИЕ.ТПл($П.Л_0$) и СЖАТИЕ.ТПп($П.П_0$) соответственно левую и правую часть правила.

ПРИМЕНЕНИЕ(*u*') :: =
= ЕСЛИ НЕ [u' = #] ТО ПОИСК.Т(*u*') КЕ
ЕСЛИ [ПОЗ = #] ТО ПОИСК.П КЕ
ЕСЛИ НЕ[ПОЗ = #] ТО ПРАВКА(ПОЗ) КЕ.

/*При заданном уникальном терминале *u*' выполняется оператор ПОИСК.Т(*u*'), который ищет заданный терминал в терминалах пар ТФ. Если уникальный терминал u' не задан и/или ПОИСК.Т(*u*') ничего не нашел, то ПОЗ=#. Тогда выполняется следующий ПОИСК.П по (n_0, u_0) из Т.Пл. И если позиция не найдена, то предикат КОНЕЦ ТРАНСФОРМАЦИИ становится истинным, сообщая о невозможности применения данного правила. В противном случае, если один из поисков нашел позицию ПОЗ, то происходит выполнение оператора ПРАВКА(*t*), который модифицирует участок пар (*n*, *u*) из ТФ парами (*j*, *k*) из Т.Пп.*/

ПОИСК.Т(u') :: =
= ПОКА НЕ [ЧТЕ.ТФ(n_m | $У_1$) = (#) И ПОЗ = #]
ЦИКЛ ЕСЛИ [ЧТЕ.ТФ(n_m | $У_1$) = u'] ТО ЕСЛИ [РАВ] ТО ЗАП.ПОЗ(n_m) КЕ ИНАЧЕ Р(У) КЕ КЦ

ПОИСК.П :: =
= ПОКА НЕ [ЧТЕ.ТФ(n_m | $У_2$) = (#) И ПОЗ = #]
ЦИКЛ ЕСЛИ [ЧТЕ.ТФ(n_m | $У_2$) = = ЧТЕ.ТПл(0)] ТО ЕСЛИ [РАВ] ТО ЗАП.ПОЗ(n_m) КЕ ИНАЧЕ Р(У) КЕ КЦ

ЧТЕ.Т – оператор чтения пары (*n*, *u*) – содержимого Т по *n*, и вывода *u*; здесь $n \in V_н$ – нетерминал, *u* – терминальная подцепочка.

РАВ – предикат, истинный, в случае, если связанные пары (поддерево) от найденной позиции и связанные пары (дерево) левой части правила одинаковы по структуре.

ЗАП.ПОЗ – оператор записи найденной позиции в ПОЗ.

РАСЖАТИЕ(Ф0) :: =
= ПОКА НЕ [d($У_4$, К)]
ЦИКЛ ЕСЛИ [(элем ∈ $V_н$ | $У_4$)]
ТО РАЗВЕР(элем) * УСТ($У_4$, Н)
ИНАЧЕ Р($У_4$) КЕ КЦ.

Оператор РАСЖАТИЕ($Ф_0$) при выполнении производит развертку (n_0, u_0) из Т.Ф в символьную цепочку заменяя все нетерминалы терминалами. В результате получаем новую формулу.

Таким образом, к особенностям реализации приведенного алгоритма можно отнести использование рекурсивной декомпозиции цепочек на основе таблиц иерархии (фактически восходящий синтаксический анализ), позволяющей обрабатывать более сложные выражения по сравнению с ДЕК\С. Кроме того, алгоритм позволяет использовать многоразовое и пакетное применение правил трансформации.

4. Пример функционирования алгоритма трансформации

Как уже упоминалось ранее, Трансформатор использует алгоритм последовательной рекурсивной декомпозиции на основе таблиц иерархии.

Рассмотрим данный алгоритм на примере преобразования формулы
((*A* * *B*) * {[U_1] (*A* * *B*)} * *C* * {[U_2] (B_2 * * {[U_3] (A_3 * B_3)} * C_2)})
по правилу (*A* *{[*U*] *A*}) => {*A* [*U*]} с контекстом для обеих частей равенства *@se = A,U*; (означает, что *A* и *U* – составные элементы (конструкции)).

Шаг 1. Построение таблицы иерархии по входной цепочке (по сути, представляет собой восходящий синтаксический анализ).

Таблица иерархии имеет следующую структуру:

− области имен – пар вида [абстракция, базовый (не составной) элемент];

− области структур – пар вида [абстракция, конструкция (составной элемент)].

В табл. 1 обозначение @operator (@predicat) представляет собой базовый оператор (предикат), @so (@sp) – составной оператор (предикат), *x* – уникальный индекс, например,
@operator_01 => *A*,
@operator_02 => *B*,
@so_01 => (@operator_01 * @operator_02).

Данная таблица может быть представлена деревом восходящего синтаксического разбора, показанным на рис. 3.

Таблица 1. Таблица иерархии формулы

@operator_x	*A*
@operator_x	*B*
@so_x	(@operator_x*@operator_x)
@predicat_x	U_1
@sp_x	@predicat_x
@operator_x	*A*
@operator_x	*B*
@so_x	(@operator_x*@operator_x)
@so_x	{[@sp_x]@so_x}
@predicat_x	U_2
@sp_x	@predicat_x
@predicat_x	U_3
@sp_x	@predicat_x
@operator_x	A_3
@operator_x	B_3
@so_x	(@operator_x*@operator_x)
@so_x	{[@sp_x]@so_x}
@operator_x	B_2
@operator_x	C_2
@so_x	(@operator_x* @so_x*@operator_x)
@so_x	{[@sp_x]@so_x}
@operator_x	*C*
@so_x	(@so_x*@so_x* @operator_x*@so_x)

Шаг 2. Построение таблицы иерархии для левой и правой части правила (см. табл. 2). Дерево в данном случае будет иметь вид, показанный на рис. 4.

Шаг 3. Далее осуществляется поиск конструкции @so_x*@so_x, находящейся в вершине дерева левой части правила, в дереве формулы (а точнее в области структур таблицы иерархии), причем индекс *x* не учитывается. Если поиск завершился успехом (см. рис. 5), происходит сравнение всех ветвей, исходящих из данных вершин, на идентичность в целом.

Таблица 2. Таблица иерархии правила

Левая часть правила	
@so_x	*A*
@sp_x	*U*
@so_x	*A*
@so_x	{[@sp_x]@so_x }
@so_x	@so_x*@so_x
Правая часть правила	
@so_x	*A*
@sp_x	*U*
@so_x	{@so_x [@sp_x]}

Кроме того, при наличии контекстного аргумента @where = «уникальный элемент» ведется поиск в найденной подобной конструкции уникального элемента (оператора, предиката). Если такой элемент отсутствует, выдается сообщение о невозможности трансформации. Также сообщение выдается при отсутствии подобной вершины.

Шаг 4. Чтобы осуществить развертку [20] (т. е., чтобы найденная структура @so_x * @so_x приобрела вид {@so_x [@sp_x]}), необходимо внести поправки в таблицу иерархии формулы, сопоставив соответствующие вершины деревьев формулы и правил (см. табл. 3).

Внеся поправки и проинтерпретировав правую часть правила на основе полученных значений нетерминалов, получаем следующую цепочку:

$$(\{(A * B)\,[U_1]\} * C * \{[U_2]\,(B_2 * * \{[U_3]\,(A_3 * B_3)\} * C_2)\}).$$

Таблица 3. Формирование поправок для таблицы иерархии формулы

Правая часть правила		Формула		Поправка	
@so_x’	*A*	@so_x	(@operator_x*@operator_x)	@so_x’	(@operator_x*@operator_x)
@sp_x’	*U*	@sp_x	@predicat_x	@sp_x’	@predicat_x
@so_x’	{@so_x [@sp_x]}	@so_x	(@so_x*@so_x* @operator_x*@so_x)	@so_x’	({@so_x*[@sp_x]}* @operator_x*@so_x)

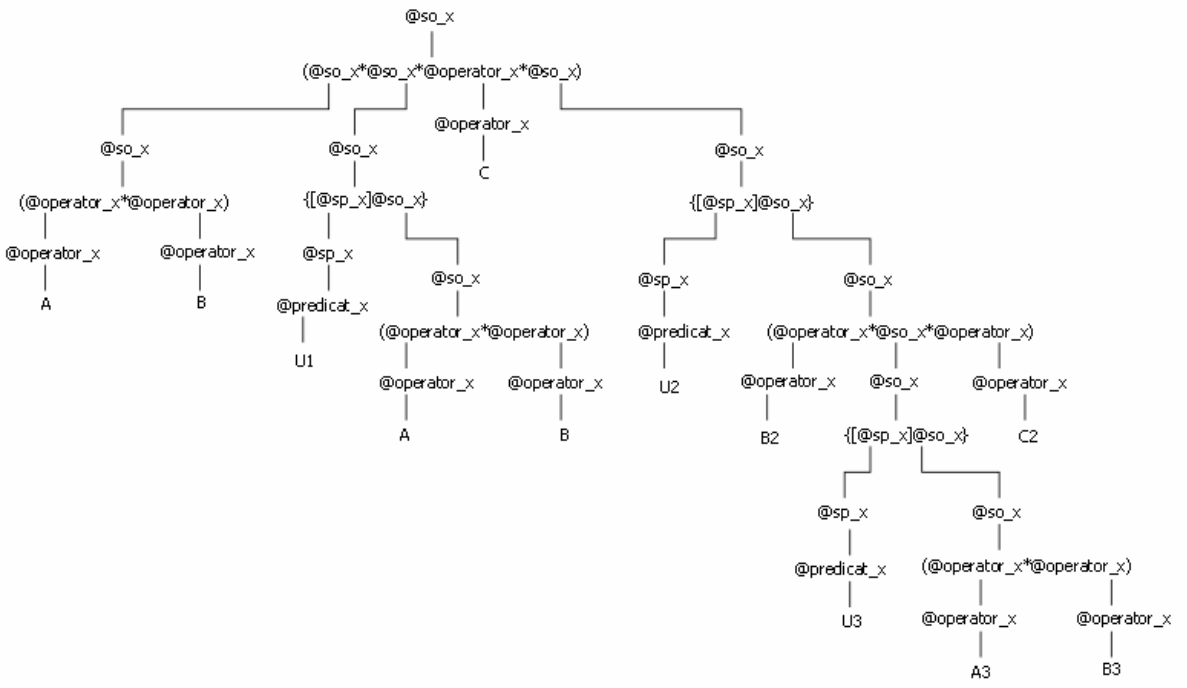

Рис. 3. Дерево для таблицы иерархии формулы

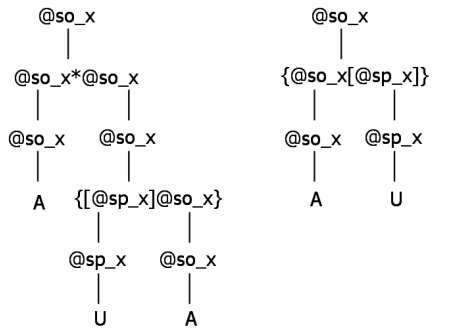

Рис. 4. Деревья для таблицы иерархии правила

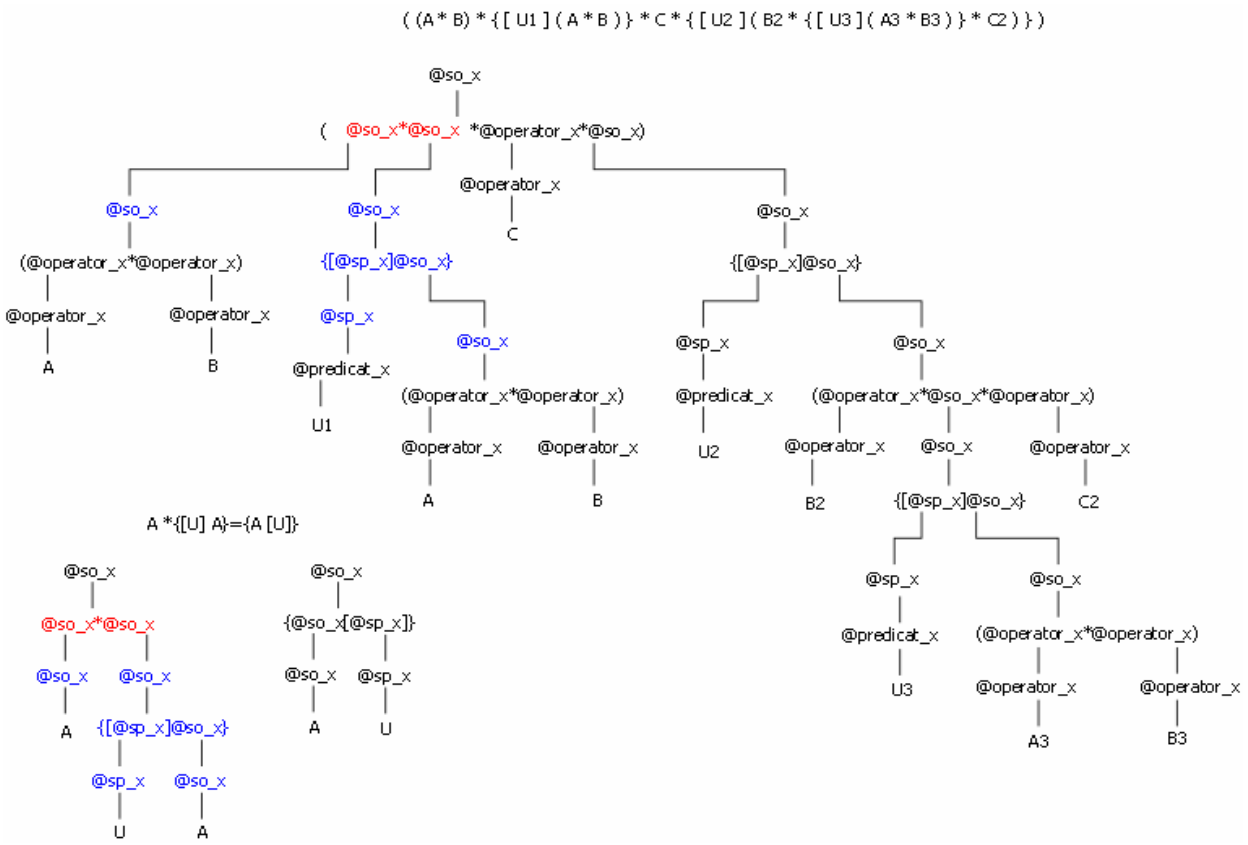

Рис. 5. Поиск конструкции и сравнение ветвей

Заключение

В данной работе представлена архитектура инструментального средства для трансформационного синтеза параллельных программ, названного Трансформатором. Рассмотрены особенности реализации Трансформатора с применением алгоритма последовательной рекурсивной декомпозиции на основе таблиц иерархии на примере одного шага преобразования (применения правила к символьной последовательности). Приведена САА-схема данного алгоритма.

В перспективе планируется перенести Трансформатор на язык Java, с целью подключения к распределенной системе автоматизации проектирования параллельных алгоритмов ОДСП [12]. Кроме того, планируется адаптация интерфейса программы для пользователей с проблемами зрения.

1. *Ноден П., Китте К.* Алгебраическая алгоритмика. – М.: Мир. – 1999. – 720 с.
2. *Чарнецкий К., Айзенекер У.* Порождающее программирование. Методы, инструменты, применение. – Изд. дом. Питер, 2005. – 736 с.
3. *Глушков В.М. , Цейтлин Г.Е., Ющенко Е.Л.* Алгебра. Языки. Программирование. – Киев: Наук. думка, 1989. – 328 с.
4. *Цейтлин Г.Е.* Алгебры Глушкова и теория клонов // Кибернетика и системный анализ. – 2003. – № 4. – С. 48–58.
5. *Цейтлин Г.Е., Иовчев В.А., Мусихин А.А.* Ментальные аспекты методов символьной мультиобработки // Проблеми програмування. – 2008. – № 1. – С. 60–67.
6. *Ющенко Е.Л., Цейтлин Г.Е., Грицай В.П. и др.* Многоуровневое структурное проектирование программ: Теоретические основы, инструментарий. – М.: Финансы и статистика, 1989. – 208 с.
7. *Цейтлин Г.Е., Суржко С.В., Ющенко К.Л., Шевченко А.И.* Алгоритмические алгебры. – Киев: 1997. – 342 с.
8. *Цейтлин Г.Е.* Конструирование алгоритмов символьной обработки // Кибернетика и системный анализ. – 1993. – № 2 – С. 17 –30.

9. *Яценко О.А.* Інтегровані алгеброалгоритмічні моделі мультиобробки // Вісн. Київськ. нац. у-ту. Серія: фіз.-мат. науки. Спец. випуск за матеріалами Міжнар. конф. “Теоретичні та прикладні аспекти побудови програмних систем” (TAAPSD’2004). – 2004.
10. *Яценко Е.А., Мохница А.С.* Инструментальные средства конструирования синтаксически правильных параллельных алгоритмов и программ // Проблемы программирования. Спец. вып. по материалам 4-й Междунар. научно-практической конф. по программированию УкрПРОГ'2004. – 2004. – № 2/3. – С. 440–450.
11. *Кнут Д.* Искусство программирования для ЭВМ // Сортировка и поиск. – 1978. – Т.3.– 824 с.
12. *Дорошенко А.Е., Цейтлин Г.Е., Иовчев В.А.* Высокоуровневые средства автоматизации проектирования параллельных алгоритмов // Проблеми програмування. – 2009. – № 3. – С. 19–30.
13. *Цейтлин Г.Е.* Проектирование последовательных алгоритмов сортировки: классификация, трансформация, синтез // Программирование. – 1989. – № 3. – С. 3–24.
14. *Цейтлин Г.Е.* Проектирование алгоритмов параллельной сортировки // Программирование. – 1989. – № 6. – С. 4–19.
15. *Цейтлин Г.Е.* Проблема тождественных преобразований схем структурированных программ с замкнутыми логическими условиями: ч. 1–3 // Кибернетика. – 1978. – № 3. – С. 50–57; № 4. – С. 10–18; 1979. – № 5. – С. 44–51.
16. *Ющенко Ю.А.* Проблема тождественных преобразований недетерминированных схем программ в системах алгоритмических алгебр // Кибернетика. – 1982. – № 3. – С. 58–64.
17. *Кирсанов Г.М., Цейтлин Г.Е., Ющенко Е.Л.* АНАЛИСТ – пакет программ для доказательства тождеств теорем в аксиоматизированных САА // Кибернетика. – 1979. – № 4. – С. 28–33.
18. *Петрушенко А.Н.* Об одном подходе к проблеме автоматизации оптимизирующих преобразований алгоритмов и программ // Кибернетика и системный анализ. – 1991. – № 5. – С. 127–137.
19. *Панфиленко В.П.* Проблема тупиков и фиктивностей в спецификациях параллельных программ // Кибернетика и системный анализ. – 1994. – № 5. – С. 142–153.
20. *Мохница А.С.* Алгебра алгоритмики и трансформационная сводимость схем алгоритмов и программ // Проблемы программирования. Спец. вып. по материалам 6-й Междунар. научно-практической конф. по программированию УкрПРОГ'2008. – 2008. – № 2/3. – С. 341–347.
21. *Цейтлин Г.Е.* Введение в алгоритмику. – Киев: Сфера, 1998. – 310 с.

Получено 20.10.2009

Об авторах:

Мохница Александр Сергеевич,
младший научный сотрудник
Института программных систем
НАН Украины,

Иовчев Владимир Александрович,
аспирант Института программных систем
НАН Украины,

Андрющенко Екатерина Андреевна,
студентка 6-го курса, факультета
информатики и вычислительной техники
Национального технического
университета Украины «КПИ».

Место работы авторов:

Институт программных систем
НАН Украины,
03680, Киев-187,
Проспект Академика Глушкова, 40,
корп. 5,

Национальный технический университет
Украины «КПИ»,
ул. Политехническая, 41, корп. 18.

e-mail:
mohnitsa@isofts.kiev.ua,
iovchev.v@gmail.com,
ekaterina.andruschenko@gmail.com