УДК 681.3.06

А.В. Малахов, В.Б. Новосельцев

К ВОПРОСУ О СИНТЕЗЕ ПАРАЛЛЕЛЬНЫХ АЛГОРИТМОВ

Рассматриваются формальные модели, используемые для генерации параллельных алгоритмов, а также налагаемые на них требования. Предлагается некоторое расширение теории вычислительных моделей для автоматического синтеза программ из класса параллельных. Показываются полнота и корректность предложенного расширения.

Для решения ряда задач в различных областях науки и техники всё шире используются параллельные компьютерные комплексы различной архитектуры. Это и параллельные суперкомпьютеры, которые включают сотни и тысячи процессоров, и рабочие станции, которые объединены компьютерными сетями, и многопроцессорные рабочие станции. Как следствие этого растет число пользователей-непрограммистов, не обладающих надлежащей профессиональной подготовкой для реализации параллельных приложений. В связи с этим повышается актуальность создания инструментальных систем, снабженных возможностями «интеллектуальной» деятельности и позволяющих автоматически генерировать параллельные программы на основании так называемых непроцедурных спецификаций.

Хорошо известна теория вычислительных моделей (ВМ), впервые предложенная Э.Х.Тыугу [1]. Указанный формализм и его модификации успешно применяются при создании различных прикладных систем, в силу того, что структура вычислительных моделей хорошо согласуется с содержанием сведений, используемых при решении широкого круга вычислительных задач. В то же время синтез параллельных программ в рамках теории ВМ исследован достаточно слабо, хотя теория вычислительных моделей обладает хорошо разработанным формальным аппаратом и предоставляет возможности для создания весьма эффективных машин вывода. Представляется заманчивым расширить теорию планирования решений задач на вычислительных моделях для синтеза параллельных программ.

За основу исследования взят вариант теории ВМ, предложенный в [2], откуда и заимствован, практически полностью, язык теории.

МОДЕЛИ ДЛЯ РАЗРАБОТКИ ПАРАЛЛЕЛЬНЫХ АЛГОРИТМОВ

В данном разделе рассматриваются некоторые принципы и этапы проектирования параллельных структур, которых придерживаются при разработке и реализации параллельных программ, а также модели, используемые при их проектировании.

Параллельные комплексы интересны тем, что они потенциально могут предоставить сосредоточенные ресурсы (процессоры, память с высокой пропускной способностью средств ввода – вывода) для решения важных вычислительных проблем. Параллельные вычисления в течение долгого времени рассматривались как редкая и экзотическая подобласть вычислений, мало интересная среднему программисту, но тенденции в приложениях, архитектуре вычислительных систем и использовании сетей показывают, что на данный момент это не так. Параллелизм становится повсеместным, а параллельное программирование становится неотъемлемой частью в разработке программного обеспечения в корпоративных масштабах.

В [3] выделяется четыре основных желательных свойства параллельных алгоритмов: параллелизм, масштабируемость, локальность и модульность. Параллелизм выражается в способности выполнять несколько действий одновременно. Это определяющее свойство, если программа выполняется на нескольких процессорах. Масштабируемость означает устойчивость к увеличению числа процессоров. Локальность означает высокий коэффициент отношения числа обращений к локальной памяти к числу обращений к удаленной памяти, что является ключом к высокой производительности на мультикомпьютерной архитектуре. И, наконец, модульность – это декомпозиция сложных сущностей на отдельные компоненты, что является важным аспектом разработки и проектирования программного обеспечения в параллельных вычислениях, также как и в последовательных.

Для теоретических исследований проектирования параллельных алгоритмов и программ предложена модель вычислительной машины, называемая мультикомпьютером. Эта модель весьма проста (что облегчает понимание и программирование) и, в то же время, реалистична (это служит гарантий того, что программа, разработанная для этой модели, будет выполняться с соответствующей эффективностью на реальных компьютерных комплексах). Мультикомпьютер – это идеализированная модель параллельного компьютера, содержащая в себе несколько машин фон Неймана или узлы, соединенные сетью. Сообщения используются для взаимодействия с другими компьютерами или для чтения или записи в удаленную память. Стоимость посылки или приема сообщения в мультикомпьютере зависит только от длины сообщения.

Из множества существующих на практике параллельных компьютеров можно выделить три основных архитектуры.

Одна из них – это компьютеры с распределенной памятью, множеством потоков данных и множеством потоков команд (distributed-memory MIMD) – их еще называют компьютерами с векторно-конвейерной обработкой данных. Распределенная память (distributed-memory MIMD) означает, что компьютер может выполнять отдельный поток инструкций над своими данными, и каждый процессор имеет собственную локальную память. Эта архитектура наиболее похожа на мультикомпьютер, различие в том, что стоимость посылки сообщений между процессорами в мультикомпьютере не зависит от расположения узлов в сети и других особенностей сетевого трафика. Примерами таких систем являются IBM SP, Intel Paragon, Thinking Machines CM5, Cray T3D, Meiko CS–2 and nCUBE.

Другую разновидность архитектуры параллельных компьютеров называют многопроцессорным компьютером или компьютером с общей памятью (shared–memory MIMD). В многопроцессорных компьютерах все процессоры обладают доступом к одному общему адресному пространству памяти обычно через шину или иерархию шин. Примерами таких систем являются Silicon Graphics Challenge, Sequent Symmetry и множество других многопроцессорных станций.

Наиболее специализированный, третий класс параллельных компьютеров называют классом с одним потоком

команд и множеством потоков данных (SIMD). В SIMD компьютерах, все процессоры выполняют один поток команд над различными частями данных. MasPar MP является примером такого класса компьютерных комплексов.

Следует оговориться, что еще два сорта компьютерных систем можно отнести к мультикомпьютерам, хотя на них накладываются дополнительные требования к безопасности и достоверности данных, и они имеют высокую стоимость передачи сообщений между рабочими станциями. Одни из них – это компьютеры, соединенные локальной сетью (LAN), а к другому сорту относятся компьютерные системы, состоящие из компьютеров, которые географически значительно удалены и соединены глобальной сетью (WAN).

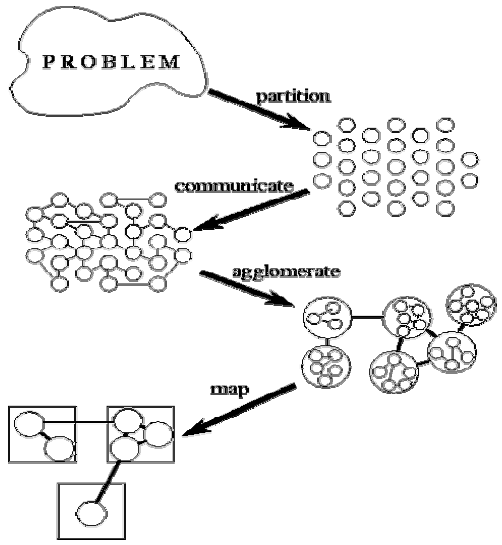

Рис. 1. Методология проектирования для параллельных программ

Для структуризации описания и для того, чтобы в машинно-независимой форме оценить свойства проектируемых параллельных алгоритмов, предлагаются различные модели (programming models – [3]). Они различаются гибкостью, механизмами взаимодействия между задачами, степенью детализации задач, поддержкой локальности, масштабируемостью и модульностью. Рассмотрим некоторые из них.

1. Модель программирования, основанная на задачах и каналах (task/channel programming model). Мы говорим о модели программирования, основанной на задачах и каналах, когда вычисления представлены множеством каналов и множеством задач. Задача инкапсулирует программу и локальную память и определяет набор портов, которые формируют интерфейс этой задачи в модели. Канал представляет собой очередь сообщений, в которую задача-отправитель может помещать сообщение и из которого задача-получатель может извлекать сообщение или блокировать его, если сообщение недоступно. Модель программирования, основанная на задачах и каналах, упрощает программирование для мультикомпьютеров за счет предоставлением абстракций, которые позволяют говорить о параллелизме, локальности и взаимосвязях в машинно-независимой форме. Последнее обеспечивает основу для модульного конструирования параллельных программ.

2. Другая модель программирования носит название модели передачи сообщений (message–passing programming model). Она основана на передаче сообщений и, возможно, является наиболее широко используемой моделью параллельного программирования на сегодня. Программы, основанные на передаче сообщений, создают множество задач, в каждой из которых инкапсулируются локальные данные. Каждая задача идентифицируется своим уникальным именем, и эти задачи взаимодействуют между собой посредством посылки и получения сообщений. Обсуждаемая модель является незначительной вариацией модели задач и каналов, отличаясь только механизмами передачи данных. Например, при посылке сообщения на «канал № 1», мы можем послать сообщение «задаче № 1».

3. Информационный параллелизм (data parallelism programming model) – еще одна модель программирования, отличительной особенностью которой является возможность применения одной и той же операции к множеству элементов структуры данных. Например, увеличить на единицу все элементы массива. Спроектированная программа состоит из последовательности таких операций.

4. Модели программирования с общей памятью (shared-memory programming model). В них задачи делят общее адресное пространство, из которого они читают и в которое они пишут данные асинхронно. Различные механизмы как, например, шлюзы и семафоры, могут быть использованы для организации доступа к общей памяти. Выгода этой модели заключается в том, что данные не имеют конкретных владельцев и, следовательно, нет необходимости определять механизмы передачи данных между задачами. Эта модель может упростить разработку программ. Однако понимание и управление локальностью становиться более сложным.

Большинство задач в программировании имеет несколько параллельных решений. Наилучшее решение может отличаться от предложенного используемыми последовательными алгоритмами. Существует методология проектирования, предназначенная для разработки параллельных приложений. Эта методология разбивает процесс проектирования на четыре отдельных стадии: декомпозиция, коммуникация, агломерация и отображение [3].

1. Декомпозиция. Вычисления, которые могут быть выполнены, и данные, которыми оперируют эти вычисления, разделяются на меньшие (локальные) задачи. Внимание сосредотачивается на оценки возможностей для параллельного исполнения.

2. Коммуникация. Определяется координирование выполнения задачи и подходящие коммуникационные структуры и алгоритмы.

3. Агломерация. Структуры задач и коммуникаций, определенные на первых двух стадиях проектирования, оцениваются в плане стоимости реализации и требований к рабочим характеристикам. При необходимости локальные задачи объединяются в большие, чтобы улучшить рабочие характеристики или уменьшить стоимость разработки.

4. Отображение. Каждая задача назначается процессору таким способом, чтобы удовлетворить конкурентным требованиям: максимизации загрузки процессоров и минимизации стоимости коммуникаций. Отображение может быть определено статически или во время выполнения при помощи алгоритмов балансирования нагрузки.

Очевидно, что рассмотренная методология проектирования параллельных программ и алгоритмов при

всех ее выгодах, тем не менее, не вводит четко сформулированного алгоритма действий для создания параллельных программ. При проектировании, основанном на этой методологии, чрезвычайно важное значение приобретает квалификация разработчика. В следующих разделах будет рассмотрены подходы к автоматизации процесса проектирования параллельных программ.

ПОДХОДЫ К АВТОМАТИЧЕСКОМУ СИНТЕЗУ ПРОГРАММ

Направление автоматического синтеза программ начало формироваться достаточно давно в области искусственного интеллекта. Уже в середине 1960-х гг. появились первые системы синтеза, основанные на принципе резолюции.

К проблеме автоматического синтеза программ существует три основных подхода: дедуктивный, индуктивный и трансформационный. При дедуктивном подходе задача синтеза трактуется следующим образом: по заданному значению x, удовлетворяющему условию $P(x)$, вычислить значение y, удовлетворяющему условию $Q(x,y)$. Здесь x и y – конечные множества, соответственно, входных и выходных величин. Считается, что спецификация $R=(P(x), Q(x, y))$ содержит достаточно информации для построения программы, которая реализует вычисление значений выходных величин по значениям входных. Задача однозначно представляется тройкой (x, y, R). Теорема существования решения задачи имеет вид: $\forall x(P(x) \supset \exists y Q(x, y))$. Построение интуиционистского доказательства теоремы существования решения обеспечивает получение требуемой программы. При таком подходе автоматическое построение программ осуществимо для формализуемых предметных областей в тех случаях, когда разработка спецификации программы по тем или иным соображениям проще разработки самой программы.

Индуктивный подход к автоматическому синтезу программ характеризуется поиском общих закономерностей на основании анализа свойств некоторого ряда объектов и, в определённом смысле, является моделированием метода обобщений, применяемого человеком.

В противоположность индуктивному подходу, где производится переход от частного к общему, трансформационный подход предполагает конкретизацию некой исходной спецификации общего вида. При этом, как правило, преследуется цель упрощения и/или повышения эффективности работы результирующей программы. Осуществляется это как за счёт знаний о предметной области, для которой строится программа, так и на основании традиционной техники оптимизации.

Достаточно хорошо известны вычислительные модели, предложенные Э.Х. Тыугу [1]. Они находят широкое применение при создании систем управления базами данных, в пакетах прикладных программ с автоматическим синтезом программ и в других аналогичных разработках. В этой работе под вычислительной моделью понимаются совокупность переменных и отношений над этими переменными. Отношения используются для вычисления значений. Под отношениями понимаются связи функционального вида, уравнения, зависимости и т.д. В таких моделях специальные машины вывода (без участия пользователя) строят вычислительные структуры над переменными для реализации поставленной задачи. Однако данная задача без введения дополнительных условий является NP-полной и, следовательно, требует огромных вычислительных ресурсов. Поэтому работа [1] не могла быть эффективно использована для практической реализации систем синтеза программ на вычислительных моделях.

Для снижения затрат на генерацию на вычислительные модели накладываются дополнительные условия. Например, переменным и отношениям приписываются дополнительные свойства [2], которые позволяют избежать избыточности вывода. Некоторые свойства позволяют обеспечивать синтез ветвящихся и рекурсивных программ. Конкретизация вида связей также способствует повышению эффективности синтеза (естественно, при этом несколько сужается и класс решаемых задач). Тем не менее, остается возможным создавать достаточно эффективные и практически полезные системы автоматического синтеза линейных, ветвящихся и рекурсивных программ. Синтезу параллельных программ в рамках теории вычислительных моделей уделено мало внимания, хотя, повторяем, теория вычислительных моделей обладает хорошо разработанным и исследованным формальным аппаратом и предоставляет возможности для реализации весьма эффективных машин вывода.

В последующих рассуждениях мы будем использовать аппарат теории, приведенный в [2]. Здесь на вычислительную модель накладываются следующие условия. Строго определен вид отношений, которые представляются в виде разрешенных уравнений $f{:}a_1,\ldots, a_n \rightarrow a_0$. Вычислительная модель в [2] называется структурной С-моделью и представляет собой конечную совокупность элементарных и неэлементарных структур, называемых схемами. Элементарная схема – это, на самом деле, конечное или счетное множество элементов (значений элементов). Неэлементарная схема представляет собой структуру следующего вида:

$$T = (a_{00} : T_{00},\ldots, a_{0N0} : T_{0N0},$$
$$\underline{\text{if}}\ p_1 \rightarrow a_{10} : T_{10},\ldots, a_{1N1}\ \square\ldots\square$$
$$p_k \rightarrow a_{k0} : T_{k0},\ldots, a_{kNk}\ \underline{\text{fi}} \mid \text{set}),$$

где a_{ij} – собственные атрибуты схемы; T_{ij} – собственные подсхемы схемы T; p_i – выбирающие символы; set – набор функциональных связей.

Постановка задачи в С-модели носит непроцедурный характер – в ней указываются лишь исходные и искомые атрибуты некоторой схемы. Содержательно постановка задачи есть задание на построение схемы алгоритма, реализующего требуемые вычисления. Схема алгоритма становится алгоритмом (программой) в результате задания интерпретации для С-модели [2].

Одним из базовых в С-моделях является понятие предложение вычислимости.

Под предложением вычислимости (ПВ) понимается выражение вида $F{:}A \rightarrow X$ где F – (программный) терм, построенный по специальным правилам, A и X – непустые наборы атрибутов некоторой С-модели. Наборы A и X называются аргументами и результатами ПВ соответственно.

Неформально ПВ означает, что программа, являющаяся её реализацией F_I, по заданным значениям аргументов A вычисляет значения результатов X.

СИНТЕЗ ПАРАЛЛЕЛЬНЫХ ПРОГРАММ В С-МОДЕЛЯХ

В [2] вводится класс рекурсивных детерминированных С-моделей (РДС), который, в отличие от моделей, введенных Э.Х. Тыугу ([1]), подходит для создания эффективных алгоритмов корректного синтеза линейных, ветвящихся и рекурсивных программ. Это достигается за счет введения на множестве величин и отношений модели дополнительных свойств, что, с одной стороны, накладывает определенные ограничения на представление вычислительной задачи, а, с другой – позволяет перейти от экспоненциальной к полиноминальной (зачастую, линейной) сложности. Кроме того, появляется возможность синтеза рекурсивных программ. Генерация целевой программы осуществляется автоматически, без участия пользователя. Основной объем работы выполняется экспертом один раз при описании предметной области (ПО) в терминах рассматриваемой теории. Конечный пользователь системы синтеза в постановке задачи указывает входные и выходные данные, а также модель, в которой необходимо осуществлять синтез, и как результат работы системы получает программный код, реализующий вычисление неизвестных данных. Под программным кодом может пониматься как код на некотором псевдоязыке, и как код на известных языках высокого или низкого уровня или завершенный машинный код. Это зависит от реализации системы вывода.

Проблема синтеза в [2] трактуется следующим образом: исходя из задачи $S=(A_0, X_0, T)$ построить программный терм-решение F такой, что для произвольной интерпретации I принадлежность кортежа r типу T_I гарантирует выполнение на r функционального соотношения, которое задаётся реализацией F_I**.** Терм-решение F извлекается из доказательства теоремы существования вида $T \vdash F \mid F{:}A_0 \rightarrow X_0$, формулируемой на основании задачи S. Доказательство этой теоремы проводится в рамках некоторой формальной теории. Содержательно это означает, что, используя информацию о структуре схемы T и соотношения между её атрибутами, мы пытаемся вывести ПВ, которое связывает атрибуты множеств A_0 и X_0 и удовлетворяет схеме T. С этой точки зрения подход к решению этой задачи относится к дедуктивным подходам к автоматическому синтезу программ.

В [2] приведено исчисление SM, в рамках которого производится поиск доказательства теоремы существования. Попытаемся расширить это исчисление с целью реализации возможности синтеза параллельных программ. В расширении исчисления SM имеется три сорта термов.

Термы первого сорта (а-термы) представляют собой упорядоченные списки (n-ки) атрибутов схем модели. Термы второго и третьего сорта определяются рекурсивно.

Пусть: p – n-местный выбирающий символ, $p \in P$; A – n-элементный терм первого сорта, такой, что его i-й атрибут связан с той же схемой, что и i-й аргумент p; P, Q – термы второго сорта; α – некоторый атрибут; «–» и «&» – операции отрицания и конъюнкции соответственно.

Тогда:

1) $\alpha p(A)$ – терм второго сорта;

2) P – терм второго сорта;

3) $P\&Q$ – терм второго сорта.

Программные п-термы третьего сорта строятся из термов всех сортов по следующим правилам.

Пусть: P – терм второго сорта; $f, g. h$ – функциональные символы; $F1$, $F2$, $F3$ термы третьего сорта и имеются вхождения символа g в $F3$; α – некоторый атрибут; $F3[h/g]$ – обозначение замены вхождения g на h в терме $F3$.

Тогда:

1) $\alpha f{:}\ldots\rightarrow\ldots$ – терм третьего сорта;

2) $F1$; $F2$ – терм третьего сорта, построенный с применением оператора композиции;

3) P then $F1$ else $F2$ fi – терм (оператор ветвления);

4) h=if P then $F3[h/g]$ else $F2$ fi – терм (оператор рекурсии);

5) $F1||F2$ – терм (оператор параллельности).

Термы первого сорта присутствуют здесь неявно.

Рассматривается одна схема аксиом и пять правил вывода.Схема аксиом теории: skip: $A \rightarrow A$.

Правила вывода:

1. $\dfrac{F : A \rightarrow X,Y}{F : A \rightarrow X}$;

2. $\dfrac{E : A \rightarrow X, If : I \rightarrow x / p}{F; f : A \rightarrow X, I, x / P \ \& \ p}$;

3. $\dfrac{F1 : A \rightarrow X, x / Q \ \& \ p \quad F2 : A \rightarrow X, x / Q \ \& - p}{\underline{\text{if } p \text{ then}}\ F1\ \underline{\text{else}}\ F2\ \underline{\text{fi}} : A \rightarrow X, x / Q}$,

4. $F1 : A \rightarrow X / Q \ \& \ p$;

$$\frac{\begin{array}{c} g1 : n^{k_1}_{\alpha_1}(A) \rightarrow n^{k_1}_{\alpha_1}(X), \ldots, gsn^{k_s}_{\alpha_s}(A) \rightarrow n^{k_s}_{\alpha_s}(X), \\ \vdash F2 : A \rightarrow X / Q \ \& - p \end{array}}{h = \underline{\text{if } p \text{ then}}\ F1\ \underline{\text{else}}\ F2[h/g1]\ldots[h/gs]\ \underline{\text{fi}} : A \rightarrow X / Q},$$

где запись вида X/P служит сокращением для $x_1/P,\ldots, x_m/P$, а n^k_α – символ отношения порядка на атрибутах модели[2];

5. $\dfrac{F1 : A \rightarrow X / P \quad F2 : A \rightarrow Y / Q, (P \ \& \ Q \neq \Lambda)}{F1 \| F2 : A \rightarrow X, Y / P \ \& \ Q}$.

Схема аксиом позволяет формировать «начальные» для вывода ПВ предложения вычислимости, отталкиваясь от заданных атрибутов.

Правило сужения (1) применяется в тех случаях, когда необходимо исключить из рассмотрения часть атрибутов, достижимость которых доказана, например, при использовании правила (4) или в конце доказательства.

Правило композиции (или транзитивности) (2) понимается как обычно: если показана достижимость I и известно, что по I вычислим x, то можно считать, что показана достижимость x с учётом условий, при которых достижимы атрибуты из I.

Правило ветвления (3) позволяет ослабить условие при атрибутах: если показана достижимость x при условии $Q\&p$ (терм $F1$) и одновременно при условии $Q\&-p$ (терм $F2$), то ПВ, у которого программный терм построен с помощью оператора ветвления, обеспечивает достижимость x, элиминируя условия p и $-p$.

Правило введения рекурсии (4). Оно применимо только для случая рекурсивной схемы. Правило означает, что если показана достижимость $X/Q\&p$ по A посредством терма $F1$, то из предположения о достижимости $n^{k_1}_{\alpha_1}(X), \ldots, n^{k_s}_{\alpha_s}(X)$ следует достижимость $X/Q\&-p$ посредством терма $F2$.

Последнее правило позволяет выполнять распараллеливание последовательного вычисления множеств атрибутов X, Y. Практический смысл правила распараллеливания заключается в следующем. Если существует программная конструкция $F1$, вычисляющая из множества величин A множество величин X, и одновременно с этим существует программная конструкция $F2$, вычисляющая из того же множества A множество величин Y, то можно построить программную конструкцию $F1||F2$, реализующую параллельное вычисление множеств X и Y.

Для исчисления SM были доказаны теоремы о полноте и корректности вывода [2]. Полнота вывода заключается в том, что решение задачи $S=(A_0, X_0, T)$ (на синтаксическом уровне) всегда может быть найдено с помощью предложенных правил. Семантическая полнота является условной – она зависит от интерпретации базовых программных термов и выбирающих условий. Корректность вывода заключается в том, что, используя рассматриваемые правила вывода, можно получить только те ПВ, которые удовлетворяют условиям задачи S.

При введении в теорию вывода дополнительного правила распараллеливания утверждается, что полнота вывода в исчислении SM не нарушится, а теорема о полноте исчисления SM будет верна и после введения дополнительного правила.

Для подтверждения корректности расширенного исчисления докажем соответствующую теорему. Для этого приведем определения из [2], описывающие условия, накладываемые на решение задачи S.

Определение 1. Рассмотрим РДС модель $M=(...,T,...)$. Пусть $a_1/P_1,\ldots, a_n/P_n$, и $x_1/Q_1,...,x_m/Q_m$ – у-атрибуты развёртки T, $\Phi =_{\text{def}} F$: $a_1/P_1,\ldots, a_n/P_n \rightarrow x_1/Q_1,\ldots,x_m/Q_m$ – предложение вычислимости, а r – кортеж, определяемый заголовком развёртки при некоторой интерпретации I**.** Будем говорить, что Φ_I имеет смысл для кортежа r, если $P_{iI} = И$, $i=1,\ldots,n$ и $Q_{jI} = И$, $j = 1,\ldots, m$.

Определение 2. ПВ $\Phi =_{\text{def}} F{:}A\rightarrow X$ удовлетворяет схеме T, если при любой интерпретации I тип T_I не содержит двух кортежей, для которых Φ_I имеет смысл, таких что их компоненты совпадают для всех атрибутов множества A, но не совпадают по крайней мере для одного атрибута множества X. При этом для произвольного кортежа r, который принадлежит отношению T_I, результатом применения F_I к компонентам r, именованным атрибутами множества A, является набор компонент, именованных атрибутами множества X.

Теорема. Расширение исчисление SM является корректным относительно понятия ПВ, удовлетворяющего схеме T.

Утверждение теоремы означает следующее: если ПВ $\Phi=F{:}A\rightarrow X$ выведено с помощью правил SM на основании информации об исходной схеме T РДС моделей M, то для любых двух кортежей типа T_I независимо от интерпретации I, таких что Φ_I имеет для них смысл, компоненты, которые соответствуют атрибутам A и X, в этих кортежах совпадают и связаны соотношением, задаваемым F_I. Для доказательства рассмотрим все правила поочерёдно и покажем, что они позволяют выводить из ПВ, удовлетворяющих исходной схеме T, только те ПВ, которые также удовлетворяют схеме T.

Для схемы аксиом, правил сужения, композиции, ветвления, рекурсии доказательство корректности приведено в [2].

Рассмотрим правило распараллеливания. Пусть посылками правила являются ПВ $F1{:}A\rightarrow X/P$ и $F2{:}A\rightarrow Y/Q$, удовлетворяющие схеме T. Рассмотрим два кортежа s и t, для которых реализации посылок имеют смысл, причём s и t совпадают по A_I. Предположим, что ПВ $F1||F2{:}A\rightarrow X$, $Y/P\&Q$ не удовлетворяет T, это означает, что s и t не совпадают по какой–либо компоненте из набора $X_I, Y_I/P_I\&Q_I$. Несовпадение по компоненте из X_I противоречит тому, что $F1{:}A\rightarrow X/P$ удовлетворяет схеме T, а несовпадение по компоненте из Y_I противоречит тому, что $F2{:}A\rightarrow Y/Q$ удовлетворяет схеме T. Набор X_I, Y_I вычисляется параллельным применением $F1_I$ и $F2_I$.

Мы показали, что правило распараллеливания в расширении исчисления SM корректно.

В настоящее время ведется экспериментальная реализация этого исчисления для оценки эффективности различных стратегий вывода. Также предполагается исследование качества синтезируемых программ относительно требований изложенных в первом разделе.

ЛИТЕРАТУРА

1. *Тыугу Э.Х.* Решение задач на вычислительных моделях. ЖВМ и МФ. 1970. Т. 10. № 3. С. 716–733.
2. *Новосельцев В.Б.* Структурные вычислительные модели – формальный базис корректного построения программ с рекурсивными конструкциями. В кн.: Синтез программ. Устинов. 1985. С.74–76.
3. *Designing* and building parallel programs. By Ian Foster. http://www–unix.mcs.anl.gov/dbpp/.

Статья представлена кафедрой программирования факультета прикладной математики и кибернетики Томского государственного университета, поступила в научную редакцию «Кибернетика» 3 июня 2004 г.