УДК 519.716,35

**МОДЕЛИ И ЯЗЫКИ ДЛЯ ОПИСАНИЯ ПАРАЛЛЕЛЬНЫХ ПРОЦЕССОВ**

**© 2018 г. В.П. Кутепов**

Федеральное государственное бюджетное образовательное учреждения высшего профессионального образования «Национальный исследовательский университет «МЭИ»,   
111250, Москва, Красноказарменная ул., д. 14. МЭИ, Кафедра Прикладной математики.

В статье рассматриваются модели и языки параллельных процессов, позволяющие адекватно описывать свойства и отношения реальных процессов как на событийном уровне, так и с учётом реального времени. Показано, что предложенные модели и языки имеют более широкие возможности для описания параллелизма по сравнению с известными аналогичными моделями и языками. Обсуждается практическое использование моделей и языков параллельных процессов для создания высокоуровневых языков параллельного программирования, распределённых вычислительных и управляющих систем.

**Введение**.

Понятие процесса является настолько универсальным, что без него невозможно дать точное определение системы, алгоритма, любой случайно протекающей или целенаправленной деятельности [1].

По тому, каким образом выполняются действия процесса во времени, их разделяют на непрерывные и дискретные, а по закону, который задаёт причинно-следственные отношения между этими действиями, процессы относят к детерминированным или случайным.

Изучая процессы, мы вынуждены рассматривать триаду: модель и язык процессов, правила их выполнения и историю (трасса, протокол, диаграмма – другие названия той же сути), отражающую следование в общем случае во времени актов конкретного наблюдаемого процесса. Точное семантическое значение понятия процесс обычно восстанавливается из контекста или уточняется.

Если модель или язык процессов и правила их выполнения известны, то множество порождаемых при выполнении конкретного процесса историй восстанавливается путём наблюдения его выполнения. Однако обратная задача определения модели по наблюдаемым историям выполнения процессов является в общем случае неразрешимой. В этом случае большее, что может быть получено, это приближенная модельная реконструкция наблюдаемого множества процессов (физика изобилует примерами достоверности этого утверждения).

В настоящей статье мы рассматриваем детерминированные дискретные процессы, выполнение которых представляется в виде следования неделимых действий процесса (актов, наблюдаемых событий, изменения состояний). При этом основной акцент сделан на модельное описание таких свойств и отношений, как асинхронность, одновременность выполняемых актов процесса, событийном и временном отображении их следования и подчиняемости. Помимо традиционного бинарного оператора, задающего отношение параллелизма двух процессов, мы вводим два новых оператора, позволяющих явно указывать для параллельных процессов одновременность их инициализации и одновременность их завершения. Это позволяет на событийном уровне задавать в описании процесса зависящие от времени условия инициализации и завершения актов процесса. Микроэлектроника даёт очевидные примеры практической необходимости рассмотрения таких условий при реализации процессов.

Иначе определён бинарный оператор выбора одного из двух процессов для продолжения выполнения исходного процесса. В отличие от его обычной трактовки, как оператора недетерминированного выбора, введён оператор детерминированного выбора, что, на наш взгляд, больше соответствует духу детерминированных процессов.

Модель языка процессов рассматриваемых в статье базируется на наших работах по созданию языков параллельного программирования и их реализации на компьютерных системах [2-6]. Заметим, что проблема описания параллелизма в программах [7-8] явилась главным стимулом необычайно широкой активности исследовательских работ в этой области.

Из многочисленных работ, посвященных этой проблеме, укажем на те, которые в той или иной степени повлияли на наши подходы к созданию моделей и языков параллельных процессов. Это прежде всего работы по асинхронной схемотехнике [9] и моделям параллельных взаимодействующих процессов [10-12]. В работах используются три оператора композиции процессов: последовательной, параллельной и недетерминированного выбора. Вместе с возможностью рекурсивного определения процессов, а также оператором синхронного взаимодействия процессов они определяют язык (модель, алгебру - другие используемые названия) параллельных процессов.

Работы [13-15] с обширной библиографией дают представление о формализации дискретных процессов на основе теории частично упорядоченных множеств и структур.

Язык сетей Петри [16], с его многочисленными расширениями, введёнными для описания функционирования распределенных систем, вначале использовал систему строго последовательного срабатывания переходов сети.

Работы [17,18] отражают подходы модельного событийного описания параллелизма и недетерминированности в сетях Петри.

Переход к модели процессов, в которых явно учитывается фактор времени выявил ряд новых проблем. Во-первых, при разных трактовках дискретного и непрерывного времени возникает проблема точности измерения и формального его отражения в модели процессов [19,20]. Другая проблема заключается в том, что в распределённых системах различные их компоненты имеют своё дискретное время и, следовательно, точность его измерения. Поэтому возникает непростая проблема временного согласования событий, происходящих в различных компонентах системы при их взаимодействии [21].

В статье рассматриваются модели и языки процессов, особенность которых, как было отмечено, связаны с созданием языков параллельного программирования и их реализацией на компьютерных системах. В конечном счете успех решения этой проблемы зависит от нашего понимания того, что представляет собой параллелизм, как условие выполнения нескольких действий, на семантическом (задачном) и процессном уровнях, и от того, насколько успешно его можно реализовать на практике.

Стандартные средства параллельного программирования PVM, MPI и Multithreading основаны на встраивании фрагментов программы в определенную модель или язык параллельных процессов [22]. В PVM и MPI в качестве таких фрагментов используются процедуры на языках последовательного программирования, которые при своем выполнении могут взаимодействовать между собой, обмениваясь сообщениями. Средства Multithreading основаны на древовидной модели динамического порождения параллельных процессов, допускающей использование рекурсии со всеми вытекающими сложностями её реализации. Таким образом каждое из названных API средств параллельного программирования имеет свою модель представления параллельных процессов, ограничения которых определяют возможности описания параллелизма.

В разделе 1 статьи мы рассматриваем общую модель дискретных процессов и способы представления историй их выполнения.

В разделе 2 предложен язык параллельных процессов, существенно расширяющий возможности описания различных свойств и отношений параллелизма.

В заключение актуализированы проблемы, требующие дальнейшего исследования.

1. Модели дискретных процессов**.**

В этом разделе мы сначала введем общий базис для определения моделей процессов, а затем рассмотрим формализм для представления историй их выполнения.

1.1. Общий базис модели дискретных процессов.

Рассматриваемый базис определения моделей дискретных процессов (назовем его *PRM*) формально задается в виде четверки , где

не более чем обычное множество актов, которые могут участвовать в выполнении процесса, Ø - акт-дедлок, *stop* - акт завершения процесса,

- отношение, которое для каждого акта определяет конечное число конечных подмножеств актов из , каждое из которых влияет на *e* и является одной из причин инициализации акта *e* после завершения всех актов подмножества,

- отношение, сопоставляющее каждому акту конечное число конечных подмножеств из , одно из которых выбирается после завершения выполнения акта *e* и на акты которого *e* влияет в качестве одного из условий их инициализации,

- конечное множество правил выполнения процессов.

Можно выделить наиболее общие характеристики правил выполнения процессов, наблюдаемые при функционировании реальных систем и процессов. Событийные правила выполнения процессов [10-12] не используют время и нацелены на отображение только следования актов на основе предписанных в описании процесса причинно-следственных зависимостей между ними.

Другой тип правил - это правила, в которых явно учитывается время (обычно длительность актов и их взаимодействий). Инициализация выполнения акта в этих моделях в общем случае есть функция, параметрами которой являются не только поступающие в реальном времени "сигналы" от других актов, но и временные отношения между ними. При этом приходится принимать во внимание точность, с которой контролируется время, в выполняемом процессе. Так как акт может быть инициализирован несколькими подмножествами других актов, необходимо предусмотреть правила, разрешающие эту проблему. При выполнении процесса акты могут порождать новый процесс (типичный случай для рекурсивно определённых процессов) или множество новых и одновременно выполняемых процессов, что также типично для параллельного программирования [23].

Процесс *PR* в базисе *PRM* определяется совокупностью следующих компонентов: , где

- множество актов, участвующих в выполнении процесса,

- подмножество начальных актов процесса,

T - конструктивное упорядоченное числовое множество (часы), посредством которых контролируется время при выполнении процесса и от которого зависит точность этого контроля,

- функция, задающая длительность выполнения акта после выполнения условий его инициализации,

- функция, которой определяется способность порождения новых процессов актом при его выполнении,

*EXEC'* - правила выполнения процесса.

1.2. Автоматное и диаграммное представления выполнения процессов и их историй.

Автоматное представление даёт возможность централизованного контроля всех событий наблюдаемого процесса, которые связаны с одновременным выполнением, завершением и инициализацией всех параллельно выполняемых им актов.

При этом предполагается, что все выполняемые процессом акты имеют конечную длительность, и отношения =, <, > на множестве *T,* значениями которого измеряется реальное время, конструктивны.

Состояние автомата ассоциируется с множеством всех одновременно выполняемых актов процесса. Изменение состояния и переход в новое состояние возникает в момент завершения одного или одновременно нескольких актов процесса. При этом в новом состоянии инициализируется выполнение всех актов, для которых возникли необходимые условия, а также продолжается выполнение всех актов исходного состояния, которые ещё не завершились.

Таким образом, автомат реализует все возникающие возможности параллельного выполнения актов процесса и является асинхронным, поскольку обладает способностью контролировать свои действия независимо от того, какова длительность выполняемых процессом актов.

Далее мы будем использовать графическое представление процесса, как последовательности изменений состояний автомата, которые изображаются в виде прямоугольников с перечислением в них всех одновременно выполняемых актов процесса в наблюдаемый момент времени. Переходы между состояниями представляются в виде дуг, взвешенных множеством всех одновременно завершивших своё выполнение актов в состоянии, из которого эти дуги исходят.

У автомата фиксируется начальное состояние, с которого начинается процесс (все его начальные акты), и, возможно, конечное состояние, которым процесс завершается и которое может не существовать, если множество состояний автомата является замкнутым по отношению к переходам между состояниями.

На рис.2 представлены все возможные процессы (истории) абсолютно параллельного выполнения программы на рис.1.

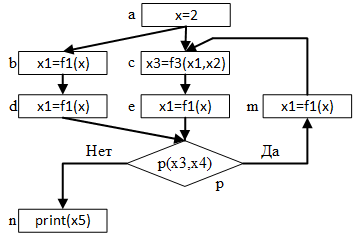


Рис.1. Блок-схема параллельной программы.

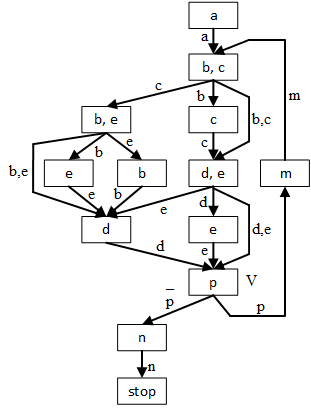


Рис.2. Автоматное представление процессов абсолютно параллельного выполнения программы.

Абсолютно параллельное выполнение процесса подчиняется правилу инициализации любого акта без задержек по его готовности, то есть завершению всех влияющих на него актов. Любой путь из состояния *a* в конечное состояние *stop* является одним из вариантов (историей) выполнения программы, который однозначно определен отношением раньше или одновременно для моментов завершения актов соответствующего состояния. Все акты в этом примере, кроме акта *p*, влияют только на одну группу других актов. Знак *V* на рис.2 означает, что после своего завершения акт *p* может влиять путем детерминированного выбора акта *n* или акта *m*.

В данном примере автомат имеет конечное число состояний. В общем случае число состояний автомата может увеличиваться неограниченно, так же, как и количество одновременно выполняемых актов, фиксируемых в состоянии.

Следующий пример рекурсивной функции:

, где - ближайшее целое к *a* и , при абсолютно параллельном вычислении *n*! и неограниченном увеличении *n* демонстрирует этот случай. В [2,3] такие функции названы функциями с не ограниченным параллелизмом. Множество историй вычисления *n*! при этом является бесконечным, но конструктивным.

На рис.3 приведён пример сети Петри при абсолютно параллельном функционировании которой и при одинаковой длительности срабатывания переходов t2 и t3 количество одновременно срабатываемых копий переходов t1 неограниченно увеличивается.

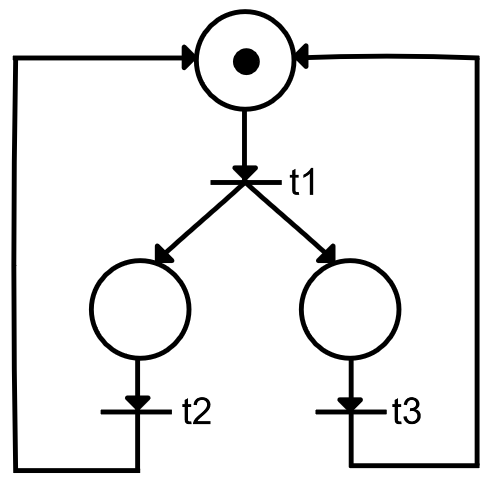


Рис.3 Сеть Петри с неограниченным параллелизмом.

Справедливо следующее утверждение.

*Утверждение.* Автоматы с конечным числом состояний могут представлять процессы только с ограниченным параллелизмом.

При задании длительности актов процесса по автоматному представлению однозначно восстанавливается все истории процесса, а по ним можно перейти к временным диаграммам.

На рис.4а приведён пример сети Петри, автоматное представление её функционирования для заданной длительности срабатывания переходов t = 1, t1 = 0,5, t2 = 1 (рис.4б) и временная диаграмма (рис.4в).

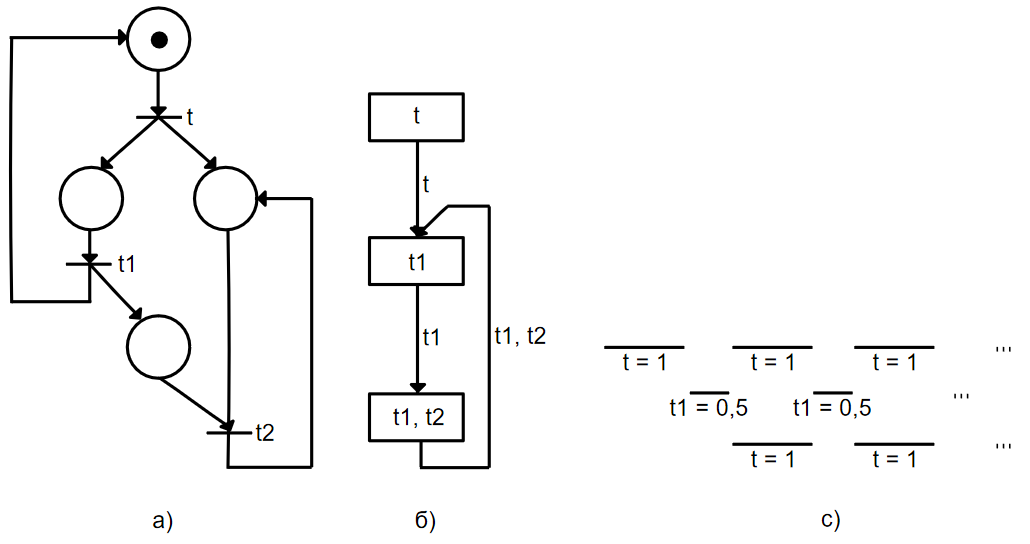


Рис.4 Автоматное представление и диаграмма функционирования сети Петри

В следующем разделе мы опишем язык параллельных асинхронных процессов, который позволяет адекватно представлять их особенности, наблюдаемые в параллельном программировании, при функционировании электронных схем, в многокомпонентных компьютерных и управляющих системах.

2. Язык асинхронных процессов.

Работы [10-12] являются классическими формализациями параллельных процессов (иногда называемых алгебрами). В этих работах используются три бинарные операции композиции процессов: последовательная композиция, априорный недетерминированный выбор и параллельная композиция, а также рекурсивные определения процессов.

Помимо априорного выбора одного из двух процессов, не менее важно иметь возможность описывать апостериорный выбор (типичный случай для программ). Для параллельных процессов часто требуется их одновременная инициализация, также как их одновременное завершение может быть необходимым условием продолжения процесса. Типичным является случай, когда результатом выполнения нескольких параллельных процессов считают результат процесса, который завершился раньше остальных. Примером этого является ситуация, когда мы заставляем одновременно выполнять несколько эквивалентных программ, разработанных для разных методов решения одной и той же задачи и имеющих разное время выполнения для различных исходных данных. Очевидно, что максимальное сокращение времени получения результата определяется минимальным временем одновременного выполнения такого рода программ. Эти и другие формы параллелизма [2,3] требуют адекватных средств их описания. Основная цель описываемого языка процессов – создание необходимых условий для адекватного описания указанных и других особенностей реальных параллельных процессов.

2.1. Формальное описание языка процессов.

Следующие условия определяют особенности процессной семантики языка.

1. Любой акт инициализируется без задержек, как только для этого выполняется необходимые условия.
2. Некоторые акты процесса могут завершаться контролируемым дедлоком, что означает невозможность дальнейшего продолжения процесса.
3. На множестве наблюдаемых событий инициализации и завершения актов выполняемого процесса определены бинарные отношения одновременно (=) и раньше (<), которые имеют однозначное представление на конструктивном множестве чисел, используемых в качестве шкалы времени.
4. Некоторые акты процесса способны инициировать выполнения множества процессов, которое определяется динамически при выполнении акта.

Пусть - не более чем счетные множества константных и переменных актов, – акт-дедлок и *stop* – стоп-финальный акт, , ( – пустое множество). Далее акты из трактуются, как акты, способные инициализировать одновременное выполнение процессов, вызываемых из некоторого заранее определенного множества процессов . Множество процессов в языке определяется индуктивно путем применения бинарных операций композиции процессов: ⊕и в общем случае систем рекурсивных уравнений.

1) Любой акт есть процессы.

2) Если P1 и P2 – процессы, то – процессы, где .

3) Пусть – не более чем счётные множества переменных, значениями которых являются процессы.

Система уравнений , где есть выражение, построенное путём применения операции композиции к процессам и переменным , определяет *k* процессов , где - минимальное по отношению включения множеств решение системы уравнений (\*).

Мы не будем здесь повторять известные теоретические основания, берущие свое начало от Д. Скотта о существовании минимального решения для систем подобных уравнений [10-11]. Отметим только возможное конструктивное построение минимальных решений для системы уравнений (\*), как точных верхних граней упорядоченной последовательности приложений:

где выражение в квадратных скобках означает одновременную подстановку .

2.2. Правила выполнения процессов на событийном уровне.

Выполнение конкретного процесса , определенного системой уравнений, реализуется путем синтаксического разбора термов обратного процедуре их индуктивного построения, процессной интерпретации, операций композиции, а также реализации операций подстановки вместо переменных актов и входящих термов системы уравнений.

Далее в записи процессов предполагается следующий порядок старшинства операций композиции: ⊕ , позволяющий опускать ряд скобок в записи процессов.

1. означает, что процесс начинает выполнение без задержек после успешного завершения процесса . Если процесс завершается дедлоком, выполнение процесса блокируется.
2. означает, что выполнение процессов начинается одновременно и завершается выполнением процесса является момент наиболее раннего успешного завершения одного из процессов или их одновременного завершения. Если оба процесса завершаются дедлоком, результатом является дедлок.

Для того, чтобы реализовать альтернативный апостериорный выбор (выбор по умолчанию) одного из двух процессов с помощью операции введем понятие так называемых ортогональных событий. Для этого множество актов разделим на два подмножества , = Ω. Любые два акта назовем ортогональными и положим, что при всяком их выполнении не более чем один из них может завершится успешно. При выполнении процесса , где и – ортогональные акты, только один из процессов соединенных операцией может завершиться успешно. В программах предикаты и можно интерпретировать как ортогональные акты, полагая при точном значении одного из них он завершается дедлоком.

Заметим, что в работах [5, 6] аналогичная операция с тем же обозначением использовалась как операция композиции функций, что подчеркивает ее универсальный характер.

1. Операции и имеют следующую процессную интерпретацию. означает, что процессы , завершаются успешно, а время выполнения равно максимальному времени завершения процессов . Интерпретация отличается от интерпретации тем, что для успешного завершения процесса требуется одновременное успешное завершение обоих процессов. Если процессы завершаются не одновременно, результатом выполнения является дедлок.
2. Выполнение любого акта приводит к одновременному выполнению множества сопоставленных ему процессов , что реализуется путем подстановки вместо процесса . На практике множество инициируемых актом процессов в общем случае может зависеть от самого акта и значений параметров, поступающих от актов, влияющих на . Формально это означает, что существует функция , которая акту для заданного времени и упорядоченного подмножества параметров из , сопоставляет конечное подмножество процессов множества PR. Также предполагается, что инициированные актом процессы сохраняют влияние на другие акты процесса, определенны отношением .
3. При готовности для выполнения любого переменного акта , осуществляется подстановка вместо него правой части определения в системе уравнений (\*), после чего выполнение процесса продолжается.

Приведем пример описания процесса абсолютно параллельного вычисления значений следующих функций:

, где и – ортогональные акты.

Процесс ниже описывает параллельное функционирование сети Петри (рис. 5), которое предполагает, что все переходы сети срабатывают, как только для этого выполнены условия готовности, т.е. на всех входных местах перехода появляются «фишки»:

Здесь и – ортогональные акты.

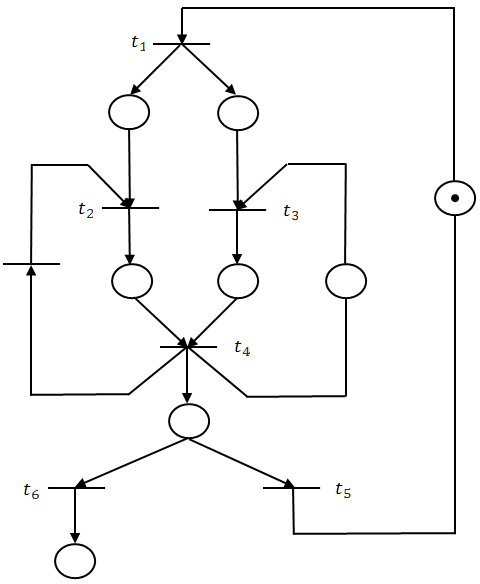


Рис.5. Сеть Петри

2.3. Описание взаимодействия процессов.

Различают синхронное и асинхронное взаимодействие двух параллельных процессов. Первое предполагает, что процесс, первым инициализирует взаимодействие и выполнение до момента, когда другой процесс также готов к взаимодействию. После выполнения операции взаимодействия, реализуемой на практике путем обмена сообщениями, процесс продолжает своё выполнение.

Формальное описание этого случая взаимодействия двух процессов в [10-12] реализуется путем специального обозначения актов процесса, которые интегрируются, как акты парного взаимодействия. Этот случай можно просто перенести на язык процессов. Асинхронное взаимодействие процессов обычно реализуется через «почтовый ящик», в который помещаются приходящие к нему сообщения, которые процесс может в определенное время доставить для чтения. При этом возникает проблема исключения доступа к «почтовому ящику» при помещении сообщений в «почтовый ящик» и чтении их из него одновременно несколькими процессами.

Известная семафорная техника позволяет решить проблему взаимного исключения путём введения семафорных целочисленных переменных, принимающих положительные значения, и двух функций и .

Функция-предикат проверяет условие для целочисленной переменной и, если оно выполнено, уменьшает на единицу и разрешает процессу обращаться к почтовому ящику. Если , то выполнение процесса задерживается до момента, когда другой процесс увеличит значение на единицу с помощью функции , сообщая тем самым, что он завершил обращение

Предполагается, что только одна функция или может исполнятся над семафорной переменной в одно и тоже время и должна инициализироваться за конечное время.

Ниже приведен пример описания взаимодействующих параллельных процессов, требующих взаимного исключения при обращении к общему буферу данных.

Положим, что каждый *i*-й процесс, , генерирует последовательно конечное число данных i , записывая их в буфер, который имеет конечное число позиций . , а процесс – обработчик данных считывает по очереди данные с буфера и выполняет их обработку. Поскольку все процессы выполняются одновременно и вынуждены работать с общим ресурсом – буфером данных, все операции обращения к буферу должны выполняться с взаимным исключением, для чего используется семафорная техника.

Введем следующие обозначения:

buff – буфер для хранения данных размера , ,

S1 иS2 – семафоры, целочисленными значениями которых определяют количество свободных и занятых позиций в буфере,

S3 – семафор, принимающий значения 0 и 1 и служащий для взаимного исключения процессов при обращении к буферу.

Следующая система уравнений описывает асинхронный процесс 1, одновременной обработки данных, который сохраняет максимальный параллелизм выполняемых действий:

*,*

Здесь – функция подготовки данных, и – запись данных в буфер и чтение данных из буфера, – операция над данными. Предикат и его отрицание интерпретируются как ортогональные акты.

3. Сетевое представление процессов.

Сетевое представление процессов позволяет графическими средствами отобразить не только структуру процесса, как композиционной организации, но также упростить его выполнение.

Сетевое представление процесса определяется следующим образом:

1. Любой акт имеет сетевое представление, представленное на рис.6а.
2. Сеть процесса , где ⊕ представляется в виде композиции сетей и процессов и (рис.6б).
3. Сеть процесса имеет представление, представленное на рис.6(в).

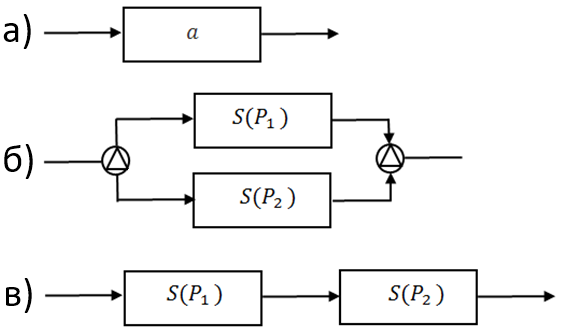


Рис.6.Сетевое представление процессов

Заметим, что сетевые (структурные) формы представления параллельных программ используются и как эффективный способ проектирования сложных объектов, и как представление, существенно упрощающее реализацию параллельных языков на компьютерных системах [4 - 6].

На рис.7 приведена сеть процесса вычисления значения функции *F(x)* приведенной ранее в разделе 2.2.

Выполнение процесса , представленного в сетевом виде для системы уравнений , начинается с сети, полученной для терма. При этом для открывающих операций, одновременно активизируются акты на обеих ветвях, исходящих из них на сети. При достижении соответствующей операции одной из ветвей для , активизируются действия, следующие за этой закрывающей операцией. Для закрывающей операции требуется завершение всех актов, которые принадлежат каждой ветви, идущей к этой операции на сети из соответствующей ей открывающей операции . Только после этого активизируется выполнение актов, следующие за этой операцией на сети. Для операции в отличие от требуется одновременное завершение всех актов на всех ветвях, связывающих её с соответствующей операцией . В противном случае закрывающая операция завершается дедлоком, т.е. следующие за ней на сети действия не могут выполняться. Для переменного акта после его инициализации осуществляется подстановка сети для терма и выполнение процесса, представленного этой сетью.

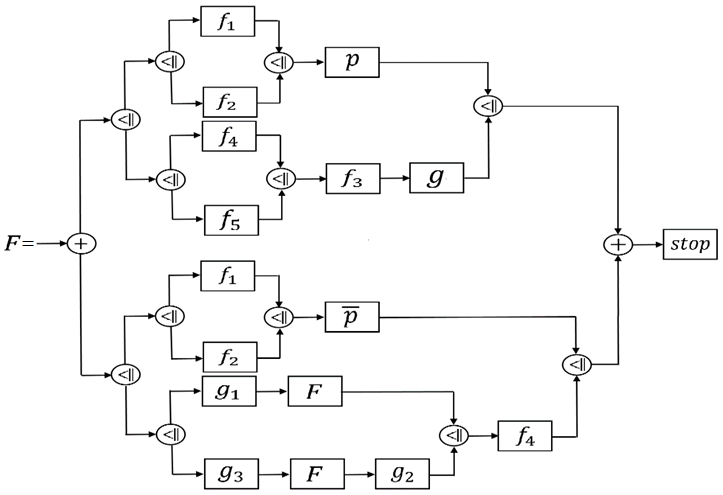


Рис.7. Сетевое представление процесса вычисления значений функции F(x)

Сделаем несколько замечаний общего характера, касающихся выразительной силы графического и текстового описания процессов и трактовки нескольких вхождений акта с одним и тем же именем в описании процесса.

Процесс абсолютно параллельного вычисления значений функции имеет текстовое представление , в котором повторяется дважды. В сетевом представлении (рис.8) повторное вхождение можно исключить.

В связи с этим истории выполнения процесса для этих двух представлений различается. Более того, если предполагать, что различные вхождения одного и того же акта в описании процесса могут иметь различную длительность при их выполнении, что наблюдается, например, в программах, то возникает проблема однозначной интерпретации различных вхождений акта с одним и тем же именем в процесс. С процессной точки зрения эти вхождения должны в общем случае рассматриваться, как различающиеся действия, происходящие при выполнении процесса, хотя и являющиеся эквивалентными при семантической их интерпретации.

Кроме того, один акт процесса может повторяться многократно из-за циклов и рекурсии в описании процесса и его длительность в общем случае может варьироваться в зависимости от очередной инициализации его выполнения.

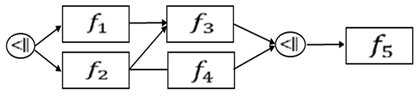


Рис. 8. Сетевое представление процесса.

4. Выполнение процессов с учетом реального времени и введение задержек актов.

Пусть для процесса задана функция, определяющая длительность его актов на множестве отсчетов времени . Таким образом, для акта , выполнение которого инициируется в момент времени , определяет длительность .

Пусть также задана функция , где значение = , определяет длительность задержки выполнения акта .

Введение задержек актов расширяет возможности языка для представления его средствами более широкого круга реальных дискретных процессов.

По этим данным, используя автоматное представление выполнения процесса, можно однозначно восстановить его и по этой истории легко реконструируется привычная временная диаграмма процесса.

5. Эквивалентность параллельных процессов.

Определение. Два процесса и назовем эквивалентными, если множество историй их автономного выполнения равны.

Ниже приведены наиболее важные аксиомы эквивалентности процессов с точки зрения приведения их к более компактной и эффективной форме параллельного выполнения.

Далее в записи аксиом эквивалентности , и обозначают произвольные процессы.

Аксиомы эквивалентности процессов:

1. (ассоциативность операций композиции процессов:

⊕

2. (коммутативность операции композиции ⊕)

3.

4.

5.

6.

7.

8. (A⊕B), при условии, что и – ортогональные процессы

9. (A⊕B),

при условии, что и – ортогональные процессы

Заключение

Представленная в статье точка зрения на модельное описание параллельных процессов в большой степени обязана, как уже отмечалось, нашими работами по созданию и реализации языков параллельного программирования [4 – 6]. Введенное в них схемное и сетевое описание процессов использовалось как эффективный инструмент их проектирования и реализации компьютерных системах. Более того, такое описание может рассматриваться как прямая аналогия между процессом и системой, реализующей процесс. Программируемые логические схемы – пример практического применения этой аналогии на практике.

Очевидно, что потребуется время, чтобы парадигма параллельного программирования стала главенствующей, а не рассматривалась как специальное расширение языков последовательного программирования. Для этого нужны новые высокоуровневые языки [23] и средства проектирования параллельных программ, анализа их сложности и приведения к оптимальной параллельной форме [2,3,5,7,9]. Существенно усложняется проблема верификации параллельных программ по сравнению с последовательными программами, заставляющая расширять классическую логику, создавая условия для её решения [24].

Наконец, практическая реализация параллелизма невозможна без создания эффективных управляющих средств планирования параллельных процессов и распределения ресурсов, новых архитектурных решений компьютерных систем [25,26].

В заключении хочу выразить признательность моему другу и коллеге профессору Фальку В.Н. за возможность многократных обсуждений вместе с ним рассматриваемых в статье проблем, полезную критику и не менее полезные советы, способствующие улучшению начальной версии статьи.

СПИСОК ЛИТЕРАТУРЫ

1. Общая теория систем. Сборник статей. Из-во «Мир», 1966. с.210.

2. Кутепов B. П., Фальк В.II. Формы, языки представления, критерии и параметры сложности параллелизма. Программные продукты и системы, 2010, №3. с.16-25.

3. Кутепов B.П. О параллелизме с разных сторон. 5-я международная конференция «Параллельные вычисления и проблемы управления». Изв. РАН, Институт проблем управления, М.2010. с.42-50.

4. Кутепов В.П., Маланин В.Н, Панков Н.А. Граф-схемное потоковое параллельное программирование: язык, процессная модель, реализация на компьютерных системах. Изв. РАН. Теория и системы управления. - 2012. - № 1. с.87-102.

5. Кутепов В. П., Фальк В. Н. Модели асинхронных вычислений значений функций в языке функциональных схем. Программирование, 1978. №3. с.15-23.

6. Кутепов B.П, Шамаль П.Н. Реализация языка функционального параллельного программирования FРТL, на многоядерных компьютерах. Изв. РАН. Теория и системы управления. - 2014. №'3. с.46-60.

7. Серия статей по параллельному программированию. Журнал для разработчиков MSDN, 2008, №1 (83). с.67.

8. Эндрюс Г.Р. Основы многопоточного, параллельного и распределенного программирования. Издательский дом «Вильяме», 2003. с.505.

9. Автоматное управление асинхронными процессами в ЭВМ и дискретных системах. Сборник статей под редакцией Варшавского В.М.: Наука, 1986.

10. Milncr. R.A. A Calculus of Communicating Systems. Lecture Notes in Computer Science, 92. Springcr-Verlag, 1980.

11. Хоар Ч. Взаимодействующие параллельные процессы, 1989. с.264.

12. J.A. Bergstra and J.W. Klop. Process algebra for synchronous communication. Information and Control, 1984, 60(1/3). с.109-137.

13. J. Winkowski, An algebraic framework for concurrent systems. Institute of Computer Science of the Polish Academy of Sciences, Report №1023, Warsaw, 2012. с.139

14. Hike Best and Agathe Nerceron. Lecture notes in computer science. 1985, v. 188.

15. В.А. Горбатов, А.В. Горбатов, М.В. Горбатова. Дискретная математика. Изд. «Астрель», 2003.

16. Дж. Питерсон. Теория сетей Петри и моделирование систем. Издательство «Мир». 1983. с.264.

17. U.Goltz and W. Reisig. The non-sequential behavior of Petru nets. Information and control, 1983, v.57, №2-3. с.125-147.

18. N. Melsen and P.S Thigarajan. Degrees of non-determinism and concurrence: A Petri net view. Lecture notes in computer science, 1984, v.181. с.89-117

19. Baeten J., Bergstra J. Real Time Process Algebra. Formal Aspects of Computing, 1991, 3(2). с. 142-188.

20. Middelburg C. A. Revisiting timing in process algebra. The Journal of Logic and Algebraic Programming, 2003, 54. с.109-127

21. И.А. Жуклинец, Д.А. Хотимский. Логическое время в распределенных программных системах. Программирование, 2000, № 6. с.3-19.

22. К. Хьюз, Т. Хьюз. Параллельное распределенное программирование с использованием С++. Изд. «Вильямс», 2004. с.667.

23. В.П. Кутепов. Высокоуровневый язык потокового параллельного программирования. Программирование (в печати).

24. L. Lamport. The temporal logic of actions. ACM Transactions on Programming Languages and Systems. 1993, V.7, №7. с. 1-52.

25. В.П. Кутепов. Интеллектуальное управление процессами и загруженностью в вычислительных системах. Изв. РАН, Теория и системы управления, 2007, № 5. с.58-73.

26. Ю.Н. Бранскикова, Ю.А. Горицкий, В.П. Кутепов, Н.А. Панков. Исследование методов прогнозирования загруженности компьютеров и компьютерных систем. Программные продукты и системы, 2015, № 2. с.135-136.