RU EN ПРОГРАММНЫЕ СИСТЕМЫ: ТЕОРИЯ И ПРИЛОЖЕНИЯ ISSN 2079-3316
научная статья ТЕОРЕТИЧЕСКИЕ ОСНОВАНИЯ ПРОГРАММНЫХ СИСТЕМ

УДК 004.432+004.272
10.25209/2079-3316-2026-17-4-3-32

Параллельная потоковая модель вычислений Буран, основанная на взаимодействии токенов

Аркадий Валентинович **Климов**✉
Национальный исследовательский центр «Курчатовский институт», Москва, Россия
✉*arkvklimov@yandex.ru*

Аннотация. Представляется абстрактная модель вычислений, являющаяся обобщением семейства архитектур и моделей, основанных на взаимодействии токенов. Токены рассматриваются как абстрактные сущности. Взаимодействия между ними порождают вычислительные кванты, которые могут порождать новые токены. Модель допускает высокую степень параллелизма независимых взаимодействий.

Модель в целом является недетерминированной. Однако, можно говорить о детерминированности отдельных вычислений. В статье дается математическое описание модели, определяются понятия вычисления и детерминированного вычисления. Доказывается принципиальная разрешимость детерминированности заданного вычисления, но остается открытым вопрос о ее более эффективном и практичном распознавании.

Будучи абстрактной, модель может иметь различные конкретизации. Одна из них – проект уникальной многоядерной архитектуры потоковой параллельной вычислительной системы (ППВС) БУРАН, предложенной В.С. Бурцевым в конце 90-х, а также некоторые ее расширения. От данного проекта и было заимствовано название самой абстрактной модели. В настоящее время работа над ним продолжается в НИЦ «Курчатовский институт».

Ближайшим известным аналогом модели являются сети Петри и их расширения, такие как Цветные Сети Петри (Colored Petri Nets, CPN). Их также можно описать как конкретизацию абстрактной модели.

Ключевые слова и фразы: абстрактная модель вычислений, параллельная модель вычислений, детерминизм, недетерминизм, токен, взаимодействие, вычислительный граф, ППВС БУРАН, сети Петри

Благодарности:
Работа проведена в рамках выполнения государственного задания НИЦ «Курчатовский институт»

Для цитирования: Климов А.В. *Параллельная потоковая модель вычислений Буран, основанная на взаимодействии токенов* // Программные системы: теория и приложения. 2026. **Т. 17**. № 4(73). С. 3–32. https://psta.psiras.ru/read/psta2026_4_3-32.pdf

© Климов А.В. 2026 CC BY

1. Введение

Модель вычислений архитектур семейства БУРАН рассматривается как абстрактная модель вычислений, в которой обобщаются и формулируются наиболее общие свойства таких архитектур. Цель абстрагирования – отделить принципиальные базовые свойства и черты, характерные для всех архитектур семейства, от особенностей конкретных архитектур, которые могут варьироваться их дизайнерами.

На пути от абстрактной модели к конкретным архитектурам можно говорить о различных уровнях абстракции. Чем выше абстракция, тем шире класс описываемых ею конкретных архитектур, и наоборот. Дополнительные требования понижают уровень абстракции и ограничивают класс, тем самым помогая архитектору определить рамки допустимых модификаций. Когда обнаруживаются какие-то интересные свойства (такие как, например, детерминизм), важно понимать, на каком уровне они появляются и на каких постулатах базируются.

Все архитектуры рассматриваемого семейства основаны на токенах и взаимодействиях между токенами. Каждый отдельный токен может рассматриваться как носитель данных и активности. Взаимодействия токенов происходят внутри особого хранилища, называемого Рабочим Пространством (РП). В РП в каждый момент времени находится некоторое мультимножество[1] токенов. В конкретных архитектурах его традиционно называют ассоциативной памятью (АП).

Абстрактная модель допускает различные возможные конкретизации. Это могут быть как конкретные архитектуры, так и промежуточные абстракции, характеризующиеся дополнительными постулатами и элементами. В рамках конкретизации может вводиться собственный язык написания программ со своей семантикой. Также могут вводится какие-либо дополнительные компоненты в токены и определяться какие-то дополнительные общие для всех программ правила работы с ними (как, например, имя узла, контекст, маски, кратность в архитектуре БУРАН или типы мест и выражения с переменными в CPN).

Проект Параллельной потоковой вычислительной системы (ППВС) БУРАН является основным примером конкретизации. Его архитектура была предложена академиком В.С. Бурцевым в конце 90х[1]. В дальнейшем работа над ним продолжалась в ИПИ РАН, ИТМ и ВТ, ИППМ РАН[2–4],

[1] Мультимножество – это, говоря неформально, множество, в котором учитываются повторы элементов. Иногда для них используют термин «мешок» (bag).

а в настоящее время – в НИЦ «Курчатовский институт», в который ИППМ РАН вошел в качестве структурного подразделения (ОППМ). Название ППВС «БУРАН» возникло около 2010 года в память об авторе: Бур-АН. От него и унаследовано название абстрактной модели Буран, которое записывается строчными буквами (кроме первой), для отличия от названия конкретизации.

Другой конкретизацией можно считать сети Петри[**5, 6**] и их расширения, такие как Цветные Сети Петри (Colored Petri Nets, CPN)[**7**]. Между этими двумя конкретизациями (БУРАН и CPN) имеется много общего. В работе [**8**] приводятся определения обеих, а затем расматриваются их сходства и различия.

Главная черта данной абстрактной модели вычислений вынесена в название статьи – это взаимодействие токенов. Группа из одного или нескольких токенов, образующих активную комбинацию (АК), реализует акт взаимодействия, или срабатывания, в результате которого порождаются новые токены. Но еще важно, что происходит при срабатывании со старыми токенами из АК. Каждый из них может быть удален или сохранен с модификацией. Есть принципиальное различие между сохранением старых токенов, выполняемым как бы мгновенно, и длительным порождением новых. Наличие этого разделения является характерной отличительной чертой модели вычислений Буран.

Статья имеет следующую структуру. Сначала (раздел 2) кратко вводятся используется обозначения и понятия. Затем, в разделе 3, после неформального описания дается строгое математическое определение абстрактной модели вычислений Буран и описание абстрактной Буран-машины. Далее выясняется, что в определении модели имеется некоторая неполнота, устранению которой посвящен раздел 4. Здесь приходит понимание, что токены это модифицируемые объекты, и на этой основе формулируются дополнительные принципы, накладывающие некоторые ограничения на работу Буран-машины.

В разделе 5 определяются понятия вычислительного процесса (вычисления) и вычислительного графа. Хотя в целом модель вычислений Буран является недетерминированной, но можно говорить о детерминированности отдельных вычислений. В разделе 6 вводятся понятия изоморфизма вычислений и графов, и на их основе – понятие детерминированного вычисления. Доказывается принципиальная разрешимость свойства детерминированности, и ставится вопрос о более эффективном распознавании детерминированности вычислительного процесса. Раздел 7 содержит описания двух конкретизаций, а Заключение подводит итоги.

2. Используемые обозначения и понятия

$\mathbb{N}$ – множество (тип) натуральных чисел, включая 0

$\mathbb{B}$ – множество (тип) логических значений $\{true,\ false\}$

$\varnothing$ – пустое множество

Пусть X – некоторое множество (тип). Мультимножество (*multiset*) типа X (или *мультимножество* на базовом множестве X) есть функция $m : X \to \mathbb{N}$, которая каждому элементу $x \in X$ ставит в соответствие его вес – неотрицательное целое число. Множество (тип) всех (конечных) мультимножеств над X обозначается как X_{MS}. Пустое мультимножество обозначается как $\varnothing_X$, или просто $\varnothing$. Размер (число элементов) мультимножества m определяется как $|m| = \sum_{x \in X} m(x)$. Обычно будут использоваться конечные мультимножества, где $|m| < \infty$, даже если тип X бесконечный. Сложение (вычитание), умножение на число и отношение вложения мультимножеств определяются естественным образом как расширение операций $+$ $(-)$, $\cdot$ и $\leqslant$ в $\mathbb{N}$.

Если X – множество (тип), а $p : X \to \mathbb{B}$ – предикат на X, то $X|p = \{x \in X | p(x)\}$, то есть подмножество элементов X, удовлетворяющих предикату p. Говорят, что p есть *характеристическая функция* этого подмножества. В дальнейшем будем иногда отождествлять множество и его характеристическую функцию, обозначая их одинаково.

3. Абстрактная модель вычислений

3.1. Неформальное описание и мотивировка

Всякая модель вычислений может быть описана как программируемый абстрактный вычислитель. Его определение обычно включает следующие компоненты:

- Класс программ, или язык программирования
- Множество состояний, среди которых выделяются начальные и заключительные
- Правила перехода из состояния в состояние
- Структура и правила работы абстрактного вычислителя

Абстрактный вычислитель модели вычислений Буран будем называть Буран-машиной (см. рисунок 1). Его состояние задается как содержимое двух хранилищ – Входного Буфера (ВБ) и Рабочего Пространства (РП). В каждый момент времени каждое хранилище содержит некоторое

мультимножество токенов. На абстрактном уровне токен – это просто элемент некоторого фиксированного множества $\mathcal{T}$.

Все новые токены сначала поступают в ВБ. Это могут быть как входные токены, поступающие извне, так и создаваемые внутри согласно программе. ВБ время от времени выбирает произвольный токен из своего состава и перекладывает его в РП. Неопределенность выбора является источником возможного недетерминизма. На токены, поступающие извне, может быть наложено условие, что они передаются в РП в своем заданном порядке, случайно перемежаясь с другими, внутренними токенами, когда таковые имеются в ВБ.

Все взаимодействия токенов происходят в РП, в котором обнаруживаются и выбираются активные комбинации. Активная комбинация (АК) – это некоторое конечное (обычно небольшое, из 1-2 токенов, но в теории может быть и больше) мультимножество токенов. Если в РП имеется АК, то одна из них (любая незаблокированная) выбирается и срабатывает. И это второй источник недетерминизма. Когда имеется несколько незаблокированных и непересекающихся АК, порядок их срабатыаний не может повлиять на результат. И это основной источник параллелизма модели.

В результате срабатывания АК значения ее токенов объединяются в пакет и отправляются в Исполнительное Устройство (ИУ) в качестве входных данных для Вычислительного Кванта (ВК). Результатом работы ВК будет некоторое мультимножество новых токенов, помещаемых в ВБ. Порожденные ВК могут выполняться в ИУ в любом порядке, общий результат как мультимножество токенов от этого не зависит. На практике это служит еще одним полезным источником параллелизма, особенно когда «кванты» большие (относительно самих срабатываний).

При срабатывании АК *одномоментно* все ее токены могут быть либо удалены из РП, либо оставлены в нем и даже, возможно, с «небольшой» модификацией. Ниже это также будет определено формально. Мы ограничиваемся лишь простыми модификациями, чтобы они могли выполняться быстро, без потери темпа, но уже давали полезные возможности, которые иначе нельзя или очень трудно выразить. Также на этой ограниченности будет основано наше понимание детерминизма. Важно, что все изменения РП производятся атомарно, то есть строго после срабатывания предыдущей АК и строго до срабатывания следующей АК.

Распознаватель АК является функцией $\mathcal{A} : \mathcal{T}_{MS} \to \mathbb{B}$, то есть умеет отвечать на вопрос, является ли некий набор токенов активной

комбинацией. Формально этого достаточно, чтобы можно было найти хотя бы одну АК, если они есть в РП, или убедиться, что их больше нет. Множество всевозможных активных комбинаций $\mathcal{T}_{MS}|_{\mathcal{A}}$, формируемое его характеристической функцией $\mathcal{A}$, будем обозначать также как $\mathcal{A}$.

Вычислительный квант вычисляет функцию $F : \mathcal{A} \to \mathcal{T}_{MS}$, определенную на множестве всех АК, ее результат – некоторое мультимножество токенов. Он зависит только от множества токенов входной АК и ничего другого. И даже от порядка токенов внутри АК не может зависеть, поскольку АК это мультимножество, а не список или вектор. Все токены результата поступают в ВБ.

Конкретный тип данных $\mathcal{T}$, распознаватель (множество) $\mathcal{A}$ и функция F определяются конкретной программой. Правила и язык описания программы должны быть определены дополнительно в рамках конкретизации модели.

Но это еще не все. По каждой АК в программе должно быть определено, тем или иным способом, какие из ее токенов удаляются из РП при срабатывании, а какие сохраняются и с какими изменениями. В абстрактной форме мы говорим о функции сохранения $H : \mathcal{A} \times \mathcal{T} \to \mathcal{T} \cup \{\varnothing\}$, где $\varnothing$ это дополнительное значение, выдаваемое функцией $H(a, t)$, когда элемент $t \in a$ нужно удалить из РП.

Сохранение токена непосредственно в РП, да еще с изменениями, это отнюдь не то же самое, что порождение нового токена взамен удаленного через квант вычисления с поступлением его в РП через ВБ. Сохранение производится в рамках одного атомарного действия в составе срабатывания. Никакой другой токен, поступивший сразу после срабатывания не может «заметить» кратковременного отсутствия токена k, сохраняемого при срабатывании. А если бы токен k передавался как результат через ВБ, то его временное отсутствие могло бы быть «замечено».

Пусть, например, придет токен r, образующий АК с токеном k ($\{r, k\}$), которая была бы более приоритетной, чем его АК с другим имеющимся в РП токеном q ($\{r, q\}$). В этом случае при временном отсутствии токена k сработает АК $\{r, q\}$, чем и будет «замечено» отсутствие токена k. Предполагается, что пришедший токен r при этом не сохраняется, и потому АК $\{r, k\}$ уже не сработает.

Таким образом, механизм сохранения добавляет в архитектуру новую функциональность, которую нельзя выразить другими средствами.

Несколько разных АК могут иметь общий токен. В этом случае сработать может только одна из них. При срабатывании токен может сохраниться в РП, после чего он может быть использован повторно в составе другой АК. А если общий токен не сохранится, то другая АК просто перестанет существовать. Тогда будем говорить, что она *гаситься*.

Среди нескольких АК с общим токеном может быть задан (программой) приоритет одних перед другими. Тем самым наличие одной АК может блокировать срабатывание другой. Но это возможно только среди АК, имеющих общий токен.

Например, пусть в РП имеется два токена, u и v, и они образуют две активные комбинации: $\{u, v\}$ и $\{u\}$, из которых первая блокирует вторую. Тогда сначала может сработать только первая, а затем, если токен u при этом сохраниться, то также вторая. А если не сохранится, то вторая погасится и исчезнет. Но вторая безусловно сможет сработать, если токен u придет в РП первым. Непересекающиеся АК не могут влиять на возможность срабатывания друг друга.

3.2. Математическое определение

Математически программу в модели вычислений Буран будем определять как набор абстрактных функций, удовлетворяющих определенным свойствам.

Определение 1. Программа *в абстрактной модели вычислений Буран определяется как структура* $P = (\mathcal{T}, \mathcal{A}, F, H, Pr)$, *где*

- $\mathcal{T}$ – *множество (тип) токенов,*
- $\mathcal{A} \subset \mathcal{T}_{MS}$ – *множество (тип) активных комбинаций (АК) токенов,*
- $F : \mathcal{A} \to \mathcal{T}_{MS}$ – *основная функция порождения новых токенов,*
- $H : \mathcal{A} \times \mathcal{T} \to \mathcal{T} \cup \{\varnothing\}$ – *функция сохранения:* $H(a, t)$, *где* $t \in a$,
- $Pr \subseteq \mathcal{A} \times \mathcal{A}$ – *отношение приоритета на множестве активных комбинаций:* $Pr(a, b)$ *означает, что «a имеет приоритет перед b» или «a блокирует b».*

Таким образом, типы токена и АК не являются чем-то фиксированным раз и навсегда, а возникают как элементы и аспекты заданной программы. Множество токенов $\mathcal{T}$ не обязано быть конечным, главное – все элементы должны быть различимыми (например, в них могут содержаться строки произвольной длины). Все функции $\mathcal{A}, F, H, Pr$ (будем называть их базисными) должны быть эффективно вычислимы. Во всякой конкретизации

модели вычислений будет определяться класс программ, обычно в форме языка. Программы будут текстами языка, причем семантика этого текста должна определять все базисные функции.

Функция H имеет два аргумента: активная комбинация a и токен $t \in a$, а результат $H(a,t)$ есть либо токен t', который вставляется в РП вместо t, либо $\varnothing$, при котором токен t будет просто удаляется из РП.

Применяя обычное расширение, результат замены всей АК a будем обозначать через $H(a,a)$ (подразумевая, что значения $\varnothing$ из множества-образа элиминируются). При этом обозначение $H(a)$, или H_a сохраним за замыканием функции H по его первому аргументу: $H_a : a \to \mathcal{T} \cup \{\varnothing\}$, то есть это функция, которая берет любой элемент мультимножества a и выдает либо какой-то элемент $\mathcal{T}$, либо «ничего». В дальнейшем в записи $H_a(t)$ будем по умолчанию подразумевать, что значение является токеном (а не $\varnothing$).

Еще нам будет важно, что новый токен $H_a(t)$ это всегда *тот же самый* токен t, хотя и модифицированный. Более точно это будет формализовано в разделе 4, а пока сосредоточимся на том, какие модификации считаются допустимыми.

В отличие от функции F, которая может породить любые новые токены, функция H_a вправе производить лишь малые модификации аргумента t. Чтобы это строго определить, нам понадобится понятие *неразличимости*.

Определение 2. *Пусть $\varphi(a)$ – функция, аргументом которой (a) является (мульти)множество токенов. Тогда мы говорим, что φ* не различает *токены t и t', если при всех a (и значениях других аргументов, если они есть), выполняется: $\varphi(a + \{t\}) = \varphi(a + \{t'\})$. Иначе говоря, результат $\varphi(a)$ не чувствителен к подмене в аргументе a токена t токеном t'.*

Теперь определим допустимые модификации.

Определение 3. *Функция H называется* сильно допустимой*, если для всех $a \in \mathcal{A}$ и $t \in C(H_a) \subseteq a$ токены t и $H_a(t)$ неразличимы для функций $\mathcal{A}$, Pr и F.*

Здесь $C(\varphi)$ – носитель функции φ, определяемый как множество тех t, для которых $\varphi(t) \neq \varnothing$.

Однако, для некоторых конкретизаций модели это требование может быть чересчур сильным ограничением. В общем случае достаточно *слабой допустимости*, когда неразличимость требуется лишь относительно $\mathcal{A}$ и Pr, но не F. Слабую допустимость будем требовать всегда, поскольку без нее будет сложно поддерживать целостность. Сильная допустимость нам будет нужна как одно из условий детерминизма. Поскольку для самого H неразличимость не требуется, то получается, что H может в сохраняемых токенах оставлять информацию как бы «для себя».

На самом деле слабую допустимость можно еще более ослабить без ущерба для целостности, а именно: пусть неразличимость относительно $\mathcal{A}$ и Pr имеет место не вообще на всех мультимножествах из $\mathcal{T}_{MS}$, а только на таких, которые в принципе в условиях данной программы могут появляться как подмножества внутри РП. И даже еще слабее: только на на таких, которые могут находиться в РП в момент исполнения рассматриваемой модификации.

В дальнейшем пару токенов (t, t'), неразличимых для функций $\mathcal{A}$, Pr и F, будем называть *конгруэнтными* (в рамках заданной программы) и обозначать это как $t \cong t'$.

Отношение Pr может иметь место только между парами пересекающихся АК: $Pr(a, b) \Longrightarrow a \bigcap b \neq \varnothing$. А порождаемый им граф должен быть ациклическим, что гарантирует возможность исполнить хотя бы одну АК, если таковые имеются в РП. Если $Pr(a, b)$, то обычно $a \nsubseteq b$, поскольку иначе b никогда не сможет сработать.

Заметим, что все базисные функции принимают на свой аргумент мультимножества токенов, то есть неупорядоченные наборы токенов, а не списки или вектора из них. В частности, это означает, что перестановка токенов внутри АК никак не может повлиять на результат. Значит, когда и если мы будем хотеть, чтобы порядок токенов в паре, тройке и т.д. имел значение, нужно будет снабдить токены признаками типа «номер входа» или «номер позиции», которыми токены одной АК будут различаться.

3.3. Буран-машина

Буран-машина – абстрактная машина, исполняющая программы в модели Буран – показана на рисунке 1. Она имеет поле программы, в которое помещается программа P, и два хранилища: буфер токенов $\mathbf{B}$ и рабочее пространство $\mathbf{W}$; в каждом из них в любой момент времени может находиться некоторое мультимножество токенов. Для их обозначения будем использовать те же буквы, но курсивом, соответственно, B и W.

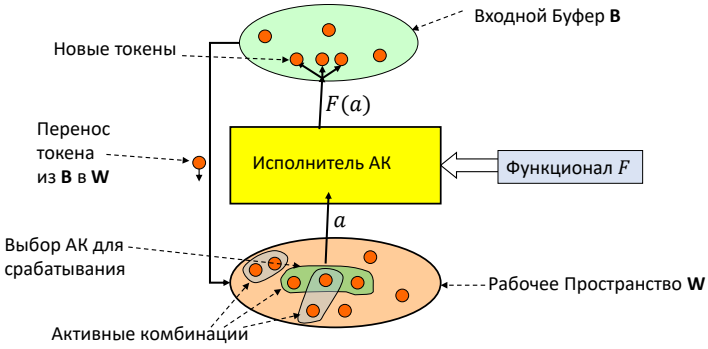


Рисунок 1. Абстрактная Буран-машина

Текущее состояние Буран-машины характеризуется парой (B, W). Начальное состояние обычно имеет вид $B, \varnothing$.

Для удобства дальнейшего изложения введем понятие *быть готовой*.

Определение 4. *Будем говорить, что АК a* готовая *(относительно текущего W), если $a \in \mathcal{A}$ и $a \subseteq W$.*

Состояние машины может изменяться в результате одного из двух действий:

1. *Прием токена*: выбирается произвольный токен $t \in B$ и перекладывается в $\mathbf{W}$. Формально: $B' = B - \{t\}$, $W' = W + \{t\}$.
2. *Срабатывание АК*: пусть имеется готовая АК a, и при этом нет другой готовой АК a', такой что $Pr(a', a)$, тогда a может быть выбрана для срабатывания. При срабатывании АК a она сначала изымается из W, после чего к остатку сразу добавляется $H(a, a)$, а к B добавляется $F(a)$. Формально: $W' = W - a + H(a, a)$, $B' = B + F(a)$.

Операции $+$ и $-$ в действии 2 выполняются как операции над мультимножествами. При этом само действие 2 выполняется как одно неделимое (атомарное) действие. Особенно существенно, что атомарной является модификация содержимого $\mathbf{W}$, поскольку следующий шаг будет выполняться уже с его новым значением W'. А вот вычисление $F(a)$ с последующей модификацией $\mathbf{B}$ «в реальности» может быть отложено на любое неопределенное время, поскольку передача результирующих токенов через буфер $\mathbf{B}$ все равно позволяет им там «отлеживаться» неопределенно долго.

В обоих действиях имеется недетерминизм. В действии 1 это произвол выбора токена в буфере **B**. В действии 2 – произвольный выбор одной из нескольких АК, присутствующих в **W** (и незаблокированных другой АК). Наконец, есть недетерминизм выбора действия 1 или 2, однако, этот недетерминизм может быть устранен введением *принципа немедленного исполнения.*

Определение 5. Конфигурацией *будем называть пару* (P, S)*, где* $P = (\mathcal{T}, \mathcal{A}, F, H, Pr)$ *– программа,* $S = (B, W)$ *– состояние, и* $B, W \in \mathcal{T}_{MS}$.

Работа Буран-машины состоит в последовательном изменении состояния в конфигурации путем последовательного исполнении описанных выше шагов. На каждом шаге состояние $S = (B, W)$ переходит в состояние $S' = (B', W')$. Тот факт, что из состояния S возможен переход в состояние S' за один шаг будем обозначать как $S \longrightarrow S'$. Если при этом выполняется прием токена t, то переход обозначается $S \xrightarrow{t} S'$. Аналогично, если это срабатывание АК a, то $S \xrightarrow{a} S'$.

Отношение $\longrightarrow$ задает на множестве всевозможных состояний $\mathcal{S}$ (заданной программы P) структуру направленного графа $\mathcal{G} = (\mathcal{S}, \longrightarrow)$. Он называется графом (или пространством) состояний. Исследование графа состояний может дать информацию для верификации программы. Если в графе состояний имеются циклы, то они говорят о возможности зацикливания программы. Если зафиксировать начальное состояние S_0, то будет полезно изучить множество всех достижимых из S_0 состояний $Reach(S_0) = \{S | S_0 \xrightarrow{*} S\}$. (Здесь через $\xrightarrow{*}$ обозначено рефлексивно-транзитивное замыкание отношения $\longrightarrow$.)

Начальное состояние S_0 содержит мультимножество исходных токенов B_0 во входном буфере **B** и пустое мультимножество в рабочем пространстве **W**. Работа заканчивается, когда не может быть выполнен шаг, то есть буфер **B** пуст, а в **W** нет ни одной АК.

Входное мультимножество B_0 не обязано быть конечным. Например, можно допустить применение, когда на вход поступает неограниченная последовательность токенов. При этом разумно считать, что они будут поступать в РП в своем порядке и с более низким приоритом по отношению к токенам, порожденным внутри. Но мультимножество W будет, тем не менее, всегда конечным, поскольку после n шагов оно не может содержать более чем n токенов (шаг типа 1 увеличивает размер B на 1, а шаг типа 2 может токены только удалять или заменять) .

Важное замечание. Изложенное в этом разделе определение модели вычислений имеет существенный дефект, и в следующем разделе будет внесено уточнение, которое его исправляет.

4. Дополнительные принципы работы Буран-машины

4.1. Токены как объекты

Выше говорилось, что токены – это элементы множества $\mathcal{T}$, и что среди токенов, находящихся в $\mathbf{B}$ или $\mathbf{W}$ могут быть одинаковые. Поэтому их содержимое мы считали мультимножествами. Однако, для ряда рассуждений этого будет недостаточно: иногда нужно будет различать одинаковые (как элементы $\mathcal{T}$) токены, а иногда, наоборот, различные по значению токены тем не менее считать *одним и тем же* токеном.

Будем считать, что в хранилищах Буран-машины каждый токен содержится как объект, имеющий ссылку, по которой его можно найти и идентифицировать. Ссылка может считаться дополнительным атрибутом самого токена – identity. Объект-оболочка создается при первом появлении токена в буфере $\mathbf{B}$ и сохраняется при его перемещении в $\mathbf{W}$. Новый объект появляется для каждого входного токена, а также для каждого отдельного элемента $F(a)$ при срабатывании a. Причем, даже когда имеется несколько одинаковых (как элементов $\mathcal{T}$) токенов, для каждого экземпляра создается свой объект-оболочка со своим identity.

Но: токены, «порождаемые» функцией H не считаются новыми. Наоборот, при срабатывании АК a всякий токен $H(a,t)$ получает identity от своего «предшественника» t. Иными словами, в токене-объекте его старое содержимое t просто подменяется новым $H(a,t)$. А поскольку для (сильно) допустимого H, токены t и $H(a,t)$ неразличимы для функций $\mathcal{A}, F, Pr$, то с точки зрения этих функций (но не H) все выглядит так, как будто токены-объекты вообще никогда не изменяются.

4.2. Принцип однократной активации

В связи с тем, что токены АК могут сохраняться после срабатывания, возникает вопрос: а могут ли сохраниться все токены АК? И если да, то, то сможет ли АК сработать повторно? На первый вопрос ответ утвердительный: такие ситуации возможны в существующих конкретизациях, и потому не хотелось бы их запрещать. При этом активация производится только один раз, после чего данная АК считается «отработавшей» и больше

никогда не срабатывает. На уровне абстрактной модели постулируется *принцип однократной активации*:

Всякая активная комбинация токенов может сработать не более одного раза.[2]

Мы пишем «не более одного раза», а не «ровно один раз», поскольку, как было отмечено, при пересекающихся активных комбинациях срабатывание одной может привести к исчезновению другой (в результате удаления одного из ее токенов). При этом важно, что идентичность АК опирается на идентичности ее токенов и только на них, то есть АК считается той же самой до тех пор, пока она состоит из тех же (в смысле identity) токенов. Поэтому, если в **W** появится новый такой же (как элемент $\mathcal{T}$) токен, но с новым identity, то это будет другая АК.

Одним из следствий этого принципа будет следующий возможный сценарий: пусть имеется АК из нескольких токенов: (a, b, c). И пусть при ее срабатывании все они сохраняются. Поместим в РП n_a токенов типа a, n_b токенов типа b, n_c токенов типа c. Отдельные токены могут различаться каким-то «полезным грузом», который не влияет на работу распознавателя АК и функции сохранения. Тогда произойдет по одному срабатыванию для каждого набора токенов $\{a_i, b_j, c_k\}$, всего $n_a n_b n_c$ срабатываний. Иначе говоря, токены будут взаимодействовать по принципу «каждая тройка по одному разу». По концу все исходные токены будут по-прежнему находиться в **W**, а все их АК будут иметь статус «отработавших».

Чтобы в модели этот принцип поддержать, следует внести уточнение в определение 4:

Определение 6. *Будем говорить, что* a готовая *(относительно текущего* W*), если* $a \in \mathcal{A}$*, и* $a \subseteq W$ *и ранее в текущем вычислении АК* a *еще не срабатывала.*

Все остальное можно оставить без изменения, поскольку мы лишь наложили ограничение на выбор очередной АК. Однако, само это ограничение не из приятных: поскольку оно апеллирует к истории вычисления, то оно не может быть поддержано без расширения понятия состояния. Необходимо, как минимум концептуально, ввести в состояние еще одно хранилище, где будут запоминаться все сработавшие АК. (На

[2] Справедливости ради, следует отметить, что в проекте БУРАН была (и формально пока остается) возможность, нарушающая этот принцип. Это кратное срабатывание АК из двух токенов с кратностями *до исчерпания одной из кратностей*. Однако, ни в каких содержательных задачах это пока не нашло применения.

практике можно, конечно, ограничиться только теми, у которых все их токены сохраняются при их срабатывании, и лишь пока они продолжают находиться в РП.) К счастью, существует более простой и, что важно, практичный метод поддержать данный принцип, к чему мы и переходим.

4.3. Принцип немедленного выполнения

Реализация принципа однократной активации может представлять непростую техническую проблему, если все токены уже лежат в РП. Тогда надо где-то еще хранить информацию о сработавших активациях, чтобы не допускать повторов. Но можно пойти другим путем. Постулируем следующий *принцип немедленного (или последовательного) выполнения*:

После того, как токен из ВБ перешел в РП, должны быть выполнены все активные комбинации с новым токеном в некотором порядке.

Иначе говоря, перед приемом следующего токена из ВБ, в РП не должно оставаться активных комбинаций, помимо «отработанных».

Чтобы этот принцип соблюсти, будем после прихода очередного токена в РП отыскивать в РП все новые АК с участием нового токена и активировать их. Когда таких АК больше нет, открывается возможность приема в РП нового токена. С учетом этой стратегии теперь нет произвола в выборе действия: 1 или 2; выполняем действие 2 всегда, когда это возможно, иначе переходим к 1.

Возникает вопрос: ограничивает ли этот принцип возможный недетерминизм. То есть, можно ли представить вычисление без него, которое в принципе невозможно с ним? В принципе да, ограничивает. Рассмотрим программу, для которой начальное состояние содержит два токена, u и v, и пусть в $\mathcal{A}$ есть три активных комбинации: $\{u\}$, $\{v\}$, $\{u, v\}$. Тогда, если без принципа немедленного исполнения возможно вычисление из одной АК $\{u, v\}$, то с ним – нет, поскольку тогда при любом порядке приема токенов u и v смогут сработать только «одиночные» АК $\{u\}$ или $\{v\}$. Но такая «свобода» вряд ли полезна на практике (разве только в теории), поэтому данное ограничение можно наложить без сожаления. Тем более, что во всех существующих проектах-конкретизациях ППВС БУРАН данный принцип соблюдается. На практике, при разработке программ, полагаться на него можно, но не рекомендуется (может усложняться проверка корректности и повышаться риск ошибок). Он не поддерживается в CPN, но там нет и сохранений (всегда $H(a, t) = \varnothing$), тогда как кратность всегда равносильна нескольким экземплярам. Отдельные конкретизации также вправе его не

соблюдать, если могут обеспечить принцип однократной активации как-то иначе. И для удобства теоретического анализа иногда этот принцип может отменяться.

4.4. Алгоритм поиска активных комбинаций

Выше в описании Буран-машины мы использовали операцию обнаружения и выбора АК без объяснения способа ее выполнения. Теперь, используя названные принципы, мы можем описать ее алгоритмически.

Пусть в РП пришел очередной токен t_1 из ВБ. Упорядочим все остальные токены РП в некотором порядке. Теперь будем перебирать все возможные АК с этим новым токеном. Чтобы найти все АК из двух элементов, достаточно пройтись один раз по всем токенам РП, обозначим отобранный токен t_2. Если возможны АК из трех элементов, будем для каждой пары (t_1, t_2) перебирать в качестве третьего (t_3) все идущие за t_2 в упорядоченном списке. Аналогично для четырех элементов потребуется третий вложенный цикл и т.д. Проходя по всем множествам, отберем все являющиеся элементами $\mathcal{A}$, то есть потенциальные АК. Глубина вложенности перебора конечна, поскольку ограничена размером текущего РП. Проход по всем токенам следующего уровня может быть ускорен с учетом знания уже отобранных токенов путем использования какого-то вида ассоциативной памяти токенов.

Функция Pr накладывает на отобранные АК частичный пред-порядок. Линеаризуем его и затем будем последовательно выполнять все АК от больших по приоритету к меньшим, учитывая, что в результате удаления токенов какие-то из отобранных АК могут исчезнуть. В пределах множества несравнимых (максимальных) АК выбор недетерминирован. Новые токены, создаваемые посредством F, отправляются «на передержку» в ВБ. Некоторые старые токены в РП могут быть модифицированы, но, благодаря слабой допустимости, ни одна из «старых» АК не перестанет быть АК, и не могут появиться новые. Поэтому процесс будет закономерно завершен.

Заметим, что мы нигде не учитываем «отработанность» АК: раз токен t_1 новый, то все они новые. С другой стороны, процесс перебирает все АК-кандидаты не более чем по одному разу, поэтому повторно они не сработают, даже если полностью сохраняться в РП. Если в РП и останутся наборы токенов из $\mathcal{A}$, то все они будут отработанными, и наш алгоритм их никогда больше не отберет. Кстати теперь видно, почему

нужна слабая допустимость. При ее нарушении могла бы появиться в РП новая АК, ранее не замеченная на этапе цикла отбора АК. Это привело бы к нарушению условия целостности, состоящего в том, что могут оставаться только отработавшие АК.

5. Понятия вычисления и вычислительного графа

5.1. Вычислительный процесс (вычисление)

Определение 7. *Пусть задана программа P и начальное состояние $S_0 = (B_0, \varnothing)$.* Вычислительным процессом *(или просто* вычислением*) назовем пару (P, σ), где $\sigma = \{S_i | i = 0, 1, \ldots, n\}$ – последовательность состояний S_i, такая что для всех $i = 1, \ldots, n$ выполняется $S_{i-1} \longrightarrow S_i$. Если $S_n = (\varnothing, W_n)$, где W_n не содержит АК, то такое вычисление будем называть* завершенным.

Из определения 7 следует, что последовательность состояний вычисления $S_i = (B_i, W_i)$ задается рекуррентными соотношениями:

$B_{i+1} = B_i - \{t\}$, $W_{i+1} = W_i + \{t\}$, если на i-м шаге выполняется прием токена $t \in B_i$,

$B_{i+1} = B_i + F(a)$, $W_{i+1} = W_i - a + H(a)$, если на i-м шаге совершается срабатывание АК $a \subset W_i$.

Вычислительный процесс называется *бесконечным*, если последовательность $\{S_i\}$ бесконечная. Конечные процессы делятся на завершенные и *незавершенные*. Незавершенный процесс может быть продолжен.

5.2. Вычислительный граф

Сущность вычисления может быть описана вычислительным графом, который позволяет абстрагироваться от определенного порядка следования действий. При этом опять придется учитывать, что токены в хранилищах являются объектами, обладающими identity. (В скобках заметим, что для формального определении вычислительного процесса как готовой последовательности состояний этого не понадобилось.)

Пусть задано вычисление EX. В нем одни переходы совершают действие 1, другие – действие 2. Нас интересует только последовательность действий 2 – срабатываний АК. Пусть a_0, a_1, a_2,... – последовательные сработавшие АК в вычислении EX. Их множество обозначим $A = \{a_i\}$. А

чтобы о действиях 1 можно было забыть, объединим хранилища **B** и **W** в одно **BW** и переформулируем определение действия 2:

$$BW_{i+1} = BW_i - a + H(a) + F(a)$$

Здесь и далее функцию H считаем сильно допустимой.

Определение 8. Вычислительным графом *вычисления* X *назовем двудольный граф* $\Gamma = (T,\ A,\ E,\ M)$*, где вершины типа* T*, или* T*-вершины, это токены-объекты, возникающие в вычислении* X*, вместе с* S_0*, то есть* $T = S_0 \cup \bigcup_i F(a_i)$*. А вершины типа* A*, или* A*-вершины это срабатывания* AK *(они же* – вычислительные кванты*), и* $E \subseteq T \times A \cup A \times T$ *– множество стрелок. Каждая* A*-вершина характеризуется активной комбинацией* a *и ее результатом* $F(a)$*. Входная стрелка в* A*-вершину проводится из каждого токена* $t \in a$*, выходная из* A*-вершины – в каждый токен* $t \in F(a)$*. Разметка* $M : T \times A \to \mathbb{B}$ *отражает режим сохранения:* $M(t, a) = true$ *тогда и только тогда, когда* $H(a, t) \neq \varnothing$*.*

Понятие графа вычисления выражает ключевую структуру конкретного вычисления. Разные (в смысле порядка действий) вычисления могут обладать одним и тем же графом. Ниже определим изоморфизм графов, который будет устранять еще некоторые различия между графами. Мы хотим, чтобы вычисления с изоморфными графами всегда давали один и тот же результат. Пример графа показан на рисунке 2.

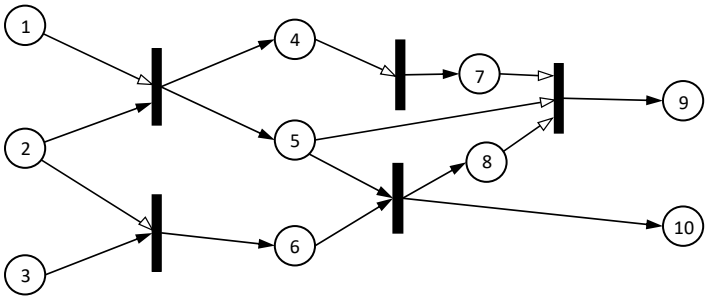


Рисунок 2. Пример вычислительного графа. Т-вершины пронумерованы и показаны кружочками, А-вершины полосками. Стрелка (из t в a) имеет белый конец, когда она стирающая, то есть когда $M(t, a) = false$, иначе – черный.

Говоря о токенах, следует различать токены как Т-вершины и токены как элементы $\mathcal{T}$. Когда нужно подчеркнуть разницу, будем обозначать

Т-вершины жирным шрифтом. Каждая Т-вершина $\mathbf{t}$ содержит значение t типа $\mathcal{T}$, которое обозначается как $Val(\mathbf{t})$. Функция Val естественным образом расширяется на наборы (множества) Т-вершин, при этом результатом будет мультимножество значений типа $\mathcal{T}$ («мульти» – поскольку в разных Т-вершинах в принципе могут содержаться одинаковые, как элементы типа $\mathcal{T}$, значения). Функция Val – это просто приведение типа, поэтому будем ее опускать, когда ее можно вывести из типа аргумента.

С вершиной a типа A связаны два набора Т-вершин: входных $\bullet a$ и выходных $a\bullet$, при этом $F(\bullet a) = a\bullet$. Т-вершина получает свое identity при порождении – либо как элемент B_0, либо как элемент выходного набора А-вершины. Отсюда следует, что в каждую Т-вершину идет не более одной стрелки, но выходить могут несколько. И поэтому вычислительный граф всегда ацикличен.

Понятие вычислительного графа позволяет говорить о зависимостях между Т-вершинами вычислительного графа.

Определение 9. *В вычислительном графе* Т-*вершина* u непосредственно зависит *от* Т-*вершины* v, *обозначается* $v \to u$, *если для некоторой* А-*вершины* a *выполняется* $v \in \bullet a \quad \& \quad u \in a\bullet$.

Определение 10. *В вычислительном графе* Т-*вершина* u зависит *от* Т-*вершины* v, *обозначается* $v \twoheadrightarrow u$, *если существуют* $v = t_0 \to t_1 \to \ldots \to t_n = u$, *для* $n > 0$. *В противном случае, вершины* u *и* v *называются* независимыми.

Поскольку А-вершина a (как срабатывание АК) полностью определяется набором ее входных вершин $\bullet a$ (представляющих собой саму сработавшую АК), будем вместо $\bullet a$ писать просто a, а вместо $a\bullet$ – $F(a)$. Кроме того, будем часто позволять себе Т-вершины называть и обозначать как просто токены (снабженые identity). А множества B и W считать множествами Т-вершин. Когда при срабатывании токен t заменяется токеном $H_a(t)$, в РП остается та же Т-вершина $\mathbf{t}$, но с другим значением. Каждая Т-вершина изначально появляется в ВБ, затем однократно переносится в РП, и находится там, пока не будет удалена срабатыванием АК, которая ее не сохраняет. Заметим, что согласно правилам работы в текущем состоянии Буран-машины не может находиться двух токенов с общим identity.

Приведем несколько простых признаков независимости двух вершин.

Лемма 1. *Если две разные* T*-вершины* u, v *в некоторый момент обе присутствуют в* $\mathbf{B}$*, то они независимы.*

Доказательство. (Хотя оно и кажется очевидным). Пусть $v \twoheadrightarrow u$, это значит, что $v \to t_1$ для некоторого t_1. Следовательно, v уже перешел в $\mathbf{W}$ и уже отсутствует в $\mathbf{B}$. Противоречие. □

Утверждать то же относительно $\mathbf{W}$ нельзя, поскольку токен v при срабатывании может сохраниться и породить новый u, который будет от него зависеть и (когда-нибудь) придет в $\mathbf{W}$.

Лемма 2. *Если две разные* A*-вершины* a, b *в некоторый момент обе готовы, то они независимы.*

Доказательство. Пусть $a \twoheadrightarrow b$, это значит, что $a \to t_1$ для некоторого t_1. Следовательно, a уже отработала, когда появилась b, а значит, потеряла готовность. Противоречие. □

Пусть X – вычисление по программе P с начальным состоянием S. Множество (возможно, бесконечное) всех таких вычислений обозначим через $\mathcal{X}_P(S)$, или просто $\mathcal{X}(S)$, когда программа P предполагается заданной. В любом случае оба компонента S – B и W (точнее, Val от них) должны быть элементами $\mathcal{T}_{MS}$. Здесь мы не ограничиваемся сугубо начальными состояниями, где $W = \varnothing$.

Множество $\mathcal{X}_P(S)$ естественным образом разбивается на три непересекающихся подмножества: завершенные вычисления, незавершенные конечные и бесконечные, $\widehat{\mathcal{X}}$, $\dot{\mathcal{X}}$ и $\ddot{\mathcal{X}}$ соответственно.

6. Понятие об изоморфизме и детерминизме вычислений

Интересен вопрос об условиях, которые бы гарантировали детерминированность результата. А чтобы точно определить понятие детерминированности, нам понадобится понятие изоморфизма вычислений, которое будет определяться через изоморфизм их графов.

Определение 11. *Пусть задана программа* P*, состояние* S_0 *и два разных вычисления* $X_1, X_2 \in \mathcal{X}_P(S_0)$*. Будем называть их графы* $\Gamma_1 = (T_1, A_1, E_1, M_1)$ *и* $\Gamma_2 = (T_2, A_2, E_2, E_2)$ *(а вместе с ними и сами вычисления)* изоморфными*,* $\Gamma_1 \leftrightarrow \Gamma_2$*, если имеется изоморфизм* $I : T_1 \leftrightarrow T_2$ *и* $I : A_1 \leftrightarrow A_2$ *между их множествами* T*- и* A*-вершин, такой что:*

(1) *I тождествен на S_0;*

(2) *изоморфные A-вершины имеют изоморфные входные и выходные наборы токенов: $I(\bullet a) = \bullet(Ia)$, $I(a\bullet) = (Ia)\bullet$;*

(3) *множества T-вершин, обладающих исходящей белой стрелкой (не сохраняющей токен), изоморфны (при этом сами их белые стрелки не обязательно идут в изоморфные A-вершины);*

(4) *изоморфные токены попарно конгруэнтны: $I(t) \cong t$.*

Определение изоморфизма выглядит как достаточно жесткое и даже кажется избыточным: зачем, например, утверждать конгруэтность условием (4), если, как представляется, оно может быть получено структурной индукцией по графу. Но проблема может возникнуть при наличии нескольких выходных токенов у одной A-вершины: изоморфизм может их просто перепутать, тем нарушив конгруэтность сопоставленных токенов. А потому и приходится конгруэнтность постулировать явно. Теперь мы можем сформулировать и доказать желаемое свойство пары изоморфных вычислений.

Лемма 3. *Если два завершенных вычисления X и X' изоморфны, то их заключительные состояния тоже изоморфны, причем так, что их соответственные токены попарно конгруэнтны.*

Доказательство. Токен $t \in T$ входит в заключительное состояние вычисления X тогда и только тогда, когда из соответствующей T-вершины графа Γ_X не выходит ни одной белой стрелки. Множества таких вершин двух графов изоморфны, поскольку получены вычитанием изоморфных подмножеств (T-вершин, обладающих исходящих белой стрелкой – по условию (3) изоморфизма графов) из изоморфных полных множеств T и T'. Конгруэнтность соответственных токенов вытекает непосредственно из условия (4) изоморфизма. □

Таким образом, состав результата вычисления однозначно выводится из структуры его графа. Например, по виду графа на рисунке 2 можно заключить, что останутся токены 3,6,9,10. Теперь мы можем определить понятие детерминированного вычисление.

Определение 12. *Вычисление $X \in \mathcal{X}_P(S)$, не являющееся незавершенным, называется* детерминированным, *если оно изоморфно любому другому вычислению $X' \in \mathcal{X}_P(S)$.*

Следствие 1. *Если вычисление $X \in \mathcal{X}_P(S)$ детерминировано, то любое изоморфное ему вычисление $X' \in \mathcal{X}_P(S)$ тоже детерминировано.*

Следствие 2. *Если вычисление $X \in \mathcal{X}_P(S)$ детерминировано и завершено, то его заключительное состояние совпадает (с точностью до изоморфизма) с конечным состоянием любого другого вычисления $X' \in \mathcal{X}_P(S)$.*

Эти следствия представляют собой основные свойства детерминированных вычислений. В частности, они показывают, что определенное нами свойство вычислений по праву называется «детерминированным». Важно, что нам удалось определить его на самом абстрактном уровне. Но пока у нас нет критерия, или признаков, по которым мы могли бы судить о детерминированости конкретного вычисления. О непосредственной проверке «в лоб» (через перебор всех возможных вычислений) можно говорить лишь теоретически. Но уже можно говорить о разрешимости свойства завершенного вычисления быть детерминированным. (О незавершенных нельзя, поскольку они бесконечные).

Следствие 3. *Во всякой конкретизации абстрактной модели вычислений задача распознавания детерминированных вычислений среди всех завершенных вычислений разрешима.*

Доказательство. Конкретизация должна позволять в любом состоянии находить все возможные активные комбинации за конечное время, проверяя их применимость (например, используя алгоритм, аналогичный описанному в подразделе 4.4, учитывая, что в любом достижимом состоянии (B, W) число токенов в $\mathbf{W}$ конечно). Поэтому в каждом состоянии имеется лишь конечный список возможных шагов. Пусть имеется завершенное вычисление X. Будем строить полное дерево перебора всех вычислений с той же начальной конфигурацией. После каждого шага будем проверять, что выбранная АК присутствует в исходном вычислении (убеждаясь, что она не была учтена ранее на прямом пути из корня, и поддерживая изоморфизм и конгруэнтность всех новых токенов и АК), иначе остановимся с выводом «недет». Изоморфизм поддерживается путем его пролонгации на выходные токены выбранной АК. По достижении конечного состояния (листа дерева) будем убеждаться, что все АК исходного вычисления были учтены (на прямом пути из корня), иначе также останов «недет». Поскольку исходное вычисление завершено, то в нем имеется только конечное число АК, и потому дерево перебора будет конечным. Завершение перебора дает вывод «дет». □

Описанную процедуру «перебора всех вычислений» можно заметно ускорить, применив динамическое программирования. Для этого потребуется память для хранения большого числа множеств активных комбинаций. Каждый узел дерева характеризуется некоторым множеством сработавших АК. Будем после каждого шага запоминать текущее множество (например, представляя его упорядоченным списком номеров АК), отбрасывая ветвь при повторах. Если в данном вычислении было M АК, то количество шагов будет ограничено числом 2^M (вместо M^M без такой оптимизации).

Теперь можно поставить вопрос о нахождении существенно более эффективной процедуры проверки детерминированности, быть может частичной, хотя бы для некоторых практических классов вычислений. Хотелось бы найти полиномиальное (от длины вычисления) решение вместо экспоненциального.

7. Конкретизации абстрактной модели

Различные вычислительные модели могут быть получены как конкретизации абстрактной модели вычислений Буран. Здесь представим две такие модели: архитектура ППВС БУРАН и Coloured Petri Nets (CPN). Их подробное и достаточное для понимания описание читатель может найти в статье [8], а здесь мы представляем их особенности как конкретизации базисных функций.

7.1. ППВС БУРАН

Пространство токенов $\mathcal{T}$ структурировано следующим образом. Программа описывает набор узлов, каждый узел имеет несколько портов. Токен посылается на порт узла. Группа токенов, идущих на все порты одного узла образует АК.

Токены имеют контекст – структуру из полей целого типа. Для каждого узла в программе прописан свой формат контекста. В токене каждое поле контекста имеет либо конкретное значение, либо «*», что значит «не важно». Токены одной АК обязаны иметь одинаковые или совместимые контексты. Последнее означает, что в токенах АК вместо каждой «*» можно поставить некоторое значение так, что их контексты будут совпадать. Иными словами, должен существовать общий числовой контекст, совместимый с контекстом каждого токена АК.

Узел с приписанным ему конкретным контекстом называется называется *виртуальным* узлом. Можно считать, что токен посылается на конкретный виртуальный узел (если все поля в его контексте имеют числовые значения) или сразу на множество виртуальных узлов (если в его контесте встречается «*»). АК есть набор токенов, направленных на все порты одного виртуального узла. С учетом «*» в контекстах разные АК могут иметь общие токены.

У каждого токена может быть задана *кратность* – число активаций, в которых он может участвовать. Поэтому функция H работает так: для каждого токена АК его кратность уменьшается на 1 (то есть токен изменяется), и если остается больше 0, токен сохраняется, иначе стирается. Такая модификация является сильно допустимой, поскольку считается, что функции $\mathcal{A}$, F, Pr кратность «не видят». Если все токены (оба, когда портов два) одной АК имеют кратность больше 1, принцип однократной активации препятствует ее повторному исполнению. В архитектуре ППВС эта ситуация называется «двойной групповой узел». Кратность может быть бесконечной, и тогда H ее не изменяет.

Механизм кратностей возникает как конкретизация функции H, если она выразима в виде $H(a,t) = h(t)$, где $h : \mathcal{T} \to \mathcal{T} \cup \{\varnothing\}$. Иными словами, действие H на совокупности токенов любой АК сводится к действию на каждом токене некоторой общей, не зависящей от a, функции h, результат которой есть всегда либо токен, либо $\varnothing$. При этом собственно кратность это сколько раз нужно итеративно примененять h до получения $\varnothing$. Если всегда получается токен, то это бесконечная кратность.

Некоторые узлы могут иметь *ветви*, когда для активации достаточно присутствия токенов на части портов. Например, узел с двумя портами A и B может активироваться также по наличию одного токена на порте A. При этом АК {A,B} имеет приоритет над АК {A}. Тем самым, токен A как бы «тестирует» присутствие токена B: если находит, то выполняется двухпортновая ветвь, а если не находит – однопортовая. В архитектуре ППВС такой токен A называется «ПП-запрос».

Другой пример – узел с портами A, B, C с тремя ветвями, активируемыми по приходу любых двух из трех токенов. В ППВС считается, что порты A и B основные, а токен C называется «косвенность»: он как бы «перехватывает» один из основных токенов на другую ветвь.

Третий пример – токен «чистки» Clear, который стирает все, с чем сопоставляется по ключу (имени узла и контексту), а затем самоуничтожается. Функция H для пары, содержащей токен Clear, всегда его оставляет,

стирая другой независимо от его кратности, а для одиночного токена Clear – выдает $\varnothing$. Пара имеет приоритет над одиночным.

Здесь хочется отметить еще одну полезную особенность архитектуры ППВС БУРАН – это суммирующие узлы. На них приходит множестве токенов-слагаемых и один особый токен-количество. Последний содержит количество остальных слагаемых и, возможно, еще одно дополнительное слагаемое. Для таких узлов активной комбинацией является вся совокупность этих токенов при условии, что количество в токене-количестве в точности равно числу токенов-слагаемых. Функция F может зависеть только от суммы всех слагаемых, но не может ничего знать об отдельных слагаемых (хотя может «видеть» их общее количество). Суммирующие узлы могут иметь контекст, но токены на них могут иметь только конкретный чисто числовой контекст (без «*») и не могут иметь (отличных от 1) кратностей. Как частный случай, токен-количество с количеством 0 сам по себе образует АК, при этом его «дополнительное слагаемое» и будет общей «суммой».

Суммирующие узлы активно применяются, наряду с двойными групповыми узлами, в таких алгоритмах как умножение матриц (в том числе разреженных) [9], а также обучение/инференс нейросетей [10, 11].

7.2. Coloured Petri Nets

Цветные Сети Петри (CPN) [7] являются обобщением обычных сетей Петри. В них активная комбинация это набор токенов, позволяющих сработать некоторому переходу. При этом всем переменным перехода T (элементам $Var(T)$) должны быть присвоены такие допустимые значения, которые превращают выражения на каждой входящей дуге перехода в значение на своем токене. В CPN нет сохраняемых при срабатывании токенов: при срабатывании все токены АК удаляются и появляются новые, образованные теми же значениями переменных из $Var(T)$ на выходных дугах. Поэтому $H(a) = \varnothing$ для всех активных комбинаций a. И тогда принцип однократной активации не актуален, а принцип немедленного исполнения не нужен и не соблюдается. Переходам в CPN могут быть приписаны приоритеты (по умолчанию у всех NORM), поэтому функция $Pr(a, b)$ работает путем простого сравнения приоритетов.

Однако, у CPN есть небольшая особенность (она свойственна сетям Петри вообще), которая нашей абстрактной моделью не покрывается: переход не определяется однозначно набором своих входных токенов. Для

одного набора входных «мест» (у нас это АК) может быть определено насколько разных переходов, один из которых выбирается недетерминированно. Соответственно, будет недетерминирован и результат срабатывания. В принципе это можно моделировать через недетерминированную функцию F. Однако, мы считаем его детерминированным, чтобы результат действия был однозначно определен выбором АК. Но сам выбор АК не детерминирован.

Заключение

В статье определен базовый понятийный аппарат, который может использоваться для описания, изучения и обсуждения свойств различных конкретных потоковых архитектур. Их основу составляет взаимодействие токенов. Правила взаимодействия представлены абстрактными функциями. В их числе: распознаватель активных комбинаций $\mathcal{A}$ (зависящий только от самой АК), отношение приоритета между ними Pr и функция порождения новых токенов F. Особую роль играет функция H, позволяющая совершать небольшие атомарные модификации токенов при обеспечении непрерывности их пребывания в РП.

Предложенный аппарат позволяет на абстрактном уровне формулировать условия, обеспечивающие выполнение разных полезных свойств. В частности – говорить о детерминизме вычислений. Определено понятие вычисления, а среди них – детерминированного вычисления. В общем случае вычисление, даже если оно детерминированное, не задается однозначно своей начальной конфигурацией. Но каждое конкретное вычисление порождает вычислительный граф, и уже этот граф для детерминированного вычисления определен однозначно с точностью до изоморфизма. Заключительные состояния изоморфных вычислений (то есть порождающих изоморфные графы) также изоморфны, что и позволяет нам называть такие вычисления «детерминированными».

Но определение детерминизма вычисления не дает нам эффективного рецепта по проверке детерминированности – если, конечно, не считать «рецептом» полный перебор всех допустимых вычислений с заданной исходной конфигурацией. Поэтому остается открытым вопрос о возможных эффективных способах и процедурах проверки. Это будет предметом дальнейших исследований.

Список использованных источников

[1] Бурцев В. С. *Выбор новой системы организации выполнения высокопараллельных вычислительных процессов, примеры возможных архитектурных решений построения суперЭВМ* // Бурцев В. С. *Параллелизм вычислительных процессов и развитие архитектуры суперЭВМ*, М.: Торус Пресс.– 2006.– ISBN: 5945880442.– 414 с.

[2] Стемпковский А. Л., Левченко Н. Н., Окунев А. С., Цветков В. В. *Параллельная потоковая вычислительная система — дальнейшее развитие архитектуры и структурной организации вычислительной системы с автоматическим распределением ресурсов* // Информационные технологии.– 2008.– № 10.– С. 2–7 ✻ URL.

[3] Климов А. В., Левченко Н. Н., Окунев А. С., Стемпковский А. Л. *Вопросы применения и реализации потоковой модели вычислений*, МЭС-2016 (Москва, Зеленоград, Россия, 3–7 октября 2016 г.) // Проблемы разработки перспективных микро- и наноэлектронных систем.– 2016.– № 2.– С. 100–106 ✻.

[4] Климов А. В. *Средства верификации распределения вычислений в потоковой архитектуре ППВС «Буран»*, МЭС-2020 (Москва, Зеленоград, Россия, 5–8 октября 2020 г.) // Проблемы разработки перспективных микро- и наноэлектронных систем.– 2020.– № 4.– С. 236–243 DOI ✻.

[5] Котов В. Е. *Сети Петри.*– М.: Наука.– 1984.– 160 с.

[6] Petri C.A. *Kommunikation mit Automaten*, PhD thesis.– University of Bonn.– 1962.– 128 pp. URL

[7] Jensen K. *Coloured Petri nets: A high level language for system design and analysis* // *Advances in Petri Nets 1990*, ICATPN 1989, Lecture Notes in Computer Science.– vol. **483**, Berlin–Heidelberg: Springer.– 1991.– Pp. 342–416.– ISBN: 978-3-540-53863-9 DOI.

[8] Климов А. В. *Цветные сети Петри и язык распределенного программирования UPL: их сравнение и перевод* // Программные системы: теория и приложения.– 2023.– Т. **14**.– № 4(59).– С. 91–122 DOI URL.

[9] Змеев Д. Н., Окунев А. С. *Разработка и исследование алгоритма задачи перемножения разреженных матриц для параллельной потоковой вычислительной системы «Буран»*, МЭС-2018 (Москва, Зеленоград, Россия, 1 октября–5 октября 2018 г.) // Проблемы разработки перспективных микро- и наноэлектронных систем.– 2018.– № 3.– С. 16–23 DOI ✻.

[10] Змеев Д. Н., Левченко Н. Н., Климов А. В. *Алгоритм прямого распространения в парадигме потока данных* // Программные системы: теория и приложения.– 2025.– Т. **16**.– № 4(67).– С. 51–79 doi URL.

[11] Змеев Д. Н., Левченко Н. Н., Климов А. В. *Алгоритм обратного распространения ошибки в парадигме потока данных* // Программные системы: теория и приложения.– 2026.– Т. **17**.– № 3(72).– С. 237–273 doi URL.

Поступила в редакцию 28.05.2026;
одобрена после рецензирования 10.09.2026;
принята к публикации 26.09.2026;
опубликована онлайн 10.10.2026.

Рекомендовал к публикации *к.ф.-м.н. С. А. Романенко*

Информация об авторе:



Аркадий Валентинович Климов

старший научный сотрудник Отделения перспективной микроэлектроники НИЦ "Курчатовский институт". Научные интересы: параллельные системы и алгоритмы, нетрадиционные языки и модели вычислений.

0000-0002-7030-1517

e-mail: *arkvklimov@yandex.ru*

Декларация об отсутствии личной заинтересованности: *благополучие автора не зависит от результатов исследования.*

EN RU PROGRAM SYSTEMS: THEORY AND APPLICATIONS ISSN 2079-3316
Research Article THEORETICAL FOUDATIONS OF SOFTWARE SYSTEMS

UDC 004.432+004.272
10.25209/2079-3316-2026-17-4-3-32

Buran: A parallel dataflow computation model based on token interaction

Arkady Valentinovich **Klimov**✉
National Research Center "Kurchatov Institute", Moscow, Russia
✉*arkvklimov@yandex.ru*

Abstract.

An abstract computation model is presented, which is a generalization of a family of architectures and models based on token interactions. Tokens are viewed as abstract entities. Interactions between them generate computational quanta, which can generate new tokens.

The model is generally nondeterministic. However, an individual computation can be deterministic. This article provides a mathematical description of the model, defining the concepts of computation and deterministic computation. The fundamental decidability of the determinism of a given computation is proven, but the question of how to recognize it more efficiently remains open.

Being abstract, the model can have various concretizations. One of them is the project of the unique multi-core architecture of the BURAN parallel dataflow computing system (PDCS), proposed by V.S. Burtsev in the late 1990s, as well as some of its extensions. The name of the abstract model itself was borrowed from this project. Work on it is currently ongoing at the Kurchatov Institute National Research Center.

The closest known analogue of the model are Petri nets and their extensions, such as Colored Petri Nets (CPN). They can also be described as a concretization of the abstract model. (*In Russian*).

Key words and phrases: abstract computation model, parallel computation model, determinism, nondeterminism, token, interaction, computational graph, PDCS BURAN, Petri nets

2020 *Mathematics Subject Classification*: 68Q09; 68T01, 68W10

Acknowledgments:
The work was carried out within the state assignment of NRC "Kurchatov institute"

For citation: Arkady V. Klimov. *Buran: A parallel dataflow computation model based on token interaction*. Program Systems: Theory and Applications, 2026, **17**:4(73), pp. 3–32. (*In Russ.*). https://psta.psiras.ru/read/psta2026_4_3-32.pdf

© Klimov A.V. 2026

References

[1] V. S. Burcev, “Selecting a new system for organizing the execution of highly parallel computing processes, examples of possible architectural solutions for building supercomputers”: Burcev V. S., *Parallelizm vychislitel’nyx processov i razvitie arxitektury superEVM*, Torus Press, M., 2006, ISBN: 5945880442, 414 pp. (in Russian).

[2] A. L. Stempkovskij, N. N. Levchenko, A. S. Okunev, V. V. Cvetkov. “Parallel dataflow computing system — the further development of architecture and the structural organization of the computing system with automatic distribution of resources”, Informacionnye texnologii, 2008, no. 10, pp. 2–7 ✻ URL (in Russian).

[3] A. V. Klimov, N. N. Levchenko, A. S. Okunev, A. L. Stempkovskij. “The application and implementation issues of dataflow computing system”, MES-2016 (Moskva, Zelenograd, Rossiya, 3–7 oktyabrya 2016 g.), Problemy razrabotki perspektivnyx mikro- i nanoelektronnyx sistem, 2016, no. 2, pp. 100–106 ✻ (in Russian).

[4] A. V. Klimov. “Tools for verification of computation distribution in the parallel dataflow computing system (PDCS) “Buran””, MES-2020 (Moskva, Zelenograd, Rossiya, 5–8 oktyabrya 2020 g.), Problemy razrabotki perspektivnyx mikro- i nanoelektronnyx sistem, 2020, no. 4, pp. 236–243 doi ✻ (in Russian).

[5] V. E. Kotov. *Petri Nets*, Nauka, M., 1984, 160 pp. (in Russian).

[6] C.A. Petri. *Kommunikation mit Automaten*, PhD thesis, University of Bonn, 1962, 128 pp. URL

[7] K. Jensen. “Coloured Petri nets: A high level language for system design and analysis”, *Advances in Petri Nets 1990*, ICATPN 1989, Lecture Notes in Computer Science, vol. **483**, Springer, Berlin–Heidelberg, 1991, pp. 342–416, ISBN: 978-3-540-53863-9 doi.

[8] A. V. Klimov. “Coloured Petri nets and the language for distributed programming UPL: their comparison and translation”, Program Systems: Theory and Applications, **14**:4(59) (2023), pp. 91–122 doi URL (in Russian).

[9] D. N. Zmeev, A. S. Okunev. “Development and investigation of algorithm of sparse matrices multiplication task for the parallel dataflow computing system “Buran””, MES-2018 (Moskva, Zelenograd, Rossiya, 1 oktyabrya–5 oktyabrya 2018 g.), Problemy razrabotki perspektivnyx mikro- i nanoelektronnyx sistem, 2018, no. 3, pp. 16–23 doi ✻ (in Russian).

[10] D. N. Zmeev, N. N. Levchenko, A. V. Klimov. “Forward propagation algorithm in the dataflow paradigm”, Program Systems: Theory and Applications, **16**:4(67) (2025), pp. 51–79 (in Russian).

[11] D. N. Zmeev, N. N. Levchenko, A. V. Klimov. “Backpropagation algorithm in the dataflow paradigm”, Program Systems: Theory and Applications, **17**:3(72) (2026), pp. 237–273 (in Russian).