ИПМ им.М.В.Келдыша РАН

Абрау-2019 • Труды конференции

А.И. Легалов, И.А. Легалов,
И.В. Матковский

Особенности семантики статически типизированного языка функционально- потокового параллельного программирования

Рекомендуемая форма библиографической ссылки

Легалов А.И., Легалов И.А., Матковский И.В. Особенности семантики статически типизированного языка функционально- потокового параллельного программирования // Научный сервис в сети Интернет: труды XXI Всероссийской научной конференции (23-28 сентября 2019 г., г. Новороссийск). — М.: ИПМ им. М.В.Келдыша, 2019. — С. 489-500. — URL: http://keldysh.ru/abrau/2019/theses/08.pdf doi:10.20948/abrau-2019-08

Размещена также презентация к докладу

Особенности семантики статически типизированного языка функционально-потокового параллельного программирования

А.И. Легалов, И.А. Легалов, И.В. Матковский

Сибирский федеральный университет

Аннотация. Рассматриваются особенности языка функционально-потокового параллельного программирования Smile, использующего статическую типизацию данных. Разработанный ранее язык функционально-потокового параллельного программирования Пифагор поддерживает динамическую типизацию, что не обеспечивает эффективной трансформации написанных программ в программы для современных параллельных вычислительных систем. Проводится анализ изменений функционально-потоковой модели вычислений и операторов языка программирования с введением статической системы типов. Показано, каким образом вносимые изменения влияют на синтаксис и семантику. В частности отмечается, необходимость использования принципа единственного присваивания, обусловленного формированием хранилищ данных конкретного типа, объявляемых по аналогии с переменными.

Ключевые слова: парадигмы программирования, параллельное программирование, функционально-потоковое параллельное программирование, статическая типизация, модели параллельных вычислений, полиморфизм.

Specifics of semantics of a statically typed language of functional and dataflow parallel programming

A.I. Legalov, I.A. Legalov, I.V. Matkovsky

Сибирский федеральный университет

Annotation. The features of the dataflow functional parallel programming language using static data typing are considered. The previously developed language Pifagor supports only dynamic typing, which does not provide an effective transformation of written programs into programs for modern parallel computing systems. The analysis of changes in the dataflow functional model of calculations and programming language operators with the introduction of a static type system is carried out. It is shown how changes made affect syntax and semantics. In particular, it is noted that it is necessary to use the principle of single assignment, due to the formation of data stores of a particular type, declared by analogy with variables.

Keywords: programming paradigms, parallel programming, functional-dataflow parallel programming, static type system, parallel computation models, polymorphism.

Введение

Современные методы разработки параллельных программ сильно привязаны к особенностям архитектур параллельных вычислительных систем (ПВС), что находит соответствующее отражение в языках программирования. Практически любые изменения архитектуры ПВС ведут к переписыванию и модификации уже разработанного и отлаженного программного обеспечения. Попыткой преодолеть это является применение концепции архитектурно-независимого параллельного программирования (АНПП), ориентированного на разработку программ с использованием языковых и инструментальных средств для абстрактных (виртуальных) параллельных систем с неограниченными вычислительными ресурсами и стратегиями управления вычислениями по готовности данных. Такие подходы развиваются в разных направлениях. В частности можно отметить язык программирования COLAMO, разработанный для систем на кристалле [1, 2]. Создание универсальных языков, напрямую не связанными с архитектурными ограничениями, можно проследить на примере функциональных языков параллельного программирования Sisal [3] и Пифагор [4].

Наиболее последовательно концепция АНПП нашла свое отражение в языке функционально-потокового параллельного программирования Пифагор. Она непосредственно учитывается в его модели функционально-потоковых параллельных вычислений. Модель программы описывается как ресурсно-неограниченный ациклический безусловный граф, в котором управление осуществляется по готовности данных, а также используется принцип единственного использования вычислительных ресурсов [4]. На уровне модели предполагается, что для выполнения любых операций выделяются свои уникальные ресурсы, реальное распределение которых осуществляется после того, как разработана и отлажена логическая структура программы. Для апробации возможностей языка разработаны инструментальные средства, поддерживающие процесс создания, преобразования и выполнения функционально-потоковых параллельных программ [5].

Однако следует отметить невысокую эффективность выполнения программ, что обуславливается использованием интерпретатора. Последнее связано с тем, что в языке применяется динамическая типизация данных, а представленные в модели вычислений операторы обладают динамическим поведением, позволяя формировать списки произвольной размерности во время вычислений. В связи с этим практически невозможна эффективная трансформация написанных программ в современные статически типизированные языки, используемые при реальном параллельном программировании.

Вместе с тем, эксперименты, проводимые с применением разработанных инструментальных средств, показали возможность эффективного использования данной парадигмы для оптимизации [6], формальной верификации [7] и отладки [8] программ, еще до того момента, как начнется их

преобразование к конкретной архитектуре. Это позволяет иметь программу, перенос которой на реальные ПВС мог бы осуществляться более формально за счет наложения ресурсных ограничений, учитывающих особенности конкретной архитектуры, с сохранением уже отлаженной общей логики функционирования.

В связи с этим перспективной видится модификация функционально-потоковой модели параллельных вычислений (ФПМПВ), направленная на учет особенностей организации данных в современных языках программирования, что позволило бы упростить процесс трансформации функционально-потоковых параллельных (ФПП) программ. В основном эта модификация связана с применением статической типизации и фиксацией размерности списковых и контейнерных структур данных, что ведет к пересмотру ряда концепций ФПМПВ. Вполне естественно, что в соответствии с этими изменениями должен измениться и язык ФПП программирования.

В результате проведенных исследований сформирована статически типизированная модель функционально-потоковых параллельных вычислений (СТМФППВ). Как и предшествующая ей ФПМПВ она определяет программу как информационный граф с управлением по готовности данных. Однако операторы, описывающие алгоритм программы, разработаны с учетом возможных преобразований в статически типизированные языки программирования, что ведет к изменению ряда аксиом и алгебры преобразований. На основе предложенной модели разработан статически типизированный язык функционально-потокового параллельного программирования (СТЯФППП) Smile.

1. Операторы модели, ориентированные на статическую типизацию

Как и в предшествующей ФПМПВ [4], операторы задают узлы информационного графа, в котором вычисления выполняются по готовности данных. Однако существует ряд особенностей, связанных с изменением требований. Необходимо обеспечить поддержку следующих свойств, характерных для статически типизированных языков программирования:

- эффективную трансформацию статически типизированных функционально-потоковых параллельных программ в другие модели вычислений вместо их интерпретации;
- повышение контроля за счет использования сильной типизации;
- сохранить принцип управления по готовности данных и общую концепцию функционально потоковой модели параллельных вычислений;
- каждый из программоформирующих операторов должен опираться на типизированные данные, контролируемые на этапе компиляции;

- контейнерные (списковые) данные должны иметь фиксированный размер, определяемый либо во время компиляции, либо во время выполнения;
- аксиоматика языка должна быть упрощена, чтобы уменьшить число динамических проверок и преобразований во время выполнения;
- упрощение алгебры эквивалентных преобразований.

Приведенные требования ведут к изменению практически всех операторов ФПМПВ, в результате чего сформирована модель вычислений, обладающая иными свойствами. Эти свойства определяются через особенности функционирования программоформирующих операторов СТМФППВ.

Оператор интерпретации описывает функциональные преобразования аргумента. Он имеет два входа, на которые через информационные дуги поступают функция **F** и аргумент **X**. Как аргумент. так и функция могут являться результатами предшествующих вычислений. Основными особенностями новой версии данного оператора являются:

- типы аргументов на входах оператора, должны быть известны во время компиляции;
- тип результата на выходе также вычисляется во время компиляции;
- на входе и выходе оператора допускаются именованные типы данных, неименованные структуры и кортежи;
- для именованных типов допустима только именованная эквивалентность;
- для неименованных структур и кортежей допускается структурная эквивалентность;
- предопределенность базовых операций, для которых на уровне языка прописаны все возможные типы данных аргументов и результатов

Исходя из этого, для базовых функций языка изначально определяются сигнатуры, задающие типы аргументов и результатов. Для функций, определяемых пользователем, типы аргументов и результатов явно задаются во время определения функций. Допускается дуализм некоторых базовых данных, которые в зависимости от использования в операторе интерпретации могут выступать как в качестве аргумента, так и функции. В этом случае для них возможно определение двойного типа (тип данных и сигнатура функции).

Оператор интерпретации запускается по готовности данных, что фиксируется появлением разметки на входных дугах. Получение результата задается разметкой выходной дуги.

Вместо группировки в список данных в СТМФППВ используются группирование в кортеж (tuple).

Можно выделить следующие основные свойства оператора группирования в кортеж:

- размер кортежа определяется во время компиляции (что обусловливается необходимостью знать типы группируемых данных и их размер);
- элементы кортежа являются данными именованных типов;
- обеспечивается сравнение на структурную эквивалентность с другими кортежами;
- для неименованных структур и кортежей допускается неявное сравнение на структурную эквивалентность;
- допускается структурная эквивалентность с неименованными структурами;
- готовность кортежа к выполнению определяется по готовности всех его данных;
- отсутствуют внутренние эквивалентные преобразования, изменяющие размер кортежа во время выполнения (сигнал, удаляемый из списка в ФПМПВ, является типом данных без значения и сохраняется в явном виде).

Изменены также аксиомы, определяющие преобразование списков данных во время вычислений, что также обуславливается введением дополнительного контроля во время компиляции.

Группировка в параллельные списки заменяется на группировку в рой (swarm). Используется для группировки данных, над которыми выполняется одна массовая операция. К свойствам роя относятся:

- размер роя определяется во время компиляции;
- элементами роя являются данные одного именованного типа или все элемента роя структурно эквивалентны;
- готовность роя к выполнению определяется по готовности хотя бы одного элемента (асинхронность в обработке отдельных его элементов);
- отсутствуют внутренние эквивалентные преобразования, изменяющие размер роя во время выполнения;
- внутри кортежей рой не вырождается в последовательность элементов кортежа, а является единым элементом.
- алгебра эквивалентных преобразований роев реализуется только во время компиляции.

Приведенные характеристики позволяют рассматривать рой в качестве набора независимых данных, запускаемых по мере их поступления. На выходе оператора интерпретации также формируется рой, состоящий из элементов одного типа.

Группировка в задержанный список заменяется на оператор задержки вычислений (delay), который отличается от ранее предлагаемой возвратом только одного значения, тип которого определяется во время компиляции и может быть любым. В языке с динамической типизацией результатом являлся

параллельный список. В новой модели выдача вместо параллельного списка роя тоже возможна, но только при явном его задании в качестве результата задержки. Раскрытие задержки осуществляется сразу же после того, как она становится аргументом оператора интерпретации. Это позволяет в ряде случаев использовать данный оператор в качестве скобочного выражения, изменяющего приоритет операций.

2. Организация данных

В отличие от языка ФПП программирования Пифагор, в котором представлены только базовые типы данных, язык программирования Smile имеет развитую систему типов, обуславливаемую необходимостью повышения контроля на этапе компиляции. Вводимые базовые типы данных во многом повторяют типы, используемые в современных статически типизированных языках. Однако помимо этого предлагаются типы, обеспечивающие возможность манипуляции с параллельными списками, что ведет к их определенному влиянию на СТМФППВ.

Можно выделить следующие базовые типы: целый, булевый, сигнальный, функциональный, ошибки. Эти типы являются основообразующими и используются не только при обработке произвольных данных, но и в ключевых операторах языка. Дополнительные типы, такие как действительные числа, символы и другие рассматриваются как расширения, определяемые проблемной ориентацией, и могут включаться в различные предметно-ориентированные версии языка. В целом можно отметить, что вопросы, связанные с расширением базовых типов не являются принципиальными на уровне модели вычислений.

К составным относятся следующие типы: массив, структура, кортеж, обобщение, рой, поток, функциональный, ссылочный. Эти типы используются для формирования производных абстракций, определяемых программистом, и состоят как из базовых, так и производных типов. Они в основном подменяют ранее используемые понятия списка данных и параллельного списка, однако при этом являются описаниями, а не операторами, что позволяет на их основе формировать соответствующие хранилища данных, используемые по принципу единственного присваивания. Массив, структура и кортеж являются специализированными разновидностями списка данных ФПМПВ.

Тип **«массив»** (array) предназначен для описания данных одного типа. Во многом он аналогичен использованию многомерных массивов традиционных императивных языков программирования. Массив имеет фиксированные размерность и длины по каждому измерению. Описание данного типа на уровне языка программирования задается с использованием следующего синтаксиса:

Массив ::= ИмяТипа «(» Размерность «)»

Размерность ::= Целое { «,» Целое }

Примеры массивов:

```
A << type int(100)
```

```
B << type bool(30, 40)
```

Тип **«структура»** обеспечивает группировку разнотипных данных по аналогии со структурными типами различных языков программирования. Структура состоит из полей, каждое из которых имеет имя и тип. Описание структуры имеет следующий синтаксис:

Структура ::= «(» ПолеСтруктуры { «,» ПолеСтруктуры } «)»
ПолеСтруктуры ::= ИмяПоля «@» ИмяТипа
| «[» ИмяПоля { «,» ИмяПоля } «]» «@» ИмяТипа

Примеры структурных типов:

```
Triangle << type (a@int, b @ int, c @int)
Rectangle << type ([x,y]@int)
```

Тип **«кортеж»** отличается от структуры отсутствием именованных полей. По сути он аналогичен вектору, но может содержать разнотипные элементы. Обращение к элементам кортежа осуществляется по номеру поля. Для задания кортежей используется следующий синтаксис:

Кортеж ::= «(» ИмяТипа { «,» ИмяТипа } «)»

Примеры задания типов кортежей:

```
C << type (int)
B << type (int, bool, signal)
```

Тип **«обобщение»** во многом аналогичен по организации и использованию обобщениям, используемым в императивных языках. Основной его задачей является описание вариантных данных структур. Существуют различные подходы к организации обобщений, включая методы, поддерживающие полиморфизм. В языке используются обобщения, поддерживающие процедурно-параметрическую парадигму программирования, которая обеспечивает более гибкую поддержку эволюционного расширения программ по сравнению с другими подходами [9]. Правила, определяющие синтаксис обобщений имеют следующий вид:

Обобщение ::= «{» ПолеОбобщения { «,» ПолеОбобщения } «}»
ПолеОбобщения ::= ИмяТипа { «,» ИмяТипа }
| ИмяПризнака «@» ИмяТипа
| «[» ИмяПризнака { «,» ИмяПризнака } «]» «@» ИмяТипа

Примеры описания обобщений:

```
Figure1 << type {Triangle, Rectangle}
Figure2 << type {trian@Triangle,
                 rect@Rectangle,
                 rhomb@Rectangle}
WeekDay << type{[Sun,Mon,Tue,Wen,Thu,Fri,Sat]@signal}
```

Тип **«рой»** используется для описание независимых данных, над которыми возможно выполнение массовых параллельных операций. Все элементы роя имеют один тип, а функция, осуществляющая их обработку, может одновременно выполняться над каждым элементом. Результатом

является также рой, размерность которого равна размерности роя аргументов. Синтаксические правила, определяющие данный тип, имеют следующий вид:

Рой ::= ИмяТипа «[» Целое «]»

Пример описания типа

```
R << type int[100]
```

Тип **«поток»** является альтернативой асинхронного списка [10]. Он используется для обработки данных поступающих последовательно и асинхронно в произвольные промежутки времени. Размерность поступающих данных при этом неизвестна, поэтому завершение обработки возможно только по признаку конца потока. Поток готов к обработке при наличии в нем хотя бы одного элемента. Тип всех элементов потока одинаков. Синтаксические правила, определяющие поток:

Поток ::= ИмяТипа «{» «}»

Пример описания потокового типа

```
A << type int{}
```

Тип **«функция»** (или функциональный тип) позволяет задать сигнатуру функции, определяя имя типа, тип аргумента, а также тип результата. В целом определение функционального типа отличается от общепринятых только тем, что любая функция имеет только один аргумент и возвращает только один результат. Синтаксические правила, определяющие описание функционального типа:

ФункциональныйТип ::= func Аргумент «->» Результат

Аргумент ::= ИмяТипа | Кортеж

Результат ::= ИмяТипа | Кортеж

Примеры описаний:

```
F << type func int -> int
F2 << type func (bool, int, int) -> (int, bool)
```

Тип **«ссылка»** (или ссылочный тип) обеспечивает поддержку указателей на различные хранилища определенного типа, что позволяет передавать значения между функциями без их копирования. Основное назначение заключается в дополнительном контроле типов в ходе передач. Синтаксические правила, определяющие описание ссылочного типа:

Ссылка ::= «&» ИмяТипа

ОткрытыйМассив ::= ИмяТипа «(» «*» { «,» «*» } «)»

3. Описание функций

В отличие от языка ФПП программирования Пифагор при описании функций используется явное задание типов аргумента и результата, что обеспечивает дополнительный контроль во время компиляции. Эти изменения затрагивают заголовок функции, что определяется следующим синтаксическим описанием:

Функция ::= func Аргумент «->» Результат ТелоФункции

Аргумент ::= ИмяАргумента «@» (ИмяТипа | Кортеж) | Структура

Результат ::= ИмяТипа | Кортеж | Структура

Примеры:

```
Factorial << func n@int -> int {...}
TrianPerimeter << func ([a,b,c]@int) -> int {...}
Sum << func t@(int, int) -> int {t:+ >> return}
```

4. Дополнительные расширения

Наличие статической типизации ведет к появлению дополнительных возможностей по разработке функционально-потоковых параллельных программ. К ним следует отнести описание хранилищ данных предопределенного типа. С ними можно взаимодействовать как с использованием принципа единственного присваивания, так и с применением многократного доступа, аналогичного с доступом к переменным в императивных языках программирования. Последнее ведет к потере надежности программ, но может иногда использоваться после формального доказательства непротиворечивости, например, для ускорения вычислений. Хранилища описываются следующими синтаксическими правилами:

Хранилище ::= (safe | var) Тип

ИменованноеБезопасноеХранилище = ИмяХранилища «@» Тип

Первое правило явно определяет безопасное (safe) хранилище, заполняемое по принципу единственного присваивания (с контролем во время заполнения) или небезопасный вариант (var), при котором не контролируется возможность повторной записи. Второе правило является «синтаксическим сахаром» для описания безопасных хранилищ. Примеры:

```
t1 << safe Triangle ≡ t1@Triangle
t << var Triangle
```

Для взаимодействия с хранилищами необходимо ввести дополнительный набор операций, ведущих к изменению семантики модели вычислений и влияющих на синтаксис языка программирования. Эти операции обеспечивают доступ к хранилищам по чтению и записи. Наличие хранилищ разного типа предопределяет и разнообразие описаний, отражаемых в соответствующих синтаксических правилах:

ЧтениеВсегоХранилища ::= ИмяХранилища «:» Функция

ЧтениеЭлементаМассива ::=

ИмяХранилища «(» индексы «)» «:» Функция

ЧтениеЭлементаКортежаИлиОдномерногоМассива ::=

ИмяХранилища «:» индекс «:» Функция

ЧтениеЭлементаСтруктуры ::=

ИмяХранилища «.» ИмяПоля «:» Функция

ЧтениеЭлементаПотока ::= ИмяХранилища «:» get «:» Функция

Запись_в_хранилище ::= Значение «:» ИмяХранилища

Запись_в_массив ::= Значение «:» ИмяХранилища «(» индексы «)»

Запись_в_кортеж ::= Значение «:» ИмяХранилища «(» индексы «)»

Запись_в_структуру ::= Значение «:» ИмяХранилища «.» ИмяПоля
Запись_в_поток ::= Значение «:» ИмяХранилища

Применение данных операций сопровождается выполнением в надежных хранилищах принципа единственного присваивания и управления по готовности данных.

Заключение

Наличие статической типизации в языке функционально-потокового параллельного программирования обеспечивает более строгий контроль данных, что повышает надежность разрабатываемых программ. Также повышается возможности по проведению более полной оптимизации и формальной верификации. Помимо этого трансформация функционально-потоковых параллельных программ в традиционные языки параллельного программирования становится более простой и эффективной, так как большинство типов данных используют практически однозначное отображение.

Исследование выполнено при финансовой поддержке РФФИ в рамках научного проекта № 17-07-00288.

Литература

1. Левин И.И., Дордопуло А.И., Гудков В.А. Программирование реконфигурируемых вычислительных узлов на языке COLAMO: учебное пособие // Ростов-на-Дону: Издательство ЮФУ, 2016.
2. Дордопуло А.И., Левин И.И. Ресурсонезависимое программирование гибридных реконфигурируемых вычислительных систем // Суперкомпьютерные дни в России: Труды международной конференции (25–26 сентября 2017 г., г. Москва). – М.: Изд-во МГУ. – 2017. – С. 714-723.
3. Kasyanov, V.: Sisal 3.2: functional language for scientific parallel programming. Enterp. Inf. Syst. 7(2), 227–236 (2013). https://doi.org/10.1080/17517575.2012.744854
4. Легалов А.И. Функциональный язык для создания архитектурно-независимых параллельных программ // Вычислительные технологии. № 1 (10), 2005. С. 71–89.
5. Legalov A.I., Vasilyev V.S., Matkovskii I.V., Ushakova M.S. (2018) A Toolkit for the Development of Data-Driven Functional Parallel Programmes. In: Sokolinsky L., Zymbler M. (eds) Parallel Computational Technologies. PCT 2018. Communications in Computer and Information Science, vol 910. Springer, Cham, pp 16-30. DOI https://doi.org/10.1007/978-3-319-99673-8_2.
6. Vasilev V.S., Legalov A.I. Loop-invariant Optimization in the Pifagor Language. / Automatic Control and Computer Sciences, 2018, Vol. 52, No. 7, pp. 843–849. ISSN 0146-4116. DOI: 10.3103/S0146411618070295.

7. Ushakova M.S., Legalov A.I. Verification of Programs with Mutual Recursion in Pifagor Language. / Automatic Control and Computer Sciences, 2018, Vol. 52, No. 7, pp. 850–866. ISSN 0146-4116. DOI: 10.3103/S0146411618070301
8. Удалова Ю.В. Методы отладки и верификации функционально-потоковых параллельных программ / Ю.В. Удалова, А.И. Легалов, Н.Ю. Сиротинина // Журнал Сибирского федерального университета. Серия «Техника и технологии». Апрель 2011 (том 4, номер 2) – С. 213-224.
9. Легалов А.И., Легалов И.А., Матковский И.В. Инструментальная поддержка эволюционного расширения программ при инкрементальной разработке. / Труды XX Всероссийской научной конференции «Научный сервис в сети Интернет» (17-22 сентября 2018 г., Новороссийск). - М.: ИПМ им. М.В. Келдыша, 2018. -- С. 346-359. ISSN 2618-9542. doi:10.20948/abrau-2018. URL: http://keldysh.ru/abrau/2018/proc.pdf.
10. Легалов А.И., Редькин А.В., Матковский И.В. Функционально-потоковое параллельное программирование при асинхронно поступающих данных. / Параллельные вычислительные технологии (ПаВТ'2009): Труды международной научной конференции (Нижний Новгород, 30 марта – 3 апреля 2009 г.). – ISBN 978-5-696-03854-4 – Челябинск: Изд. ЮУрГУ, 2009. С. 573-578. (Электронное издание).

References

1. Levin I.I., Dordopulo A.I., Gudkov V.A. Programmirovanie rekonfiguriruemyh vichiclitelnyh uzlov na yazyke COLAMO: uchebnoye posobie // Rostov-na-Donu: 2016.
2. Dordopulo A.I., Levin I.I. Resursnonezvisimoe programmirovanie gibrydnyh rekonfiguriruemyh vichiclitelnyh system // Russian Supercomputing Days: International conf. (25–26 September 2017, Moskow). – 2017. – pp 714-723.
3. Kasyanov, V.: Sisal 3.2: functional language for scientific parallel programming. Enterp. Inf. Syst. 7(2), 227–236 (2013). https://doi.org/10.1080/17517575.2012.744854
4. Legalov A.I. Funktsionalnyy yazyk dlya sozdaniya arkhitekturno-nezavisimykh parallelnykh programm // Vychislitelnyye tekhnologii. No 1 (10). 2005. S. 71–89.
5. Legalov A.I., Vasilyev V.S., Matkovskii I.V., Ushakova M.S. (2018) A Toolkit for the Development of Data-Driven Functional Parallel Programmes. In: Sokolinsky L., Zymbler M. (eds) Parallel Computational Technologies. PCT 2018. Communications in Computer and Information Science, vol 910. Springer, Cham, pp 16-30. DOI https://doi.org/10.1007/978-3-319-99673-8_2.
6. Vasilev V.S., Legalov A.I. Loop-invariant Optimization in the Pifagor Language. / Automatic Control and Computer Sciences, 2018, Vol. 52, No. 7, pp. 843–849. ISSN 0146-4116. DOI: 10.3103/S0146411618070295.

7. Ushakova M.S., Legalov A.I. Verification of Programs with Mutual Recursion in Pifagor Language. / Automatic Control and Computer Sciences, 2018, Vol. 52, No. 7, pp. 850–866. ISSN 0146-4116. DOI: 10.3103/S0146411618070301.
8. Udalova J., Legalov A., Sirotinina N. Debug and verification of function-stream parallel programs. // Journal of SibFU. Engineering & Technologies, Vol 4, No 2 (2011). P. 213-224.
9. Legalov, A.I., Redkin, A.V., Matkovsky, I.V. Functional streaming parallel programming with asynchronously incoming data / Parallel Computing Technologies (PCT'2009). 2009 – ISBN 978-5-696-03854-4.
10. Legalov, A.I., Legalov, I.A., Matkovsky, I.V. Instrumental support of the evolutionary expansion of programs using a incremental development. / CEUR Workshop Proceedings, Volume 2260, 2018, Pages 346-359. 20th Conference Scientific Services and Internet, SSI 2018; Novorossiysk-Abrau; Russian Federation; 17 September 2018 - 22 September 2018; Code 142666.