К формальной верификации программных комплексов суперкомпьютерного моделирования с использованием предметно-ориентированных языков

В.А. Васенин, М.А. Кривчиков

НИИ механики МГУ имени М.В. Ломоносова

План доклада

- Введение
- К постановке основной задачи исследования
- Разработка и реализация базового языка
- Формальные модели программ и языков программирования
- Апробация подхода
- Заключение

Введение

Программы окружают нас

Например, в следующих сферах жизни общества:

- развлечения
- рабочие места
- коммуникации
- транспорт
- медицина
- наука
- производство
- энергетика
- вооружение

Пример: «Виртуальная АЭС»

- код CMS — моделирование теплогидродинамических процессов в первом и втором контурах реакторов ВВЭР
- разрабатывается ВНИИАЭС с 1990-х гг.
- используется:
  - в тренажёрах для персонала АЭС
  - в том числе — при отработке действий при чрезвычайных ситуациях
  - при проектировании новых АЭС на базе ВВЭР
  - в перспективе — для предупреждения чрезвычайных ситуаций
- задача: реинжиниринг для параллельной вычислительной среды

Программы модифицируются и усложняются

GNU gzip (утилита для сжатия данных)
версии 1.2.4 – 1.6 (1993–2013)
×**6** файлов, ×**7** строк[1]

Ядро ОС FreeBSD i386
версии 2.0.5 – 8.4 (1995-2013)
×**8** файлов, ×**12** строк[1]

[1] Данные получены утилитой CLOC 1.60 для следующих типов файлов: C, C++, C/C++ Header

Программы содержат ошибки

Coverity Scan Open Source Report 2013:
Статический анализ кода 741 Open Source проекта от ~10 тыс. строк до ~8 млн строк, в среднем ~340 тыс. строк на проект
(обнаружено дефектов на 1000 строк кода)

2008 — 0.30

2009 — 0.25

2010 — 0.81

2011 — 0.45

2012 — 0.69

2013 — 0.59

Ошибки в программах могут приводить к катастрофическим последствиям

- «Фобос-Грунт» (2011) — утрачена автоматическая межпланетная станция, ущерб ~5 млрд руб.
- Панама, установки лучевой терапии (2000-2001) — пострадало 28 человек, погибло не менее 5
- Therac-25, установки лучевой терапии (1985-1987) — 6 пострадавших, 3 погибших
- FADEC (Chinook HC.2), отказ ПО управления двигателем вертолёта — одна из вероятных причин крушения в 1994 г. — 29 погибших

Необходим контроль качества программных продуктов

верификация — соответствие продукта требованиям

валидация — адекватность требований практическим потребностям

Подходы к верификации

визуальный контроль

тестирование

статический анализ

формальная верификация

Распространённые подходы не гарантируют отсутствия ошибок

- Визуальный контроль
  - зависит от экспертного мнения;
  - не автоматизирован;
  - на больших объёмах кода сложен в применении.
- Тестирование
  - большой объём кода тестов по сравнению с кодом продукта (SQLite: 80 тыс. строк кода / 90 млн. строк тестов);
  - тесты необходимо поддерживать в актуальном состоянии;
  - тесты не гарантируют отсутствия дефектов/ошибок.
- Статический анализ
  - нетривиальные свойства неразрешимы согласно теореме Риса;
  - большое количество ложных срабатываний;
  - на практике встречаются дефекты, которые не способны обнаружить средства статического анализа (OpenSSL Heartbleed bug: информация раскрыта 7 апреля 2014 г., адаптированные средства статического анализа появились 18 апреля 2014 г.).

Формальная верификация не имеет широкого применения

- верификация на моделях (model checking)
  - исчерпывающая проверка моделей
  - типичные модели: автоматы, сети Петри
- **дедуктивная верификация**
  (deductive, proof-theoretic verification)
  - формальная семантика языка программирования → формальная система
  - доказательство свойства — вывод в формальной системе
  - возможна только частичная автоматизация
- **программирование с зависимыми типами**
  (programming with dependent types)
  - система типов допускает задание формальных спецификаций
  - программы корректны по построению: исходный код содержит доказательства, компилятор осуществляет проверку
  - развитие: автоматизированный вывод корректных по построению программ из формальных спецификаций

Объект исследования

математические модели и основанные на них средства разработки и анализа исходного кода программ с целью удовлетворения требований по их формальной верификации. Такие модели и средства должны допускать использование нескольких языков программирования, в том числе предметно-ориентированных.

Предмет исследования

технологические процессы построения (разработки) и модификации верифицируемого программного обеспечения с использованием предметно-ориентированных языков, в том числе, процессы верификации существующего (унаследованного) кода.

Цели исследования

- построение перспективных моделей технологических процессов создания новых и реинжиниринга унаследованных больших программных комплексов с высокими требованиями по их верификации на основе анализа знаний о области их применения;
- разработка предметно-ориентированных языков на основе модели λ-исчисления с зависимыми типами и их апробация для адекватного описания больших комплексов программ.

К постановке основной задачи исследования

Предыстория и современное состояние исследований

На фоне общих тенденций развития вычислительной техники и инженерии программ рассмотрено $\sim$70 публикаций за 1930-2014 гг.:

- формальные модели вычислений в целом;
- основные этапы развития формальной верификации программ;
- история используемого формализма (λ-исчисление с зависимыми типами);
- применение предметно-ориентированных языков при разработке.

Предыстория и современное состояние исследований

Сделаны следующие выводы:

- в целом может быть выделена тенденция к повышению уровня абстракции от оборудования в используемых и вновь разрабатываемых языках программирования;
- известны результаты по успешному применению лямбда-исчисления с зависимыми типами к задачам, связанным с формальной верификацией программ (компиляторы CompCert и CerCo, исследовательский язык F^*).

Отмечен также следующий подход к распространению программного обеспечения:

- код, несущий доказательства (proof-carrying code) — подход, при котором код программы распространяется совместно с сертификатом, содержащим проверяемое компьютером доказательство его соответствия спецификациям.

Практические аспекты

1. Вопросы удовлетворения требований к верификации продукта, в том числе, с использованием формальных методов, целесообразно рассматривать на этапе предпроектных исследований.
2. На практике в настоящее время актуальна задача реинжиниринга средних и крупных унаследованных программных комплексов. Под реинжинирингом понимается процесс существенной переработки программного продукта с целью удовлетворения новых нефункциональных требований с сохранением полезных свойств исходной версии.
3. Решению задачи формальной верификации может способствовать использование предметно-ориентированных языков — языков программирования, адекватно отражающих специфику предметной области.

Пример предметно-ориентированного языка[1]

```
pumping program P1 for AtLeastOneZone + WithAlarm +
                       SuperPowerCompartment[f=comp1]  {
    parameter defaultWaterLevel : int
    parameter superWaterLevel: int
    event superPowerTimeout

    init {
        set comp1->targetHeight = defaultWaterLevel
    }

    start:
        on (comp1->needsPower == true) && !(comp1->isPumping) {
            do comp1->pumpOn
        }
        on comp1->enough {
            do comp1->pumpOff
        }
        on comp1.superPumping->turnedOn {
            set comp1->targetHeight = superWaterLevel
            raise event superPowerTimeout after 20
        }
        on comp1.superPumping->turnedOff or superPowerTimeout {
            set comp1->targetHeight = defaultWaterLevel
        }
}
```

[1] 2011. Voelter *et al.* Product Line Engineering using Domain-Specific Languages

Подход

Новая разработка

- применяется языково-ориентированное программирование
- предметно-ориентированные языки имеют заданную формальную семантику
- в некоторых случаях возможна автоматическая верификация

Реинжиниринг унаследованного кода

- с использованием предметно-ориентированных языков строится модель целевого компонента
- для прочих компонентов задаётся внешняя формальная спецификация

Задачи исследования

Основная задача исследования: разработать макет средства формальной спецификации и верификации программ, записанных на предметно-ориентированных языках программирования и провести апробацию такого средства.

- определить промежуточное представление на основе модели λ-исчисления с зависимыми типами («базовый язык»);
- задать в промежуточном представлении набор моделей для описания семантики программ, записанных на DSL и языках общего назначения;
- разработать параллельную версию фрагмента программного комплекса моделирования теплогидравлических процессов в АЭС с использованием предметно-ориентированных языков и показать корректность параллельного исполнения кода.

Разработка и реализация базового языка

Исходная модель: исчисление конструкций

1986. T. Coquand, G. Huet, "The Calculus of Constructions".

1989. Z. Luo. "ECC, an Extended Calculus of Constructions".

Типизированное λ-исчисление с конструкторами типов, полиморфизмом и зависимыми типами.

Зависимые произведения:

$$\begin{aligned} A : \mathrm{Type}, B(a : A) : \mathrm{Type} &\Rightarrow \Pi[a : A].B : Type \\ A : \mathrm{Type}, B(a : A) : \mathrm{Type}, b(a : A) : B(a) &\Rightarrow \lambda[a : A].b : \Pi[a : A].B \\ t : \Pi[a : A].B, u : A &\Rightarrow t\,u : B(a) \end{aligned}$$

Зависимые суммы:

$$\begin{aligned} A : \mathrm{Type}, B(a : A) : \mathrm{Type} &\Rightarrow \Sigma[a : A].B : Type \\ A : \mathrm{Type}, B(a : A) : \mathrm{Type}, b(a : A) : B(a) &\Rightarrow \mathrm{pair}_{\Sigma[a:A].B}[a : A].b : \Sigma[a : A].B \\ t : \Sigma[a : A].B &\Rightarrow \mathrm{fst}\,t : A \\ &\Rightarrow \mathrm{snd}\,t : B(\mathrm{fst}\,t) \end{aligned}$$

менее выразительно, чем машина Тьюринга

как формальная система эквивалентно ZFC + infinitely many inaccessible cardinals

Соответствие Карри-Говарда

λ-исчисление с зависимыми типами может быть интерпретировано в рамках интуиционистской логики. При этом типы термов соответствуют утверждениям, а термы, имеющие заданный тип — доказательствам.

Примеры

- функция, возвращающая вектор заданной длины

$$\Pi[x : \mathbb{N}].\mathrm{Vector}(\mathbb{N}, x)$$

- эквивалентность регулярных выражений и ДКА

$$\Pi[A : \mathrm{Type}][r : \mathrm{RegEx}(A)].\quad \Sigma[Q : \mathrm{Type}].[d : \mathrm{RecDFA}(Q, A)].\ \Pi[w : \mathrm{List}(A)].\ (regex\_rec(r, w) \to dfa\_rec(d, w))$$

Гомотопическая теория типов

2013. IAS, "Homotopy Type Theory: Univalent Foundations of Mathematics" / Univalent Foundations Program, 609p.

Типы идентичности (равенства)

$$\frac{x : A \qquad y : A}{x =_A y : \mathrm{Type}} \qquad \frac{x : A}{\boldsymbol{refl}\, x : x =_A x}$$

$$\frac{x, y : A \quad C : A \to \mathrm{Type} \quad z : C(x) \quad \alpha : x =_A y}{J(A, x, y, C, z, \alpha) : C(y)}$$

Каноническое правило редукции: $J(A, x, x, C, z, \boldsymbol{refl}\, x) \longrightarrow z$

- существуют нетривиальные элементы типов идентичности
- элементы типа идентичности можно интерпретировать как пути с точки зрения теории гомотопий

$$\mathbb{Z}_3 =_{\mathrm{Type}} \mathbb{Z}_3$$

- открытый вопрос: свойство каноничности (гипотеза Воеводского)

  пример:

  $plus0 \equiv \lambda[x : \mathbb{N}].0 + x =_{\Pi \mathbb{N}.\mathbb{N}} \lambda[x : \mathbb{N}].x + 0$

  $J(\Pi \boldsymbol{N}.\boldsymbol{N}, \lambda x.0 + x, \lambda x.x + 0, \mathbb{N}, 0, plus0) \not\longrightarrow 0$

Новый результат: дополнительные правила редукции

Задача: разработать способы редукции нетривиальных элементов типов идентичности, таким образом, чтобы некоторые "интуитивно" равные термы приводились правилами редукции к одному виду

Мотивация: использование в рамках формальной теории принципа "изоморфные объекты равны"

Результаты: предложены правила редукции, доказано сохранение типизации предложенными правилами.

$x =_A y \Leftrightarrow t$ (новое отношение: раскрытие типа идентичности)

$$J(\Sigma[x : X].Y, a, b, \lambda[c, \delta].f(c, \delta, \mathrm{fst}\, c), z, \alpha) \longrightarrow$$
$$J(\dots, \lambda[c, \delta].f(c, \delta, \underline{\mathrm{fst}\, b}), \alpha, J(X, \mathrm{fst}\, a, \mathrm{fst}\, b, \dots, z, \mathrm{fst}\, \boldsymbol{out}(\alpha)))$$

Существующие подходы:

2012. D. Licata et al. "Canonicity for 2-dimensional type theory" / ACM SIGPLAN POPL, 2012.

2014. T. Coquand et al. "A model of type theory in cubical sets" (препринт).

Новый результат: базовый язык

В.А. Васенин, М.А. Кривчиков. Приложение одной разновидности типизированного лямбда-исчисления к построению формальных моделей программ. Ломоносовские чтения – 2014. Секция механики – М.: Изд-во Московского Университета, 2014

Язык программирования/спецификации на основе построенной разновидности исчисления конструкций

Синтаксис — Lisp-подобный:

Литералы 2, 3.14, `"ABCDEF"`

Символы `refl`

List `(a b c d ...)` — приложение

Brackets `[a b c d ...]` — формы с именами

Braces `{ a b c d ... }` — аннотации типов

Макросы

- типизированные синтаксические макросы
- предметно-ориентированный язык для сопоставления синтаксических конструкций с образцом
- типизированные предметно-ориентированные процедуры частичного разрешения утверждений

Существующие реализации

- Coq (INRIA, первая версия — 1989, актуальная версия — 2014)
  ML-подобный синтаксиси, поддержка инфиксных операторов, поддержка нетипизированных процедур разрешения, записанных в синтаксисе, отличном от используемого для спецификаций.
- Agda (Chalmers, первая версия — 2007, актуальная версия — 2014)
  Haskell-подобный синтаксис, поддержка инфиксных операторов, поддержка типизированных процедур разрешения.
- Idris (E. Brady, в процессе разработки, актуальная версия — 2014)
  Haskell-подобный синтаксис, поддержка инфиксных операторов
- PVS (SRI, использовался с 1992, актуальная версия — 2006)
  Поддерживает нетипизированные процедуры разрешения, записанные в Lisp (для спецификаций синтаксис собственный)

Формальные модели программ и языков программирования

Общий подход

- не рассматриваются вопросы описания конкретного синтаксиса (правил построения программы как строки символов)[2]
- в рамках базового языка описание абстрактного синтаксиса может быть объединено с описанием статической семантики с помощью макросов
- формальная семантика задаётся с использованием подхода на основе монад

---

[2] Н. Непейвода. Стили и методы программирования. Курс лекций. Учебное пособие. 320 с. М.:Интуит, 2005

Представление формальной семантики программ

1989. E. Moggi. Computational Lambda-Calculus and Monads

Монады (конструкции в теории категорий) задают функциональные блоки, с помощью которых может быть построена формальная семантика языков программирования.

- $M : \mathrm{Type} \to \mathrm{Type}$
- $fmap : \Pi[A, B : \mathrm{Type}].(A \to B).(M\,A \to M\,B)$
- $join : \Pi[A : \mathrm{Type}].(M(M\,A) \to M\,A)$
- (+ monad laws)
- выводимые функции: $bind$, $return$.

Преобразователи монад позволяют строить стеки монад:

- $MT : \Pi[m : \mathrm{Monad}].\Sigma[n : \mathrm{Monad}].[\Pi[A].M_m(A) \to M_m(M_n(A))]$

- Стек монад → DSL

Стандартные монады

1998. N.S. Papaspyrou. Formal Semantics for the C Programming Language

`Error(E)` — поддержка исключений

$A \mapsto A + E$

Стандартные функции: *throw*, *catch*

`State(S)` — поддержка изменяемого состояния

$A \mapsto (S \to \Sigma A.S)$

Стандартные функции: *update*, *get*, *put*

`Partial` (`Res`) — частичность

$A \mapsto Partial\ A$ (коиндуктивный тип)

Стандартные функции: *wait*

`Pow` — многовариантность

$A \mapsto FinSet(A)$

Стандартные функции: *variate*

`Reader(C)`, `Writer(W)`, `Continuation(R)`

Новый результат: статическая формальная семантика стандарта ECMA-335

В.А. Васенин, М.А. Кривчиков. Статическая формальная семантика стандарта ECMA-335. Программирование, №4, 2012. с. 3-16.

- широко используемый промежуточный язык (Microsoft .NET)
- построена денотационная модель статической семантики
- фрагменты модели адаптированы для использования в базовом языке
- рассмотрены обобщённые типы
- может быть задана динамическая семантика с использованием стека монад M: $[\![\cdot]\!] : \mathrm{Ecma335} \to \mathrm{M\ R}$
- фрагмент описания:

$$
\begin{array}{lcl}
\boldsymbol{Method}_{\mathrm{ref}} & = & \boldsymbol{Scope} \times \boldsymbol{Ide} \times \boldsymbol{Method}_{\mathrm{sig}} \\
\boldsymbol{Param} & = & \boldsymbol{Type} \times \boldsymbol{B}^3 \times \boldsymbol{Ide} \\
\boldsymbol{Method}_{\mathrm{code}} & = & \mathrm{Cil} \mid \mathrm{Native} \mid \mathrm{Runtime} \\
\boldsymbol{Method}_{\mathrm{attr}} & = & \boldsymbol{Access} \times \boldsymbol{B}^{10} \times \boldsymbol{Method}_{\mathrm{code}} \times \boldsymbol{Special} \\
\boldsymbol{Method}_{\mathrm{def}} & = & \boldsymbol{Method}_{\mathrm{ref}} \times \boldsymbol{Method}_{\mathrm{attr}} \times (\boldsymbol{N} \to \boldsymbol{Param})
\end{array}
$$

Существующие результаты:

2008. N. Fruja. Towards proving type safety of .NET CIL

2002. G. Butler *et al.* A Formalization of the Common Type System and CLS Rules of ECMA Standard 335: «Common Language Infrastructure»

Апробация подхода

Новый результат: модель динамического параллельного исполнения программ

В.А. Васенин, М.А. Кривчиков. Модель динамического параллельного исполнения программ. Программирование, №1, 2013. с. 45-59.

описывает отложенные вычисления как модификатор семантики

аналогичные модели широко используются на практике

- Lisp Futures
- T-система
- ECMAScript 6 Promises
- C# 5 async

недостаток: интеграция с существующими языками

недостаток: не поддерживает разделяемое состояние

язык управления потоком исполнения:

- `comp` c — произвести вычисления (определяются внешним образом)
- s_1 ; s_2 — последовательность команд
- вызов функции, условный оператор, цикл
- `spawn` t, c, u — порождение новой задачи
- `wait` u, m — ожидание результата от задачи

Разработка параллельной версии программного комплекса моделирования теплогидравлических процессов в АЭС

- теплогидравлическая система задаётся графом объектов различного типа
- шаг моделирования состоит из нескольких десятков стадий, на которых вычисляются различные свойства объектов
- свойства могут зависеть от состояния смежных объектов на предыдущих стадиях
- на определённом этапе производится решение разреженной СЛАУ
- исполнение в режиме реального времени (порядка 80 мс на шаг)
- на целевом аппаратном обеспечении пересылка данных достаточно дорога (порядка единиц мс)
- состояние системы к концу шага должно быть записано в определённую область памяти
- код генерируется автоматически

Первичный анализ

- разработан статический анализатор подмножества языка Fortran, позволяющий определить связность стадий по операциям чтения/записи
- анализатор показал, что степень связности высока, следовательно, затруднительно выбрать гранулы достаточно большого размера для получения ускорения на параллельной системе
- цель: разбить систему на области, в которых производится решение СЛАУ меньшего размера, тем самым сокращая количество необходимых пересылок до двух: в начале и в конце шага вычислений
- затруднение: недоверие со стороны специалистов-предметников к сохранению устойчивости используемых численных методов при разбиении

Задача в рамках настоящего исследования

- подготовка кода к верификации с использованием предметно-ориентированных языков
- выбран следующий набор DSL:
  - язык арифметических выражений с контролем единиц физических величин абстрактной реализацией арифметики (могут быть использованы как числа с плавающей точкой, так и рациональные числа с вычислениями произвольной точности)
  - язык описания типов объектов модели
  - язык объявления стадии вычислений, указывающий объект для стадии и набор используемых свойств внешних объектов
  - язык описания параллельного исполнения

Контроль единиц физических величин

1991. M. Wand, P.M. O'Keefe. Automatic Dimensional Inference
1996. A.J. Kennedy. Programming Languages and Dimensions

Требования к языку:

- расширение — получение значения из состояния модели
- подключение дополнительных типов и функций (массивы, вспомогательные функции, полиморфные по единицам)

Фрагмент описания модели:

$$
\begin{array}{lcl}
float & : & \text{Type};\quad \texttt{+}, \texttt{-}, \texttt{*}, \texttt{/}, \texttt{sqrt}, \ldots \\
\textbf{\textit{Dimension}} & = & \#k \\
\textbf{\textit{UnitSpec}} & = & \textbf{\textit{Vector}}\ k\ \mathbb{Q} \\
\textbf{\textit{Unit}} & = & \text{Base}\ \textbf{\textit{Unit}} \quad | \quad \text{Derived}\ (\textbf{\textit{Unit}} \times (float \to float) \times (float \to float)) \\
\textbf{\textit{Quantity}} & = & \textbf{\textit{Unit}} \to \text{Type} \\
\textbf{\textit{of}} & : & \Pi\{u : \textbf{\textit{Unit}}\}.float \to \textbf{\textit{Quantity}}\ u \\
\textbf{\textit{conv}} & : & \Pi\{u : \textbf{\textit{Unit}}\}.\{fg : float \to float\}. \\
 & & \textbf{\textit{Quantity}}\ (\text{Base}\ u) = \textbf{\textit{Quantity}}\ (\text{Derived}\ ufg)
\end{array}
$$

Статическая семантика: $\Pi\{I\ J : \text{Type}\}.\{ext : J \to \text{Type}\}.\text{Type}$

Объекты модели

- типы объектов — «вершины» и «ребра»
- выделяются переменные и постоянные характеристики объекта
- состав переменных характеристик может зависеть от значений постоянных:

$$
\begin{aligned}
\textbf{\textit{Path – Point – Point}} \quad & \{p_1\, p_2 : \text{Point}\} \\
\text{const} : \quad & \textit{chkValve} : \textbf{\textit{Maybe}}\ (\text{const ChkValve}) \\
\text{var} : \quad & \text{match}\ (\textit{chkValve}) \\
& \text{Just v} \ \Rightarrow\ \textit{vchkValve} : \text{var ChkValve}
\end{aligned}
$$

- типы полей — единицы физических величин, целые числа, объекты-«вершины»

Объявление стадии вычислений

- тип объекта, состояние которого модифицируется на данной стадии
- предикат на постоянных характеристиках
- для «ребер» — предикат на постоянных характеристиках связанных вершин
- для «ребер» — список полей, допустимых к считыванию для каждой связанной вершины

Разделяемое состояние

задача: адаптировать модель динамического параллельного исполнения программ к рассматриваемой системе

подход к решению:

- состояние с гранулируемым доступом на чтение и запись
- из языка управления потоком исполнения исключаются циклы и вызовы внешних функций
- независимость по данным гарантируется при проверке типов базовым языком

- тип состояния: $GS \equiv \Sigma S.(List\ \Sigma G.\{read \mid write\})$
- при запуске в типе идентификатора задачи сохраняется последовательность доступов на чтение/запись
- при получении результата статический контроль типов:

$$\texttt{wait} : \Pi[currState : GS].[taskId : \Sigma T.GS].\\ [independent(currState, \mathbf{snd}\ taskId)].\\ (merge(currState, \mathbf{snd}\ taskId))$$

Результаты

В.А. Васенин, М.А. Кривчиков. Распараллеливание теплогидравлического расчетного кода CMS в составе полномасштабной суперкомпьютерной модели «Виртуальная АЭС». Ломоносовские чтения — 2013. Секция механики — М.: Изд-во Московского Университета, 2013

- разработаны предметно-ориентированные языки, позволяющие описать стадии вычислений в целевом программном комплексе
- предметно-ориентированные языки имеют заданную формальную семантику
- с использованием таких языков построена формальная модель одной из стадий расчёта
- получено формальное доказательство корректности параллельного исполнения для этой стадии
- с использованием статической формальной семантики стандарта ECMA-335 может быть получена статическая формальная модель исходной версии стадии расчёта

Заключение

Общие результаты

1. Базовый язык
   - новая разновидность λ-исчисления с зависимыми типами
   - реализация с поддержкой типизированных синтаксических макросов и предметно-ориентированных процедур разрешения (базовый язык)
2. Формальные модели программ и языков программирования
   - адаптирован к базовому языку существующий подход к описанию формальной семантики с использованием монад
   - построена статическая формальная семантика стандарта ECMA-335
3. Апробация подхода
   - построена модель динамического параллельного исполнения программ;
   - разработаны предметно-ориентированные языки, гарантирующие корректность параллельного расчёта для модели теплогидравлических процессов в АЭС

Дальнейшая работа

- Базовый язык
  - исследование свойств исчисления
  - компиляция
  - средства поддержки разработки
- Формальные модели программ и языков программирования
  - стандартная библиотека для разработки предметно-ориентированных языков
  - средства для работы с синтаксисом
- Апробация подхода
  - формальное доказательство эквивалентности исходной и полученной версии фрагмента кода
  - использование предметно-ориентированных процедур разрешения для автоматизации таких доказательств
  - обобщение и развитие используемых предметно-ориентированных языков

Спасибо за внимание!