Формальная семантика параллельных программ на базе λ-исчисления с зависимыми типами

М.А. Кривчиков

НИИ механики МГУ, механико-математический факультет МГУ

22 апреля 2014 г.

План доклада

- Введение
  - Актуальность
  - Подходы к формальной верификации
  - Практические аспекты
  - Характеристика используемого подхода
- Задача
  - Объект исследования
  - Предмет исследования
  - Цели исследования
  - Задачи исследования
- Результаты
  - Базовый язык
  - Формальные модели программ
  - Верификация параллельных программ
  - Спецификация многоязыковых приложений
- Заключение
  - Новые результаты
  - Дальнейшая работа

Введение

Программы окружают нас

Например, в следующих сферах жизни общества:

развлечения

рабочие места

коммуникации

транспорт

медицина

наука

производство

...

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

Контроль качества продуктов

верификация — соответствие продукта требованиям

валидация — адекватность требований практическим потребностям

Подходы к верификации

визуальный контроль

тестирование

статический анализ

формальная верификация

Контроль качества продуктов

верификация — соответствие продукта требованиям

валидация — адекватность требований практическим потребностям

Подходы к верификации

визуальный контроль

тестирование

статический анализ

формальная верификация

Недостатки распространённых подходов

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.).

Подходы к формальной верификации

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

Подходы к формальной верификации

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

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

Модификации могут быть обусловлены нефункциональными требованиями

Формальная верификация существующей программы

Требования сертификации, нормативных актов

Пример: Common Criteria (ГОСТ Р ИСО/МЭК 15408)

Эффективность выполнения в современных вычислительных средах

Перенос на другую платформу

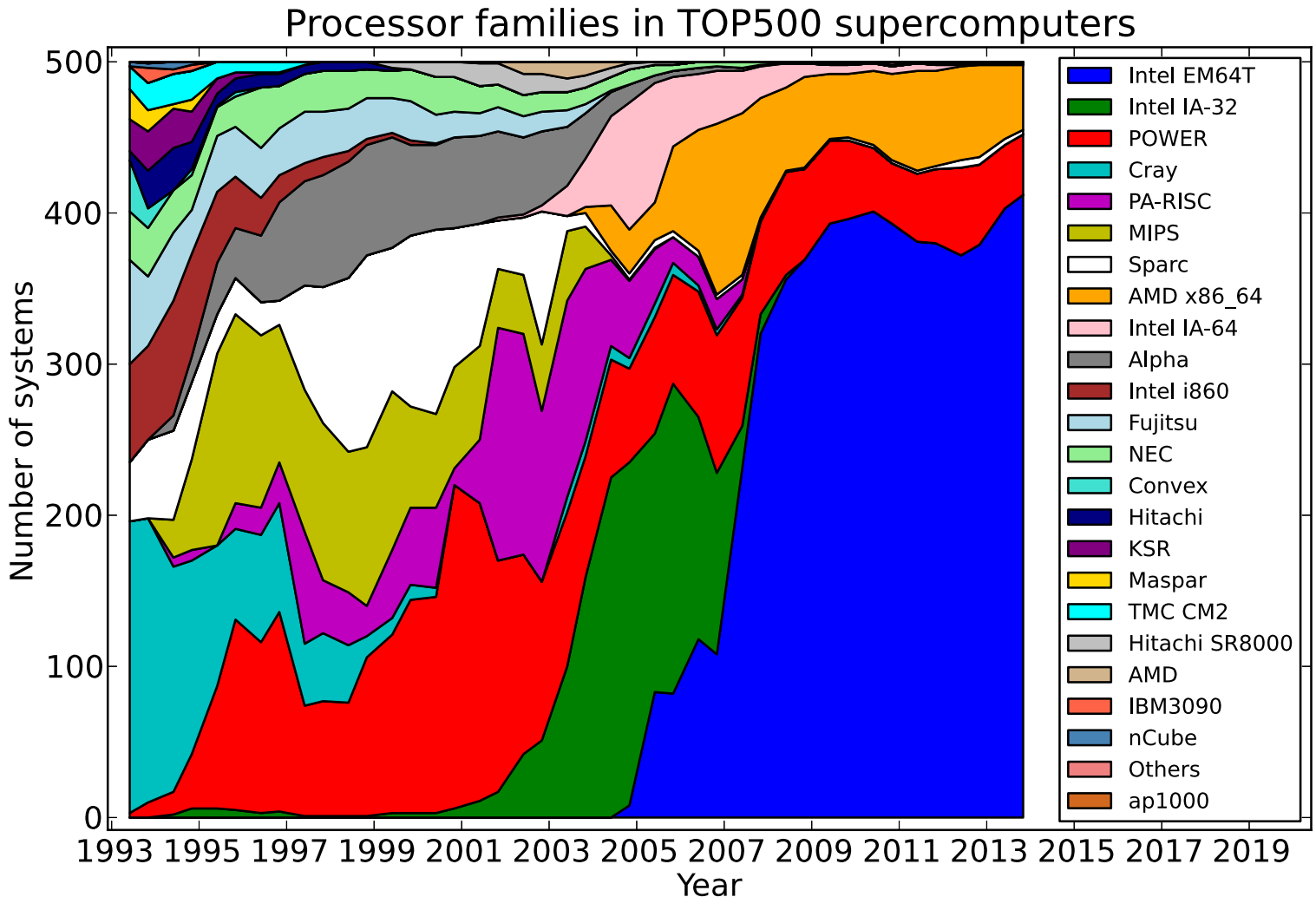
Распараллеливание

Вычислительные ускорители

Распределённые вычислительные среды

Перспективные гетерогенные суперкомпьютеры производительностью порядка экзафлопс (10^{18} операций в секунду)

Наиболее распространённые семейства процессоров в TOP500



Реинжиниринг программ

существенная переработка программного продукта

цель: удовлетворение новых нефункциональных требований

необходимо сохранить полезные свойства исходной версии

затруднения вызывает унаследованный код (legacy code)

Разработка на нескольких языках программирования

Репозитории GitHub, top 1256 (stars, forks), 13.10.2013:

40.5% — один ЯП (51.6% без JavaScript)

26.8% — два ЯП (25.6% без JavaScript)

26.7% — три и более ЯП (22.8% без JavaScript)

остальные — без ЯП (книги, презентации и т.п.)

Предметно-ориентированные языки

(языки, адекватно отражающие специфику предметной области, DSL)

проблемно-ориентированные языки и среды разработки (1970-е)

языково-ориентированное программирование (M. Ward, 1994)

основная идея: декомпозиция решения задачи на DSL и предметно-ориентированный код на нём

DSL могут иметь разрешимые нетривиальные свойства

Примеры DSL (I)

SQL
```
SELECT * FROM accounts WHERE balance > 0
          GROUP BY customer
```

C# LINQ
```
var elements = from element in array
   orderby element descending
   where element > 2
   select element;
```

printf
```
printf("String %s, number %05d, hex %#x, "
   "float %5.2f.\n",
       "str", 89, 255, 3.14159);
```

Примеры DSL (II)[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

строится модель целевого компонента

для прочих компонентов задаётся внешняя формальная спецификация

Задача

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 оценка эффективности их использования для адекватного описания больших комплексов программ.

Экспериментальные задачи

1. Разработать параллельную версию фрагмента программного комплекса моделирования теплогидравлических процессов в АЭС с использованием предметно-ориентированных языков и показать корректность параллельного исполнения кода.
2. Построить формальную спецификацию фрагмента существующей автоматизированной информационной системы, написанной с использованием нескольких языков программирования.

Задачи исследования

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;

решить экспериментальные задачи.

Результаты

Нетипизированное λ-исчисление

1932. A. Church, "A set of postulates for the foundation of logic".

1936. A. Church, "An unsolvable problem of elementary number theory".

$$\boldsymbol{x} \qquad \{\boldsymbol{F}\}(\boldsymbol{X}) \qquad \lambda\boldsymbol{x}[\boldsymbol{M}]$$

$$\{\lambda\boldsymbol{x}[\boldsymbol{M}]\}(\boldsymbol{N}) \longrightarrow_\beta M[x := N]$$

эквивалентно машине Тьюринга

противоречиво как формальная система (парадокс Клини-Россера)

λ-исчисление с простыми типами

1940 A. Church, "A Formulation of the Simple Theory of Types".

Синтаксис:

Типы $\tau ::= \tau \to \tau | T$, где $T \in B$ (множество базовых типов)

Термы $e ::= x \mid \lambda x : \tau . e \mid e\, e \mid c$, где c --- константы

Правила типизации:

$$\Gamma \vdash e : \sigma \qquad \frac{\Gamma, x : \sigma \vdash e : \tau}{\Gamma \vdash (\lambda x : \sigma . e) : (\sigma \to \tau)} \qquad \frac{e_1 : \sigma \to \tau \quad e_2 : \sigma}{e_1\, e_2 : \tau}$$

менее выразительно, чем машина Тьюринга

непротиворечиво как формальная система

проверка типов используется как "защита"от противоречивости

проверка типов разрешима

Расширения типизированного λ-исчисления

1991. H.P. Barendregt, "Introduction to generalized type systems"

Полиморфное λ-исчисление ($\lambda 2$)

1972. J.-Y. Girard, "Interprétation fonctionnelle et élimination des coupures de l'arithmétique d'ordre supérieur"

1974. J. Reynolds, "Towards a Theory of Type Structure".

Универсальная квантификация над типами:

$\tau ::= \ldots \mid \alpha \mid \forall \alpha . \tau$

λ-исчисление с зависимыми типами ($\lambda \Pi$)

1980. N.G. de Bruijn, "A survey of the AUTOMATH project".

Типы могут зависеть от термов:

$$\frac{A : * \qquad \Gamma, x : A \vdash B : Kind}{\Pi x : A . B : Kind} \qquad \frac{A : * \qquad \ldots \qquad \Gamma, x : A \vdash y : B}{\lambda x : A . y : \Pi x : A . B}$$

λ-исчисление с конструкторами типов ($\lambda_{\underline{\omega}}$)

1987. G. Renardel de Lavalette, "Strictness analysis for a language with polymorphic and recursive types"(preprint)

Конструкторы типов:

$Kind ::= * \mid Kind \rightarrow Kind \qquad (\lambda t : * . u) : * \rightarrow * \ldots$

Исчисление конструкций

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

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

Объединяет все расширения (полиморфизм, конструкторы типов и зависимые типы)

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

$$\mathbb{Z}_{256} =_{\mathrm{Type}} (\mathbb{Z}_2')^8$$

открытый вопрос: свойство каноничности

(гипотеза Воеводского)

пример:

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

$J(\Pi \boldsymbol{N}.\boldsymbol{N}, \lambda x.0 + x, \lambda x.x + 0, \mathbb{N}, 0, plus0) \twoheadrightarrow 0$

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"

Недостатки: на настоящее время правила разработаны не для всех видов термов, доказательство некоторых свойств теории пока не получено

$x =_A y \Leftrightarrow t$ (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).

Следствия из нового результата (I)

1 Реализация высших индуктивных типов

определение типа как "носителя"и набор предопределённых элементов типов идентичности на нём

пример применения (Patch Theory):

$$Repo = (repo, commit(a \leftrightarrow b\,@\,i) : repo =_{Repo} repo,$$
$$commute : (a \leftrightarrow b\,@\,i) \circ (c \leftrightarrow d\,@\,j) =_{\dots} (c \leftrightarrow d\,@\,j) \circ (a \leftrightarrow b\,@\,i))$$

правила вывода:

$$\frac{A : \mathrm{Type} \quad e : \Pi\, A.A.\mathrm{Type} \quad \epsilon : \mathrm{Groupoid}(e)}{\boldsymbol{M}(A, e, \epsilon) : \mathrm{Type}}$$

$$\frac{a : A}{\boldsymbol{inT}\ a : \boldsymbol{M}(a, e, \epsilon)} \qquad \frac{a, b : A \quad \alpha : e(a, b)}{\boldsymbol{inE}(a, b, \alpha) : \boldsymbol{inT}(a) =_{\boldsymbol{M}(A,e,\epsilon)} \boldsymbol{inT}(b)}$$

$$\frac{\begin{array}{l} C : \Pi \boldsymbol{M}(A, e, \epsilon) \to \mathrm{Type} \quad z : \Pi[a : A].C(\boldsymbol{inT}\ a) \quad a : M(A, e, \epsilon) \\ w : \Pi[a, b : A].[\beta : e(a, b)].J(\dots, z(a), \boldsymbol{inE}(\beta)) =_{C(\boldsymbol{inT}\ b)} z(b) \end{array}}{rec_{\boldsymbol{M}}(C, z, w, a) : C(a)}$$

Следствия из нового результата (II)

2 Индуктивные типы как фактор-типы

Натуральные числа задаются аксиоматически ($\mathbb{N}, 0, S, rec_{\mathbb{N}}$)
Фактор-типы задаются как высший индуктивный тип
($\sim: \Pi A.A.\mathrm{Type}$):

$$A/{\sim} \equiv M(A, \mathrm{FreeGpd}(\sim))$$

Индуктивные типы задаются с помощью функтора $F : K \to K$
и вложения $\boldsymbol{in} : \Pi[X : K].X \to F(X)$.
Семантика индуктивных типов — фактор-тип $\Sigma[n : \boldsymbol{N}].F^n(\mathbb{0})$
по отношению, порождаемому вложением $\boldsymbol{in}$.
(работа не завершена)

Новый результат: язык на основе исчисления с синтаксическими макросами

Синтаксис

Литералы 2, 3.14, `"ABCDEF"`

Символы `refl`

List `(a b c d ...)` — приложение, стандартные формы

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

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

Макросы

`(define-macro <name> R)`

`R` : Syntax → Context → Partial(*Term*)

`(typing T C)` : Type + TypingError

T : Term

`(check t T C)` : T + TypingError

t : Term, T : Type

`(compat t u T C)` : #1 + TypingError

t, u : Term

Замечания по реализации

Кроме рассмотренных выше термов введены следующие:

- конечные типы (`#0`, `#1`, ...)
- внешние функции / аксиомы и примитивные типы
  (`(admit "dprintf"` $\Pi String.\#1$`)`)
- в дальнейшем примитивные типы будут интернированы

Макросы могут быть использованы для следующих целей:

- Синтаксический сахар (tuple, ADT)
- Задание предметно-ориентированных языков
- Внешние процедуры разрешения

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: модели систем частичных функций

Типизированное λ-исчисление не допускает прямое определение частичных (потенциально незавершимых) функций.

Стандартным подходом является использование монады `Partial`.

Новый результат: модель системы частичных функций *в рамках λ-исчисления с зависимыми типами* на базе монады `ReactT`

Мотивация: bootstrapping базового языка (разрешимость алгоритма проверки типов невыводима в рамках языка).

Результат:

$Index : \mathrm{Type}$ — индекс (набор идентификаторов функций).

$arg, result : \Pi Index.\mathrm{Type}$ — сигнатура (типы аргумента и результата для каждого идентификатора).

Для индекса i определяется функция на следующем языке:
$return(r : result\, i)$, $call(j : Index, a : arg\, j, c : result\, j \to \texttt{GR i})$

Для интерпретации вводится понятие стека вызовов.

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

Экспериментальная задача 1

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, чтобы их было удобно расставить вручную и получить ускорение

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 методов при разбиении

Задача в рамках настоящего исследования

подготовка кода к верификации с использованием предметно-ориентированных языков

выбран следующий набор DSL:

- Fortran-подобный язык задания стадии вычислений с контролем единиц измерения и возможностью моделирования в рациональных числах (с произвольной точностью)
- язык объявления стадии вычислений, указывающий объект для стадии и набор используемых свойств внешних объектов
- язык описания параллельного исполнения
- язык задания отображения объектов в память

Задача в рамках настоящего исследования

подготовка кода к верификации с использованием предметно-ориентированных языков

выбран следующий набор DSL:

- Fortran-подобный язык задания стадии вычислений с контролем единиц измерения и возможностью моделирования в рациональных числах (с произвольной точностью)
- язык объявления стадии вычислений, указывающий объект для стадии и набор используемых свойств внешних объектов
- **язык описания параллельного исполнения**
- язык задания отображения объектов в память

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:

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

Многоязыковые приложения

приложение использует РСУБД

схема БД и запросы задаются с помощью ORM (Object-Relational Mapping)

схема БД $\longrightarrow$ DDL

операторы ORM $\longrightarrow$ SQL + DML

существующие результаты:

2006. A. Silva, J. Visser. Strong Types for Relational Databases

БД может быть представлена с помощью гетерогенных списков

Новый результат: развитие модели на зависимые типы

(работа находится на ранних этапах)

$$RDB \equiv Dictionary(name : String, table : Table)$$

$$Table \equiv \Sigma[s : Schema].Data(s)$$

$$Schema \equiv \Sigma[d : RDB].[numAttrs : \mathbb{N}].Vector(numAttrs, [name : String, type : \mathrm{Type}, constr : List(Constraint(type, d))])$$

$$Constraint \equiv content(c : \Pi[type].\#2) \mid foreign(table : \Sigma[name : string].inDictionary(d, name), attr : \Sigma[attrName : string].inDictionary(\dots))$$

Подход к верификации

спецификация ORM-провайдера интерпретируется в терминах модели БД

в рамках общего базового представления могут быть сформулированы и доказаны требуемые свойства (например, выполнение политики безопасности на уровне полей)

Заключение

Общие результаты

1. Базовый язык

   новая разновидность λ-исчисления с зависимыми типами
   реализация с поддержкой синтаксических макросов

2. Формальные модели программ

   модель динамического параллельного исполнения программ
   состояние с гранулируемым доступом

3. Подходы к верификации программ

   предметно-ориентированный язык, гарантирующий корректность параллельного расчёта для модели теплогидравлических процессов в АЭС
   подход к верификации приложений, использующих РСУБД

Дальнейшая работа

- Базовый язык
  - исследование свойств исчисления
  - подключение внешних процедур разрешения
  - компиляция
- Формальные модели программ
  - средства для более удобного задания DSL
  - средства для работы с синтаксисом
- Подходы к верификации программ
  - разработка необходимых DSL для реинжиниринга модели теплогидравлических процессов в АЭС
  - использование предметно-ориентированных процедур разрешения

Спасибо за внимание!