Принципы построения системы детерминированного параллельного программирования

Алексей И. Адамович[1], **Андрей В. Климов**[2]

[1] Институт программных систем им. А.К. Айламазяна РАН
г. Переславль-Залесский

[2] Институт прикладной математики им. М.В. Келдыша РАН
г. Москва

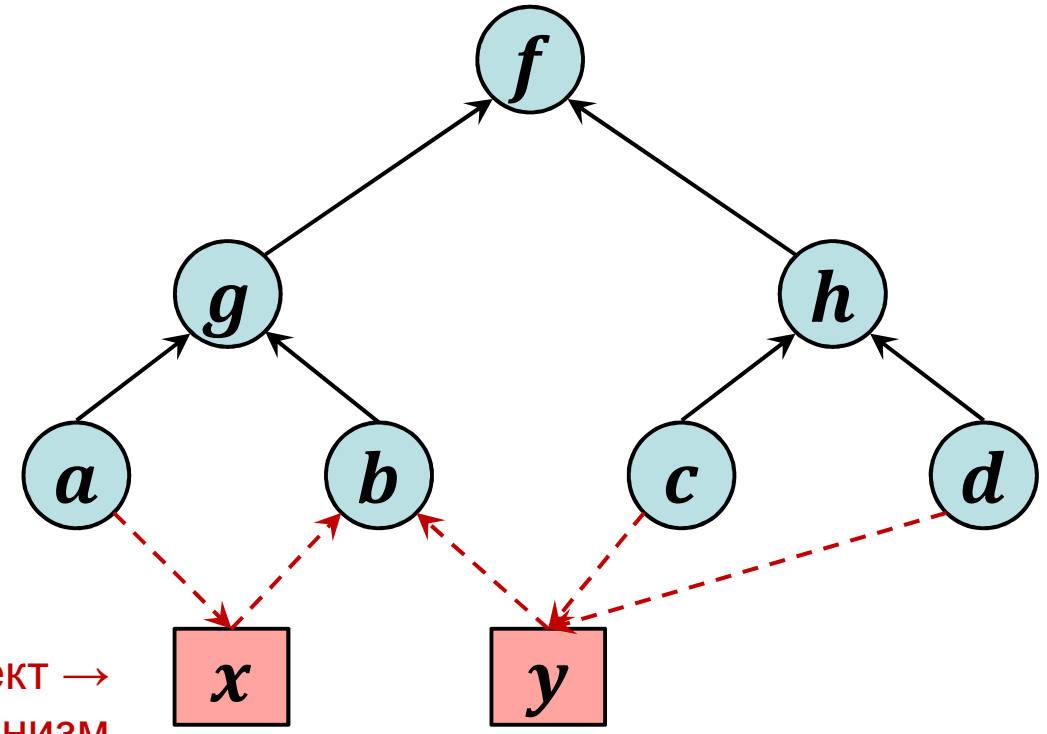
Семинар «Наукоемкое программное обеспечение» (НПО'19)
2–3 июля 2019 г., Академгородок, Новосибирск

Общая постановка задачи

- **Цель**
  - высокая **продуктивность** параллельного программирования = высокая **производительность труда разработчика**, легкость программирования, отладки и сопровождения
- **Серьезное препятствие**
  - **недетерминизм параллельных программ** (parallel & concurrent)
- **Задача**
  - создание **гарантированно детерминированных программ несмотря на недетерминизм базовых** средств
- **Разрешима ли она?** (в практическом смысле)
  - Да, есть «тривиальное» решение (но не достаточное)
    - **чисто функциональное программирование**
  - Работы под лозунгом «**детерминизм по умолчанию**» для **Java**
    - Robert L. Bocchino, Jr., Vikram S. Adve, Sarita V. Adve, and Marc Snir. **2009.** **Parallel programming must be deterministic by default** // *Proceedings of the First USENIX conference on Hot topics in parallelism* (HotPar'09). USENIX Association, Berkeley, CA, USA.

Детерминизм чисто функциональных программ

$$f(g(a,b),h(c,d))$$



Вычисляются параллельно

с детерминированным результатом

Побочный эффект → вносит недетерминизм

Идея: ограничить операции над объектами так, чтобы сохранить детерминизм

Параллелизм в функциональных языках

- Большинство языков функционального программирования – **«грязно» функциональные**, то есть допускают **побочный эффект**
  - для ввода-вывода
  - для работы со сложными структурами данных (массивы, объекты, **представления графов**)
- Существует лишь один **чисто функциональный язык**, достаточно **широко распространенный** и используемый **для практических задач** –
  - **Haskell** (https://www.haskell.org/)
  - **Parallel & Concurrent Haskell** – **библиотеки**, а не средства языка
    - **детерминированный** параллелизм (модуль Control.**Parallel**)
      - управление параллельными процессами, стратегии
    - **недетерминированная** часть (модуль Control.**Concurrent**)
      - процессы и их взаимодействие через общие переменные

Даже в Haskell'е есть «грязь»

Библиотеки созданы **на уровне средств реализации**, а не на Haskell'е, используя **unsafe features**

Параллелизм в функциональных языках

- Большинство языков функционального пр
  «**грязно**» **функциональные**, то есть допу
  - для ввода-вывода
  - для работы со сложными структурами
    (массивы, объекты, **представления гр**
- Существует лишь один **чисто функциона**
  достаточно **широко распространенный** и
  используемый **для практических задач** –
  - **Haskell** (https://www.haskell.org/)
  - **Parallel & Concurrent Haskell** – **библи**
    - **детерминированный** параллелиз
      - управление параллельными проце
    - **недетерминированная** часть (модуль Control.**Concurrent**)
      - процессы и их взаимодействие через общие переменные

Даже в Haskell'е есть «грязь»

Библиотеки созданы **на уровне средств реализации**, а не на Haskell'е, используя **unsafe features**

Выводы из опыта функциональных языков

Проблема
- Требуются **высокая квалификация** разработчиков и трудозатраты **для создания библиотек** параллельного программирования

Вопрос
- Можно ли создать **одну параллельную библиотеку** на все случаи жизни?

Ответ
- Конечно, нет!

Структура языка и среды программирования

Язык «высокого» уровня и его реализация гарантируют детерминизм при использовании «правильной» библиотеки

Библиотека «нижнего» уровня на языке реализации: гарантии дает ее автор

Требуются средства программирования (язык, среда, компилятор, run-time)
- с удобным **универсальным** входным языком (для «нижнего» уровня)
- обеспечивающие **детерминизм на подъязыке**, проверяемом системой

Построение среды детерминированного параллельного программирования

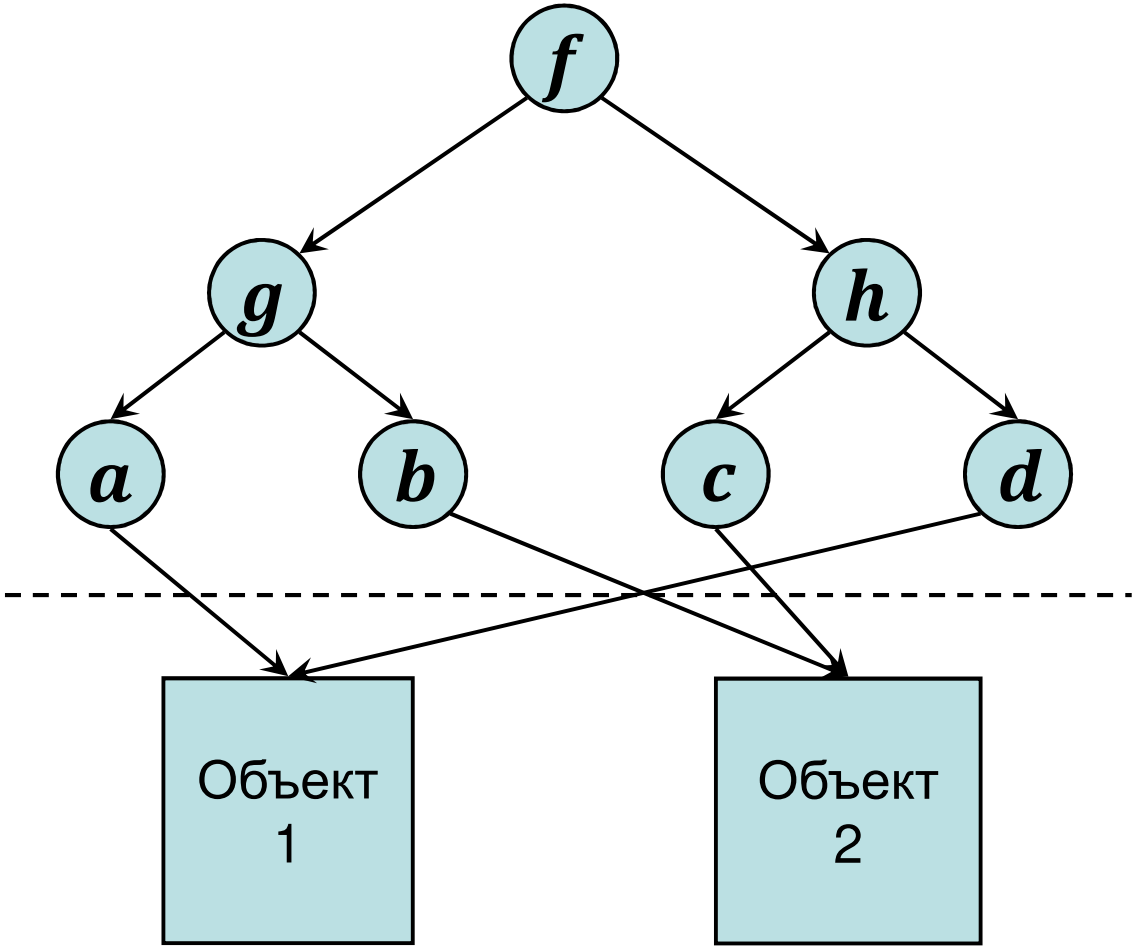
- **Берем универсальный язык программирования**
  - **объектно-ориентированный** (куда уж меньше)
  - со **средствами параллельного** программирования
- **Выделяем подмножество языка**
  - примерно **функциональное подмножество**
  - обеспечивающее **детерминизм**
  - и другие полезные свойства
- **Использование языка**
  - **весь язык** – для **квалифицированных программистов**, создающих библиотеки классов «нижнего уровня»
  - **подмножество языка** – для **прикладных разработчиков**
    - где **детерминизм гарантирован авторами библиотек** и устройством этого подъязыка

А как? По какой методике?

Гибрид функционального и объектно-ориентированного языка

- **Идея**
  - делим на 2 части язык программирования и программы
- **В «верхней» части**
  - **неявный** параллелизм
  - гарантированный **детерминизм**
- **В «нижней» части**
  - **явный** параллелизм
  - **детерминизм надо доказывать** (тестировать)

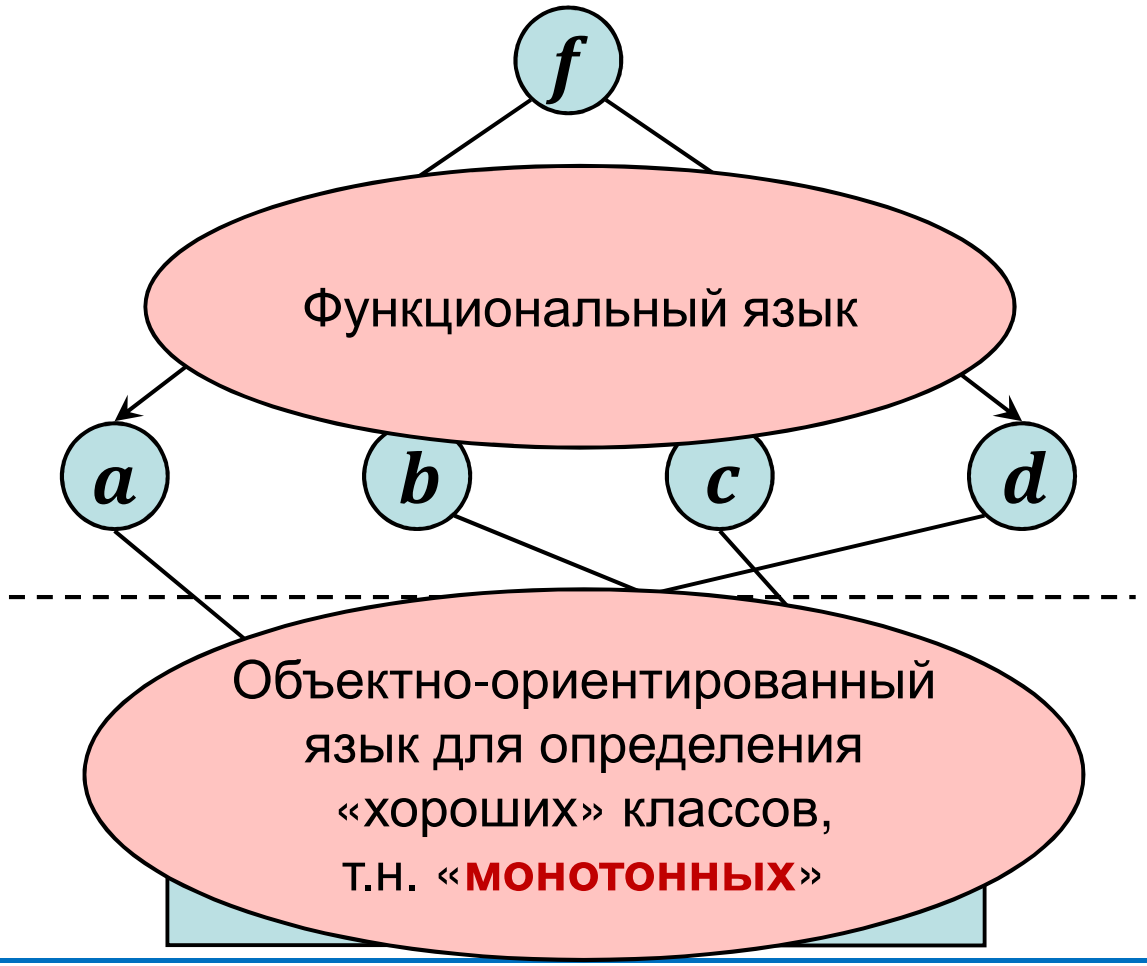
$$f(g(a,b),h(c,d))$$



Гибрид функционального и объектно-ориентированного языка

- **Идея**
  - делим на 2 части язык программирования и программы
- **В «верхней» части**
  - **неявный** параллелизм
  - гарантированный **детерминизм**
- **В «нижней» части**
  - **явный** параллелизм
  - **детерминизм надо доказывать** (тестировать)

$$f(g(a,b),h(c,d))$$



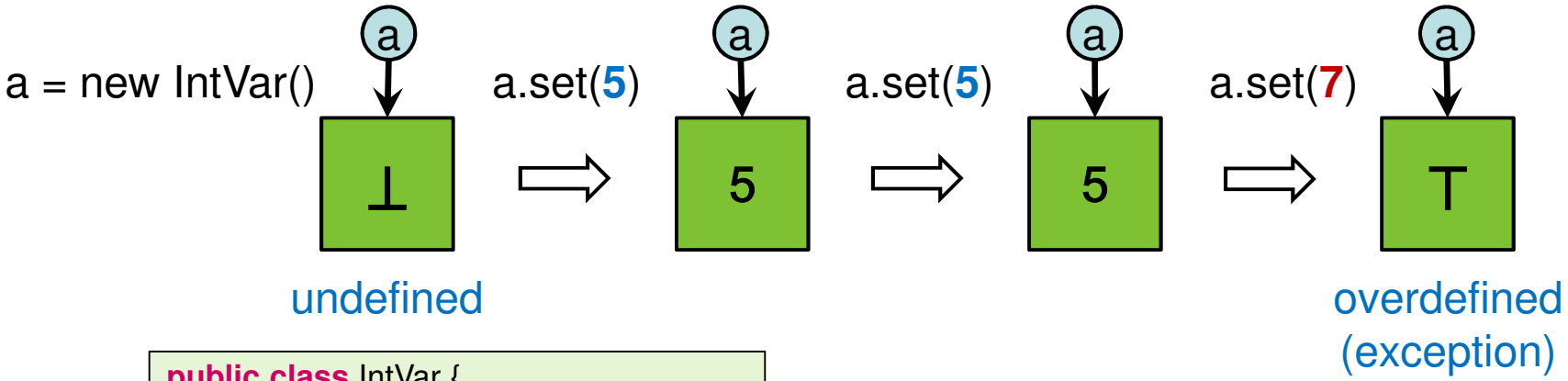
Существующие подходы к детерминизму ООЯ

- **Выделение функционального подмножества**
  - Функциональное подмножество легко проверяется компилятором
  - Разрешаются только **immutable objects**
    - Alex Potanin, Johan Östlund, Yoav Zibin, and Michael D. Ernst. **2013**. **Immutability** // ***Aliasing in Object-Oriented Programming***, Dave Clarke, James Noble, and Tobias Wrigstad (Eds.). LNCS 7850. Springer-Verlag, Berlin, Heidelberg 233-269.
- **Методы анализа программ, расширения систем типизации** для **выявления детерминизма** фрагментов программы
  - эффекты (effects), алиасы (alias analysis)
  - разрешения на доступ (permissions), области доступа (domains)

  Работы по «**детерминизму по умолчанию**» используют все эти методы
    - Robert L. Bocchino, Jr., Vikram S. Adve, Sarita V. Adve, and Marc Snir. **2009. Parallel programming must be deterministic by default** // *Proceedings of the First USENIX conference on Hot topics in parallelism* (HotPar'09). USENIX Association, Berkeley, CA, USA.
- **Методики создания библиотек классов нижнего уровня**
  - **объекты изменяется монотонно на некоторых полурешетках**
    - Lindsey Kuper. **2015**. **Lattice-based Data Structures for Deterministic Parallel and Distributed Programming**, Ph.D. Thesis, Indiana University, USA.

Эту идею развиваем дальше

Пример монотонного класса: Arvind's I-Structure



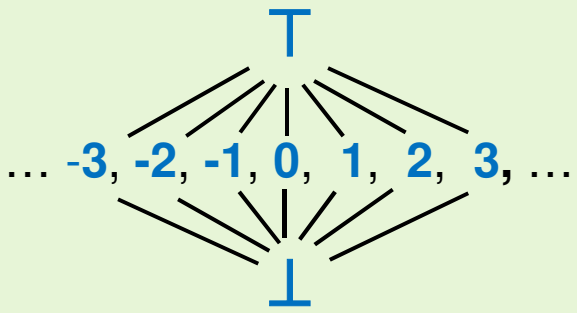

```
public class IntVar {

  boolean defined = false;
  int value;

  public synchronized int get() {
    if (!defined) wait();
    return value;
  }

  public synchronized void set(int x) {
    if (!defined) {
      value = x;
      defined = true;
      notifyAll();
    }
    else if (value != x)
      throw new RuntimeException();
  }
}
```

Здесь используется
плоская полурешетка



Разрабатываем прототип системы

На базе **JVM**, используя

- языки программирования **Java**, **Kotlin**
- собственный язык **Ajl** для синтаксического сахара и других расширений
  - А.И. Адамович. **Язык программирования Ajl: автоматическое динамическое распараллеливание для платформы JVM** // Программные системы: теория и приложения, 7:4 (31), 2016. С. 83–117.
- современные средства эффективного параллельного (parallel & concurrent) программирования в терминах **легких тредов (lightweight threads), файберов (fibers)** с использованием **сопрограмм**
  - **Quasar** – библиотека фирмы Parallel Universe с преобразователем байткода в момент загрузки
    - https://github.com/puniverse/quasar
  - **Kotlin coroutines** – языковые средства и библиотека параллельного программирования фирмы JetBrains
    - https://kotlinlang.org/docs/reference/coroutines/coroutines-guide.html

Строим библиотеки монотонных классов на следующих классах задач

- **Построение и обработка графов, циклических структур данных**
  - важно: **в чисто функциональных языках** эффективно представимы **только деревья**, и нужно **преодолеть это ограничение**
  - трудно: ставим цель **сохранить явные ссылки на объекты** в расширенном функциональном языке, а этого больше никто не делает
  - вчерашний (1.7.2019) доклад на PSSV'19 «**Building Cyclic Data in a Functional-Like Language Extended with Monotonic Objects**»
- **Переборные алгоритмы методом ветвей и границ**
  - важно: **детерминированность результата** этих алгоритмов сейчас **доказывается для их математического определения**, а **не реализации**, однако **требуются детерминированные программы**
  - хорошо: в них есть **своя монотонность**, которую надо явно выразить
    - монотонно изменяющийся **рекорд**
    - монотонно растущие **пути в дереве перебора**
  - интересно: есть **«отсев» (pruning) поддеревьев перебора**, и **требуется выразить это «монотонно» в программе**

Выводы

- В мире развиваются **два подхода к детерминированному параллельному** программированию:
  1. Выявлять **статическим анализом**, когда программа является **детерминированной** в рамках в общем **недетерминированного** языка
     - **ограничен** слабостью статического анализа
  2. Предоставить **язык и библиотеки, гарантирующие детерминированность** параллельного программирования, «спрятав» недетерминизм в реализацию «нижнего» уровня
     - **практичен**, фактически так строятся **специализированные** библиотеки параллельного программирования

- Требуются **общее решение** в виде расширяемой системы из двух частей:
  1. **Верхний уровень**: **гарантированно детерминированный подъязык** для прикладного программирования – **функциональный, расширенный объектами со ссылками**
  2. **Нижний уровень**: **объектно-ориентированный язык** для разработки базовых классов (называемых нами «**монотонными**» по семантическим причинам), устроенный так, чтобы **гарантировать детерминированность** программам верхнего уровня