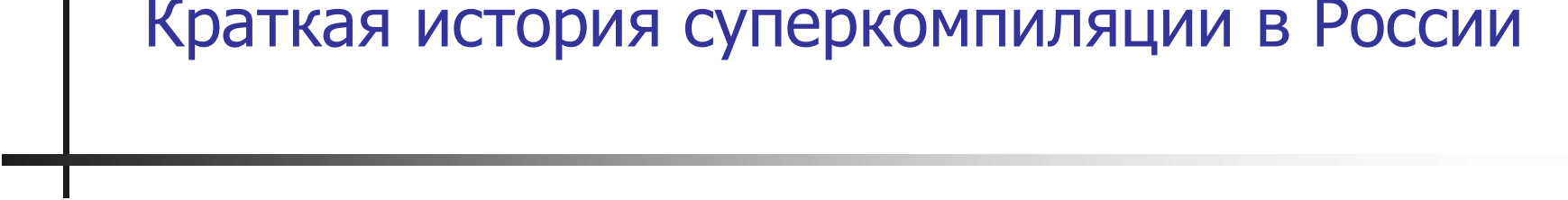


Введение в суперкомпиляцию  
И  
Краткая история суперкомпиляции в России



Андрей В. Климов, Сергей А. Романенко

Институт прикладной математики им. М.В. Келдыша РАН

Москва

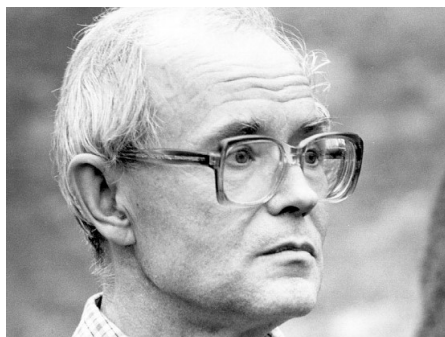
Всероссийская научная конференция памяти А.Л. Фуксмана  
«Языки программирования и компиляторы 2017»  
3 апреля 2017

# План

- Введение
  - основатели и контекст возникновения метавычислений
  - основные задачи метавычислений
  - краткая история суперкомпиляции
- Суперкомпиляция
  - основная идея суперкомпиляции
  - пример: применение суперкомпилятора для доказательства коммутативности сложения с 1:  $x+1 = 1+x$
  - метод суперкомпиляции: резюме
- Краткая справка
  - о работах последних лет
  - о проблемах и современных целях

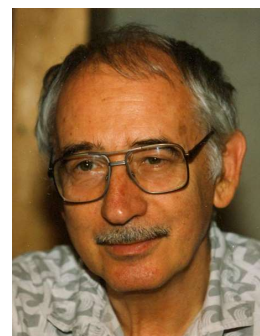
## Основатели и классики метавычислений (1970–80-е)

**Mixed computation,  
смешанные вычисления**



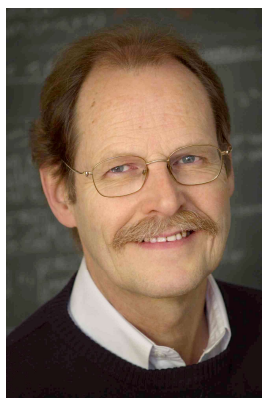
Андрей П. Ершов

**Supercompilation,  
суперкомпиляция**



Валентин Ф. Турчин

**Partial evaluation**



Neil D. Jones

**Generalized partial computation**



Yoshihiko Futamura

## Исторический контекст (1970-е)

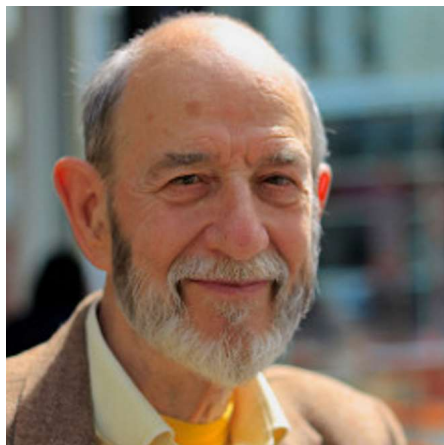
# A Transformation System for Developing Recursive Programs

R. M. BURSTALL AND JOHN DARLINGTON

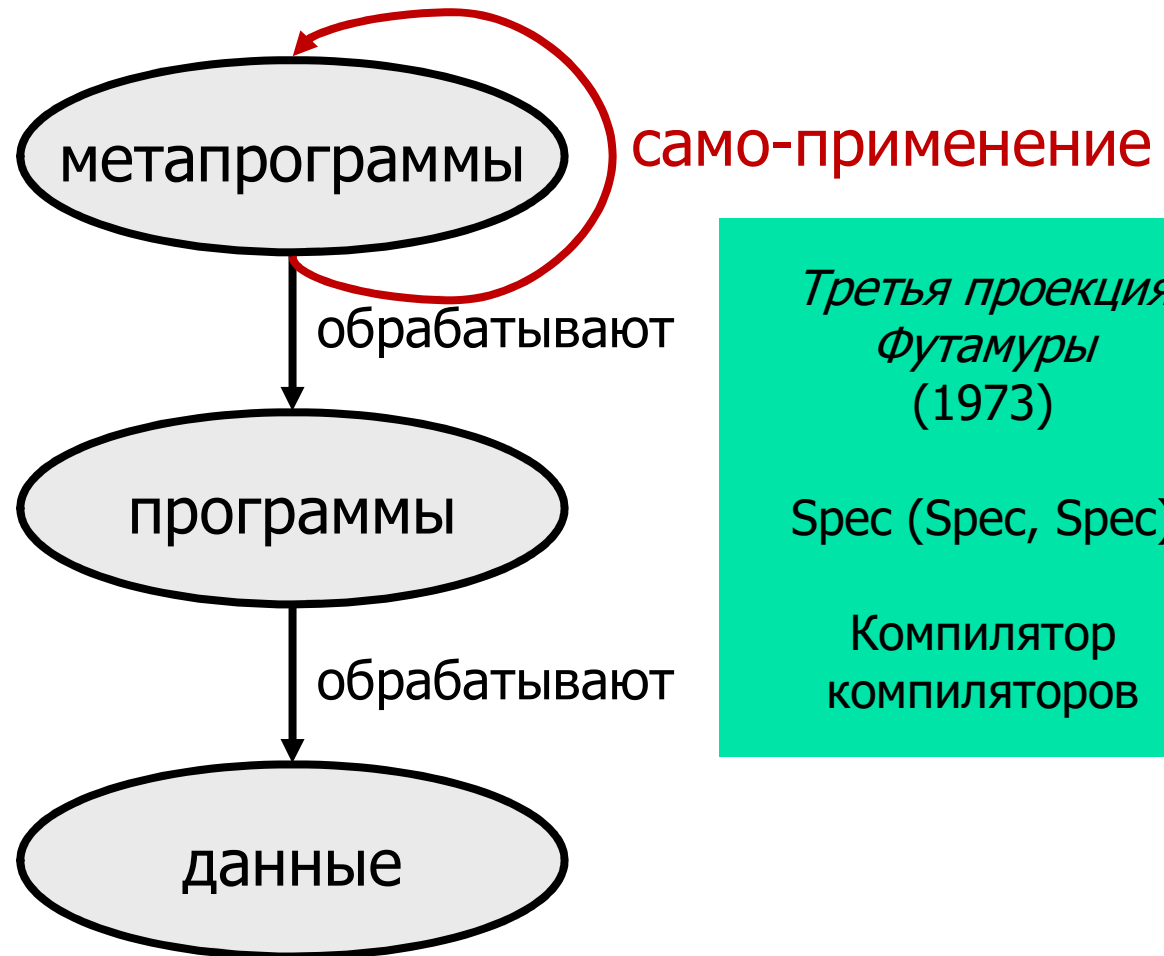
**unfold / fold  
transformations**

*University of Edinburgh, Edinburgh, Scotland*

**Journal of the Association for Computing Machinery, Vol 24, No 1, January 1977, pp 44-67**



# Метавычисления = преобразования программ



*Валентин Турчин*  
(1971)  
“Мы должны научиться так обрабатывать программы на ЭВМ, как сейчас обрабатываем числа на Фортране”

*Третья проекция*  
*Футамуры*  
(1973)

Спец (Спец, Спец)

Компилятор  
компиляторов

# Метавычисления = преобразования программ

*Валентин Турчин  
(1971)*

“Мы должны  
научиться так  
обрабатывать  
программы на ЭВМ,  
как сейчас  
обрабатываем числа  
на Фортране”

Каждые два года в конце июня – начале июля  
проводим в Переславле-Залесском

## **International Valentin Turchin Workshop on Metacomputation**

<http://meta2008.pereslavl.ru/>

<http://meta2010.pereslavl.ru/>

<http://meta2012.pereslavl.ru/>

<http://meta2014.pereslavl.ru/>

<http://meta2016.pereslavl.ru/>

*Приглашаем!*

## Основные задачи метавычислений

- Специализация программ
  - $f(x, y) \Rightarrow f_A(y) = f(A, y)$
- Композиция программ
  - $f(x), g(x) \Rightarrow f_g(x) = f(g(x))$
- Инверсия программ
  - $f(x) \Rightarrow f^{-1}(y) = x$ , когда  $y = f(x)$
- Верификация программ
  - $f(x), P(x, y) \Rightarrow P(x, f(x)) = \text{true}$

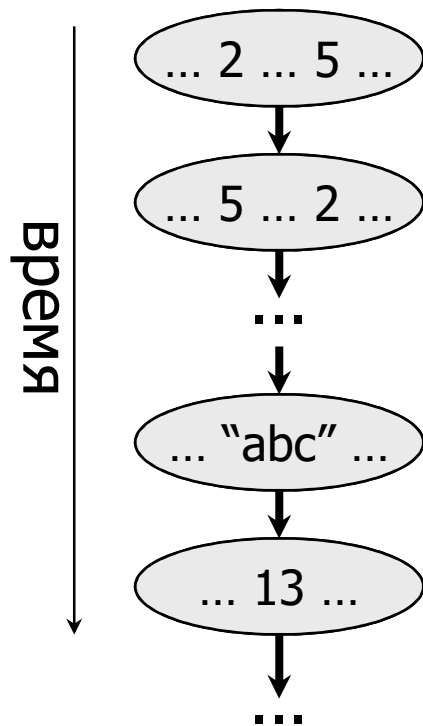
# Краткая история суперкомпиляторов в России

- 1974 В.Ф. Турчин рассказал о суперкомпиляции группе студентов на серии семинаров в Москве
- 1980-е В.Ф. Турчин разрабатывает **первые суперкомпиляторы** для функционального языка **Рефал** (CUNY, Нью-Йорк)
- 1996 Серия статей В.Ф. Турчина по **суперкомпиляции Рефала**
- 1990-е Работы по теории суперкомпиляции и **простым суперкомпиляторам** в Дании (N. Jones, R. Glück, M. Sørensen, et al, DIKU – часть в соавторстве с нами)
- 1995 Книга С.М. Абрамова «**Метавычисления и их приложения**»
- 1993 – А.П. Немытых (ИПС им. А.К. Айламазяна РАН, Переславль-Залесский) завершил работу В.Ф. Турчина над суперкомпиляторами **для Рефала**
- 2000-е
- 2007 Книга А.П. Немытых «**Суперкомпилятор SCP4: общая структура**»
- 1998 – Суперкомпилятор **языка Java** (Анд.В. Климов, Арк.В. Климов, А.Б. Шворин)
- 2000-е **Пока только 2 для практических языков**
- 2008 – **Многоуровневый** суперкомпилятор языка с **функциями высших порядков** (И.Г. Ключников, С.А. Романенко, ИПМ им. М.В. Келдыша РАН, Москва)
- 2010
- 2006 – Суперкомпиляция **как отношение** и **специализированные многорезультатные** суперкомпиляторы (Анд.В. Климов, И.Г. Ключников, С.А. Романенко)
- 2012
- 2012 – **Многорезультатная** суперкомпиляции с **насыщением равенствами** (С.А. Гречаник, ИПМ им. М.В. Келдыша РАН, Москва)
- 2017



# Основная идея суперкомпиляции

## Обычное вычисление



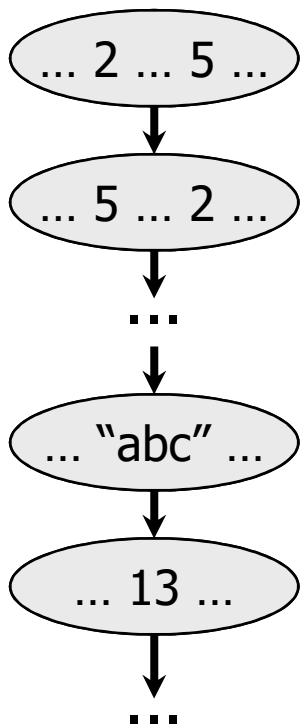
# Основная идея суперкомпиляции

Дерево процессов является программой, хотя и бесконечной

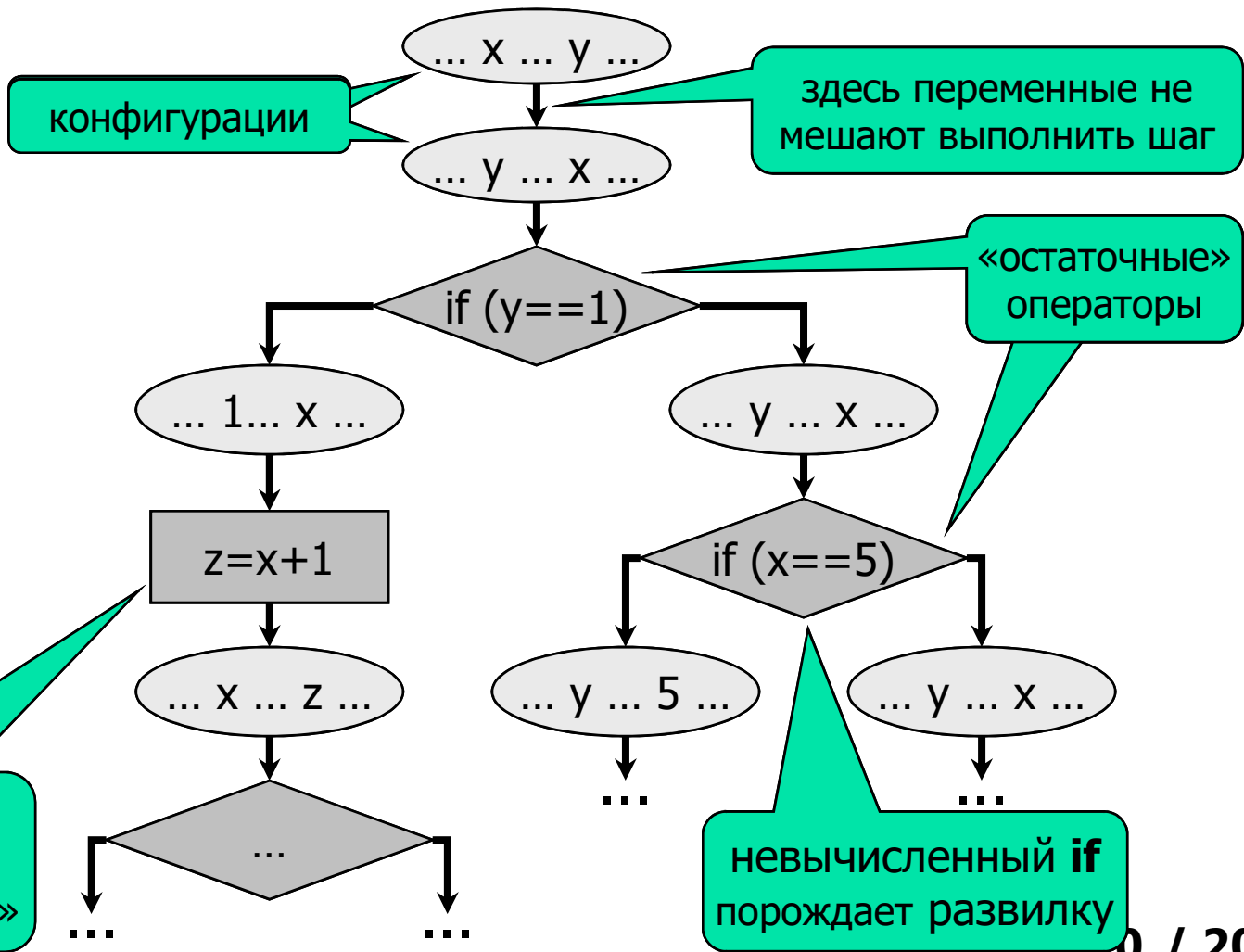
Обычное вычисление

Прогонка

время



оператор, который нельзя вычислить, «резидуализируется»



# Введение в суперкомпиляцию на примере: Доказательство коммутативности сложения с 1: $x+1 = 1+x$

**0 = Z**  
**1 = S Z**  
**2 = S (S Z)**  
**3 = S (S (S Z))**  
...

- Исходная программа на языке Haskell
  - **data N = Z | S N**
  - **add x Z = x**
  - **add x (S y) = S (add x y)**
  - **eq Z Z = True**
  - **eq Z (S y) = False**
  - **eq (S x) Z = False**
  - **eq (S x) (S y) = eq x y**
  - **p x = eq (add x (S Z)) (add (S Z) x)**
- Надеемся прооптимизировать функцию **p** до вида
  - **p x = True**

# Доказательство коммутативности сложения с 1: $x+1 = 1+x$

## Интерпретация $p(2)$

- Трасса вычислений  $2+1 = 1+2$

- $p (S (S Z))$
- $eq (add (S (S Z)) (S Z)) (add (S Z) (S (S Z)))$
- $eq (S (add (S (S Z)) Z)) (add (S Z) (S (S Z)))$
- $eq (S (add (S (S Z)) Z)) (S (add (S Z) (S Z)))$
- $eq (add (S (S Z)) Z) (add (S Z) (S Z))$
- $eq (S (S Z)) (add (S Z) (S Z))$
- $eq (S (S Z)) (S (add (S Z) Z))$
- $eq (S Z) (add (S Z) Z)$
- $eq (S Z) (S Z)$
- $eq Z Z$
- **True**

Ленивый  
порядок  
вычислений

- *Состояние* – терм из конструкторов и функций

- $S ::= (K^{(n)} S_1 \dots S_n)$   
|  $(F^{(n)} S_1 \dots S_n)$
- $K^{(n)}$  – конструкторы арности  $n$   
 $K^{(0)} ::= Z, K^{(1)} ::= S$
- $F^{(n)}$  – функции арности  $n$   
 $F^{(2)} ::= add \mid eq, F^{(1)} ::= p$

- Исходная программа на языке Haskell

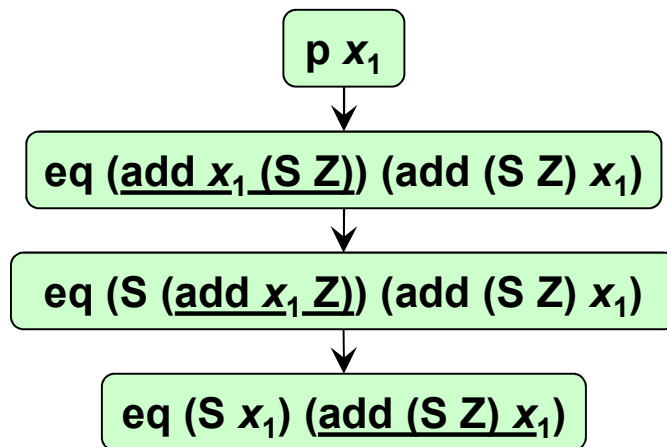
- `data N = Z | S N`
- `add x Z = x`
- `add x (S y) = S (add x y)`
- `eq Z Z = True`
- `eq Z (S y) = False`
- `eq (S x) Z = False`
- `eq (S x) (S y) = eq x y`
- `p x = eq (add x (S Z)) (add (S Z) x)`

- Надеемся прооптимизировать функцию  $p$  до вида

- `p x = True`

# Доказательство коммутативности сложения с 1: $x+1 = 1+x$

## Суперкомпиляция $p(x_1)$



- Конфигурация – обобщенное состояние: терм из конструкторов, функций и переменных

- $C ::= (K^{(n)} S_1 \dots S_n)$
  - |  $(F^{(n)} S_1 \dots S_n)$
  - |  $V$

- $V ::= x_1 \mid x_2 \mid \dots \mid y_1 \mid y_2 \mid \dots$  – конфигурационные переменные

- Конфигурация  $C$  изображает множество состояний  $\llbracket C \rrbracket$ , получающееся подстановкой всевозможных значений вместо конфигурационных переменных

- $\llbracket C \rrbracket = \{\delta C \mid \delta \in \Delta\}$ , где  $\Delta$  – множество всех подстановок

- Исходная программа на языке Haskell

- $\text{data } N = Z \mid S \ N$

- $\text{add } x \ Z = x$

- $\text{add } x \ (S \ y) = S \ (\text{add } x \ y)$

- $\text{eq } Z \ Z = \text{True}$

- $\text{eq } Z \ (S \ y) = \text{False}$

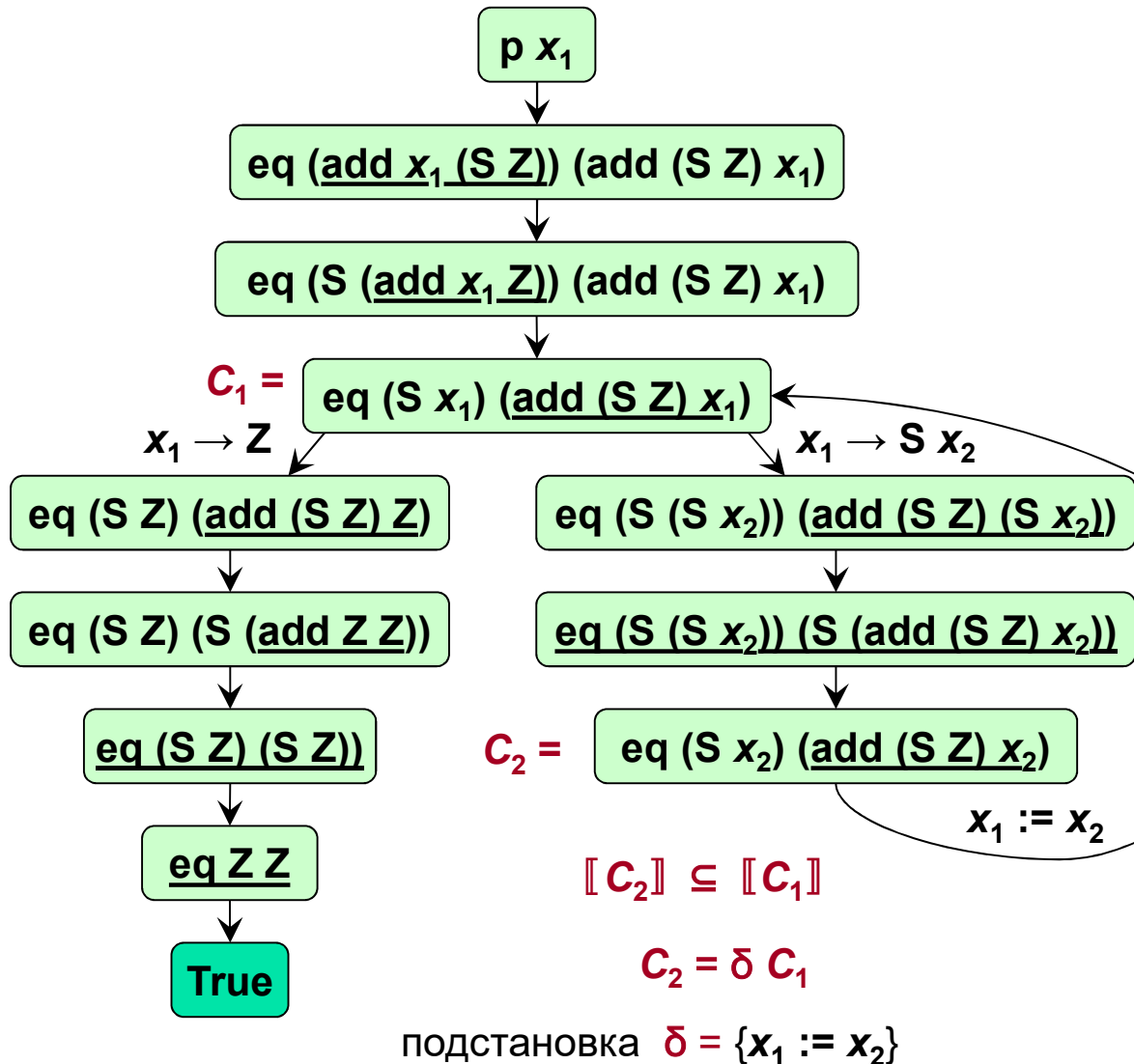
- $\text{eq } (S \ x) \ Z = \text{False}$

- $\text{eq } (S \ x) \ (S \ y) = \text{eq } x \ y$

- $p \ x = \text{eq } (\text{add } x \ (S \ Z))$   
                   $(\text{add } (S \ Z) \ x)$

# Доказательство коммутативности сложения с 1: $x+1 = 1+x$

## Суперкомпиляция $p(x_1)$



■ Исходная программа на языке Haskell

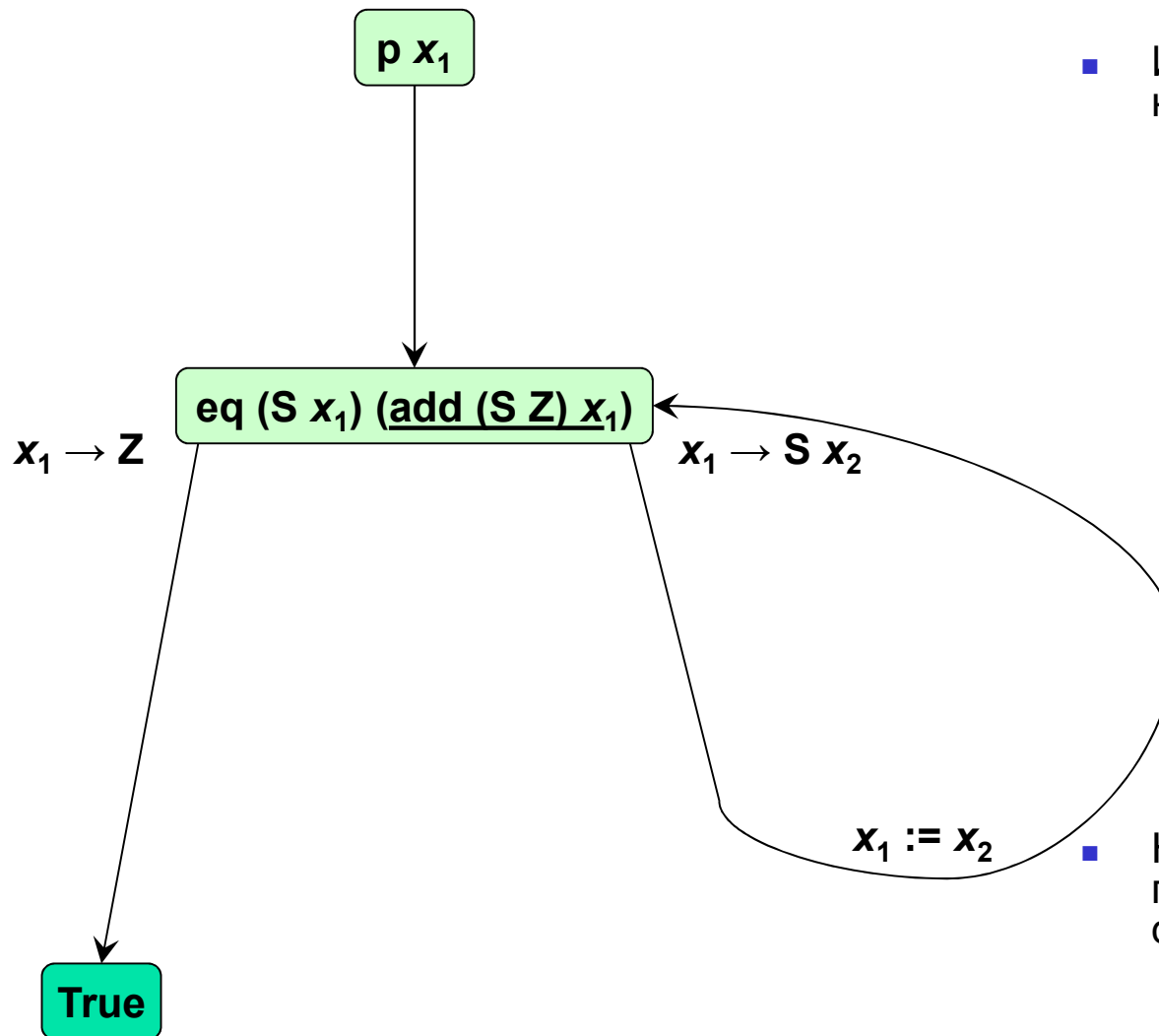
- `data N = Z | S N`
- `add x Z = x`
- `add x (S y) = S (add x y)`
- `eq Z Z = True`
- `eq Z (S y) = False`
- `eq (S x) Z = False`
- `eq (S x) (S y) = eq x y`
- $p\ x = eq\ (add\ x\ (S\ Z))\ (add\ (S\ Z)\ x)$

■ Надеемся прооптимизировать функцию  $p$  до вида

- $p\ x = True$

# Доказательство коммутативности сложения с 1: $x+1 = 1+x$

## Построение остаточной программы (1)



- Исходная программа на языке Haskell

- `data N = Z | S N`
- `add x Z = x`
- `add x (S y) = S (add x y)`

- `eq Z Z = True`
- `eq Z (S y) = False`
- `eq (S x) Z = False`
- `eq (S x) (S y) = eq x y`

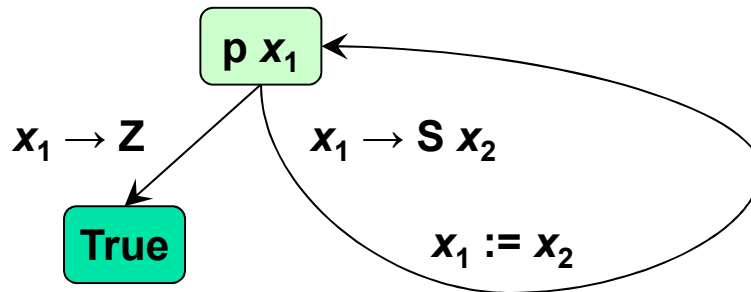
- `p x = eq (add x (S Z)) (add (S Z) x)`

- Надеемся прооптимизировать функцию `p` до вида

- `p x = True`

# Доказательство коммутативности сложения с 1: $x+1 = 1+x$

## Построение остаточной программы (2)



- Остаточная программа

- $p\ Z = \text{True}$
- $p\ (S\ x2) = p\ x2$



- $p\ x = \text{True}$

- Исходная программа на языке Haskell

- `data N = Z | S N`
- `add x Z = x`
- `add x (S y) = S (add x y)`
- `eq Z Z = True`
- `eq Z (S y) = False`
- `eq (S x) Z = False`
- `eq (S x) (S y) = eq x y`
- $p\ x = \text{eq}\ (\text{add}\ x\ (S\ Z))\ (\text{add}\ (S\ Z)\ x)$

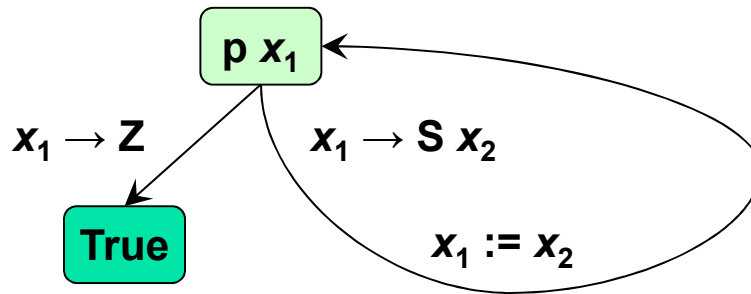
- Надеемся прооптимизировать функцию `p` до вида

- $p\ x = \text{True}$



# Доказательство коммутативности сложения с 1: $x+1 = 1+x$

## Построение остаточной программы (2)



- Остаточная программа

- $p\ Z = \text{True}$
- $p\ (S\ x_2) = p\ x_2$



- $p\ x = \text{True}$

- Исходная программа на языке Haskell

- `data N = Z | S N`
- `add x Z = x`
- `add x (S y) = S (add x y)`
- `eq Z Z = True`
- `eq Z (S y) = False`
- `eq (S x) Z = False`
- `eq (S x) (S y) = eq x y`
- $p\ x = \text{eq}\ (\text{add}\ x\ (S\ Z))\ (\text{add}\ (S\ Z)\ x)$

Это дополнительное преобразование, а не собственно суперкомпиляция.

Неэквивалентное преобразование:  
Расширение области определения!

# Метод суперкомпиляции: резюме

Детерминированный алгоритм

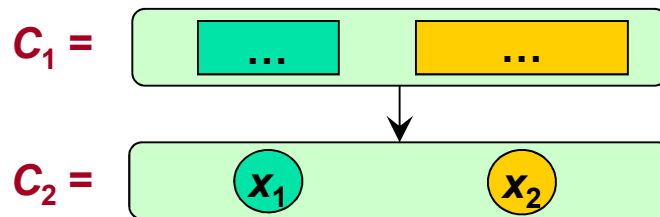
- Понятие конфигурации
  - состояние с переменными
- Прогонка
  - вычисления с переменными
  - построение потенциально бесконечного дерева путей вычислений
- Конфигурационный анализ
  - Зацикливание
  - Обобщение конфигураций
    - переход от конфигурации  $C$  к более общей  $C'$ :  
 $C \rightarrow C'$ , где  $C \subset C'$ ,  $C = \delta C'$
  - Рассечение конфигураций
    - разбиение конфигурации на части

Здесь есть недетерминизм, стратегии

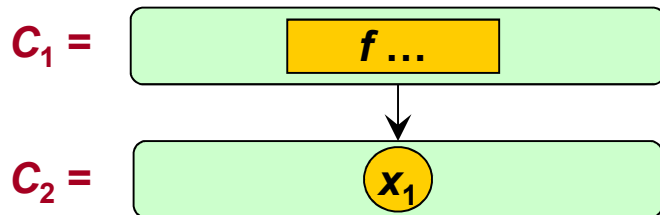
# Обобщение и рассечение конфигураций

$$[[C_1]] \subseteq [[C_2]]$$

$$C_1 = \delta C_2$$



$$\delta = \{ x_1 := \text{cyan box with } \dots ; x_2 := \text{yellow box with } \dots \}$$



$$\delta = \{ x_1 := \text{yellow box with } f \dots \}$$



# Краткая история суперкомпиляторов в России

- 1974 В.Ф. Турчин рассказал о суперкомпиляции группе студентов на серии семинаров в Москве
- 1980-е В.Ф. Турчин разрабатывает **первые суперкомпиляторы** для функционального языка **Рефал** (CUNY, Нью-Йорк)
- 1996 Серия статей В.Ф. Турчина по **суперкомпиляции Рефала**
- 1990-е Работы по теории суперкомпиляции и **простым суперкомпиляторам** в Копенгагенском университете (N. Jones, R. Glück, M. Sørensen, DIKU)
- 1995 Книга С.М. Абрамова «**Метавычисления и их приложения**»
- 1993 – 2000-е А.П. Немытых (ИПС им. А.К. Айламазяна РАН, Переславль-Залесский) завершил работу В.Ф. Турчина над суперкомпиляторами **для Рефала**
- 2007 Книга А.П. Немытых «**Суперкомпилятор SCP4: общая структура**»
- 1998 – 2000-е Суперкомпилятор **языка Java**  
(Анд.В. Климов, Арк.В. Климов, А.Б. Шворин) **Пока только 2 с установкой на практические языки**
- 2008 – 2010 **Многоуровневый** суперкомпилятор языка с **функциями высших порядков**  
(И.Г. Ключников, С.А. Романенко, ИПМ им. М.В. Келдыша РАН, Москва)
- 2006 – 2012 Суперкомпиляция **как отношение** и **специализированные многорезультатные** суперкомпиляторы (Анд.В. Климов, И.Г. Ключников, С.А. Романенко)
- 2012 – 2017 **Многорезультатная** суперкомпиляции с **насыщением равенствами**  
(С.А. Гречаник, ИПМ им. М.В. Келдыша РАН, Москва)

# Заключение: Состояние дел и цели работ

Спасибо! Вопросы?

- На фоне интересных теоретических и экспериментальных работ, стремимся к **практическому применению** суперкомпиляторов, но **пока не достигли**
- Сейчас суперкомпиляторами SCP4 для Рефала и JScp для языка Java могут пользоваться **только авторы** и то с трудом
  - Требуется **управление от пользователя**
    - преобразования программ вышли за пределы «**чернощичного режима**»
  - Надо **погружать в студии (IDE)**, чтобы **программисту было интересно**
    - пользователь мог **управлять стратегиями** суперкомпиляции
    - получать информацию о процессе и результате суперкомпиляции не труднее, чем из **диалоговых отладчиков**
    - сделать применение суперкомпиляторов таким же естественным как **рефакторинг программ**
- Приложения, поиск «killer application»
  - Раньше главной целью мы считали **оптимизацию**
  - «Конкуренты» – закон Мура и параллельные процессоры
    - **эффективность** волнует ограниченный круг пользователей
  - Кандидаты в «killer application»
    - **анализ и верификация** программ и проблемно-ориентированных систем и моделей
    - распараллеливание и преобразования программ между различными **моделями параллельных вычислений**
    - масштабируемое **моделирование (симуляция)** больших систем
    - генетическое и эволюционное программирование, машинное обучение, ИИ