Реверсивные вычисления

Непейвода Н.Н., Непейвода А.Н.

Физические предпосылки

$$R = \frac{N_{ops}}{t} = \frac{N_{ops}}{E_{diss}} \times \frac{E_{diss}}{t} = F_E \times P_{diss}$$

N — число полезных операций E — выделяемая энергия P — средняя потребляемая мощность

- Единственный путь повысить производительность — увеличивать количество операций на единицу энергии (F_E).

Обобщенный Принцип Ландауэра — фон Неймана

С учётом соотношения $T = \frac{dE}{dS}$ с потерей

- единицы информации теряется энергия

$$E_{diss} \geq T \times k_B \times \ln\ p$$

- p --- число состояний элементарной ячейки. Это соотношение показывает предел производительности нереверсивных вычислений.

Первые алгоритмические предпосылки

Ч.Беннет (1973), Лесерф (1963): Любая вычислимая функция может быть вычислена реверсивно.

Независимо разработали аппарат Реверсивных Машин Тьюринга (РМТ) для работы с реверсивными алгоритмами.

Реверсивные машины Тьюринга

Главное отличие: невозможно одновременно считать символ с ленты и переместить головку.

$\langle p, a, b, q \rangle$ - на символе a в состоянии q пишем символ b и переходим в состояние p.

$\langle p, *, s, q \rangle$ - в состоянии q двигаемся по ленте согласно s (-1 — влево, +1 — вправо) и переходим в состояние p.

Реверсивные машины Тьюринга

Если РМТ перешла из состояний q_1 и q_2 в одно и то же состояние p, то q_1 и q_2 оба принадлежат записывающему типу и записали на ленту разные символы.

Отсюда следует — существование обратного просчёта пути вычислений.

Игра в камни

Пусть G — список гнёзд $\{0, 1, 2, \ldots, T_G\}$, и имеется n камней, которые можно в них класть.
0 — зарезервированное гнездо, в котором всегда лежит камень.
На каждом шаге игры игрок может положить камень в гнездо или убрать его оттуда согласно правилу: «если гнездо i занято камнем, то можно либо положить камень в гнездо i+1 (если оно пустое), либо убрать камень из гнезда i+1».
Какое минимальное число камней потребуется, чтобы достичь гнезда с номером i?

Игра в камни — алгоритм Беннета

Локальный шаг

Пусть есть 3 гнезда i_1, i_2, i_3, причём i_1 полное, а два других пустые.

1) Положим камень в i_2
2) Положим камень в i_3
3) Уберём камень из i_2

Алгоритм Беннета с n камнями позволяет достичь гнезда с номером максимум 2^n-1.

Игра в камни — реверсивная интерпретация

Камень --- хранящееся состояние вычислений
Количество камней n--- увеличение в n раз требуемой памяти по сравнению с нереверсивным алгоритмом

Приведение нереверсивных вычислений к реверсивным

Классические вычисления, проходящие за время T и требующие S памяти, могут быть преобразованы в реверсивные, требующие

$3^k \times 2^{O(\frac{T}{2^k})}$ времени и $S \times (1 + O(k))$ памяти

(k — параметр, свободно выбираемый от 1 до $\log_2 T$)

Условия реверсивных вычислений (Тоффоли, Фредкин)

1) Скорость распространения информации ограничена.
2) Количество информации, хранимое в конкретном состоянии конечной системы, ограничено.
3) Хранение информации и её передача эквивалентны с релятивистской точки зрения.
4) Осуществима обратимость.
5) Существует биективная композиция (без копирования сигнала).
6) Существует аддитивный инвариант (Ландау и Лифшиц, 1961).
7) Локальная топология пространства-времени Евклидова.

Реверсивные гейты. Гейт Фредкина

Входы A, B, C. Если A=1, B и C передаются неизменно. Если A=0, B и C меняются местами; A передаётся в неизменном виде.

Реализация «и». На входе A, B, 0. Тогда на втором выходе (соответствующем B) передастся значение A&B.

Реализация «не». На входе A, 0, 1. На втором выходе (соответствующем 0) передастся ~A.

Реализация «или». На входе A, 1, B. На втором выходе (соответствующем 1) передастся AvB.

Реверсивные гейты. Основной базис

1) NOT: вход A, выход ~A.

2) Feynman: входы A, B, выходы A, A+B.

3) Toffoli: входы A, B, C, выходы A, B, A&B+C.

4) Обобщённый KNOT (обобщённый Toffoli): входы A_1, A_2, ... , A_n, выходы A_1, A_2, ... , A_n+ A_1& A_2 &... &A_{n-1}.

Общее свойство — самообратность.

Идея реверсивных бинарных вычислителей

Биты памяти делятся на две области: те, которые содержат нужную информацию, и те, которые нужны для поддержания обратимости.
Каждый раз, когда в рабочей зоне меняется бит, во вспомогательной происходит противоположный переход.

Реверсивные базисы

Результат Кернторпфа:

Всякий реверсивный гейт n*n, не сохраняющий прямую сумму (не линейный), является базисом пространства реверсивных функций n*n.

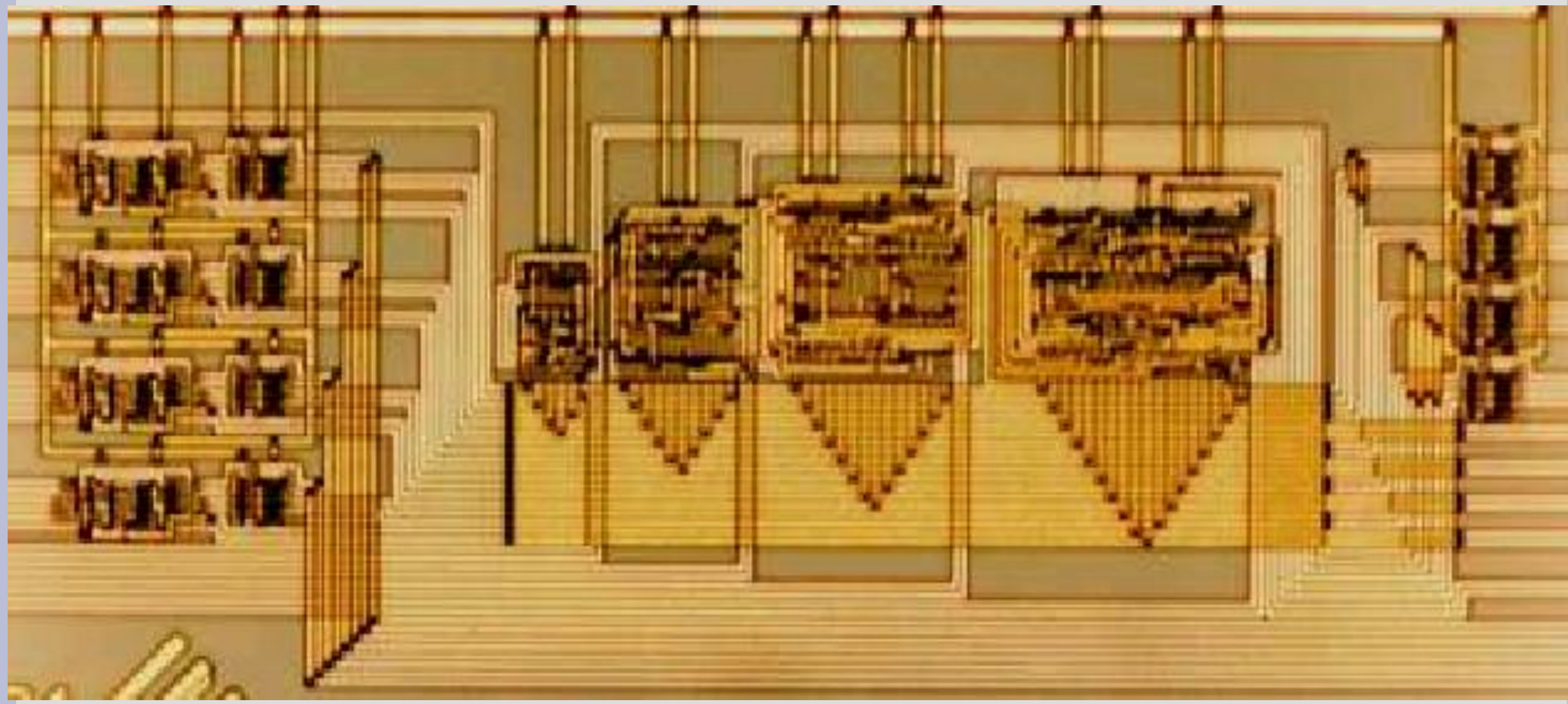
Синтез реверсивных схем

1) Динамическое программирование (свойство оптимальности подсхемы оптимальной схемы).

2) Традиционные средства, адаптированные под реверсивные вычисления (например, синтез схемы с помощью построения КНФ).

3) Алгебраический подход (гейты размером n*n образуют группу, изоморфную симметрической группе S_{2^n})

Микрофотография реверсивного сумматора



Броуновская машина (Беннет)

Идея — случайные блуждания, корректируемые с помощью незначительных количеств управляющей энергии.

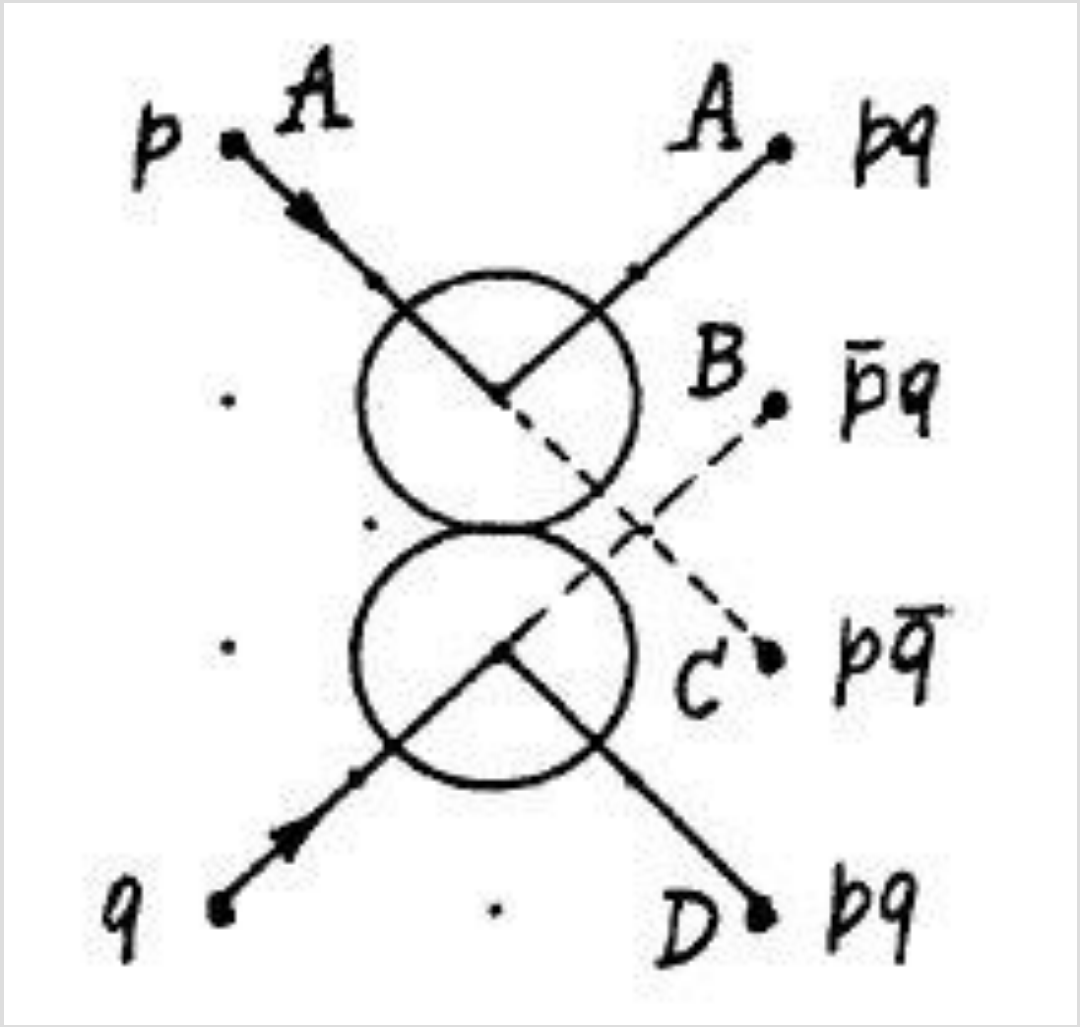
Естественная модель — полимеризация ДНК. В нормальных условиях (например, при делении клетки) происходит со скоростью порядка 1000 нуклеотидов в секунду, с расходом энергии примерно 40 $k_B T$ за шаг.

Адиабатное переключение (Меркле)

Идея — все переключения на гейтах делаются при одинаковом напряжении.

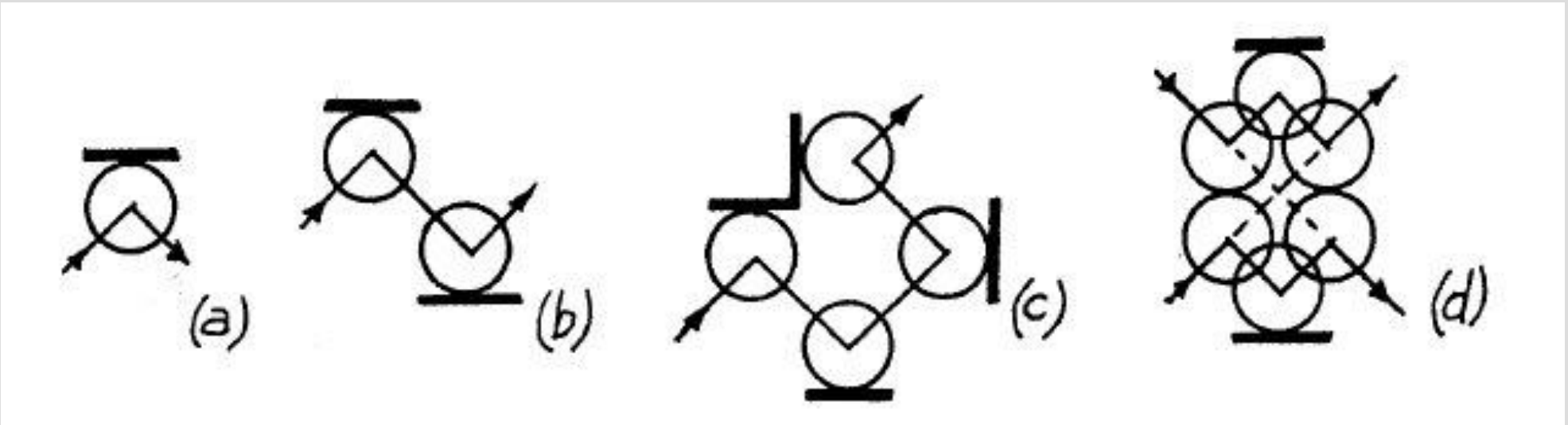
1) Энергия на переключение не тратится.
2) Энергия на приведение напряжения между переключениями тратится тем меньше, чем более плавно оно меняется. В идеале — почти нулевые выбросы тепловой энергии. Минус — низкая скорость работы.

Биллиардная модель Тоффоли



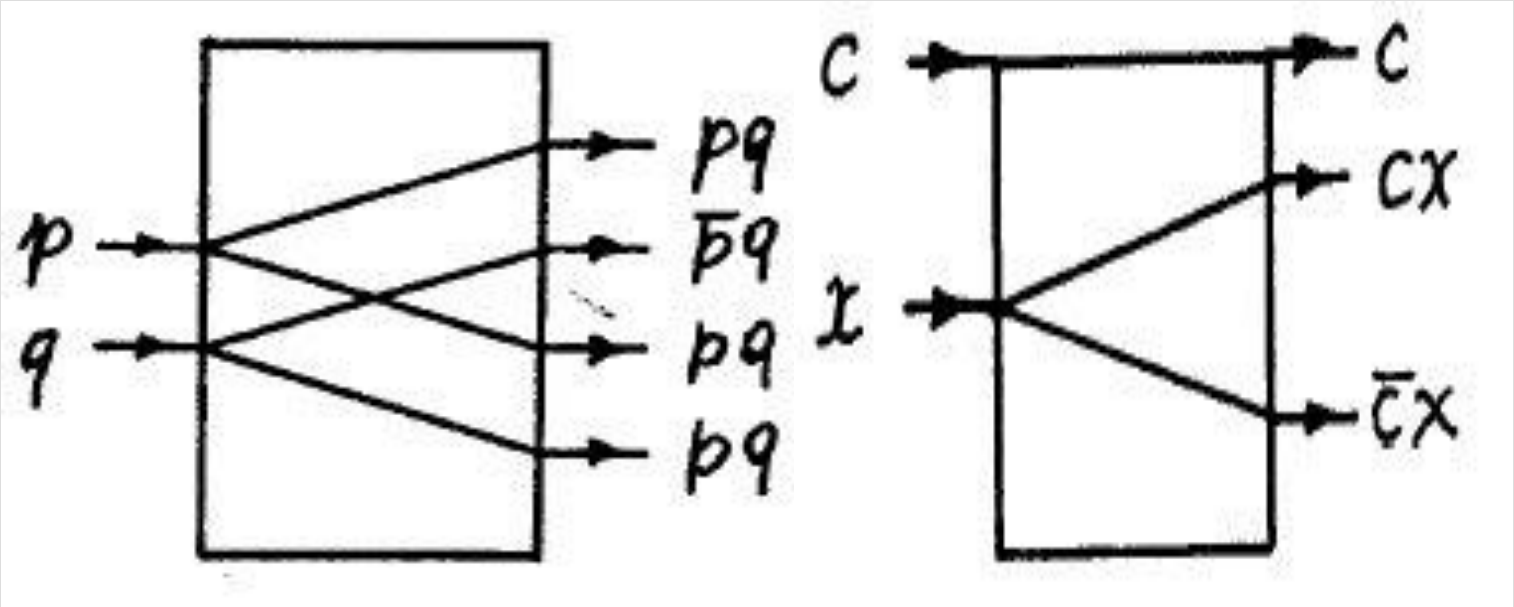
Пространство — двумерная решётка с расстоянием между узлами 1.
Объекты — одинаково тяжёлые круги радиусом $\frac{1}{\sqrt{2}}$, которые движутся со скоростью 1.
Столкновение — абсолютно упругое.

Биллиардная модель Тоффоли



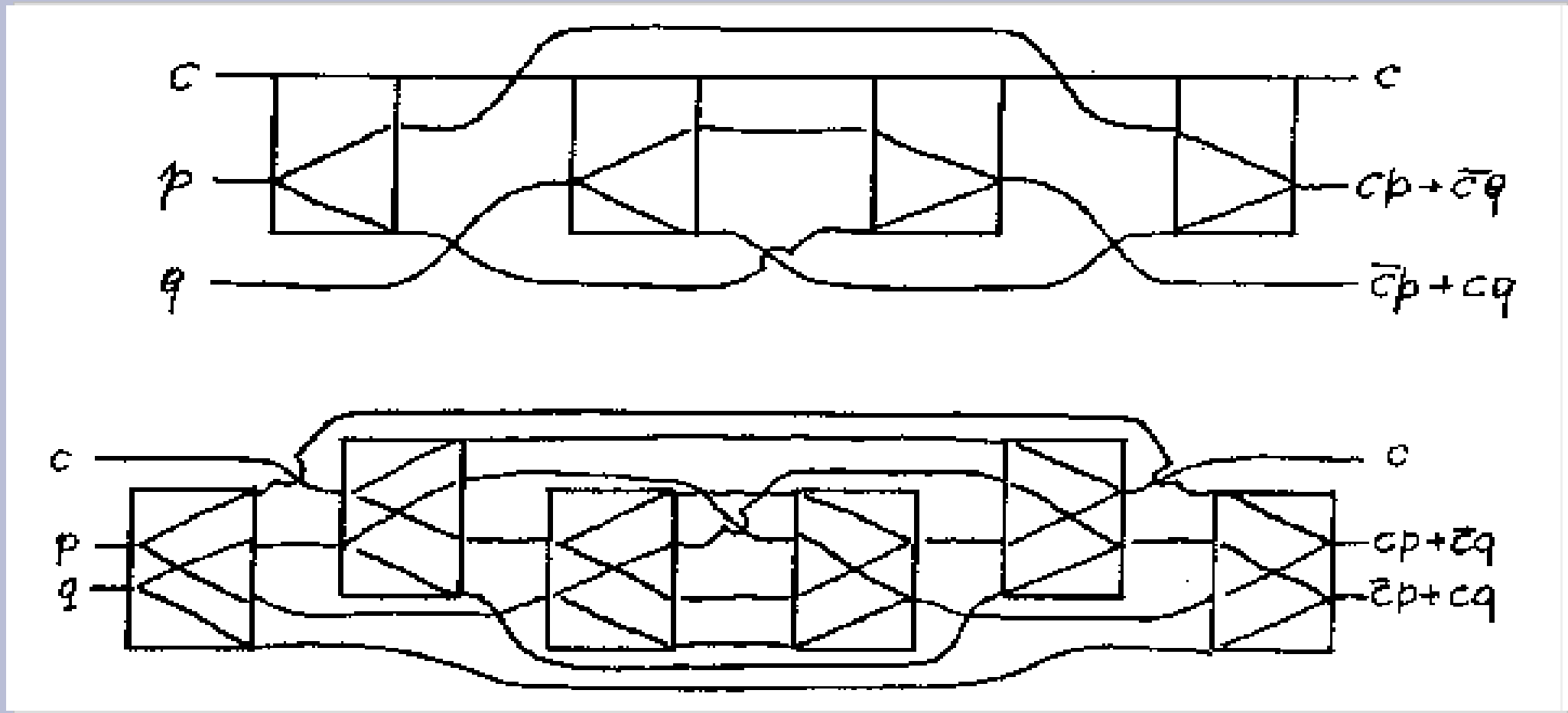
Упругая стенка и модели управления с её помощью движением шаров.

Биллиардная модель Тоффоли



Гейты взаимодействия шаров и взаимодействия шара со стенкой.

Биллиардная модель Тоффоли



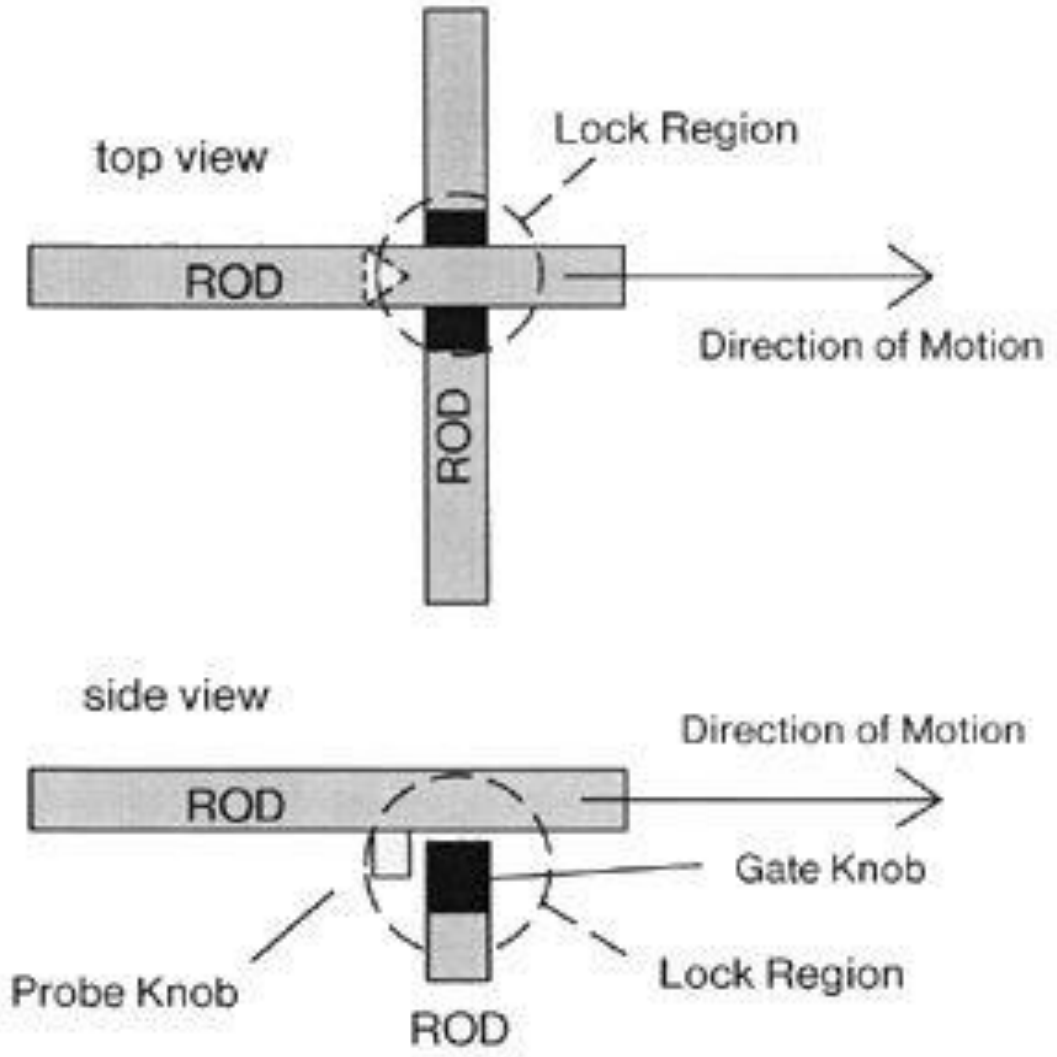
Так из гейтов столкновения и отражения строится гейт Фредкина

Модель биллиардных шаров (физическая реализация)

Тоффоли, Фредкин (MIT) — электронная модель, в которой пучки зарядов перемещаются по индуктивным контурам между конденсаторами.

Ричард Фейнман (Калифорнийский технический институт) — реализация идеи Тоффоли на практике.

Модель стержней и пазов (Дрекслер, Меркле)



Элемент управления — стержень с выступом, имеющий одну степень свободы.

Взаимодействие — блокировка при попадании выступов в одну область.

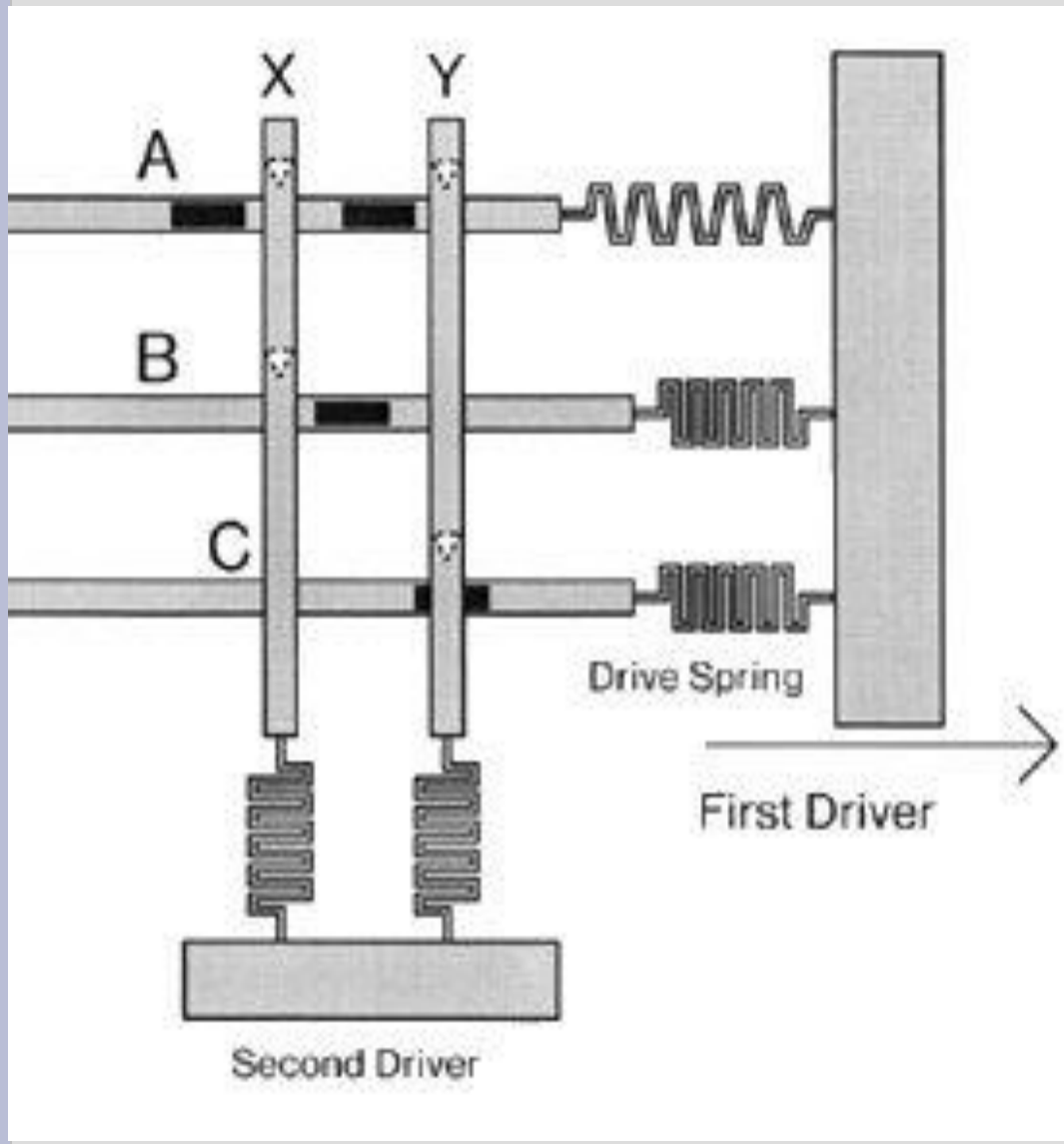
Модель стержней и пазов (Дрекслер, Меркле)

1) Стержни размещены внутри матрицы с фиксированными каналами и снабжены эластичными фиксаторами, ограничивающими движение по оси и обеспечивающими дополнительное сдерживание стержня, который претерпел столкновение.

2) Стержни не двигаются, пока их не толкнут, и только с толчком можно выяснить, заблокирован ли стержень.

3) В каждый момент времени стержень можно толкнуть, освободить или оставить как есть.

Модель стержней и пазов (Дрекслер, Меркле)



Внешний уровень — без регулировщика (driver). С него начинается активация системы. Когда она дойдёт до 1 уровня, начнут отключаться в обратном порядке регулировщики, и результирующий сигнал вновь достигнет внешнего уровня, где может быть считан.

Модель стержней и пазов (физическая реализация)

Показано, что реализация вычислений на уровне атомов обязана быть реверсивной.

Helical computer (Merkle)

Дрекслер — молекулярная модель атомов углерода (rod logic).

Меркле — электронная модель движения электронов в системе потенциальных ям (buckled logic).

Реверсивные языки программирования

1) Обратимая надстройка над функцией f(y) функция $g(x,y)=(x \circ f(y),y)$, такая, что операция $\circ$ допускает левое сокращение. У всех функций существуют обратимые надстройки.

2) Обратная детерминированность программы.

3) Возможность вычисления программы в другую сторону («инверсность»).

Реверсивные языки программирования

Язык Janus (Yokoyama):

1) Все функции превращены в «обратимые надстройки».

2) В циклах и условиях возвращается информация о пройденном пути («история» Беннета).

Пример: вычисление n-ого числа Фибоначчи в Janus программируется с запоминанием пары (n-ого и n-1-ого чисел).

Реверсивные языки программирования (2)

Язык Janus (Yokoyama):

3) Практически не учитываются существенные особенности реверсивного программирования
4) Язык низкого уровня (нет даже процедур с параметром процедура)

Конструктивизм

Методологические принципы

Универсальность

- Универсальность – отрицательный эпитет. Универсальная система – специализированная, забывшая рамки своей применимости (см. США в мире).
- Универсальные системы – те, которые переросли рамки своей разумности
- Знание слабостей и границ сильного метода является важнейшим условием настоящего мастерства при владении этим методом.

Новизна

- «Новое» - не положительный и не отрицательный эпитет, а **скоропортящийся**.
- Если что-то *рекламируется* как «новое», лучше бегите от него: значит, реальных достоинств нет
- Самый тяжкий случай, когда нечто десятилетиями является «новым и перспективным» (пример – параллельное программирование). Значит, это больно с самого рождения (а генетические болезни неизлечимы).

Основные логические достижения XX века

- Разрушение иллюзии доступности истины человеку и познаваемости мира (теоремы о неполноте Геделя и Чейтина)
- Осознание необходимости как реальных, так и идеальных объектов (программа Гильберта)
- Появление рациональной альтернативы традицонному рационализму (Брауэр, Марков)

Причина конструктивизма

Появление в математике грязных теорем существования
(существует, но построить никак нельзя).
Побочная причина: кризис оснований математики

Появление конструктивизма

Брауэр: причина грязных теорем не какой-то конкретный математический принцип, а сама классическая логика.

Доказать — это решить задачу, скрытую в теореме

Назначение

Конструктивзм – математика, в которой ориентируются не на «истинность», а на построения.

Определение конструктивизма

Конструктивизм — форма математики, в которой доказательство должно дать идеальное умственное построение искомого объекта.

Теорема должна быть реализована.

Псевдопроблема Фреге

В конструктивной математике логические значения — вырожденная абстракция

Теорема ценна не своей «истинностью», а своим конкретным доказательством, дающим конкретное построение.

Основное изменение взгляда

Изменяются в первую очередь не логика и не объекты, а само понимание математических высказываний: они из утверждений становятся задачами.

Это лучше соответствует даже «практической (спортивной) математике»

Ошибка Брауэра

Классическая логика в эллинской математике не привела к грязным теоремам существования.

Причина: разрешимость геометрии.

Так что логика математики сразу создавалась не для конечных объектов.

Зародыш конструктивизма

Эллинская математика

Построения циркулем и линейкой

Построить и обосновать!

Конструктивизм и информатика

Реализация - программа

Построить и обосновать! – неэкономно

Построить одновременно с обоснованием - конструктивизм

Основной принцип конструктивизма

Принцип конечной информации: конечная информация о результате преобразования может быть получена по конечной информации о его параметрах.

По сути дела топологический принцип.

Внимание! «Новое» понятие!

- Концептуальное противоречие.

Спрятано, никто не видит.
На первый взгляд безопасно и привлекательно, опасности начинаются далее.

Совсем новое понятие

- Священная корова.

Торчит на самом виду. Страшно мешает. Никто не осмеливается заметить и сказать.

Старое понятие (Лесков, XIX в)

- Благоглупость.

Приятно, гладко, было бы хорошо, если бы оно так было.

- При малейшем отклонении получается гадость

Применения конструктивизма

Анализ концептуальных противоречий и выявление священных коров как минимум в информатике.

Почти не используется (противоречит «позитивному мышлению»).

Применения конструктивизма

Попытка синтеза программ (информатика созрела до функциональных языков, а они прямо соответствуют выводам).

Полностью провалилась в конце 70-х- начале 80-х.

Методологические выводы о приложениях

Неприятное, но полезное принимают лишь в случае очевидной тяжелой болезни, и то не всегда.

Лучшие приложения высокой теории — косвенные.

Прямой приятный результат — ловушка Дьявола.

А как же благоглупости?

Их можно успешно ловить и в классической математике (неустойчивость или переход к модели более высокого порядка)

Стили программирования

Первоуровневые стили

Действия Условия	Локальные	Глобальные
Локальные	**Структурное**	**Автоматное**
	Циклическое	Последовательное
	Рекурсивное	Параллельное
Глобальные	**Событийное**	**Сентенциальное**
	Демоны	Конкретизация
	Обработчики	Унификация

Разные логики

Все стили концептуально противоречивы, требуют разной логики рассуждений

Особенно коварны противоречия внутри одной большой клетки

Универсальность — не похвала!

Структурное программирование

- Его логика интуиционистская; почти что классическая, но $\neg\forall x(A(x)\vee\neg A(x))$ - не противоречие.
- Получающаяся программа нейтральна по отношению к последовательному исполнению и параллелизму
- Задача распараллеливания возникла из-за отвратительного дизайна современных языков программирования.

Что опускается в ЯП?

- Призраки (Цейтин, 1970): нет в программе, но нужно для ее понимания.
- Примеры: цель, ограничение числа шагов (можно бесконечными числами!)
- Подпорки (2005): есть в программе, но вредно для ее понимания.
- Примеры самых коварных: ; if then else

Что опускается в интуиционистской логике?

Предполагается, что ресурсы не ограничены.
Лишь бы все вычислялось за конечное время с конечной памятью.
Результат: простые логические построения порою приводят к исключительно ресурсоемким программам.
Логика чистых построений и чистого знания

Нильпотентная логика автоматного программирования или логика конечного времени (1982)

Автоматное программирование

Таблица переходов — естественный способ программирования.

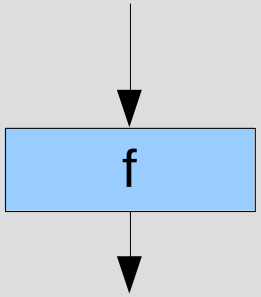
Go to здесь на месте, циклы не на месте.

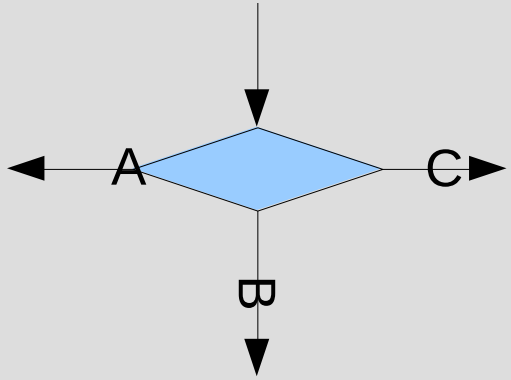
Присваивания страшно мешают, вызовы функций-нет

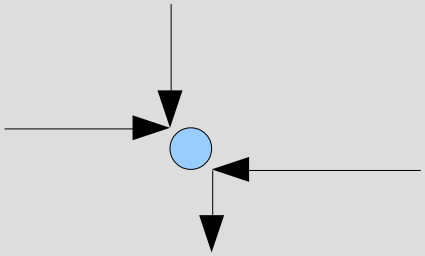
Автоматные модели

Таблица переходов и состояний
Машина
Полугруппа
Грамматика
Схема Янова
…

Недетерминированные схемы Янова







Недетерминированные схемы Янова (2)

Схема невырождена, если в любом цикле и на любом пути из входа в выход есть хотя бы одно действие

Модель ресурсов и действий

Множество состояний S и фундированный чум оценок ресурсов R.

r(s) — функция ресурсов

Действие f — такое отношение, что

$\forall s,t\ (s\ f\ t \rightarrow r(t)<r(f))$

Недетерминированность, частичность

Модель ресурсов и действий (2)

Исполнение — путь из входа в выход, такой, что каждой дуге приписаны состояния. Состояние изменяется, дишь если дуга выходит из действия и тогда s_n f s_{n+1}

Если дуге приписана формула, то она истинна в данном состоянии

Модель ресурсов и действий (3)

Если все предикаты и действия разрешимы, то отношение, реализуемое схемой, тоже разрешимо

Если они перечислимы, то оно перечислимы.

Если недетерминизм конечный и необратимы, то недетерминизм остается конечным

Спецификации

Специфицированная схема — формулы сопоставлены всем дугам.

Корректное исполнение — все формулы верны.

Спецификация действий — пары пропозициональных формул $A \to B$ (конструктивная импликация)

Формулы 1 порядка

$A \rightarrow B$ – конструктивная импликация

Классические лишь внутри:

$A \vee B$, $A \& B$, $\neg A$, $A > B$,

Исчисление 1 порядка

$A \Rightarrow B$ $C \Rightarrow D$ тогда $A \vee C \Rightarrow B \vee D$

$A \vee B \Rightarrow A \vee C$ тогда $B \Rightarrow C$

$A \supset B$ $B \Rightarrow C$ $C \supset D$ тогда $A \Rightarrow D$

False $\rightarrow$ false

$A \Rightarrow A$ тогда $\neg A$ (правило исключенного застоя)

Естественно добавляется конструктивная дизъюнкция. Конъюнкция и отрицание – нет.

Выражение последовательности

$A \rightarrow B$ $B \rightarrow C$

$A \vee B \rightarrow B \vee C$

$A \rightarrow C$

Стандартная форма: дейкстровский if внутри цикла.

Каждое действие достаточно записать в схеме один раз.

Проблема высших типов

Как работать с функциями над функциями?

Для обычных функций решена Шейнфинкелем в 1920 г.

((K x) y)= x

(((S x) y) z)= ((x z) (y z))

Для автоматов не удавалось продвинуться 50 лет.

Реверсивная логика обратимых вычислений (2008)

Логический язык CRL

Классические связки &,V,>,¬.
Конструктивные связки =>,&,~,E.
Они полностью интероперабельны.
Модель: группа
Реализация: подмножество группы.
Классические связки булевы.
c ® A⇒B iff для всех a®A a°c®B
c® ~A iff c^{-1}® A
C® A&B iff существуют a®A, b®B c=a°b

Разрешимость и кванторы

Реализуемость выразима в теории групп с конечным числом дополнительных одноместных предикатов. Она разрешима (Ю. Л. Ершов), значит, разрешима CRL

В CRL на пропозициональном уровне выразимы кванторы

$\forall A$ iff $E \supset (A \vee \neg A \Rightarrow A)$

$\exists A$ iff $\neg \forall \neg A$

Некоторые математические результаты

Все конструктивные связки независимы (Н.А.Н)
Логика не описывается конечной таблицей истинности (Н.А.Н.)
Другие результаты переходим к pdf статьи

Программистские следствия

Реверсивные вычисления – стиль программирования со своей собственной логикой.
Архитектурные решения обычных языков для них заводят в тупик.
Двоичная логика в качестве основы реверсивных компьютеров достаточно бесперспективна.
Нужны элементы, реализующие вычисления в группах.

Программистские следствия (2)

Реверсивное программирование с самого начала функционально, поскольку в CRL функции представляются как элементы группы
В чистом реверсивном языке не должно быть условных операторов.
В реалистичном реверсивном языке использование условных операторов должно быть резко ограничено, поскольку каждый из них требует выделения дополнительной реверсивной памяти.

Программистские следствия (3)

Операции ввода и вывода информации существенно нереверсивны.
Следствие 1: Реверсивный процессор должен работать в составе системы, в которой главный процессор традиционного типа.
Следствие 2: реверсивный язык программирование не должен быть полон по Тьюрингу.
Следствие 3: Первостепенная задача – интерфейсы реверсивного и традиционного вычислителей.

Предупреждение и прогноз

Недостаточно аппаратной реализации групп вычетов. Нужно реализовывать прямую сумму рабочей и вспомогательных групп. Обойтись без этого можно лишь на задачах, идеально «реверсивно программируемых», но для каждой такой задачи будет своя группа.
Идеальные для реверсивной работы задачи, судя по всему, прежде всего задачи линейной алгебры и небесной механики (где обратимость по существу и «увеличения энтропии» в ходе решения нет).

Базы данных с реверсивностью

В базах данных возникает практическая задача обеспечить реверсивность хотя бы на некоторое время («время исковой давности»).
Здесь дополнительный расход памяти на запоминание вариантов условных операторов ничтожен по сравнению с объемом информации при тупом запоминании контрольной точки.

Полуреверсивность

Часто возникает задача обеспечить частичную реверсивность: возможность отката системы внутри некоторого горизонта.
Здесь возникает не группа, а моноид с операцией правого обратного: $a*a^{-1}=e$.
При функциональном программировании каждая процедура должна проектироваться одновременно с процедурой отката. F Undo_F

Энергетическая емкость задач

Возникает новая характеристика задачи: минимальная энтропия, возникающая при ее решении за заданное время.

Список ссылок на классические работы

1) Fredkin E.F., Toffoli T. Conservative logic. Int. J. Theoretical Physics, 21, 3 / 4 (1982), 219-253.

2) Bennett C.H. Logical reversibility of computation. IBM J. Res. Dev., 17, 6 (1973), 525-532.

3) Bennett C.H. Time/space trade-offs for reversible computation. SIAM J. Comput., 18, 4 (1989), 766-776.

4) Merkle R.C., Drexler K.E. Helical logic. Nanotechnology, 7, 4 (1996), 325-339.

5) Landauer R. Irreversibility and heat generation in the computing process. IBM J. Res. Dev., 5 (1961), 183-191.

Ссылки на современные работы

1) Vitanyi T. Time, space and energy in reversible computing. Conference on Computing Frontiers. ACM Press, 2005.

2) Frank M.P. Introduction to reversible computing: motivation, progress and challenges. Conference on Computing Frontiers. ACM Press, 2005.

3) Kerntopf P. A new heuristic algorithm for reversible logic circuit synthesis. 41-th Design Automaton Conference, San Diego (2004).

4) Yokoyama T., Gluck R. A revesible programming language and its invertible self-interpreter. Partial Evaluation and Program manipulation. ACM Press, 2007.

Ссылки на современные работы (2)

5) S. Abramsky, **A Structural Approach To Reversible Computation**, *Theoretical Computer Science* vol. 347(3), 441--464, 2005

6) WILLE, R., OFFERMANN, S., AND DRECHSLER, R. 2010a. SyReC: A Programming Language for Synthesis of Reversible Circuits. In *Forum on specification & Design Languages.*

7) YOKOYAMA, T., AXELSEN, H. B., AND GL¨UCK, R. 2008. Principles of a reversible programming language. *Comput. Frontiers, 43–54.*