К практическому использованию формальной верификации при разработке ПО

И.В.Шошмина, Ю.Г.Карпов
ishoshmina@dcn.ftk.spbstu.ru
karpov@dcn.ftk.spbstu.ru
РВКС, ФТК, СПбГПУ
2011

Примеры недавних ошибок

- 09.09.2009. Компания Volvo объявила об отзыве по всему миру 26 тысяч автомобилей, у которых обнаружена неисправность в блоке управления двигателем. Из-за этого дефекта мотор может не запуститься или заглохнуть после того, как автомобиль проедет несколько сотен метров.
  - Все владельцы неисправных машин получат письмо с предложением посетить сервисную станцию для бесплатного перепрограммирования блока управлением двигателем
- 14.09.2009. Авария на Саяно-Шушенской ГЭС: ущерб более миллиарда долларов (>40 млрд руб), погибли 75 человек
  - Глава Ростехнадзора Николай Кутьин: «*Основные причины аварии на Саяно-Шушенской ГЭС лежат в технологической области. Они связаны с недостатками эксплуатации, проектирования и недостатками в работе систем контроля. Стали очевидными системные недочеты в работе автоматики и самой системы контроля*» (т.е. в программном комплексе АСУ). Вышедший из строя второй агрегат был модернизирован в I квартале 2009 г, на него поставили новую систему управления.
  - Кутьин: у его ведомства есть ряд вопросов: «как была спроектирована система управления, какие алгоритмы были заданы, ...»

Последствия программных ошибок – каждый день

- Апрель, 2010. Авария в Мексиканском заливе: возможна ли программная ошибка? *Don Shafer, Phillip Laplante. The BP Oil Spill: Could Software be a Culprit? IEEE IT Pro September/October 2010, IEEE, 2010.*
  В апреле 2010 года на буровой платформе Deepwater Horizon произошел взрыв, причинивший огромный ущерб окружающей среде, экономике и будущему всей отрасли глубоководной добычи полезных ископаемых в США. Не могла ли одной из причин бедствия стать ошибка в программном обеспечении?

- 05.07.2010. Apple: ошибка связи iPhone 4 — программная. Компания Apple признала существование в iPhone 4 проблем, касающихся качества связи.
- 06.12.2010. Спутники ГЛОНАСС и ракету “Протон-М” утопили программисты. Ракета-носитель «Протон-М» со спутниками «Глонасс-М» отклонились от заданного курса из-за допущенных ошибок в математическом обеспечении, заложенном в бортовой компьютер ракеты (основная причина - неправильно написанная формула в документации на заправку кислородом разгонного блока)
- 08.12.2010 — Японский зонд «Акацуки» не смог выйти на орбиту Венеры

  Космический исследовательский аппарат «Акацуки», запущенный Японией для исследования Венеры, не смог выйти на орбиту планеты, сообщает РИА «Новости» со ссылкой на специалистов Японского аэрокосмического агентства JAXA

Задача повышения качества программ является актуальной современной проблемой

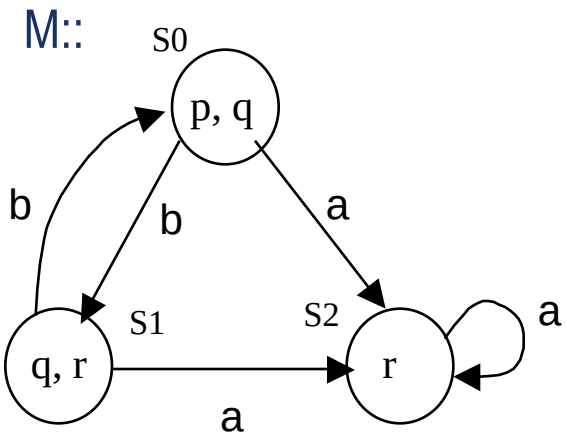
Подходы к обеспечению качества программ

- Статические методы
- Динамические методы:
  - Тестирование
  - Имитационное моделирование
  - Формальная верификация
    - применяется редко

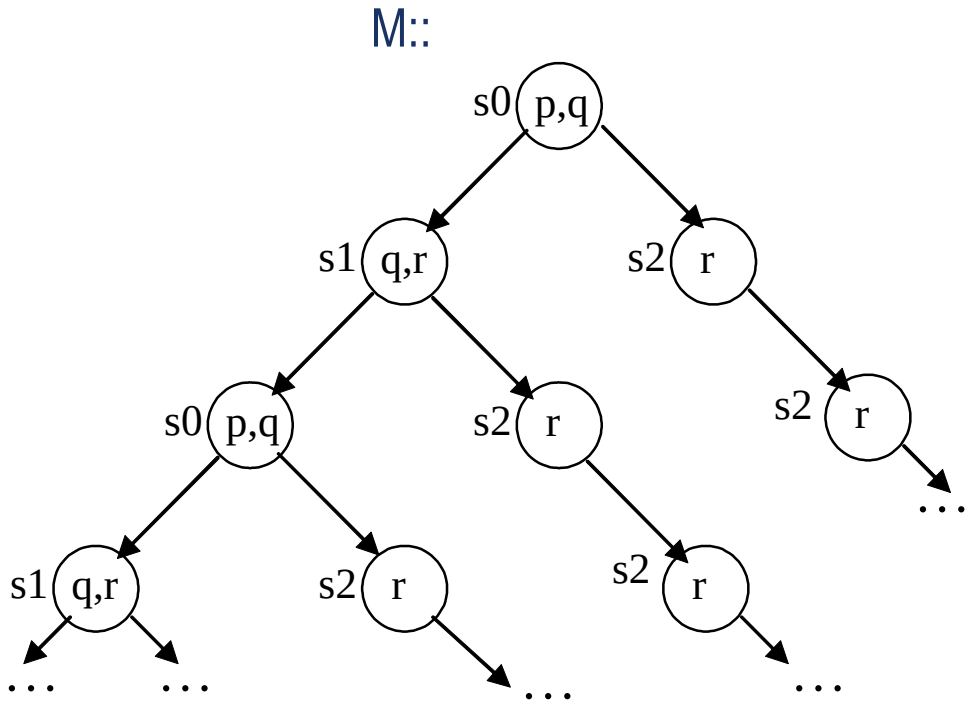
В последнее время – качественный прорыв в одном из направлений исследований в области верификации ПО и дискретных систем, основанный на изящных формальных методах – model checking (проверка модели)

Проверка модели – Model Checking

Тип систем: реагирующие системы (reactive systems)



Свойства вычислений описываются темпоральные логики φ=FGr



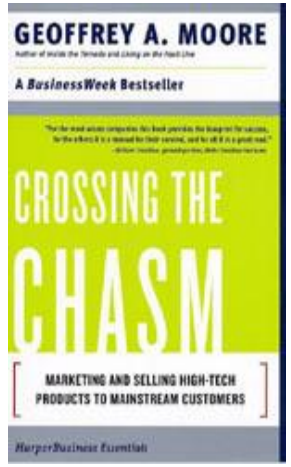
Разработаны методы проверки модели для разных темпоральных логик

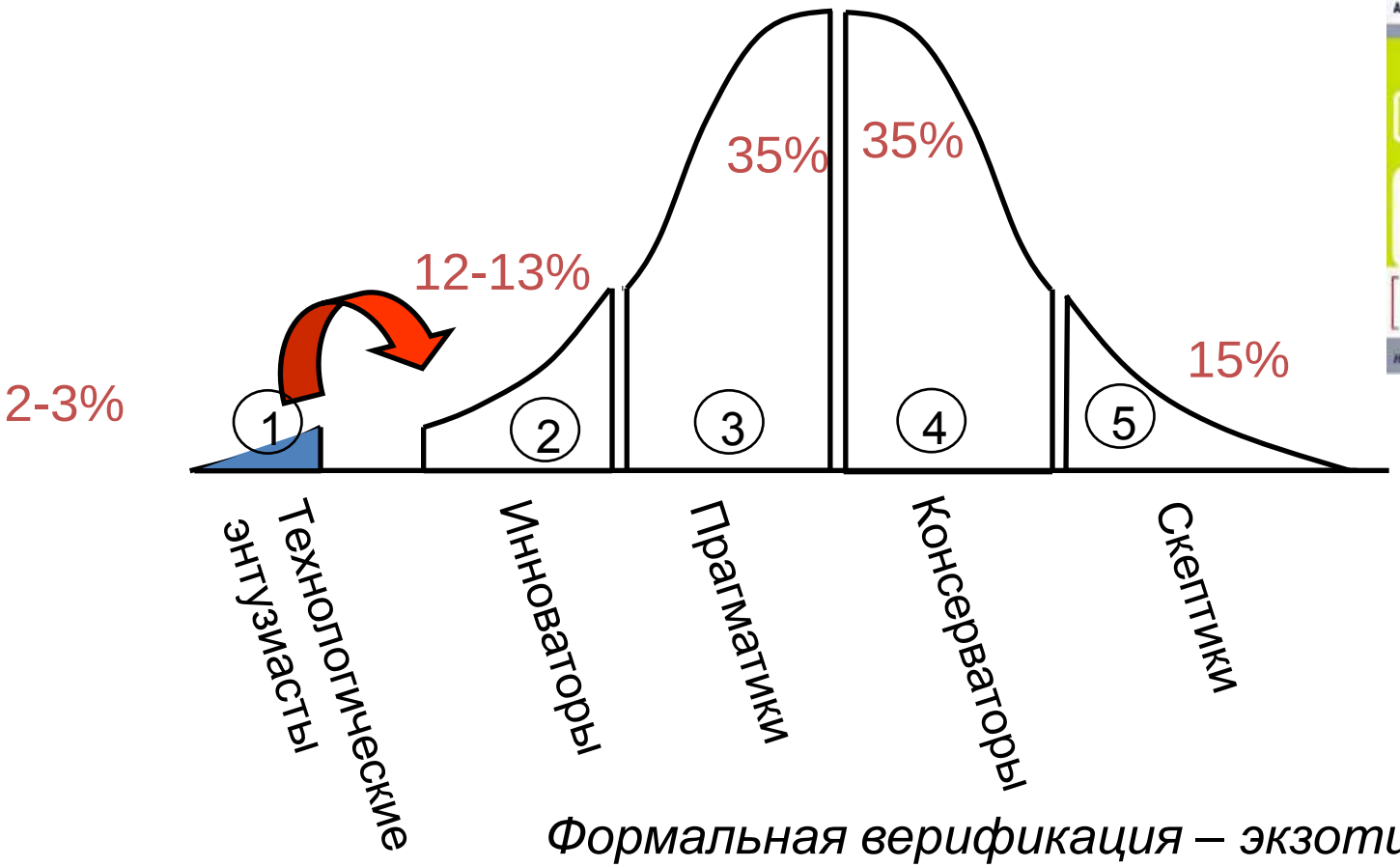
- Новые методы верификации потенциально позволяют существенно улучшить качество разрабатываемого ПО
- НО! в практику эти методы входят медленно

Причины:

- уровень разработки теоретической и технологической базы методов формальной верификации

Динамика внедрения новых технологий (technology adoption cycle)





Формальная верификация – экзотика

Задача: развитие теоретических методов и технологических решений для внедрения метода проверки на модели в проектирование программ

Ведущие компании – разработчики встроенных систем управления занимаются разработками по включению формальных методов в практику корпоративного проектирования

NASA, Lockheed Martin, Rockwell Collins, IBM, Microsoft, Airbus

Пример: отчет фирмы Rockwell Collins 2007г.

ADVANCED COMPUTING SYSTEMS

Formal Verification of Flight Critical Software

Dr. Steven P. Miller
Advanced Computing Systems

Elise A. Anderson
Commercial Systems Flight Control

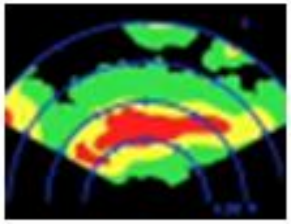
Rockwell Collins
400 Collins Road NE, MS 108-206
Cedar Rapids, Iowa 52498

{spmiller,eaanders}@rockwellcollins.com

Who Are We?

A World Leader In Aviation Electronics And Airborne/ Mobile Communications Systems For Commercial And Military Applications

- Communications
- Navigation
- Automated Flight Control
- Displays / Surveillance
- Aviation Services
- In-Flight Entertainment
- Integrated Aviation Electronics
- Information Management Systems



Methods and Tools for Flight Critical Systems Project

- Five Year Project Started in 2001
- Part of NASA's Aviation Safety Program (Contract NCC-01001)
- Funded by the NASA Langley Research Center and Rockwell Collins
- Practical Application of Formal Methods To Modern Avionics Systems

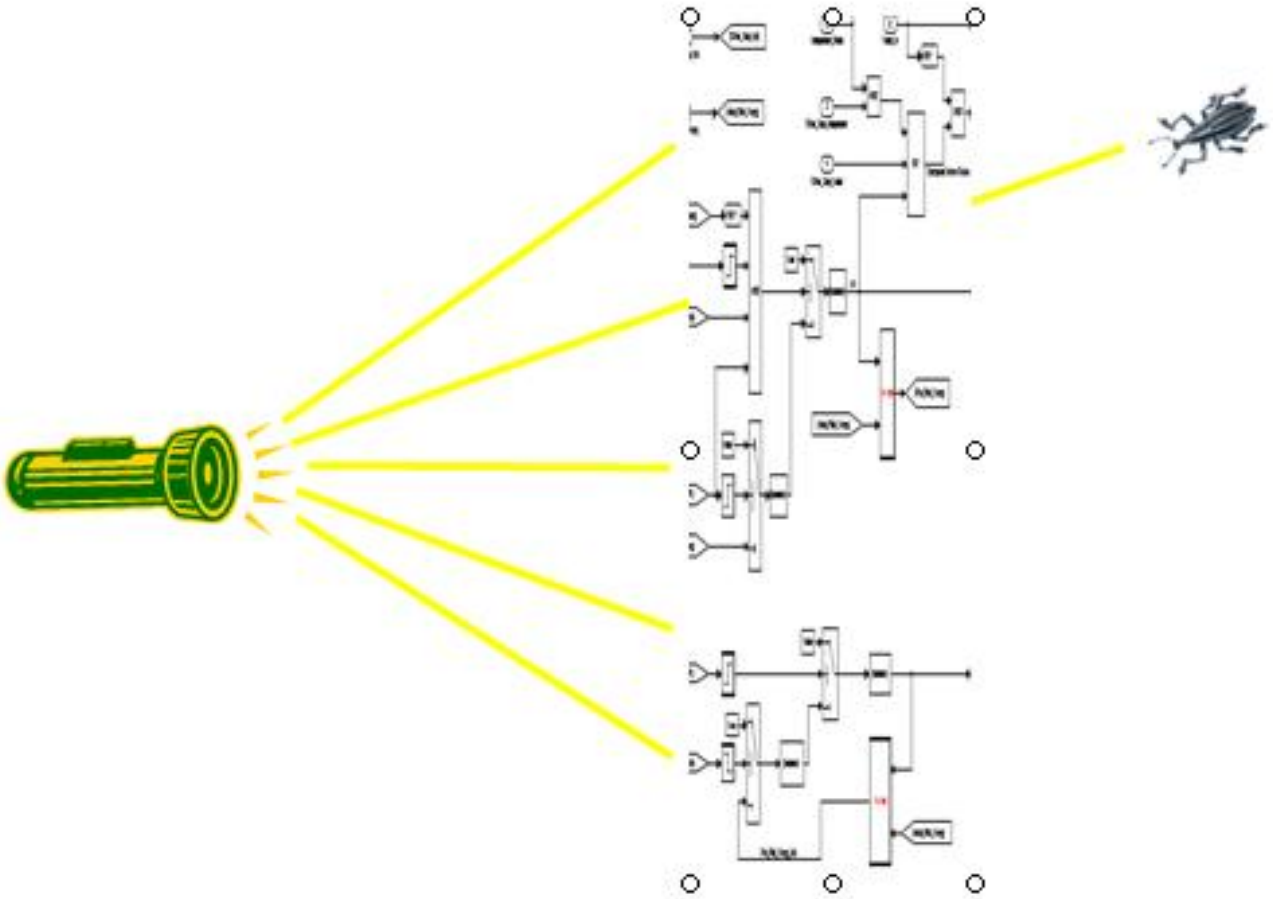
What Are Model Checkers?

- Breakthrough Technology of the 1990's
- Widely Used in Hardware Verification (Intel, Motorola, IBM, …)
- Several Different Types of Model Checkers
  - Explicit, Symbolic, Bounded, Infinite Bounded, …
- Exhaustive Search of the Global State Space
  - Consider All Combinations of Inputs and States
  - Equivalent to Exhaustive Testing of the Model
  - Produces a Counter Example if a Property is Not True
- Easy to Use
  - "Push Button" Formal Methods
  - Very Little Human Effort Unless You're at the Tool's Limits
- Limitations
  - State Space Explosion (10^{100} – 10^{300} States)

Advantage of Model Checking

Testing Checks Only the Values We Select

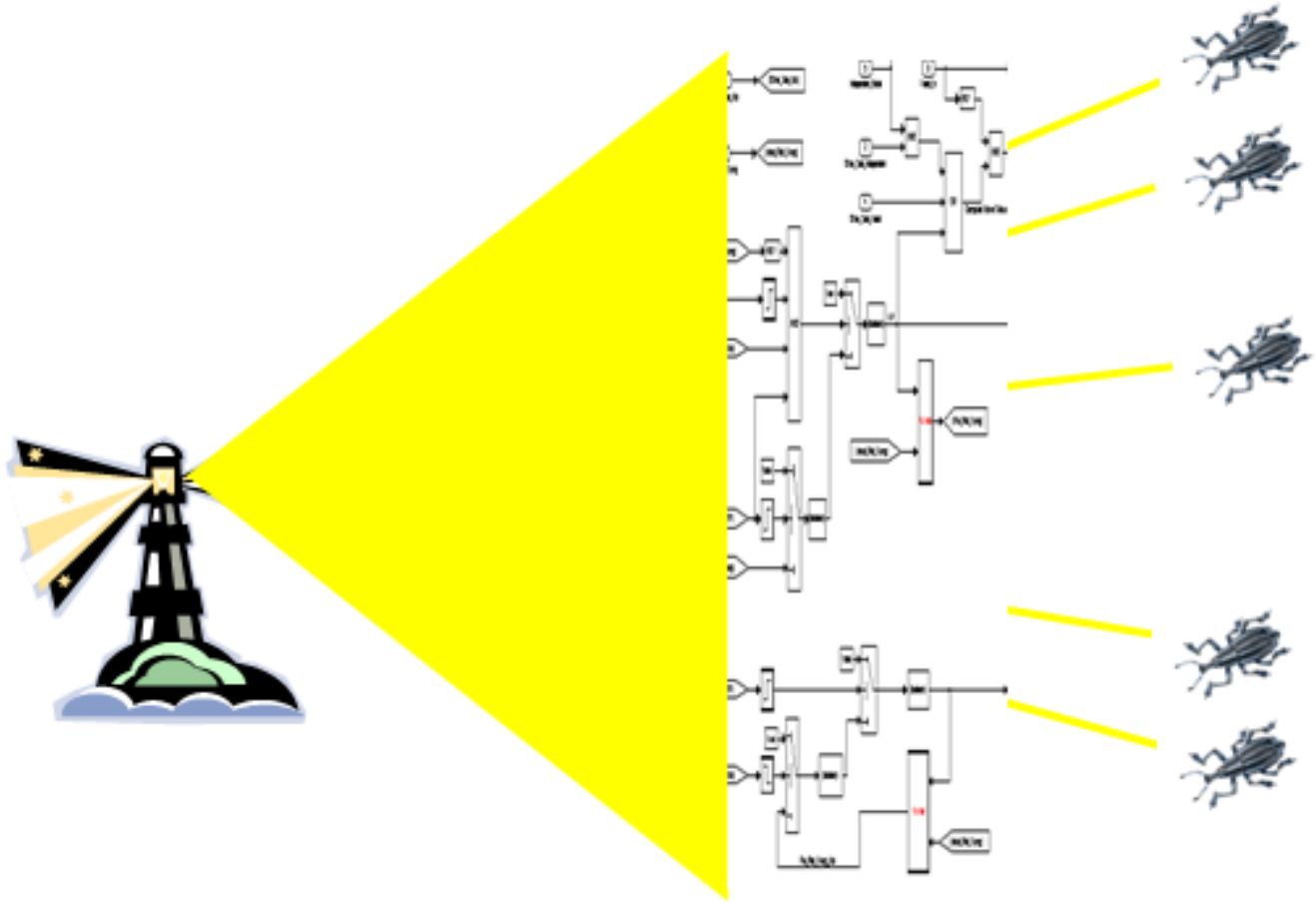
Even Small Systems Have Trillions (of Trillions) of Possible Tests!



Advantage of Model Checking

ADVANCED COMPUTING SYSTEMS

Model Checker Tries Every Possible Input and State!



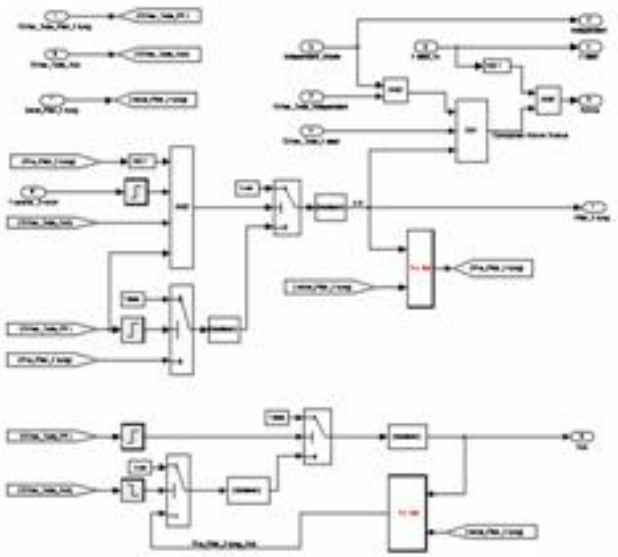
Rockwell

Example - ADGS-2100 Adaptive Display & Guidance System

883 Subsystems

9,772 Simulink Blocks

2.9 x 10^{52} Reachable States



Requirement

Drive the Maximum Number of Display Units Given the Available Graphics Processors

Counterexample Found in 5 Seconds!

Checking 373 Properties
Found Over 60 Errors

Summary of Errors Found

Dectected By	Likelihood of Being Found by Traditional Methods				
	Trivial	Likely	Possible	Unlikely	Total
Inspection			1	2	3
Modeling		5	1		6
Simulation					
Model Checking	2	1	13	1	17
Total	2	6	15	3	26

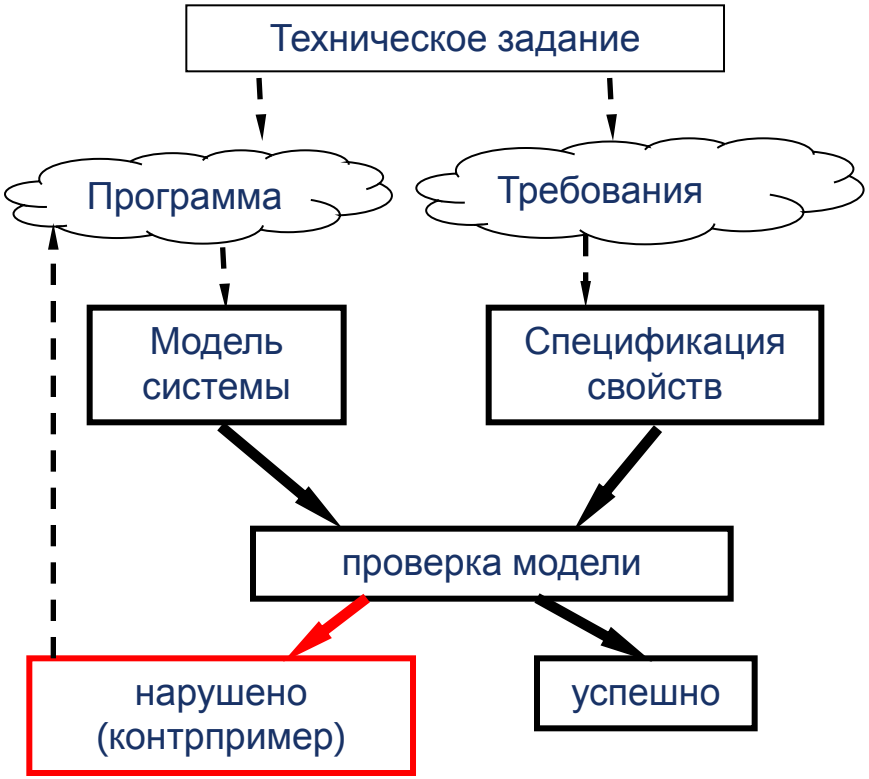
- **Model-Checking Detected the Majority of Errors**
- **Model-Checking Detected the Most Serious Errors**
- **Found *Early in the Lifecycle* during Requirements Analysis**

Conclusions

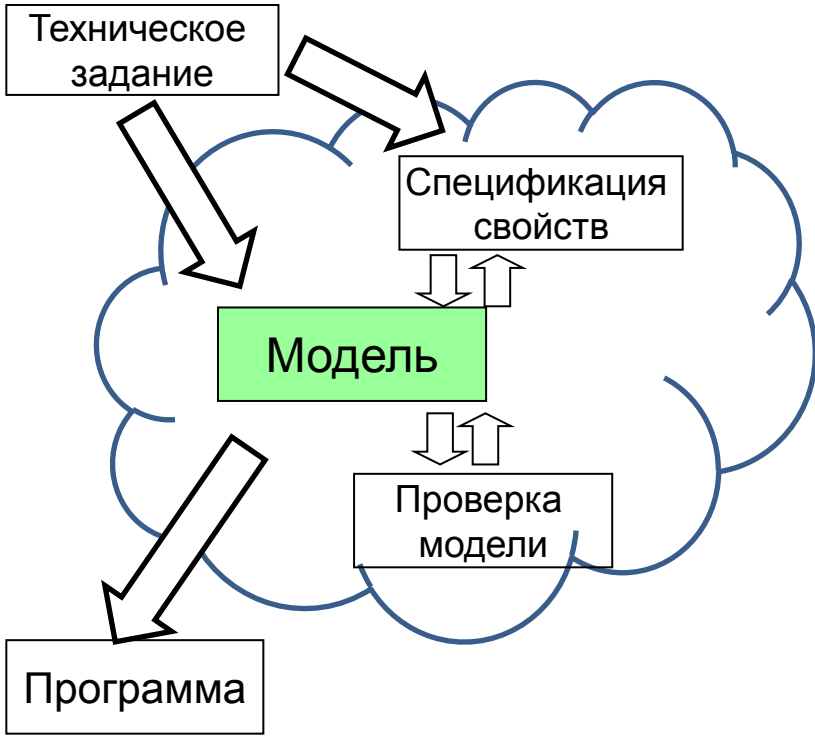
- Model-Based Development *is* the Industrial Use Formal Specification
- Convergence of Model-Based Development and Formal Verification
  - Engineers are Producing Specifications that Can be Analyzed
  - Formal Verification Tools are Getting More Powerful
- Model Checking is Very Cost Effective
  - Simple and Easy to Use
  - Finds All Exceptions to a Property
  - Used to Find Errors Early in the Lifecycle
- Applied to Models with Only Boolean and Enumerated Types

Основные подходы к применению Model Checking на практике

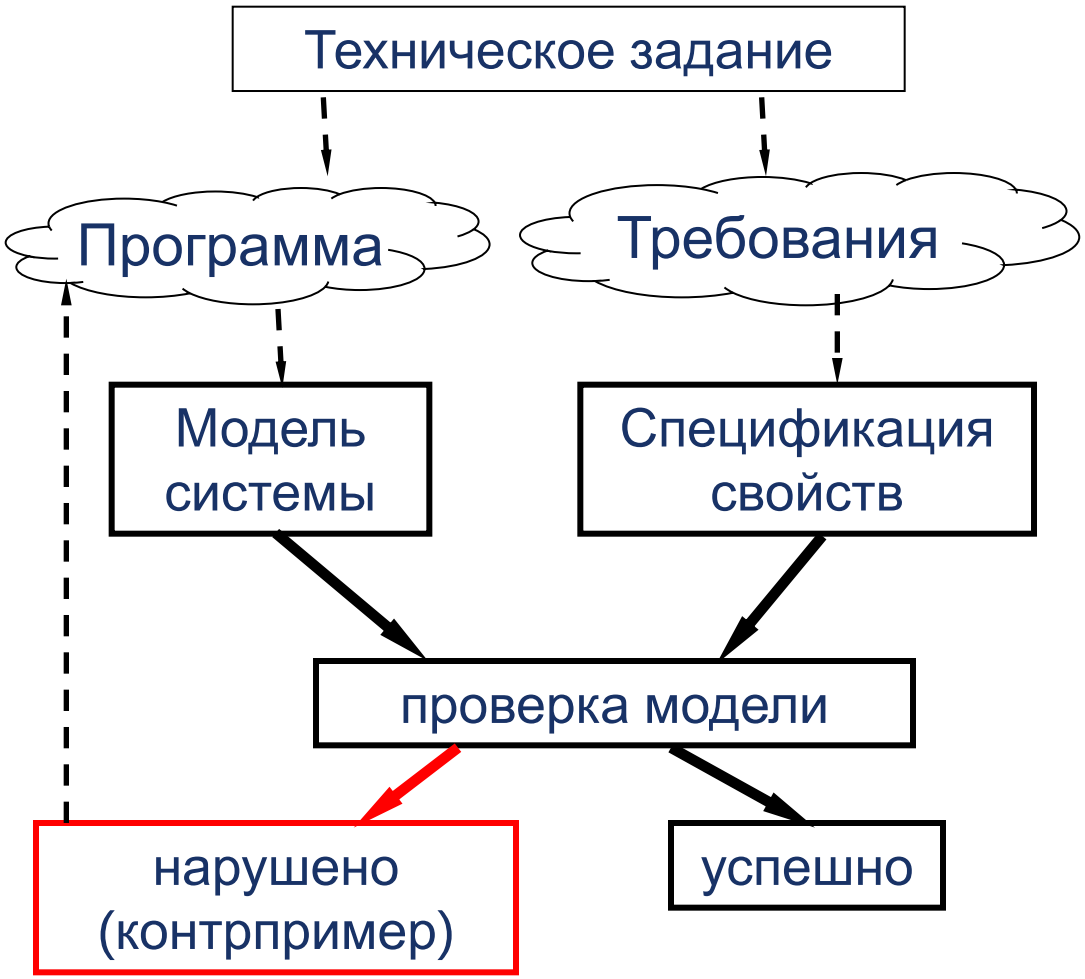
- Модель из кода программы



- Код программы из модели



Модель из кода программы



- Преимущества подхода:
  - Привычная технология разработки
  - Компактный код
- Недостатки подхода:
  - Большой размер построенной модели
  - Идея авторов программы не может быть восстановлена
  - В случае ошибок модификация кода производится на поздних стадиях проектирования

Подход противоречит парадигме инженерной деятельности

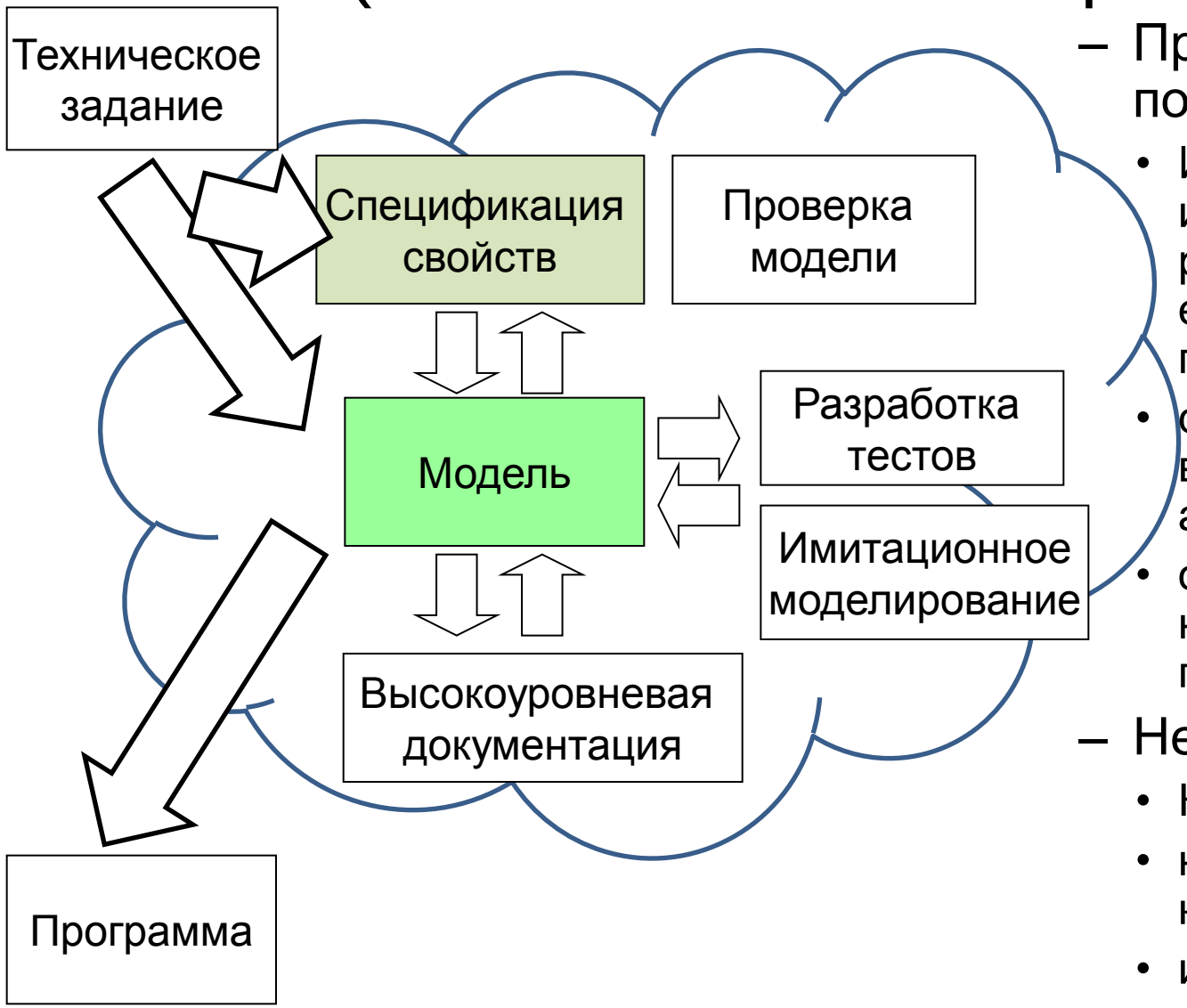
Эдсгер Дейкстра о верификации программ

- *Вместо того, чтобы программы сначала разрабатывать, а ПОТОМ пытаться доказать, что эта программа правильна,*

  программы нужно сразу строить корректными

Программа должна выводиться строго логически из точно представленной спецификации, а не только исходя из опыта и общих туманных соображений

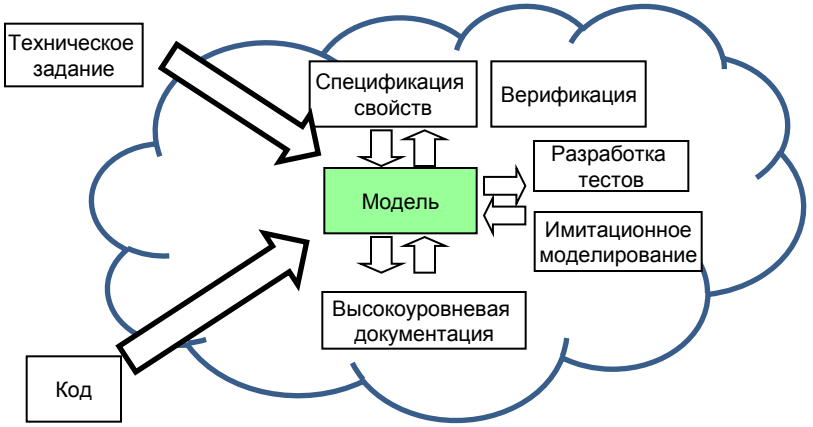
Проектирование ПО на основе моделей (Model-Driven Development)



– Преимущества подхода:
  - Идея выражена четко и ясно с начала разработки, модель и есть проект программной системы
  - систематизация включения методов анализа
  - обнаружение ошибок на ранней стадии проектирования
– Недостатки подхода:
  - Критика адекватности
  - неэффективность кодогенерации
  - изменение технологии

Система управления электронным микроскопом для Philips Electron Optics

- Для построения модели был проведен реинжениринг существующей системы в стейтчартах



В результате:

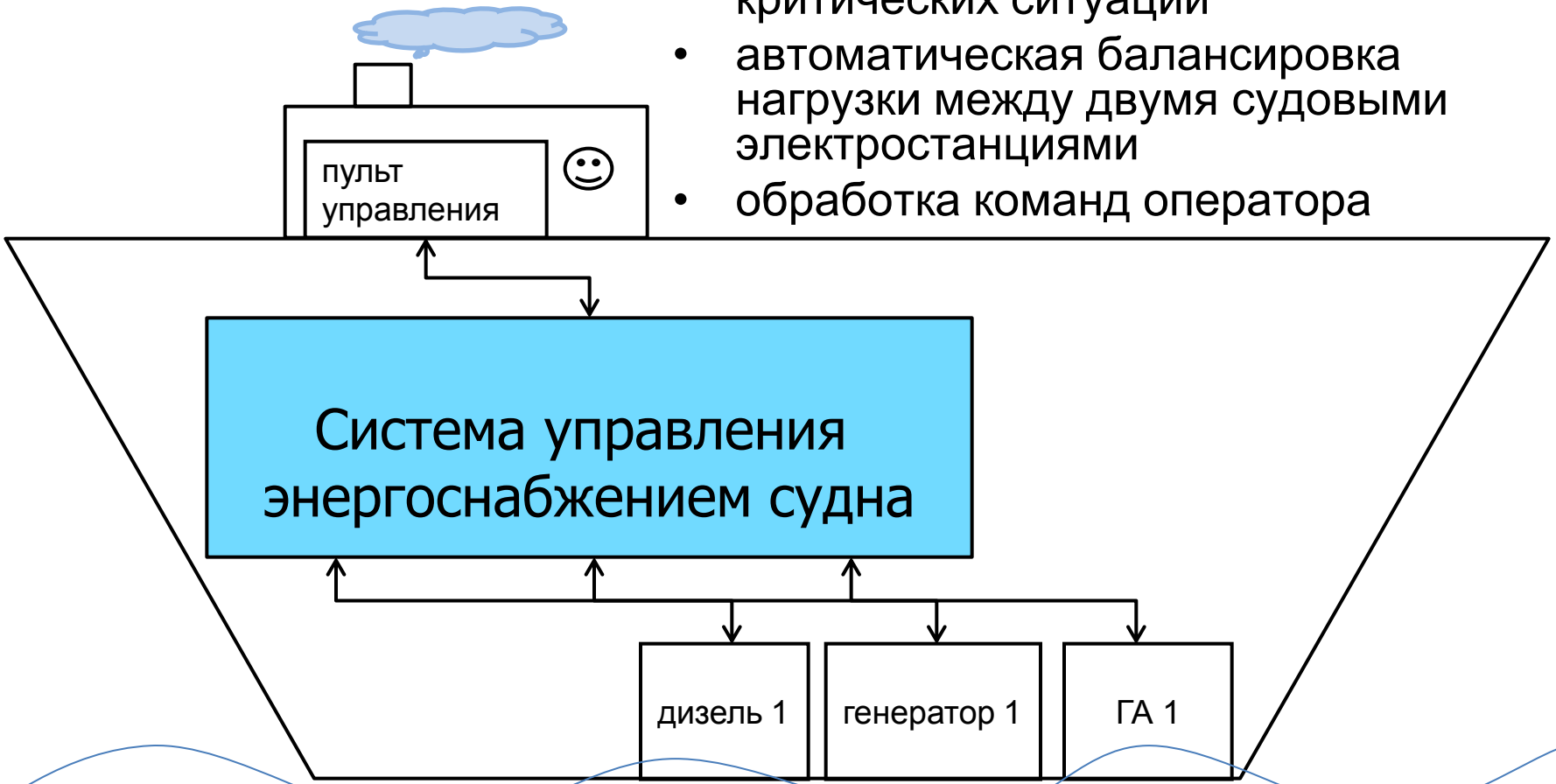
- Был обнаружен мертвый код, заплатки, ошибки
- Раньше код был понятен только программистам. Модель понятна всем разработчикам (физикам, менеджерам, инженерам)
- Интерфейсные модули постоянно модифицировались. Модель позволяет прозрачно вносить модификацию и отслеживать ее влияние, согласовывать со всеми разработчиками. Только после согласования изменений со всеми разработчиками генерировался код
- Эта модель использовалась при сопровождении системы, как основной документ, на котором фиксируются все модификации

Проект «Верификации СУЭС»

- в содружестве ЦНИИ судовой электротехники и технологии (СЭТ)
- ЦНИИ СЭТ предоставил:
  - протестированный код на C++ реальной системы энергоснабжением судна
  - на момент проведения наших работ эта система находилась на этапе сдачи заказчику
- Мы проводили исследования по практическому применению Model Checking

Система управления энергоснабжением судна (СУЭС)

- контроль состояния судовых электростанций и обработка критических ситуаций
- автоматическая балансировка нагрузки между двумя судовыми электростанциями
- обработка команд оператора



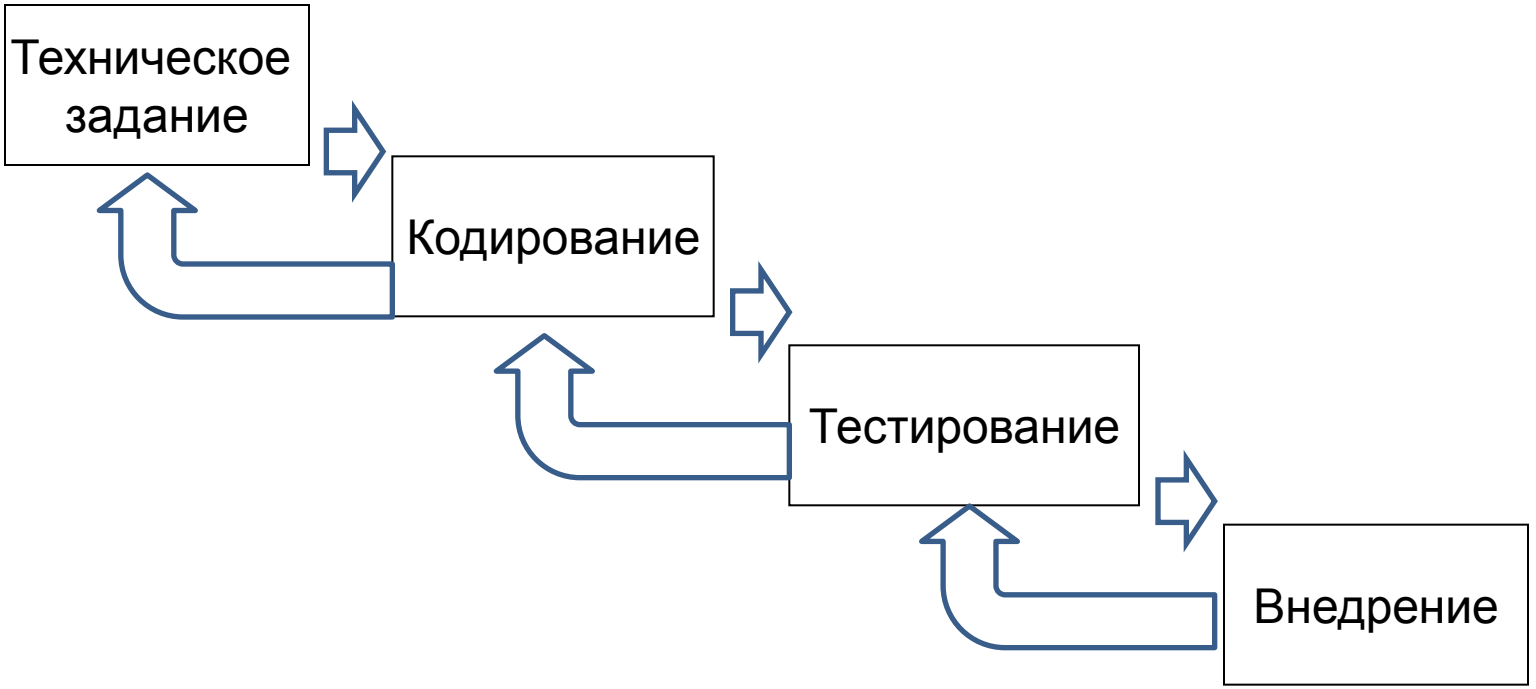
Задачи исследования

- построить четкую формальную спецификацию СУЭС
- построить представление системы, пригодное для верификации
- провести верификацию системы на основе наших теоретических разработок

Трудности проекта:

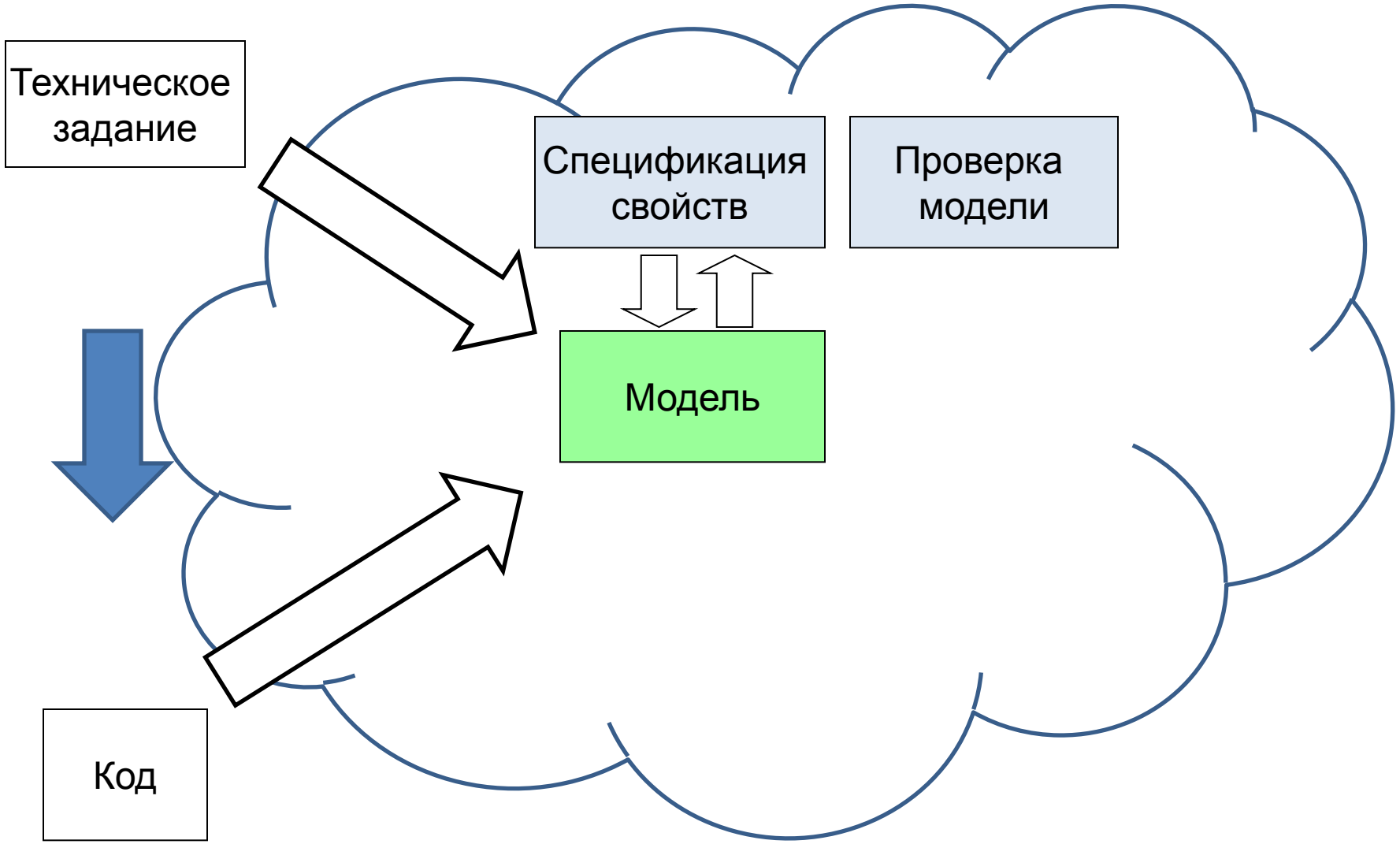
- техническое задание СУЭС не имеет четкого описания, допускает несколько толкований критически значимых ситуаций
- документация, описывающая общую архитектуру системы управления, отсутствует
- описание поведения конкретных модулей отсутствует

Модель проектирования ПО, использовавшаяся ЦНИИ СЭТ

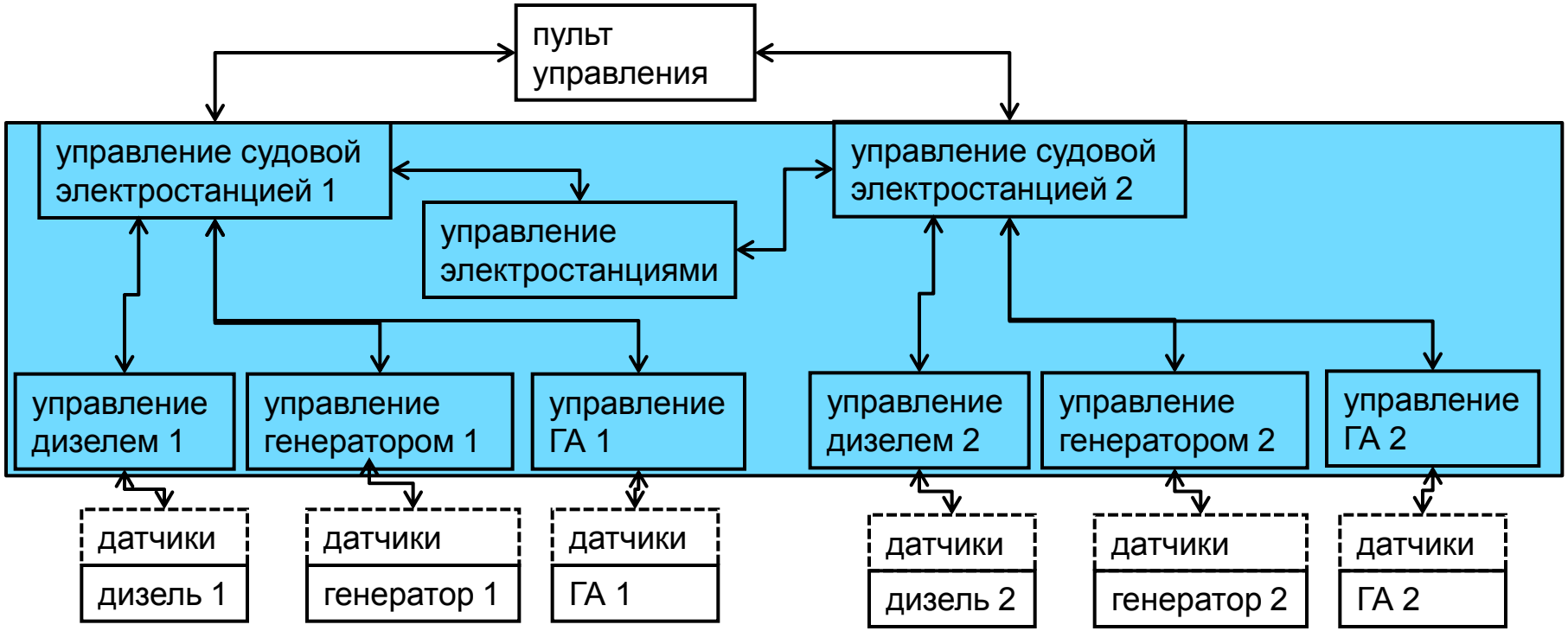


Тестирование проводилось в основном на поздних этапах разработки ПО

Проектирование СУЭС на основе моделей



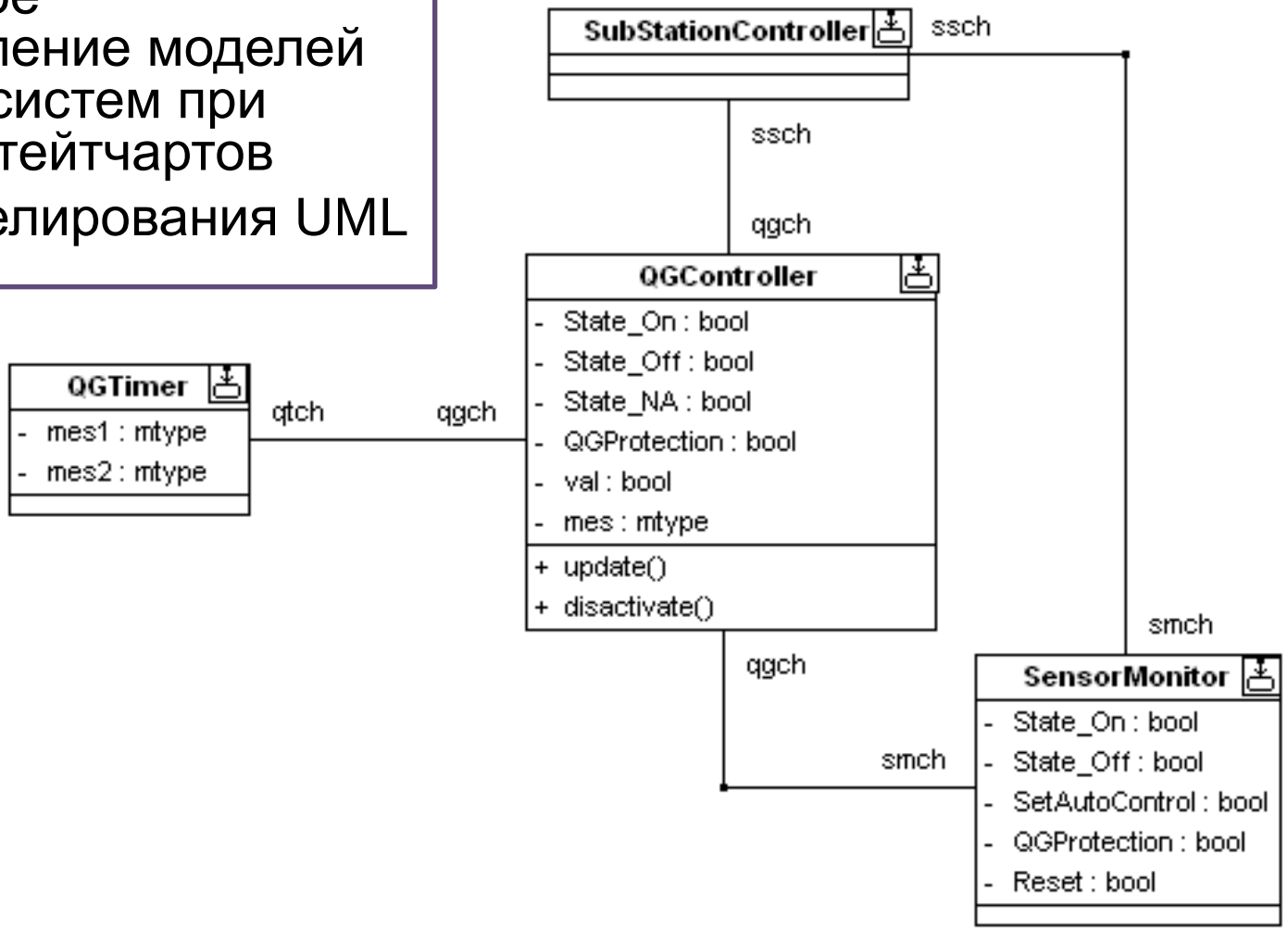
Модель системы



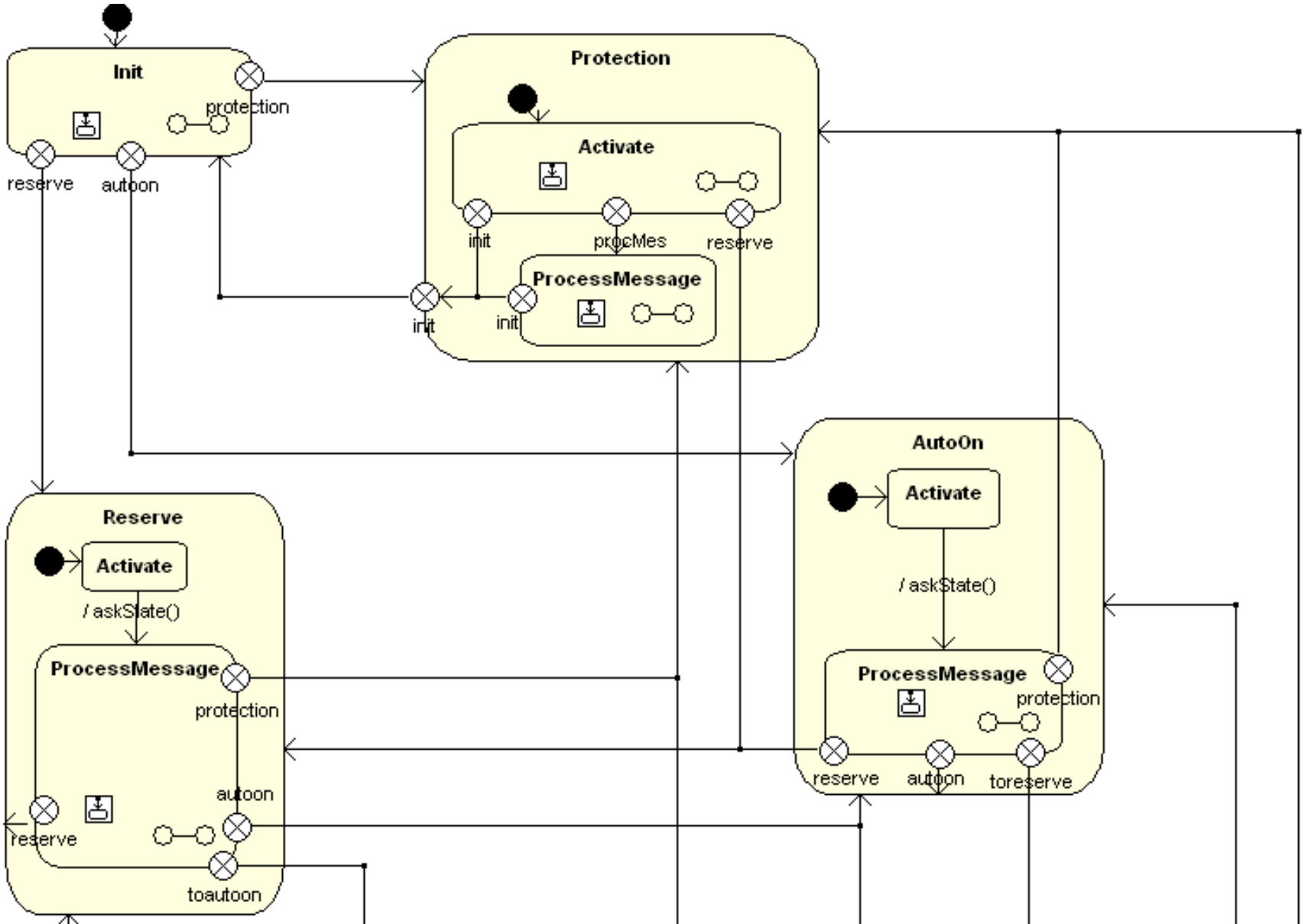
- 9 основных модулей – по процессу на каждый модуль
- процессы параллельны и работают асинхронно
- 28 состояний у каждого модуля – моделируются системы с конечным числом состояний
- 9 асинхронных каналов конечной емкости

Модель СУЭС на языке моделирования UML

- компактное представление моделей больших систем при помощи стейтчартов
- язык моделирования UML



Часть модели генераторного автомата



Формальная спецификация свойств и верификация

- Свойства формулируются на языке линейной темпоральной логики
- SPIN – система верификации, разработана Bell Labs
- Разработан транслятор UML->Spin

Функциональная спецификация свойств

- *Базовые свойства* – отсутствие блокировок, наличие недостижимого кода модели и другие (типичные проблемы синхронизации)
- *Требования управляемости* – достижение состояний, характеризующих выполнение функции
- *Требования надежности* – выполнение критических и опасных режимов, переход в такие режимы по управляющим командам
- *Требования корректности* – выполнение базовых сценариев, предписанных техническим заданием

Требования управляемости

- условие активного автоматического режима ГА

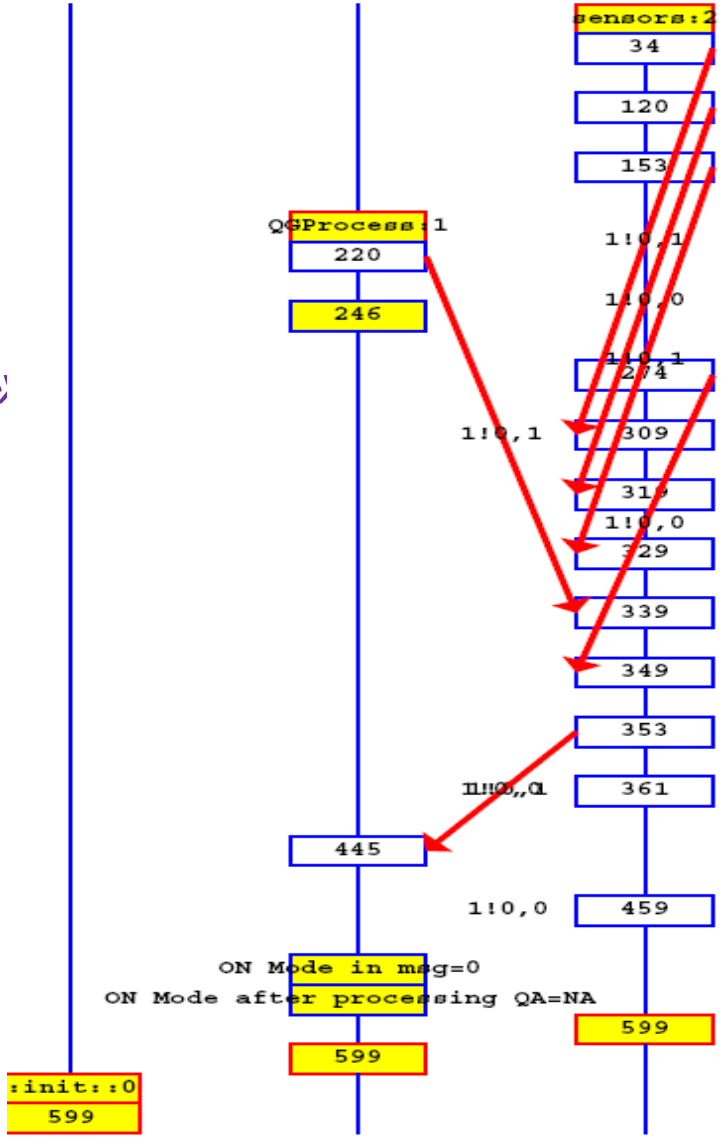
«в активном автоматическом режиме ГА датчик ГА находится в непротиворечивом состоянии»

$$G!(s \,\&\, r)$$

- s – ГА в активном автоматическом режиме
- r – локальное противоречивое значение датчика

Свойство нарушается

исправляется добавлением дополнительного перехода



Требование надежности

– условие выхода из режима защиты

«выход ГА из режима защиты возможен только по нажатию кнопки СБРОС»

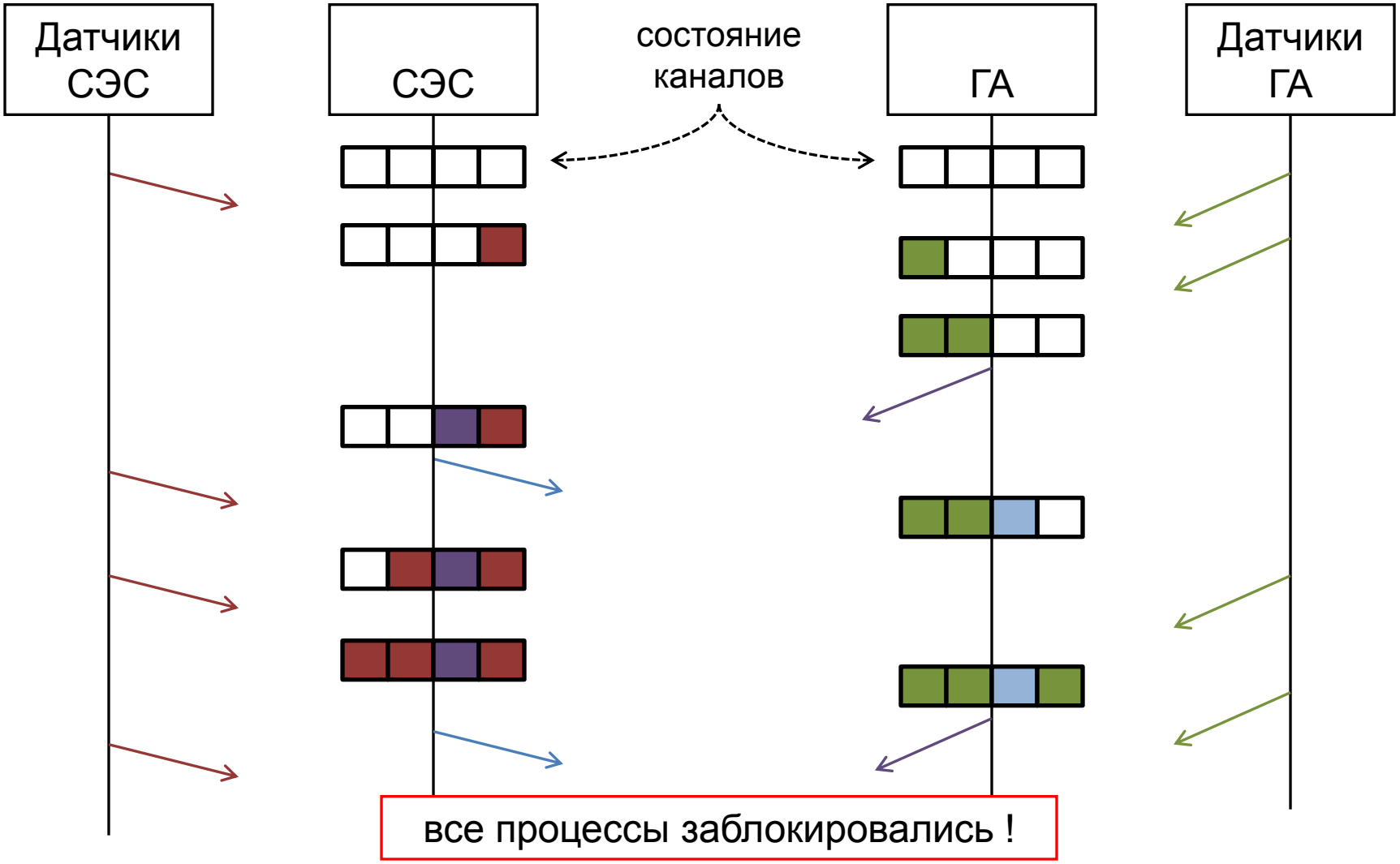
$$Fq \Rightarrow (!qUp)$$

- q – выход ГА из режима защиты
- p – нажатие кнопки СБРОС

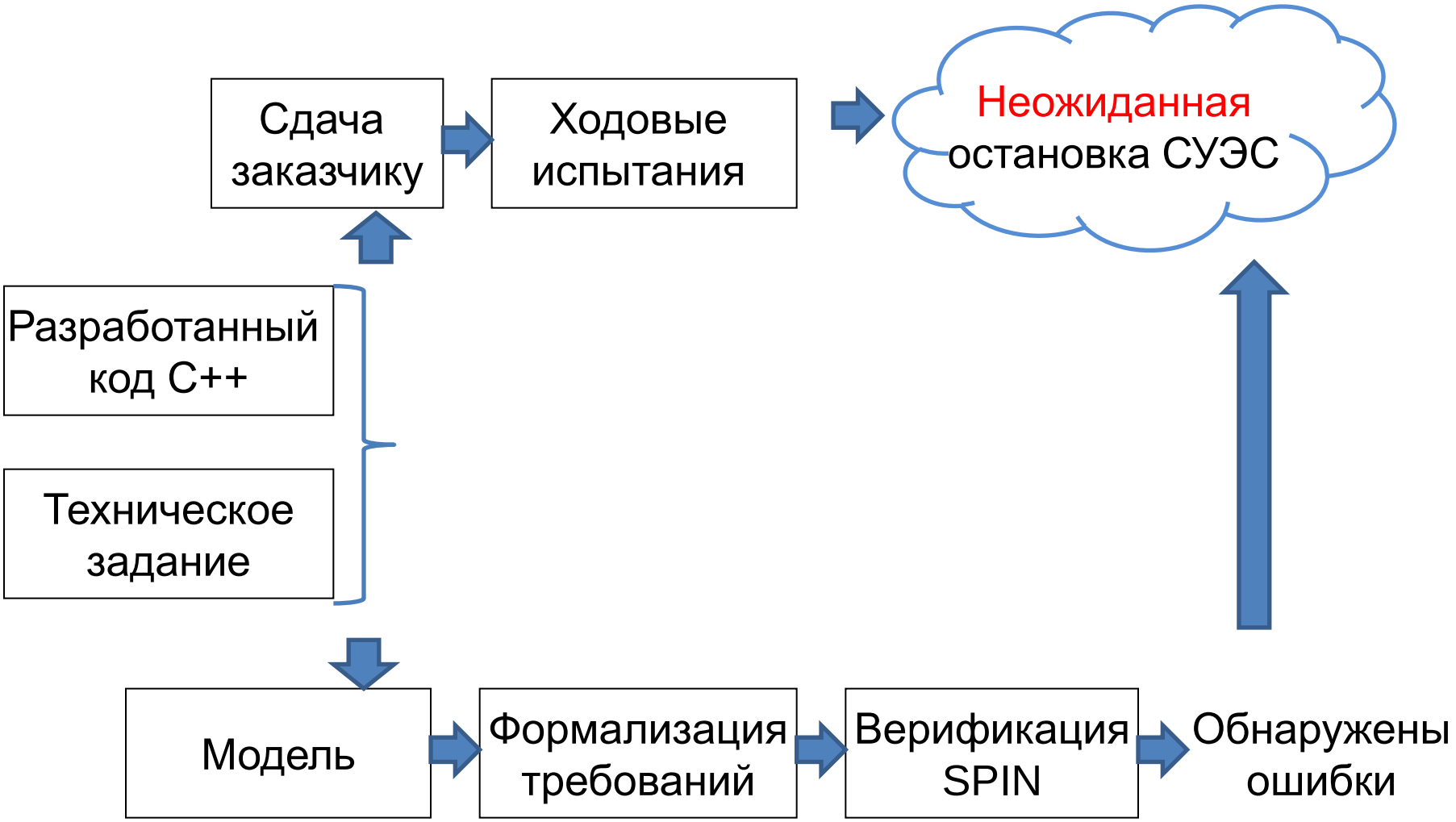
Свойство нарушается!

Простейшее исправление ошибки – удаление перехода
Масштабируемое решение – добавление дополнительного переходного режима

Базовое свойство: отсутствие блокировок



Верификация СУЭС



Результаты проекта

- При моделировании:
  - найден избыточный код, мертвый код, структурные несоответствия
  - исправлены в модели
- При верификации:
  - найдены нарушения базовых свойств распределенных систем (блокировки)
  - найдены нарушения требований технического задания
  - ошибки подтверждены трассами кода и испытаниями
  - в модели ошибки исправлены

Найдено около 10 ошибок, большинство из них редкие, не были обнаружены при тестировании

Обнаружение ошибок в программах до сих пор является научным результатом
Выводы подтверждают результаты Rockwell Collins

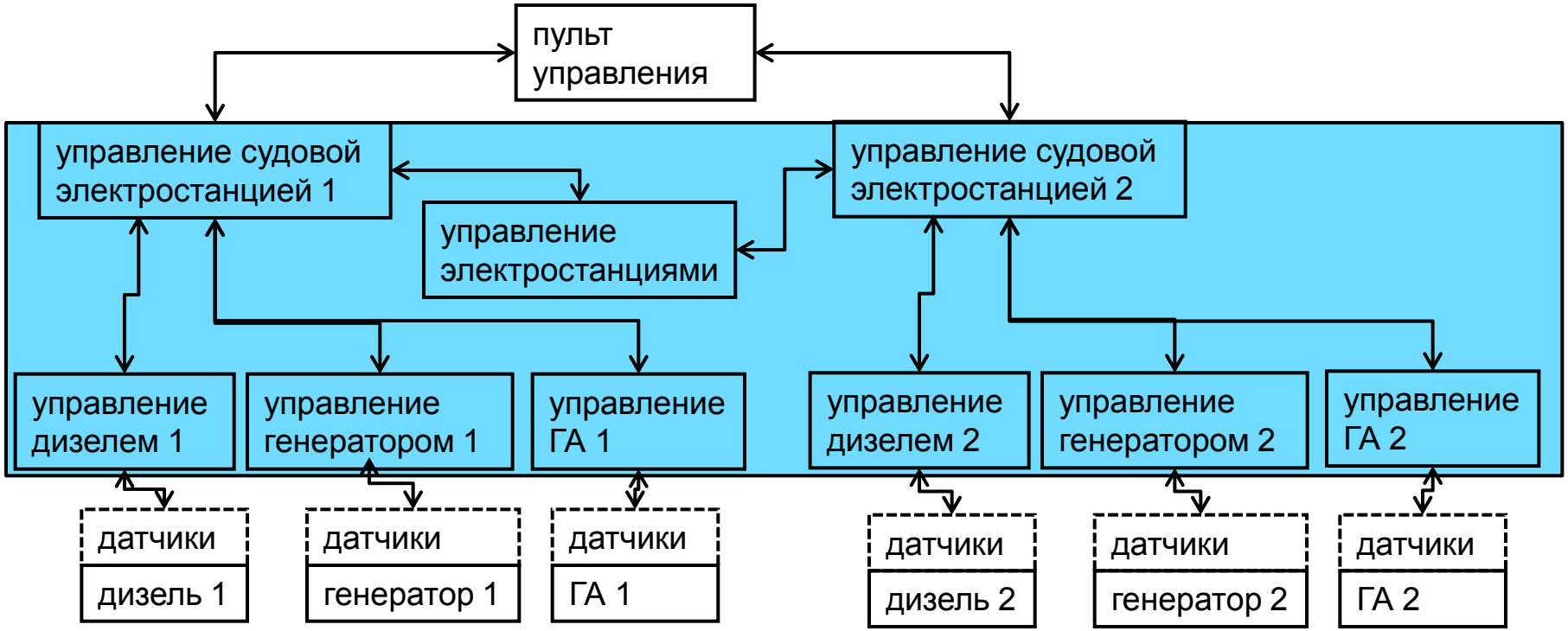
Извлеченные уроки

Согласно теоретическим результатам для любых реактивных систем с конечным числом состояний можно построить структуру Крипке и верифицировать любую формулу, заданную LTL

Однако:

1. Размер практических задач достаточно велик. Модель не может быть верифицирована целиком. Выход -> верификация частей моделей – композициональная верификация
2. На практике для LTL формул определенной длины соответствующий автомат Бюхи не строится

Композициональная верификация

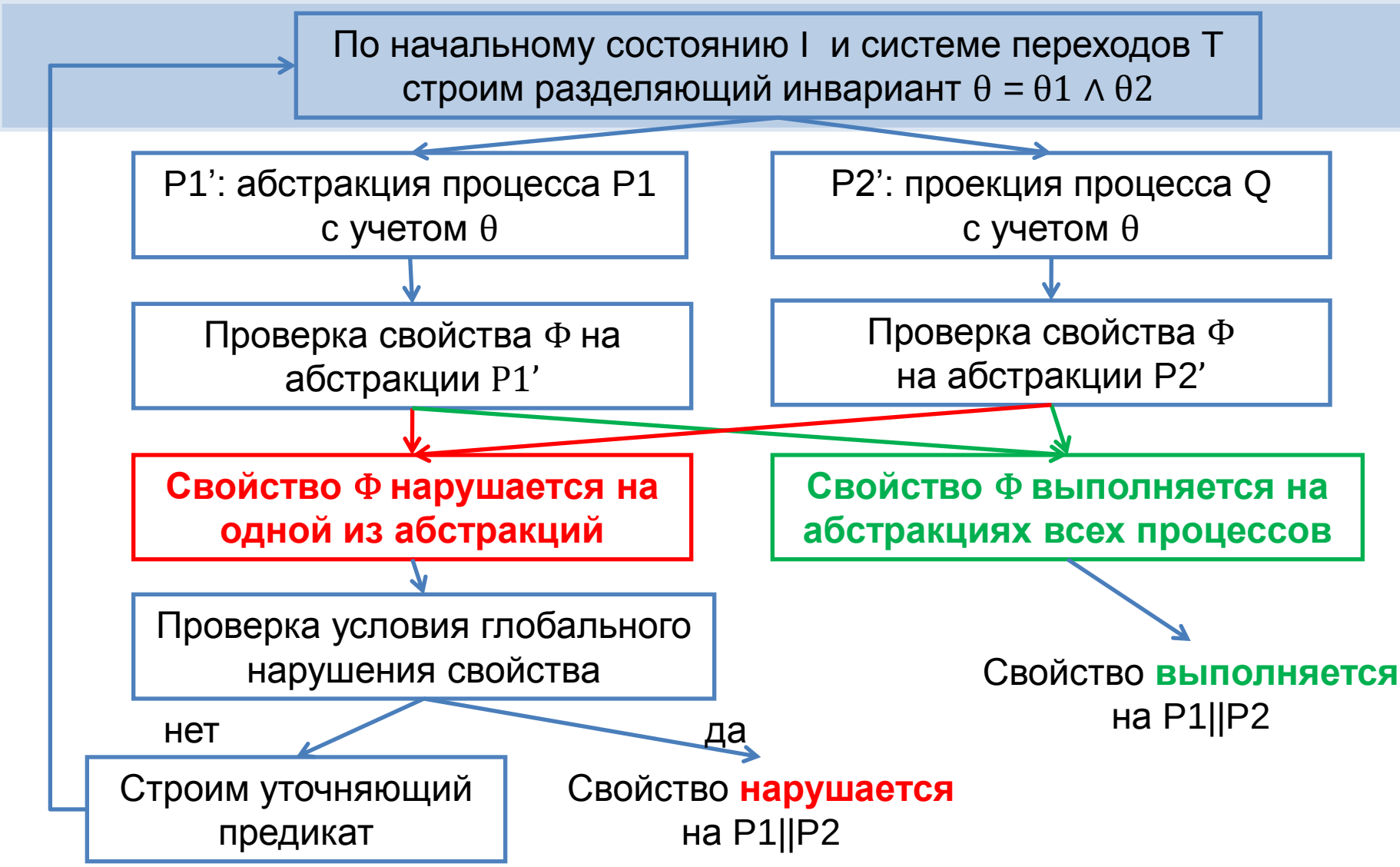


- 9 основных модулей, 28 состояний у каждого модуля, 9 асинхронных каналов глубины 6, 76 форматов сообщений + асинхронная композиция процессов
- Количество состояний системы переходов > 2^{400}

Композиционная верификация
$\Rightarrow$ разбить модель на части
$\Rightarrow$ верифицировать отдельные модули

Алгоритм верификации на основе локальных доказательств

Алгоритм верификации на основе локальных доказательств



Пример СУЭС: реакция на нажатие кнопки СБРОС защиты

Всегда, по нажатию кнопки СБРОС защиты когда-нибудь в будущем система энергоснабжения станет активна

$$G(q \rightarrow Fr)$$

Выполняется ли это требование (на корректной модели)?

Нет. *Существует поведение системы, когда датчик генераторного автомата с какого-то момента всегда будет в противоречивом состоянии*

- Но мы формулировали свойство, мысленно исключая подобную трассу! Добавим ограничение справедливости

$$GFp_1 \rightarrow G(q \rightarrow Fr)$$

И в этой формулировке свойства не выполняется

потому, что существуют несправедливые трассы относительно других параметров СУЭС: температуры дизеля, давления масла в дизеле, датчиков защиты генераторного автомата и т.д.

Длина формулы растет

Свойство с ограничениями справедливости

В общем случае, размер автомата Бюхи, строящегося по LTL формуле, зависит от ее длины экспоненциально

$$(GFp_1 \wedge \ldots \wedge GFp_n) \rightarrow G(q \rightarrow Fr)$$

- Для указанного свойства выхода из состояния защиты СУЭС n=10
- SPIN не справляется с подобными формулами n>8
- SPIN. При n=8 процесс never claim, соответствующий данной формуле, генерируется за 3 часа

Использовать другой алгоритм генерации автомата Бюхи по LTL формуле

- Один из наиболее эффективных– алгоритм на основе *альтернирующих автоматов* (по числу состояний результирующего АБ)
- Мы использовали характеристические булевы функции при разработке своего алгоритма

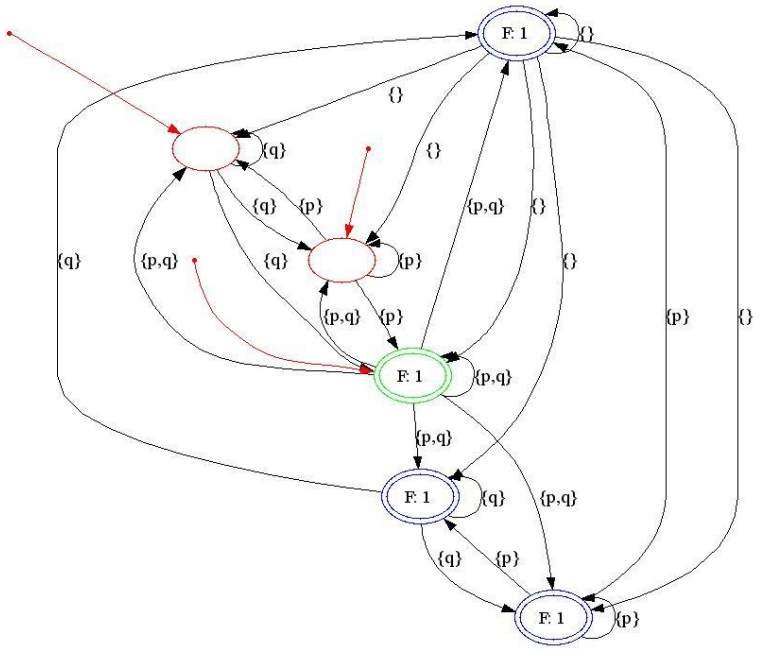
Альтернирующие автоматы над бесконечными словами

- Альтернирующий автомат : $A = (Q, \Sigma, I, \delta, F)$

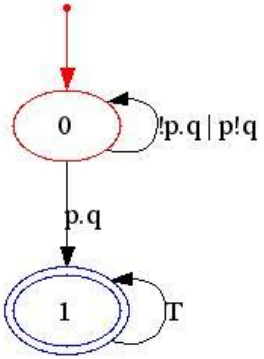
  Q – конечное множество состояний;
  Σ - конечное множество – входной алфавит
  $I \subseteq Q$ - конечное множество начальных состояний
  $\delta \subseteq Q \times \Sigma \times B^+(Q)$ – отношение перехода
  $F \subseteq Q$ – множество *принимающих* состояний

- δ – переходы с экзистенциальным выбором (ИЛИ) и переходы с универсальным выбором (И)
- Вычисление автомата A над бесконечным словом $w=a_0a_1... \in \Sigma^\omega$ - это граф
- Вычисление ρ допускается, iff для любой бесконечной цепочки β, принадлежащей вычислению ρ, $(\exists q \in F)\ q_i=q$ для бесконечного числа $i \in N$ (т.е. $\inf(\beta) \cap F \neq \varnothing$)
- Язык $L_A \subseteq \Sigma^\omega$ - множество бесконечных слов, для которых $\exists$ допускаемое вычисление ρ

Сравнение алгоритмов генерации автомата Бюхи по LTL формуле $(p \vee q)U(p \wedge q)$



По алгоритму на основе атомов



По алгоритму на основе альтернирующих автоматов

Результаты сравнения реализации символьного алгоритма с SPIN, LTL2BA

Классы LTL формул

$\neg[(GFp_1 \wedge \ldots \wedge GFp_n) \rightarrow G(q \rightarrow Fr)]$ $\neg(p_1 U(p_2 U \ldots (p_{n-1} U p_n)\ldots)$

n	SPIN	LTL2BA	Символьный алгоритм
2	0.19	<0.01	<0.01
3	4	<0.01	<0.01
4	155	<0.01	<0.01
5	4607	0.05	0.03
6	5232	0.57	0.11
7	8113	4	0.33
8	11212	45	2
9	-	375	11
10	-	4500	16

n	SPIN	LTL2BA	Символьный алгоритм
2	0.02	<0.01	<0.01
3	0.03	<0.01	<0.01
4	0.17	<0.01	<0.01
5	1.23	<0.01	<0.01
6	38	0.02	<0.01
7	127	1.15	0.02
8	-	150	0.03
9	-	4200	0.17
10	-	-	0.63

Заключение

- Model Checking многообещающий метод повышения качества программ
- Проектирование программ по модели согласуется с инженерной парадигмой и позволяет естественно использовать Model Checking
- Проведенные проект даже без значительных доработок показал эффективность Model Checking
  - Model Checking можно и нужно применять
- Практическая задача высветила направления работы, которые позволяют приблизить Model Checking к проектированию встроенных бортовых систем управления
- Получены результаты по композициональной верификации и эффективному построению автоматов Бюхи

Многие иностранные компании провели проекты по внедрению верификации в корпоративный процесс проектирования

СПАСИБО ЗА ВНИМАНИЕ

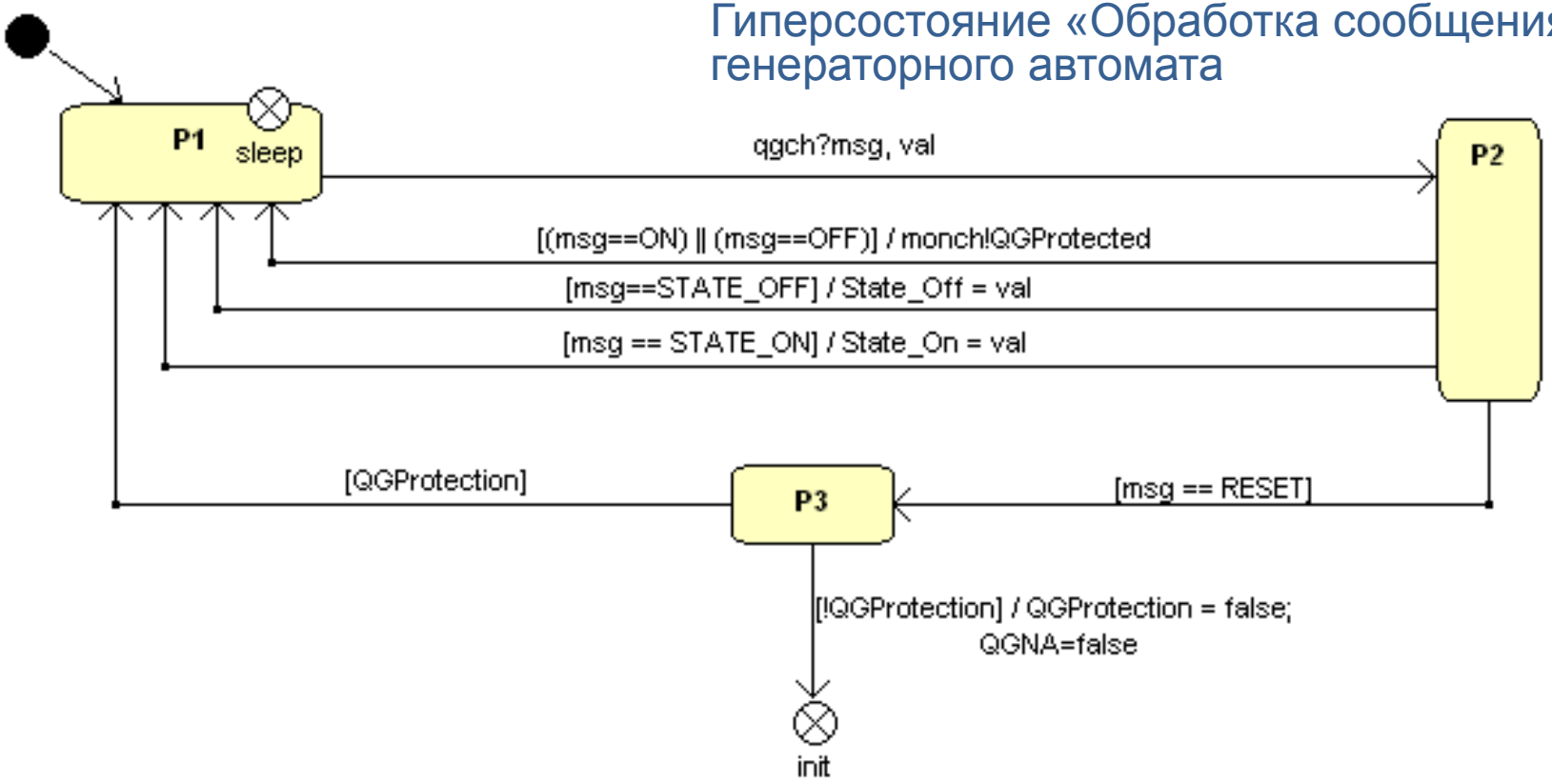
ОТВЕТЫ НА НЕКОТОРЫЕ ВОПРОСЫ

Моделируются ли ограничения UML state machine

См. следующий слайд

Пометки на переходах

Гиперсостояние «Обработка сообщения» генераторного автомата



Переходы могут быть помечены:

- событие[сторожевое условие]/набор действий
- набор действий – или присваивание значения переменной, или отправка сообщения

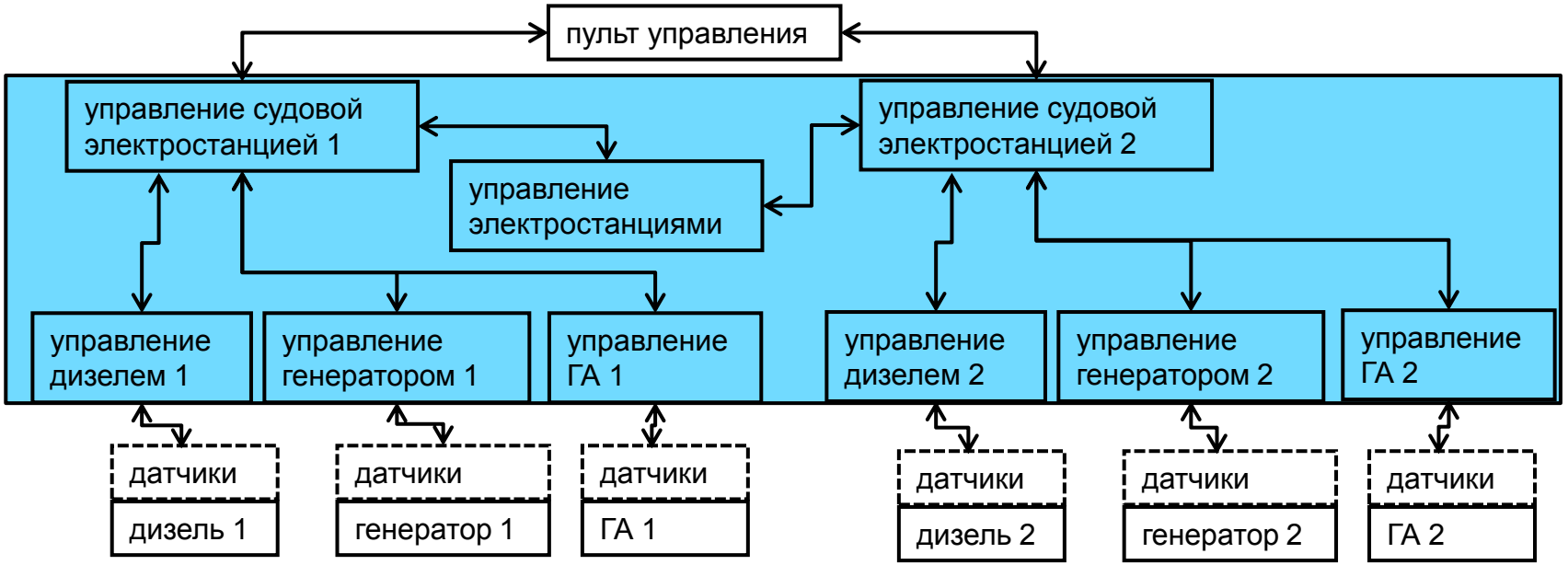
Количество строк кода в исходной программе СУЭС на C++

- Около 18000 строк кода

Была ли исходная система управления энергоснабжением судна асинхронной (сл.29)

следующий слайд

Структура системы управления энергоснабжением судна



Реализация:

- 9 основных модулей
- каждый модуль выполняется в отдельном процессе
- процессы параллельны и работают асинхронно
- 28 состояний у каждого модуля
- 9 асинхронных каналов конечной емкости

Модель:

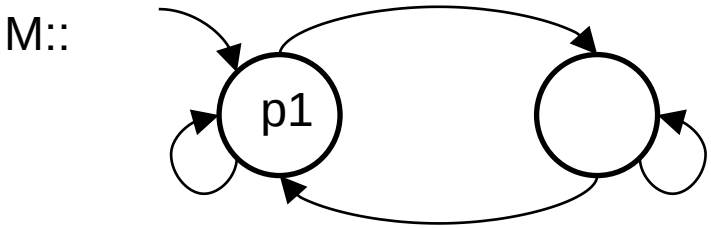
- каждый модуль моделируется системой с конечным числом состояний
- каждый модуль в отдельном процессе
- процессы параллельны и работают асинхронно
- взаимодействие по асинхронным каналам конечной емкости

Зачем нужны условия сильной справедливости (сл.42)

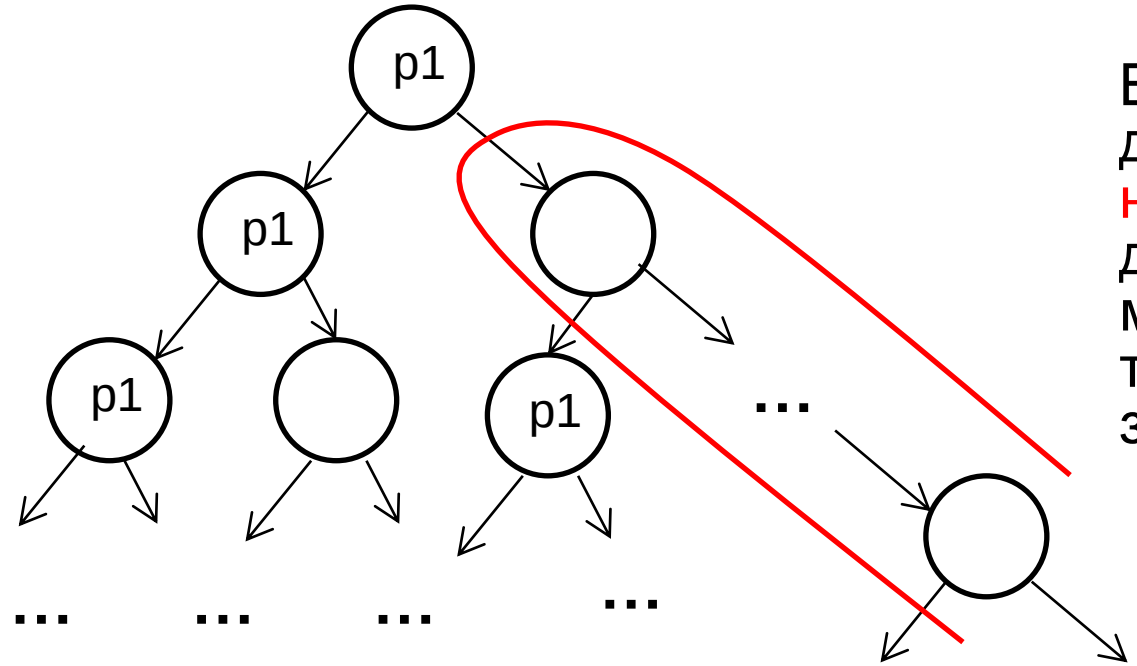
следующий слайд

Пример СУЭС: реакция на нажатие кнопки СБРОС защиты

- Поведение среды моделируется недетерминировано
  - т.е. предположения об алгоритме ее поведения не делаются



- M – модель датчика непротиворечивого состояния ГА
- p1 = true, если датчик в непротиворечивом состоянии; p1= false, иначе



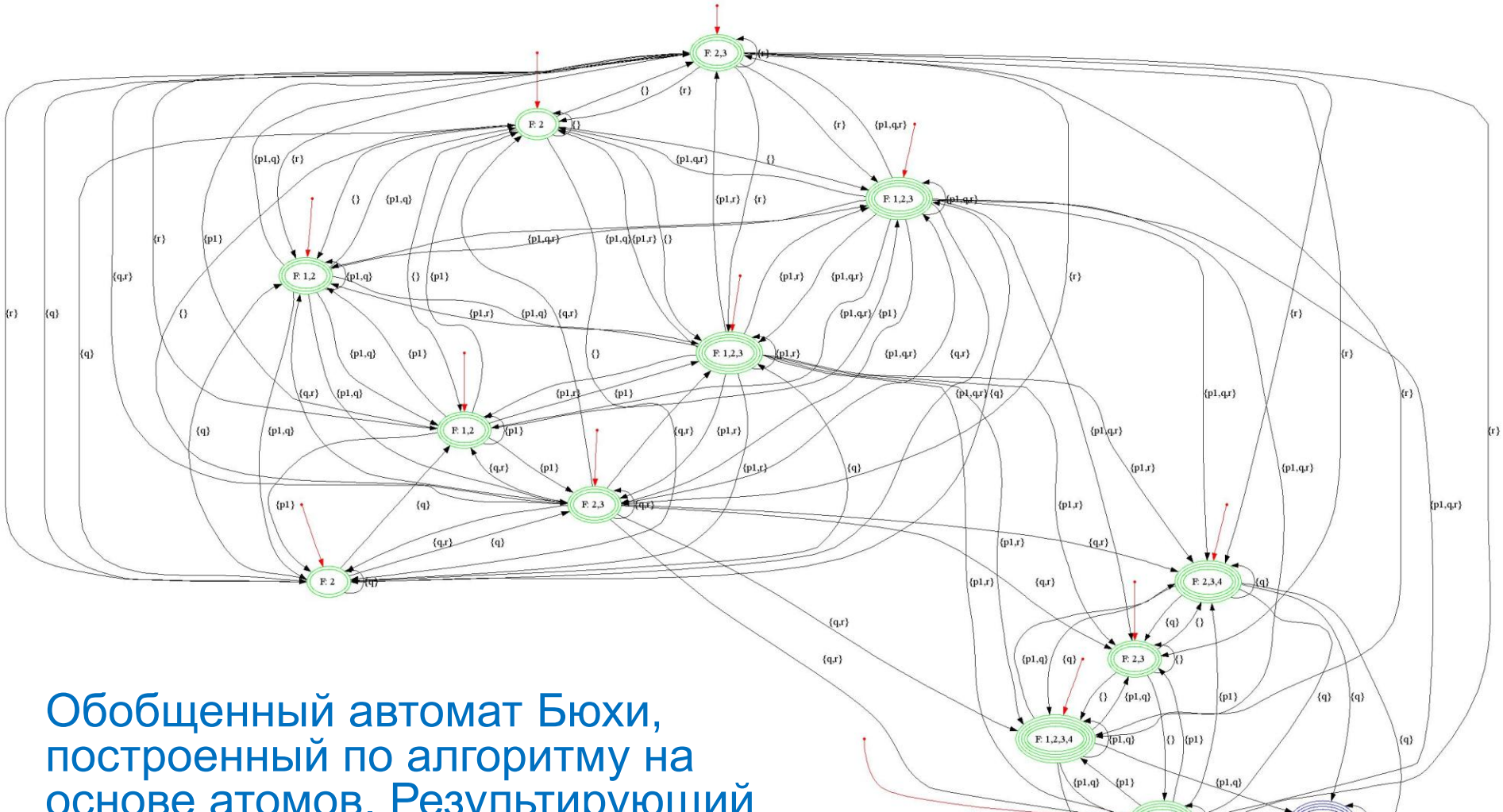
В поведении модели датчика M есть несправедливые трассы : датчик с некоторого момента всегда принимает только противоречивое значение

Сравнение алгоритмов генерации автомата Бюхи по числу состояний

для LTL формулы $\neg \left[GFp_1 \rightarrow G(q \rightarrow Fr) \right]$ (сл.42,45)

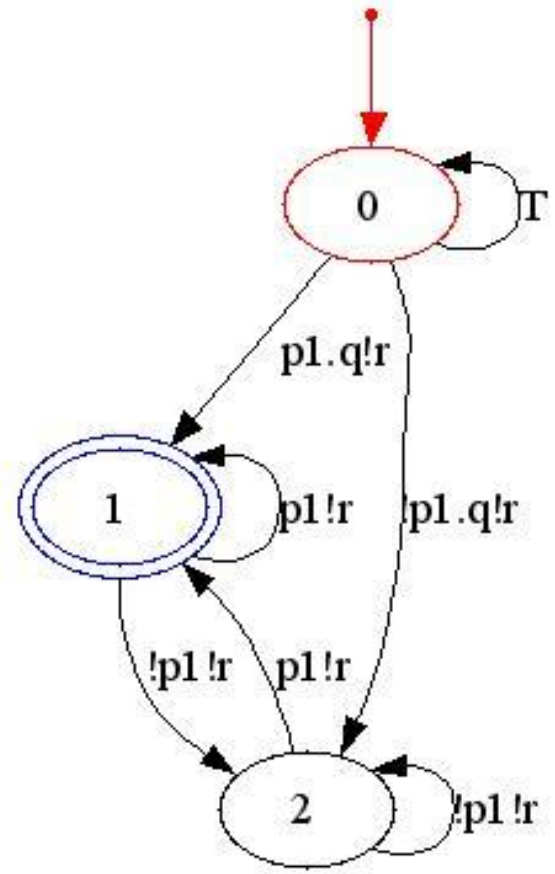
следующий слайд

Сравнение алгоритмов генерации автомата Бюхи по числу состояний для LTL формулы $\neg \left[GFp_1 \rightarrow G(q \rightarrow Fr) \right]$



Обобщенный автомат Бюхи, построенный по алгоритму на основе атомов. Результирующий

Сравнение алгоритмов генерации автомата Бюхи по числу состояний для LTL формулы $\neg [GFp_1 \rightarrow G(q \rightarrow Fr)]$



Автомат Бюхи по алгоритму на основе альтернирующих автоматов

Задается ли разделяющий инвариант вручную (сл.40,41)

следующий слайд

Разделяющий инвариант

- $\varphi = \varphi(s)$ – предикат состояния (или assertion)
- φ – *инвариант программы P*, если он выполняется во всех достижимых состояниях P
- предикат состояния ξ – *индуктивный инвариант программы P*, если $I \Rightarrow \xi$

$$\xi \Rightarrow wlp(T, \xi)$$

- θ_i – *локальный assertion программы* P_i, если он определен только относительно переменных V_i
- вектор $\{\theta_i\}$ – *разделяющий assertion программы* P
- $\theta = \bigwedge \theta_i$ – *разделяющий инвариант программы P*, если θ – индуктивный инвариант P

Th. [Namjoshi] Наиболее сильный разделяющий инвариант θ может быть найден, как минимальная неподвижная точка уравнений $\theta_i \equiv \exists L \setminus L_i : I \vee (\vee_j sp(\hat{T}_j, \theta))$

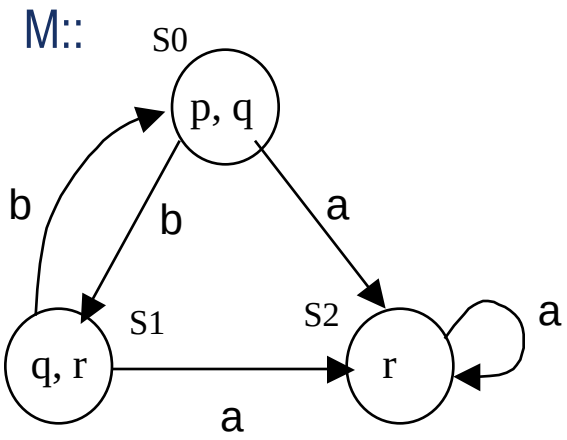
- Pi - асинхронные процессы P
- I – начальное условие
- T, Ti – отношения переходов

Что такое Model Checking (сл.6)

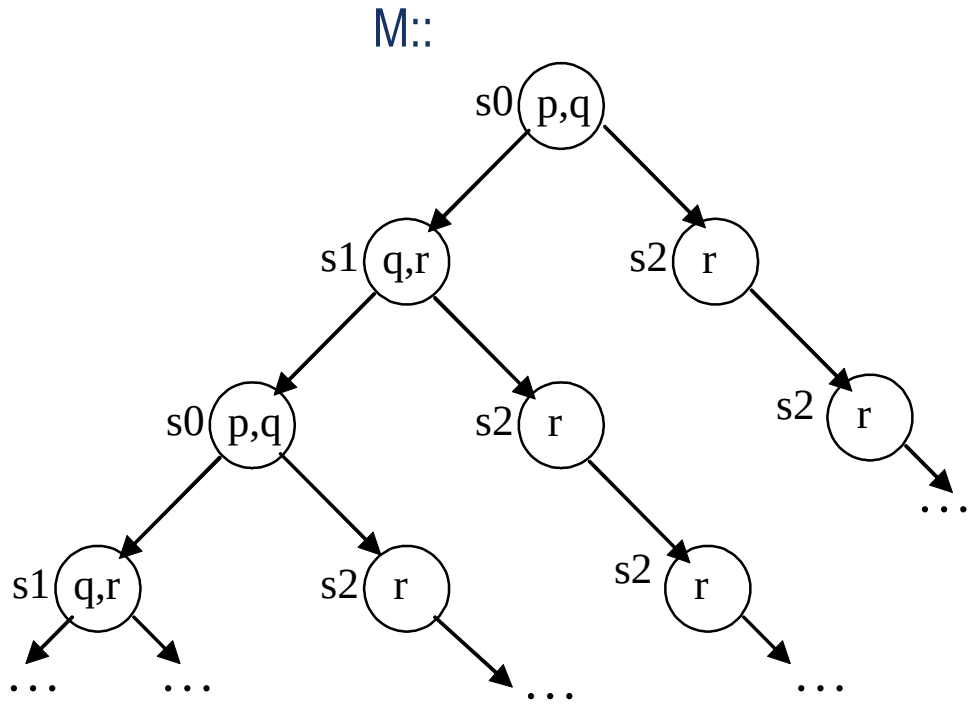
следующий слайд

Проверка модели – Model Checking

Тип систем: реагирующие системы (reactive systems)



Свойства вычислений описываются темпоральные логики φ=FGr



Разработаны методы проверки модели для разных темпоральных логик

Дана формула φ языка L, и система M. Требуется определить, будет ли φ выполняться на M, т.е. будет ли M МОДЕЛЬЮ φ?

Виды темпоральных логик

следующие слайды

LTL и CTL – подклассы CTL*

В LTL - формулы пути, которые должны выполняться для всех вычислений, т.е. предваряются квантором пути A

В CTL каждый темпоральный оператор предваряется квантором пути A или E

Формулы LTL:

AG(p ⇒**F** q)

A (¬a∨ **G**b & (a**U** ¬c))

A (a **U** ¬b)

Формулы CTL:

AG(p&¬**EF**(q⇒r))

EF(a & **E**(a**U** ¬c))

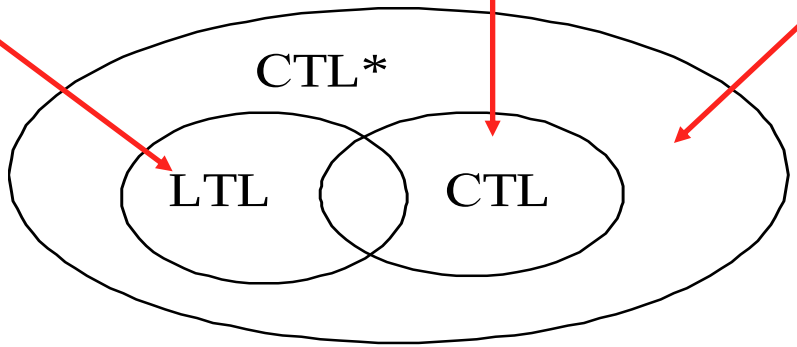
A (a **U** ¬ b)

Формулы CTL*:

E(¬p & **X A F** q)

EX (a & **AX**(b**U**c)]

A (a **U** ¬ (**F** b))



Вероятностная CTL – PCTL (Hansson & Jonsson'94)

PCTL (Probabilistic CTL) заменяет кванторы E и A в CTL вероятностным оператором $Pr_{\sim p}(\alpha)$, где $p \in [0,1]$, $\sim \in \{\leq, <, \geq, >\}$, например, $P_{>0.3}$ (Fq)

Формула состояния: $\varphi ::= q \mid \varphi_1 \vee \varphi_2 \mid \neg \varphi \mid P_{\sim p}(\alpha)$

где α -*формула пути:* $\alpha ::= X \varphi \mid \varphi_1 U \varphi_2$

Операторы F и G выражаются через *Until:* $F\varphi = True\ U\ \varphi$, $G\varphi = \neg F \neg \varphi$

Семантика вероятностного оператора: (α - формула пути, s – произвольное состояние, $Path_s$ – все пути из состояния s, σ – путь)

$s \models P_{\sim p}(\alpha)$ iff $Pr\{\sigma \in Paths_s \mid \sigma \models \alpha\} \sim p$

В состоянии s вероятностная мера ~p выполняется для формулы пути α, iff с этой мерой может быть выбран путь из состояния s, на котором выполняется формула α

Примеры:

$P_{>0.7}(q\ U \neg r)$ – вероятность того, что на путях из данного состояния выполнится формула пути (q U ¬r), больше 0.7

$P_{< 0.1}$ ($(P_{>0.2}\ Xq)\ U \neg r$) вероятность того, что на путях из данного состояния выполнится формула пути $(P_{>0.2}\ Xq)\ U \neg r$, меньше 0.1

Timed Temporal Logic – структура формул

TCTL – Timed CTL – естественное расширение операторов U, F, ... логики CTL количественной информацией.

Грамматика TCTL (= CTL + Time):

$\phi ::= p \mid \alpha \mid \neg\phi \mid \phi \wedge \phi \mid z \text{ in } \phi \mid E[\phi\, U\, \phi] \mid A[\phi\, U\, \phi]$

p – атомарный предикат
α - ограничение на таймеры и формульные часы
z – формульные часы
z in ϕ - введение новых часов в формулу ϕ
E [ϕ U ϕ], A [ϕ U ϕ] – как в CTL

Обозначение: $E[\phi\, U_{\alpha}\psi] \equiv z \text{ in } E[(\phi \,\&\, \alpha)\, U\, \psi]$

Выводимые операторы $EF_{\alpha}\phi \equiv E[\text{True}\ U_{\alpha}\ \phi]$ и т.д.

Формулы TCTL включают $E[\phi\, U_{\sim k}\psi]$, $A[\phi\, U_{\sim k}\psi]$, $EF_{\sim k}\phi$, $EG_{\sim k}\phi$, $AF_{\sim k}\phi$, $AG_{\sim k}\phi$
где ~ - любой символ из {<,≤,=, ≥, >} и k – рациональное число

Для верификации временного автомата A можно анализировать R(A)

- Подробнее о Model Checking и темпоральных логиках можно найти в книге Ю.Г. Карпова «Model Checking. Верификация параллельных и распределенных программных систем», 2010