

Верификация параллельных программных и аппаратных систем



Курс лекций

Карпов Юрий Глебович
профессор, д.т.н., зав.кафедрой
“Распределенные вычисления и компьютерные сети”
Санкт-Петербургского политехнического университета

karpov@dcn.infos.ru



Лекция 1

Введение



Проблема

- Информационные технологии проникли во все сферы жизни современного общества. Интенсивно развивается направление *ubiquitous computing* (всепроникающие вычисления)
- Наступила пора параллельного программирования. Простые последовательные программы - очень узкое применение
- Нетривиальные параллельные программы непостижимы для человеческого мозга: они очень часто неправильны, содержат ошибки
- Разработчик параллельной программы должен контролировать не последовательности возможных событий, как в последовательных программах, а все комбинации частично упорядоченных событий.
- Параллельные системы часто работают правильно “почти всегда”, они годами могут сохранять тонкие ошибки, проявляющиеся очень редко
- Большие программы после их разработки всегда содержат ошибки. По оценкам Microsoft, ОС Windows 95 имеет около 5000 ошибок.



Ошибки в системах и их последствия

- INTEL: Микропроцессоры содержат ~5 М переходов. В 1994 выпущен чип с ошибкой. Замена *миллионов* дефектных процессоров \Rightarrow потери ~\$500 М
- 4.06.96 ракета *Ариан 5* (аналог Протона) взорвалась через 39 с. - ошибка переполнения при преобразовании 64-битового вещественного числа в 16-битовое целое. Ущерб > \$600 млн.
- Конец 80-х: Therac-25 прибор лучевой терапии. Пациенты получили передозировку, двое умерли, несколько стали инвалидами
- Война в Ираке, 23.03.2003. Ошибка в программе \Rightarrow система Patriot определила свой бомбардировщик Tornado как приближающуюся ракету. До 24% потерь в живой силе в \perp Иракской войне – *"Friendly Fire"*
- Boeing 757, 1995 г (рейс из Майами в Кали, Колумбия). Ошибка в одном символе в Flight Management System привела к катастрофе. Погибли 159 чел
- Связь с советской АМС "Фобос-1" прервалась 2.09.88 г. из-за ошибочной команды, посланной с Земли. АМС потеряна, где-то летает сама по себе. То же с "Фобос-2"



Примеры недавних ошибок

- 09.09.2009. Сообщение в СМИ. Компания Volvo объявила об отзыве по всему миру 26 тысяч автомобилей
 - У них обнаружена **неисправность в блоке управлением двигателем**. Из-за этого дефекта мотор может не запуститься или заглухнуть после того, как автомобиль проедет несколько сотен метров
 - Все владельцы неисправных машин получают 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. Конкурс на выполнение работ производился в 2005 г., его выиграла петербургская НПФ «Ракурс»
 - Кутьин: у его ведомства есть ряд вопросов: **«как была спроектирована система управления, какие алгоритмы были заданы, ...»**
 - Комментарий блоггера (gazeta.ru): *“Вот они, наши хваленые программисты и быдлокодеры: ни строчки кода без бага”*



Примеры недавних ошибок

- 02.2010. Toyota отозвала с рынков >8 млн проданных автомобилей из-за неисправности педали газа и дефекта ПО тормозной системы. Это обойдется ей в \$2 млрд. Глюк автомобилей Toyota стал причиной не менее 5 смертей.
- Этот инцидент подрывает доверие к торговой марке Тойоты на многие годы. Акио Тойода, президент компании Тойота, 24.02.2010 выступил в Конгрессе США и принес извинения за проблемы, связанные с качеством автомобилей
- По мнению экспертов, проблемы Toyota вызваны переходом автомобильной индустрии на электронные системы **даже в критически важных функциональных узлах**
 - Эксперты отмечают, что основная **проблема именно в компьютерной системе управления**, в которой не было предусмотрено экстренное торможение: педаль тормоза просто не срабатывала, когда сенсор акселератора командовал автомобилю разгоняться.
 - Эта история поднимает новые **проблемы перед конструкторами и программистами, которые разрабатывают системы управления**. Ясно, что проблемы с электроникой и компьютерные глюки будут встречаться всё чаще

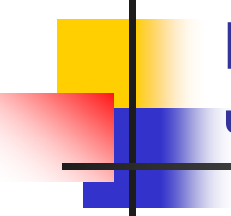


Panel Says Computer Error Felled Mars Orbiter

The New York Times

Published: April 14, 2007

- Although it circled Mars longer than any other spacecraft, the Mars Global Surveyor did not die a death of old age last November, a review board assembled by NASA announced yesterday.
- Instead, an errant computer command five months earlier had been placed in the wrong location of the computer memory for the spacecraft. That, in effect, implanted a fatal defect in the spacecraft, disabling a safety feature to prevent the solar panels from rotating too far and mangling its ability to communicate with Earth in case of a mishap.
- Last Nov. 2, a run-of-the-mill command to the Global Surveyor set off the latent time bomb. Because the safety feature had been disabled, the solar panels rotated as far as they could, causing the spacecraft to "think" erroneously that the panels had jammed.
- In its last 13-minute contact, the Global Surveyor reported numerous alarms to mission controllers but gave no indication that it was in immediate danger.
- As the spacecraft tried to recover, it ended up in an orientation such that the Sun was shining directly on a battery, causing it to overheat. The Global Surveyor misinterpreted that signal, sensing that it had overcharged the battery and stopped charging its other battery, as well.
- Meanwhile, because the June error caused the craft's antenna to point in the wrong direction, mission controllers on Earth could not get in touch with the craft again.
- "It was a sort of unfortunate concatenation of events," said Dolly Perkins, deputy director-technical of the NASA Goddard Space Flight Center in Greenbelt, Md., who was chairwoman of the review board.
- Within half a day, the Global Surveyor batteries drained dead.
- The chain of events leading to the craft's doom started in September 2005 on a routine technical update. For redundancy, the information was copied to two separate locations in the computer memory, but at different times with slightly different values. That led to a slight inconsistency, and when the engineers tried to repair that in June, they caused the bigger problem.
- The review board said that the mission team had properly followed its procedures, but that the procedures were inadequate and that more thorough checks of its commands could have found the errors before they were sent to the spacecraft.
- The Mars Global Surveyor was launched in 1996 and arrived at Mars 10 months later. After gradually moving into a circular orbit, its science mission started in March 1999 and was to last one Martian year, or 687 Earth days, until January 2001. But the spacecraft continued operating long after that, sending back more than 240,000 photographs.



Несет ли профессионал ответственность за
человеческие жизни?

Программирование является
единственной областью инженерной
деятельности,
где разработчик не отвечает
за качество своей работы

Обычно ПО имеет 10 ошибок на 1000 строк кода,
ПО высокого качества – 3 ошибки на 1000 строк кода

Современное ПО содержит миллионы строк кода,
уже сданные программы наполнены ошибками



Параллельные системы

Нетривиальные параллельные и распределенные системы
непостижимы для человеческого мозга

“Существует обширный печальный опыт того, что
параллельные программы упорно сопротивляются
серьезным усилиям по выявлению в них ошибок”

S. Owicki, L. Lamport “Proving liveness properties of concurrent programs”, ACM
TOPLAS, N4, 1982

Пример ошибки в параллельной системе

Process A::	Process B::
...	...
$x := x + a$	$x := x + b$
...	...

```
x = 0;  
a = $ 2000;  
b = $ 15;
```

Есть ли ошибка?

Могут ли быть ошибки в программе из одной строки???

A1. $x \rightarrow R_A$

B1. $x \rightarrow R_B$

A2. $R_A + a \rightarrow R_A$

B2. $R_B + b \rightarrow R_B$

A3. $R_A \rightarrow x$

B3. $R_B \rightarrow x'$

Чему равно x после выполнения процессов A и B?

1) \$ 2015

A1, A2, A3, B1, B2, B3

2) \$ 2000

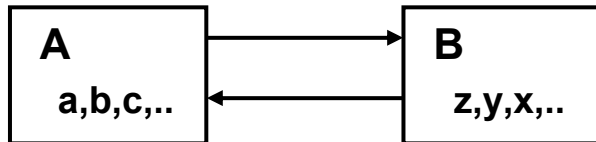
A1, A2, B1, B2, B3, A3

3) \$ 0015

B1, B2, A1, A2, A3, B3

Пример: протокол W.C.Lynch (1968)

Посимвольная полнодуплексная передача по телефонным линиям без потерь с *обнаруживаемыми* ошибками



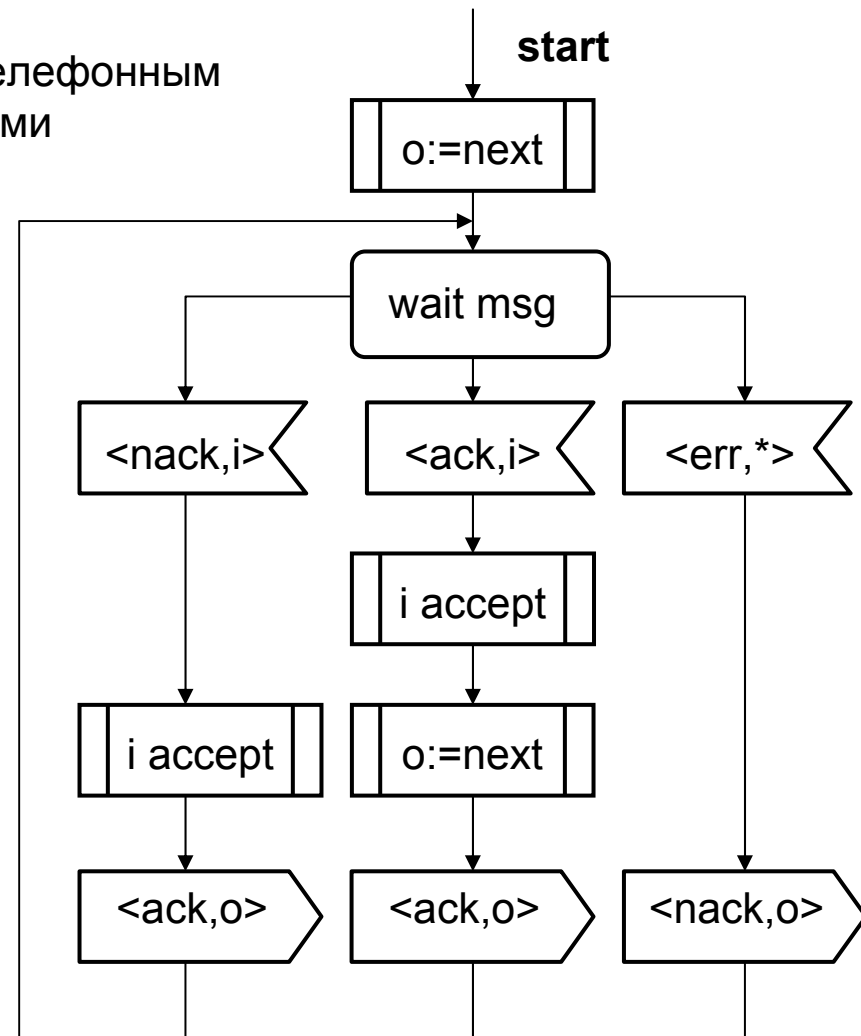
Сообщение: < C, X >

C - управляющий символ

X - информационный символ

Правила протокола:

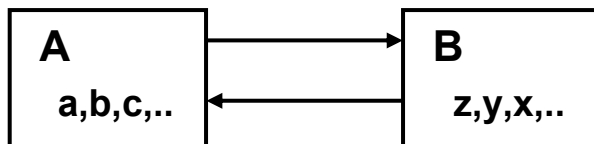
1. *Управляющий символ:* возвращаем
Если пришло без ошибок, то *ack*,
если пришло с ошибкой, то *nack*
2. *Информационный символ:* посылаем
Если получено *nack* или *err*, то
старый символ,
если нет - следующий символ



Корректен ли протокол Линча?

Некорректности протокола W.C.Lynch

Правила выглядят абсолютно логично, но протокол содержит ошибки!



Ошибки протокола Линча:

1. *Останов передачи:*

Если в одном направлении нечего передавать, то и в другом направлении передача останавливается (передавать пустую информацию);

2. *Проблемы инициализации:*

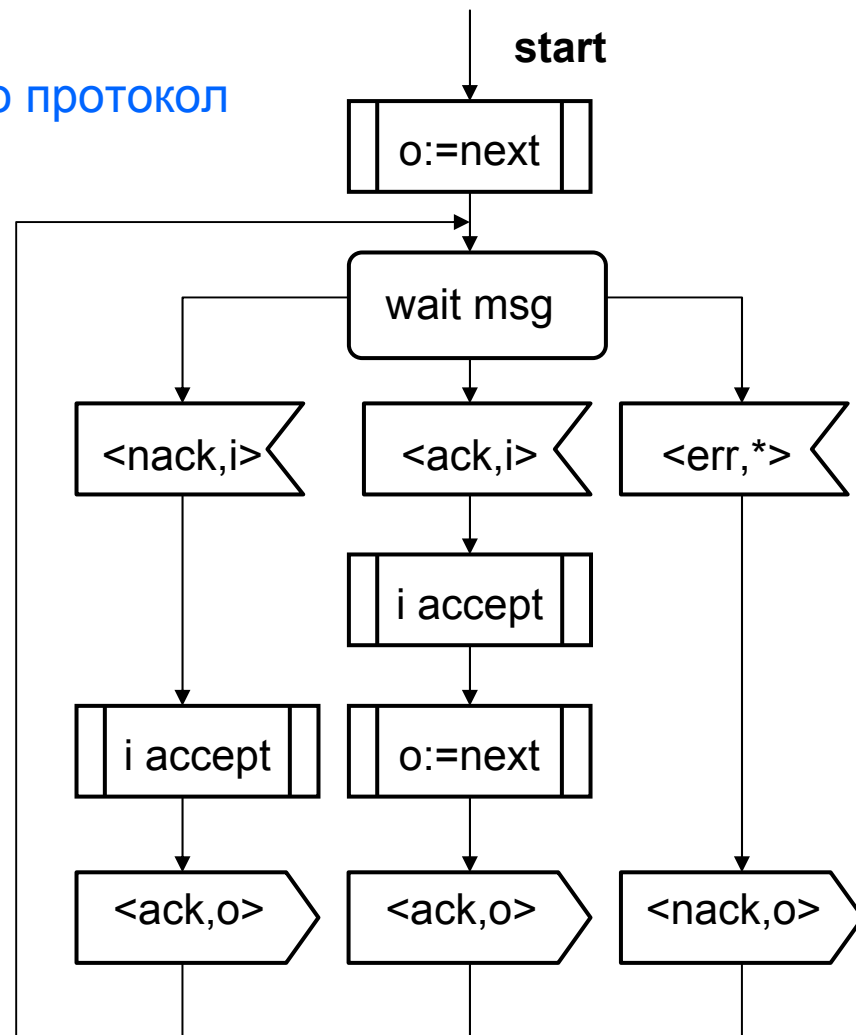
После запуска каждый ждет сообщения от партнера

3. *Проблемы с завершением:*

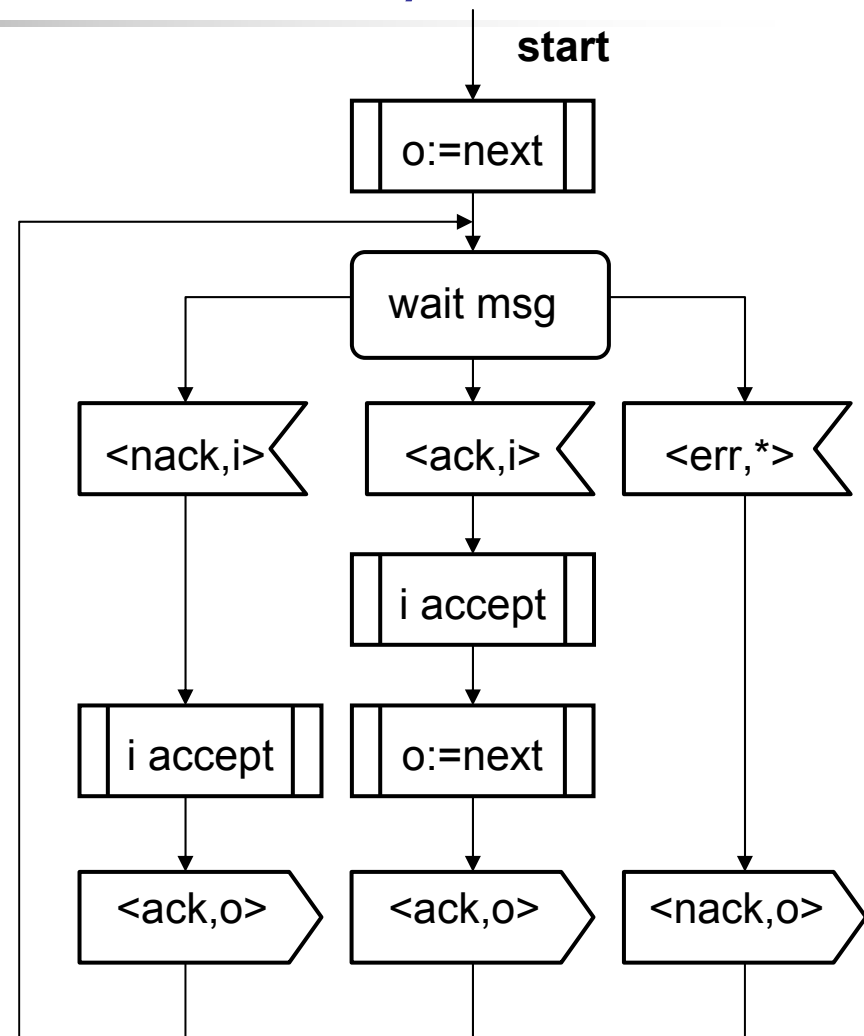
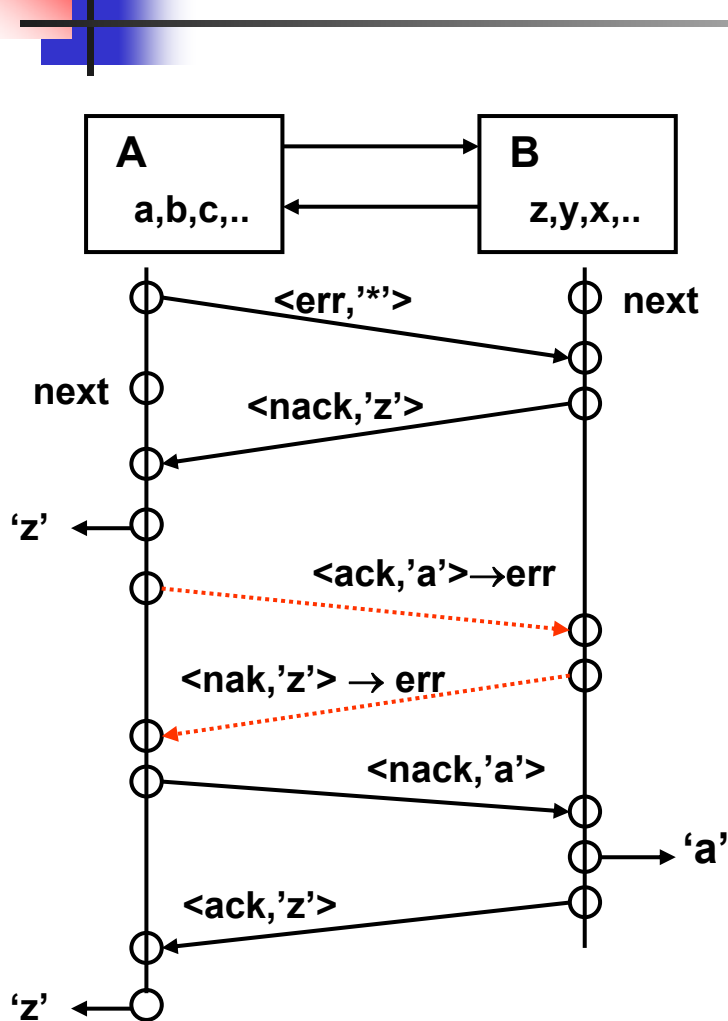
Как завершить сеанс?

4. *Логическая ошибка в организации передачи*

Какая?



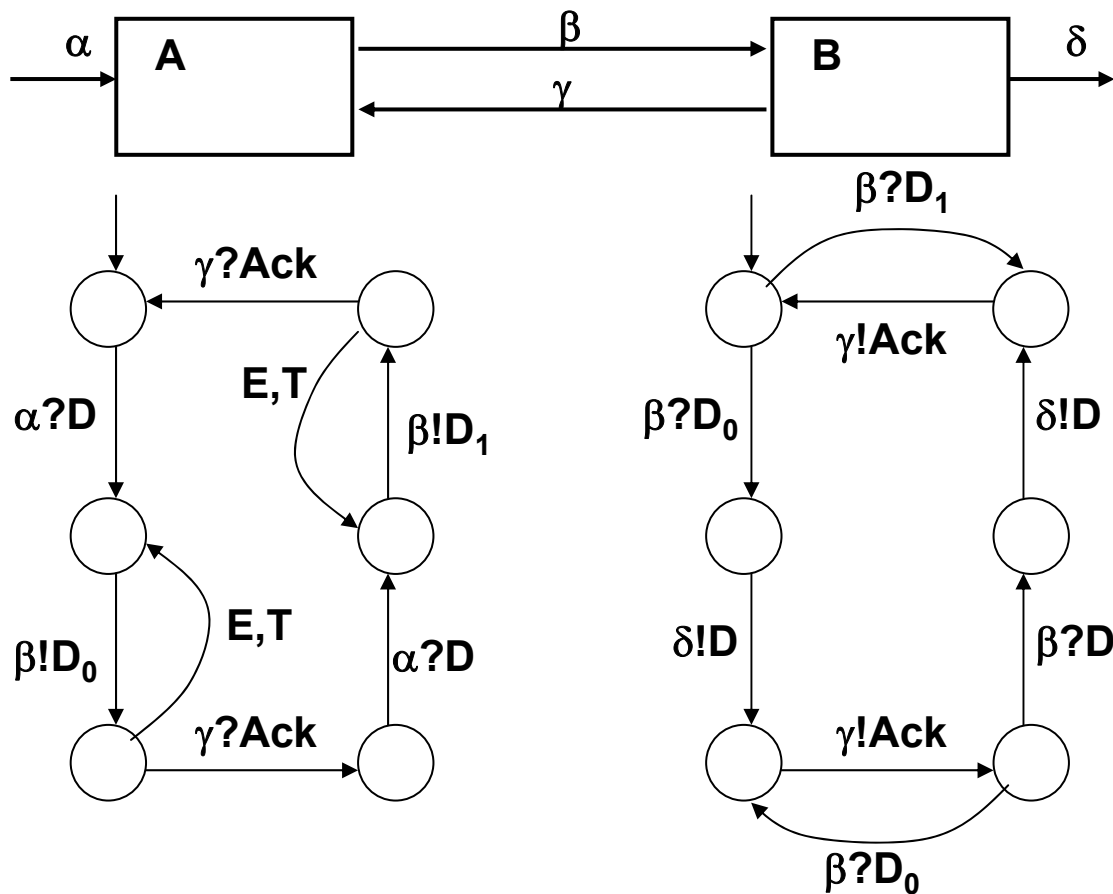
Логическая ошибка в протоколе W.C.Lynch



Редкая ошибка: повторная ошибка в двух противоположных направлениях в линии приводит к дублированию. Тестированием выявить нельзя

Пример ошибки в протоколе: протокол PAR

Передача с потерями и с обнаруживаемыми ошибками с однобитным окном: нумерованные сообщения и нумерованные подтверждения



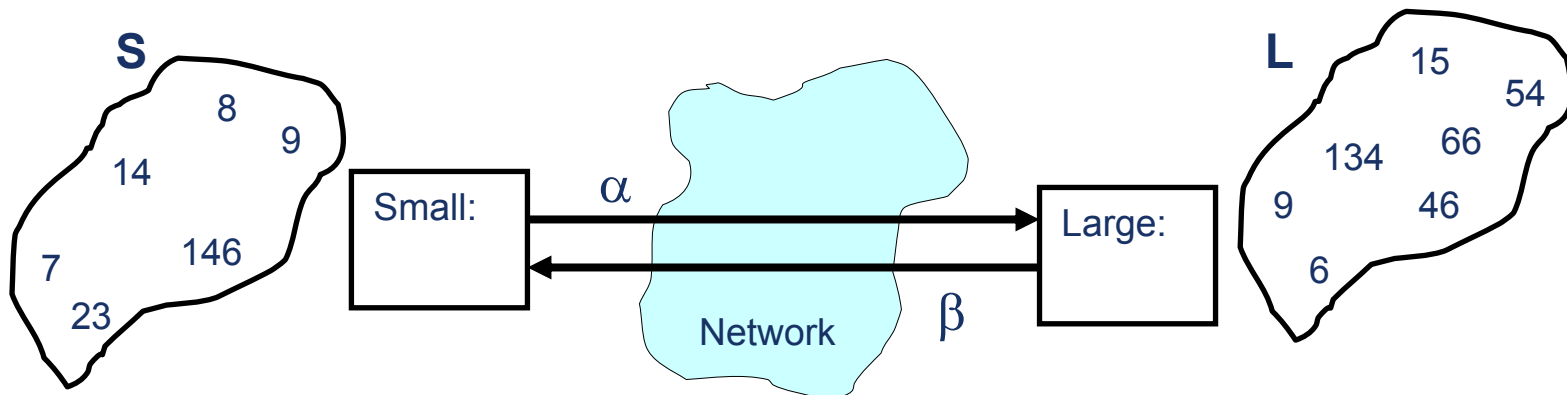
Упражнение: Найти ситуации, при которых происходят ошибки

Если использовать не только нумерацию сообщений, но и нумерацию подтверждений, то протокол становится корректным

Как ДОКАЗАТЬ?

Пример параллельной программы

Параллельная программа разделения множеств (Дейкстра):



▪Small::

```
mx := max(S);  $\alpha!$  mx; S := S - {max(S)};  
 $\beta?$  x; S := S  $\cup$  {x}; mx := max(S);  
while mx > x do {  
     $\alpha!$  mx; S := S - {max(S)};  
     $\beta?$  x; S := S  $\cup$  {x}; mx := max(S);  
}
```

▪Large::

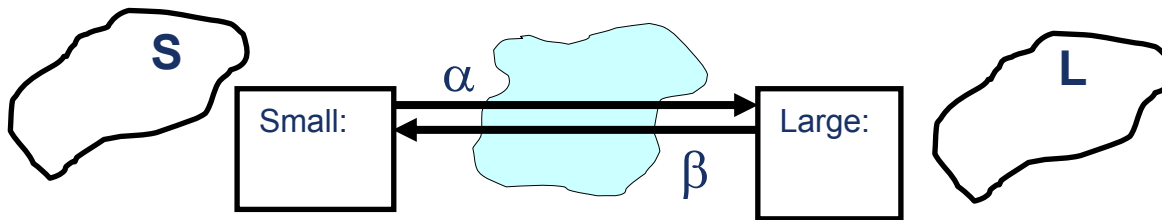
```
 $\alpha?$  y; L := L  $\cup$  {y}; mn := min(L);  
 $\beta!$  mn; L := L - {mn}; mn := min(L);  
while mn < y do {  
     $\alpha?$  y; L := L  $\cup$  {y}; mn := min(L);  
     $\beta!$  mn; L := L - {mn}; mn := min(L);  
}
```

Карпов Ю.Г. Анализ корректности параллельной программы разделения множеств // Программирование, 1996, N6,

Требования к программе

S^0 и L^0 – начальные значения множеств

S^{Term} и L^{Term} – заключительные их значения



Упражнение:

Найти ВСЕ ситуации, при которых происходят ошибки

C1. Объединение множеств не изменилось:

$$S^{\text{Term}} \cup L^{\text{Term}} = S^0 \cup L^0;$$

C2. Мощности множеств сохранились:

$$|S^{\text{Term}}| = |S^0|, |L^{\text{Term}}| = |L^0|;$$

C3. Элементы S^{Term} не больше элементов L^{Term} : $\max(S^{\text{Term}}) \leq \min(L^{\text{Term}})$.

- *Частичная корректность*: если множества S^0 и L^0 непусты и конечны, то *после завершения* процессов Small и Large условия C1, C2 и C3 выполняются
- *Полная (тотальная) корректность*: если множества S^0 и T^0 непусты и конечны, то *процессы Small и Large завершаются* и условия C1, C2 и C3 выполняются

Эта || программа частично корректна, но тотально некорректна



Ошибки в параллельных системах

- || системы обычно работают правильно “**почти всегда**”, они годами могут сохранять тонкие ошибки, проявляющиеся в редких, исключительных ситуациях
- Ошибки очень редко проявляются
- При повторных прогонах ошибки обычно не повторяются, нельзя поставить диагноз (“гонки”, race conditions)
- Ошибки в || программах проявляются при специфических сочетаниях времен. Трассирование программ при их отладке скрывает ошибки, зависящие от времени!
- Необходимы модели и методы, позволяющие **гарантировать** правильную работу программных и аппаратных систем во всех режимах

Любая программа, особенно параллельная, должна рассматриваться как некорректная до тех пор, пока не ДОКАЗАНО обратное

Важность проблемы валидации

- До 80% средств при разработке встроенного ПО тратится на валидацию – проверку соответствия ПО тому, что нужно потребителю
(*\$60 billion annually for US economy – данные 2005 г.*)
- Последствия оставшихся в системах ошибок:
 - Огромные материальные потери
 - Потери жизней
- КТО БУДЕТ ОТВЕЧАТЬ? Страховка?
 - низкая надежность продукта \Rightarrow падает прибыль
 - увеличивается время разработки (time-to-market)
 - риск по страховке \Rightarrow компания разоряется

В последнее десятилетие сложность разрабатываемых компьютерных систем дошла до критического уровня: требуются все более сложные системы, а промышленность неспособна выпускать их с требуемым качеством





Тестирование, верификация, валидация

Процессы и методы, направленные на повышение качества продукта

- **Валидация** – набор приемов и методов подтверждения того, что разработано то, что требует заказчик
- **Верификация** – набор приемов и методов подтверждения того, что разрабатываемая система удовлетворяет установленным требованиям (спецификации)
- **Тестирование** – управление выполнением готовой системы с целью обнаружения несоответствия ее поведения установленным требованиям

Обычно тестирование ПО – проверка работы готовой системы на наборах тестовых данных в фиксированных сценариях

Формальная верификация – проверка формально определенных требований по формальной модели системы



Тестирование vs верификация

■ Тестирование

- тестирование и симуляция (практические методы валидации) позволяют проверить лишь некоторые сценарии поведения систем
- только тривиальные программы могут быть оттестированы исчерпывающе (exhaustive):
 - *"In practice exhaustive testing exhaust the testers long before it exhausts the system ..."* (из курса по верификации CalTech Uni, 2005)
- тестирование может показать только присутствие, но не отсутствие ошибок (Дейкстра, 1968)

■ Верификация.

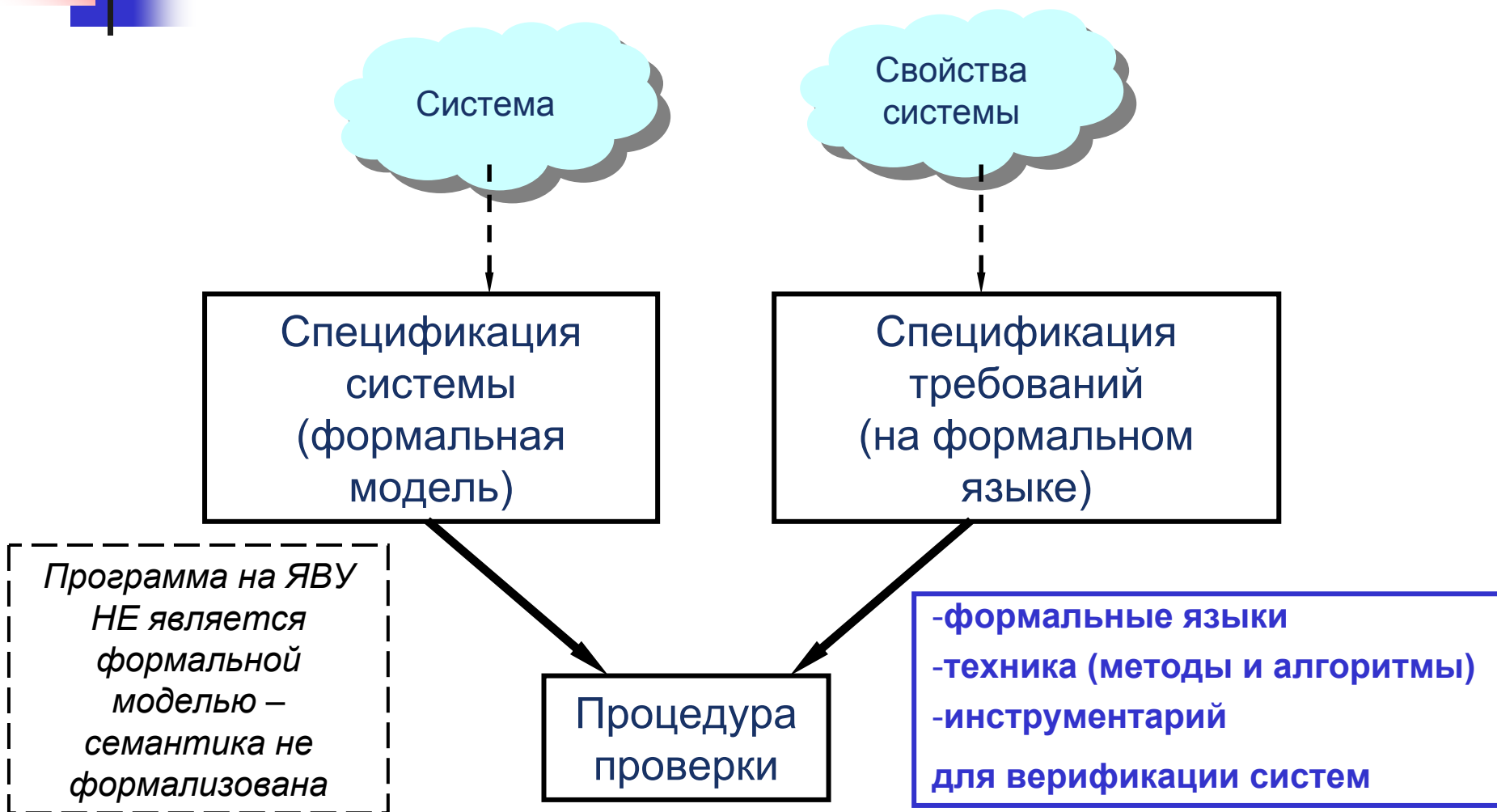
- Необходима уверенность в правильной работе систем **при всех обстоятельствах, т.е. верификация**
- Верификация требует использования формальных методов. В этом курсе мы рассматриваем формальные методы, направленные на проверку правильности программ
- При разработке современных систем верификация требует все больших затрат времени и усилий



Ненадежность тестирования

- Параллельные процессы, работающие с разделяемыми данными, обычно работают некорректно *в очень редких ситуациях*, которые воспроизвести невозможно. Примеры:
 - два параллельных оператора программы ($x := x + a \parallel x := x + b$) чрезвычайно редко пересекаются при выполнении;
 - параллельная программа разделения множеств “почти все время” работает правильно, она работает некорректно лишь на узком классе входных данных, и никакие методы тестирования не гарантируют нахождения ошибки
 - протокол Линча и протокол PAR работают правильно “почти все время”, обнаружить ошибку тестированием очень маловероятно – необходимо сочетание редко наступающих условий
- Тестирование не может гарантировать правильности программ
- Нужен строгий формальный анализ, формальная верификация программных систем по тексту программы и зафиксированной спецификации программы – что она должна делать

Общий подход к верификации систем



Формально проверить можно только формальные объекты. Адекватность формальных объектов исходным системам – вне теории



Недостатки и достоинства верификации

■ Недостатки:

- Всегда доказывается модель, а не реальная система, модель м. б. неадекватна
- Спецификации свойств могут быть неполны, неточны
- Программа автоматической верификации м.б. неправильна
- Создает ложное чувство безопасности
- Процедура проверки может требовать помощи пользователя
- Результаты верификации м.б. непонятны разработчикам систем
- Обычно требует высокой квалификации пользователей

■ Достоинства

- Верификация даже упрощенной системы повышает уровень доверия к программе, ведет к надежным системам
- Верификация “критической части” системы, отражающей ее управляющую структуру, обычно приводит к более понятной и защищенной системе
- Возрастающая степень автоматизации верификации ведет к возможности проверки более сложных систем



Выводы: Тестирование и верификация

- *Тестирование* – проверка *некоторых* поведений *реальной* системы
- *Верификация* – исчерпывающая проверка *всех ВОЗМОЖНЫХ* поведений *модели* системы

Тестирование и верификация являются взаимодополнительными методами валидации программ, каждый имеет свои достоинства и недостатки, их надо применять совместно



Важность верификации

- Для некоторых областей именно верификация наиболее важна:
 - Бортовая электроника систем космического и военного назначения
 - Контроллеры медицинским оборудованием
 - Автоматические системы управления тормозами
 - Криптографические протоколы
 - Контроллеры атомных реакторов

А теперь, как мы знаем, и

- Контролеры и системы управления крупных технических систем: электростанций, химических производств ... (Авария на Саяно-Шушенской ГЭС)



Состояние проблемы верификации

- Еще 20 лет назад использование формальных методов и верификации было весьма далеким от практики:
 - нотация сложна и неясна, ошибки встречались и в доказательствах
 - методы анализа были не масштабируемы (not scalable)
 - автоматические инструменты неадекватны и трудны в использовании; фактически, верификация проводилась интерактивно
 - были доказаны только тривиальные примеры
 - требовалась высокая квалификация для спецификации и анализа

До последнего времени методы верификации могли использоваться для проверки корректности только “игрушечных” систем, но и для них требовалось огромные усилия



Прорыв в области верификации

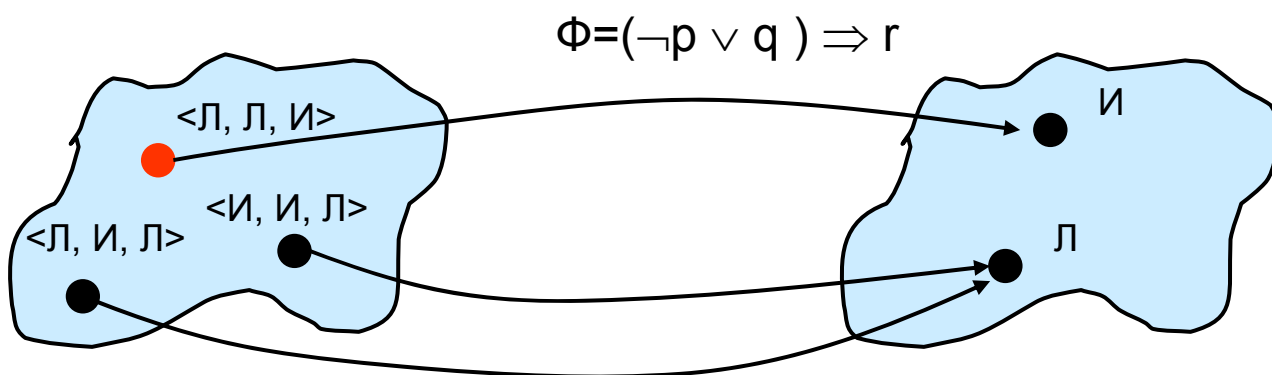
В последнее время – качественный прорыв в области верификации ПО и дискретных систем, основанный на изящных формальных методах, основной из них – model checking

Model checking (проверка модели):

методы и алгоритмы верификации аппаратуры и программ разработаны от теоретических изысканий до индустриальной технологии

Происхождение термина Model checking

Логика высказываний

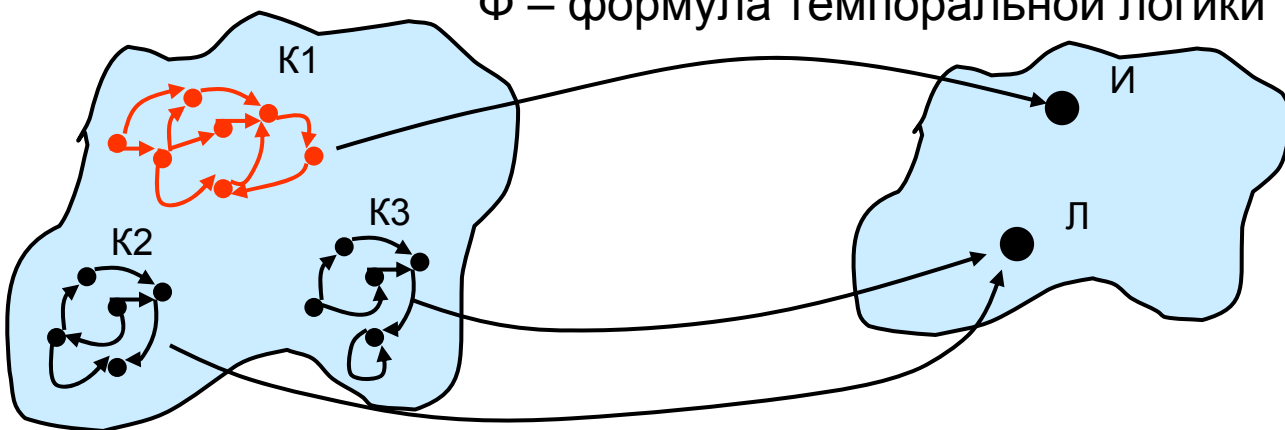


Интерпретации Φ –
наборы значений
переменных p, q, r
(конечное число)

Интерпретация $\langle \text{Л}, \text{Л}, \text{И} \rangle$ - модель формулы Φ

Темпоральная логика

Φ – формула темпоральной логики



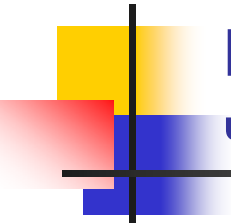
Интерпретации Φ –
системы переходов, в
каждом состоянии
которых свой набор
значений переменных
 p, q, r
(бесконечное число)

Интерпретация $K1$ - модель формулы Φ

21 июня 2008 г премия Тьюринга была вручена трем создателям техники MODEL CHECKING, внесшим в нее наиболее существенный вклад

Edmund M. Clarke (CMU),
E. Allen Emerson (U Texas, Austin),
Joseph Sifakis, (Verimag, France)

“за их роль в превращении метода Model checking в высокоэффективную технологию верификации, широко используемую в индустрии разработки программного обеспечения и аппаратных средств”



Несет ли профессионал ответственность за человеческие жизни?

Программист НЕ ЗНАЕТ

какие формализмы использовать

как определить требования к результату работы

как проверить, что эти требования выполняются

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

Rockwell Collins

Advanced Technology Center Slide 1



Who Are We?

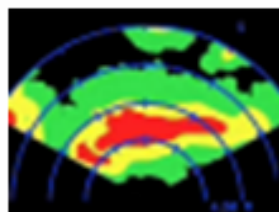
ADVANCED COMPUTING SYSTEMS

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



► **Communications**

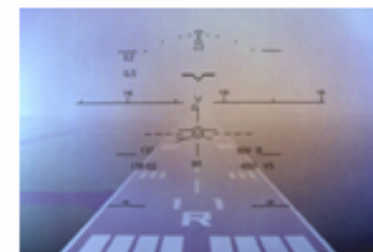
► **Navigation**



► **Automated Flight Control**

► **Displays / Surveillance**

► **Aviation Services**



► **In-Flight Entertainment**

► **Integrated Aviation Electronics**

► **Information Management Systems**





Methods and Tools for Flight Critical Systems Project

ADVANCED COMPUTING SYSTEMS

- **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?

ADVANCED COMPUTING SYSTEMS

- **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)

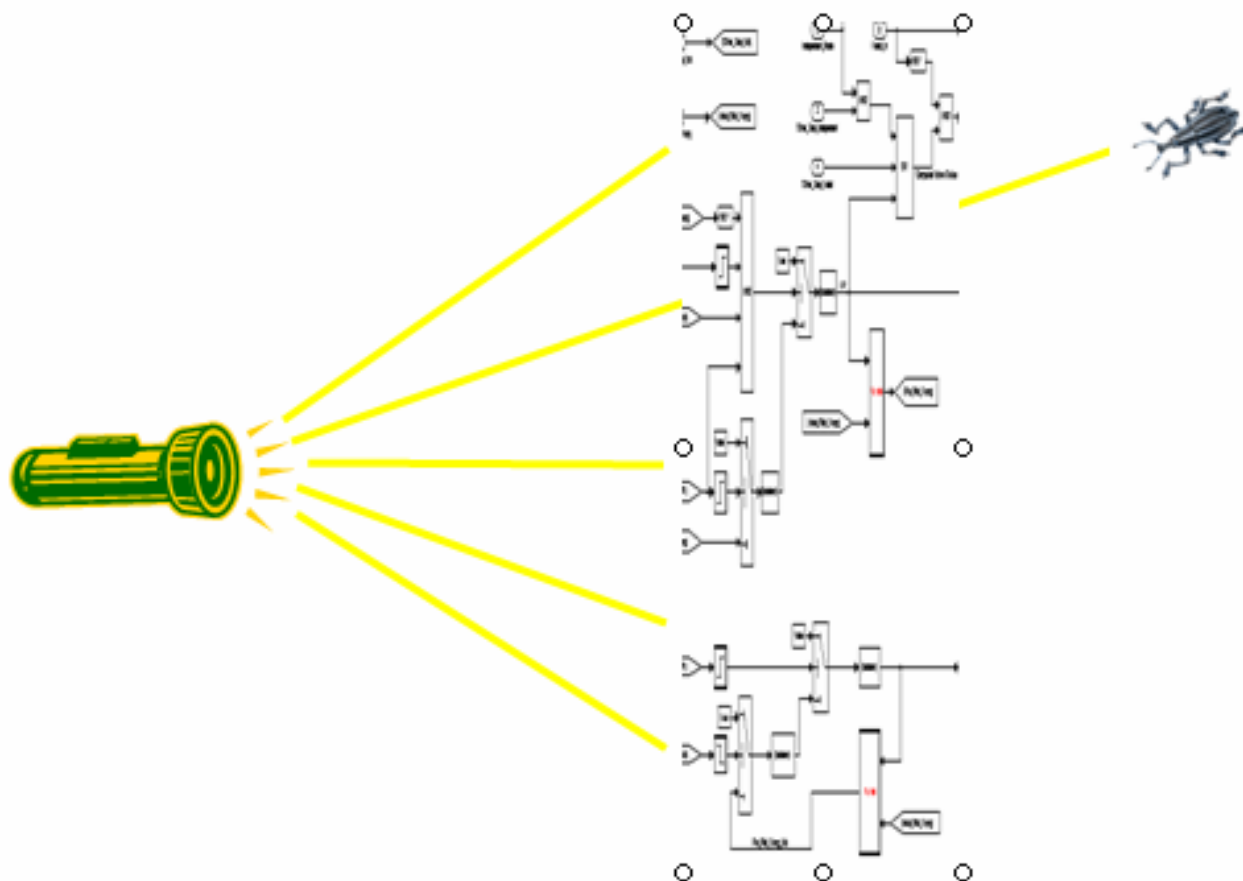


Advantage of Model Checking

ADVANCED COMPUTING SYSTEMS

Testing Checks Only the Values We Select

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

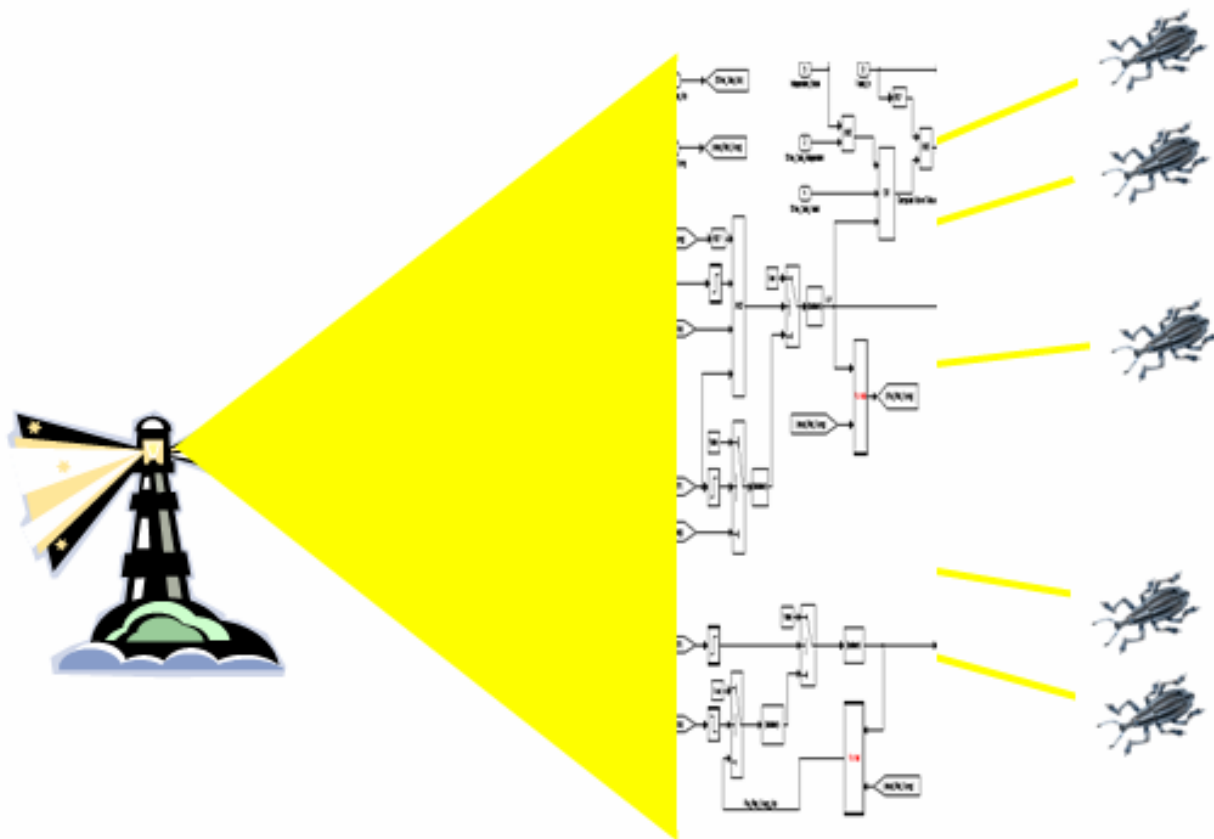




Advantage of Model Checking

ADVANCED COMPUTING SYSTEMS

Model Checker Tries Every Possible Input and State!



Rockwell



Example - ADGS-2100 Adaptive Display & Guidance System

ADVANCED COMPUTING SYSTEMS



Requirement

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

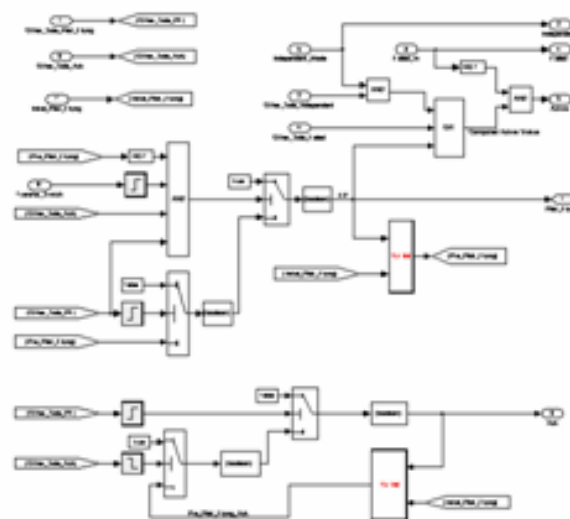
Counterexample Found in 5 Seconds!

**Checking 373 Properties
Found Over 60 Errors**

883 Subsystems

9,772 Simulink Blocks

2.9×10^{52} Reachable States



**Rockwell
Collins**



Summary of Errors Found

ADVANCED COMPUTING SYSTEMS

Decteded 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**



- **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**



Примеры использования подхода

Верифицированные системы, в которых выявлены ошибки

- Cambridge ring protocol
- IEEE Logical Link Control protocol, LLC 802.2
- фрагменты больших протоколов XTP и TCP/IP
- контроллеры реагирующих систем
- отказоустойчивые системы, протоколы доступа к шинам, протоколы контроля ошибок в аппаратуре,
- криптографические протоколы
- протокол Ethernet Collision Avoidance
- протоколы самостабилизации
- DeepSpace1 (NASA). Одна из критических ошибок – при моделировании сценария полета на расстоянии 96 млн км

Lockheed Martin: “Формальная верификация позволила найти несколько тонких ошибок, которые, скорее всего, были бы упущены в процессе традиционного тестирования”



Типичная цитата конца 90х – начала 2000х

- Formal verification of programs and systems is a very active field for both theoretical research and practical developments, especially since impressive advances in formal verification technology provided feasible in several realistic applications from the industrial world. The highly successful model checking approach suggested that a working verification technology could well be developed for finite systems



Верификация как этап технологии

ACM President Stuart Feldman :

“Это великий пример того, как технология, изменившая промышленность, родилась из чисто теоретических исследований”

- В ведущих фирмах методы и алгоритмы верификации аппаратуры и программ на основе Model checking разработаны от теоретических изысканий до индустриальной технологии



Обучение формальным методам

Обеспечение качества разработанной технической системы является важнейшей составляющей каждой инженерной специальности

Конструкторы самолетов изучают аэродинамику, строители – сопротивление материалов, машиностроители изучают теорию механизмов и машин

Каждая инженерная специальность имеет свою область прикладной математики, на которой основаны теории, позволяющие гарантировать качество инженерных разработок в этих областях

В области разработки ПО необходимые разделы прикладной математики – это формальные методы, на которых основывается верификация

“Формальные методы должны стать частью образования каждого исследователя в области информатики и каждого разработчика ПО так же, как другие ветви прикладной математики являются необходимой частью образования в других инженерных областях.

J.Rushby, NASA contractor Report 4673 *“Formal Methods and Their Role in Digital System Validation for Airborne Systems”*



Формальные методы в разработке ПО

Известно, что программисты боятся формальных методов, не используют их, рассматривают их как необязательные, искусственные трудности. Им интересны только конкретные технологические приемы, рутинные методы разработки ПО

“Формальные методы обеспечивают интеллектуальную основу нашей области, они помогают оформить наши мысли и направить их на плодотворный путь при решении проблем; они обеспечивают нотацию при документировании требований и разработки, позволяют выразить мысли при общении разработчиков в точной и ясной манере, они обеспечивают нас инструментарием для анализа свойств систем”

J.Rushby, NASA contractor Report 4673 *“Formal Methods and Their Role in Digital System Validation for Airborne Systems”*



Методика преподавания курса



Обучение верификации в Санкт-Петербургском Политехническом университете

- Курс “Технология программирования. Верификация параллельных и распределенных программ”
Читается 10 лет
- Студенты:
 - ФТК (кафедры РВКС, ИУС)
 - ФМФ (кафедра Прикладной математики),
 - ЦНИИ РТК (кафедра Телематики)
- Кроме теоретических знаний и решения задач, студенты получают первоначальный опыт использования системы верификации Spin: выполняют курсовую работу по проверке правильности нетривиальной параллельной программной системы управления
- Система Spin разработана Bell Labs, распространяется свободно, она использовалась для проверки правильности многих бортовых систем управления космического назначения в NASA



Чем мы будем заниматься

- Во всех инженерных областях существуют научные методы построения формальных моделей, которые позволяют гарантировать качество инженерной разработки
 - электротехника - теоретическая электротехника
 - авиастроение – аэродинамика
 - строительство – сопротивление материалов
- Эти модели и методы используются на этапе проектирования в технологическом цикле изготовления продукта
- В программировании до последнего времени этого не было
- Мы будем заниматься проверкой правильности (корректности) программных систем

Курс даст в руки студентов

теорию

элементы технологии

инструментарий

с помощью которых они могут проверять корректность
нетривиальных параллельных и распределенных
программ



Цели курса

- Дать понимание идей, которые используются при верификации дискретных систем, в частности, параллельных и распределенных систем
- Дать понимание теорий и формальных методов, на которых основаны современные инструменты, методы анализа и верификации
- Предоставить возможность получения начального опыта применения теорий и алгоритмов для решения задач в этой области (ручное решение задач)
- Предоставить возможность получения практического опыта использования формального моделирования и анализа систем с помощью языка Promela и системы верификации SPIN (и/или других инструментов)



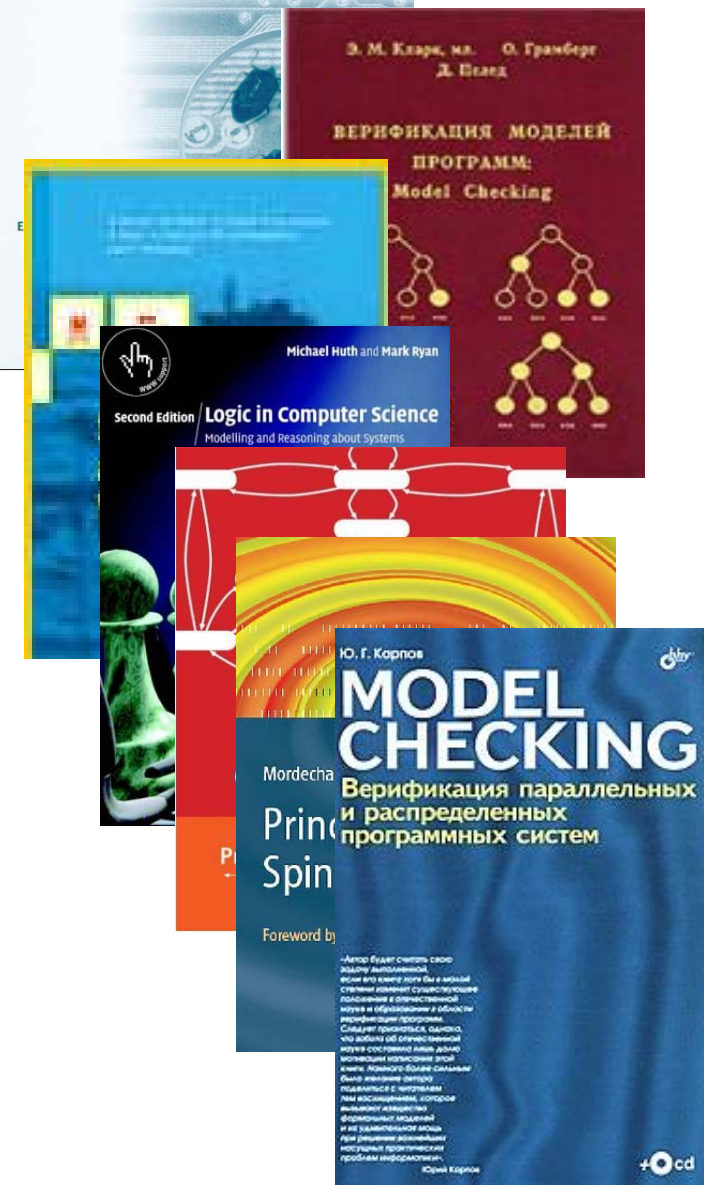
План курса

1. Введение
2. Метод Флойда-Хоара доказательства корректности программ
3. Алгебры процессов
4. Темпоральные логики
5. Алгоритм model checking для проверки формул CTL
6. Автоматный подход к проверке выполнения формул LTL
7. Структура Крипке как модель реагирующих систем
8. Темпоральные свойства систем
9. Система верификации Spin и язык Promela. Примеры верификации
10. Применения метода верификации model checking
11. BDD и их применение
12. Символьная проверка моделей
13. Количественный анализ дискретных систем при их верификации
14. Верификация систем реального времени (I)
15. Верификация систем реального времени (II)
16. Консультации по курсовой работе

Литература

- E. M. Clarke, O. Grumberg, D. Peled. Model checking. 1999
 - (Русский перевод: Э.М.Кларк, О.Грамберг, Д.Пелед. Верификация моделей программ: Model Checking. М., 2002)
- B. Berard et al. Systems and Software Verification. Model checking Techniques and Tools // Springer, 1999
- M. Huth, M. Ryan. Logic in Computer Science: Modelling and Reasoning about Systems Cambridge U // 1999
- C. Baier, J. P. Katoen. Principles of Model checking // 2008
- M. Ben-Ari. Principles of Spin Model Checker // 2008
- Ю. Г. Карпов. Model checking. Верификация параллельных и распределенных программных систем // БХВ. Петербург, 2010
- Множество статей, выложенных в Интернет курсов (в частности, Ю. Лифшиц-CalTech, Б. Конюх – U. Liverpool)

Model Checking



- Посещаемость
- Курсовая работа по верификации параллельных систем с помощью системы Spin (Prism)
- Письменный экзамен по решению задач.
В каждый вариант будет входить набор базовых обязательных задач:
 - проверка CTL формулы на структуре Крипке
 - проверка PCTL формулы на вероятностной структуре Крипке
 - построение BDD структуры для заданного фрагмента программы
 - проверка TCTL формулы для временного автомата



Методические и учебные материалы

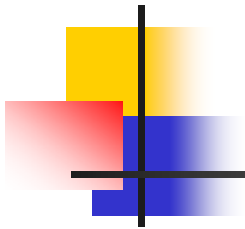
- Слайды лекций будут еженедельно выкладываться на сайте кафедры *“Распределенных вычислений и компьютерных сетей”* по адресу: [//dcn.infos.ru/~karpov](http://dcn.infos.ru/~karpov)
- Книга
 Ю.Г.Карпов. *Model checking*
 есть в библиотеке. Кроме того, можно купить по цене издательства на кафедре.
- Пособие по установке системы Spin и верификации систем с ее помощью будет выложено на сайте [//dcn.infos.ru/~karpov](http://dcn.infos.ru/~karpov). В пособии определена одна типовая задача из нескольких параллельных процессов управления светофорами на перекрестках, взаимодействующих через общие переменные. Задача параметризуется топологией перекрестка
- Ирина Владимировна Шошмина, ассистент кафедры РВКС – консультант и инструктор по Spin’у и курсовой работе

Организационные вопросы

- Курсовая работа – решение нетривиальной задачи синхронизации процессов – модификация изначально некорректной программы и ее верификация – **зачет**
- Теоретический курс – письменный **экзамен** (решение задач) 3 часа
- По каждому виду учебной работы выставляются баллы:
 - Посещение лекций до 15 баллов
 - Курсовая работа от -15 до + 15 баллов (норма – 0)
 - Письменный экзамен до 80 баллов
- Посещения лекций фиксируется: 1 балл за 1 посещенную лекцию
- Курсовая работа:
 - 0 получают студенты, представившие стандартное решение и понимающие его
 - Больше 0 (до 15) баллов за курсовую работу получают студенты, представившие оригинальное глубокое решение и полностью его понимающие.
 - Меньше 0 (до -15) баллов получают студенты, плохо понимающие, что они представили в качестве решения
 - Студенты, **не понимающие**, что они представляют в качестве решения, зачета не получают
- Общая оценка выводится по сумме набранных баллов:

0 - 19	баллов	оценка	– 1;
20 - 39			– 2;
40 - 59			– 3;
60 - 79			– 4;
85 - 110			– 5
-

- # Ю.Г.Карпов



Спасибо за внимание



Пример задач к лекции 1

- 1.1. Рассмотрите параллельную программу $P_1 || P_2$ с двумя процессами, которые выполняются асинхронно:

$$\begin{array}{lll} P_1:: & \{ a_1: A:=1; & b_1: A:=2; & c_1: A:=3; \} \\ P_2:: & \{ a_2: B:=A; & b_2: C:=A+B; & c_2: D:=A; \} \end{array}$$

Значение A вначале равно 0. Рассмотрите все возможные варианты значений, которые могут иметь переменные B , C и D по завершении программы (после завершения обоих процессов).

- 1.2. а) Протестируйте программу разделения множеств для различных наборов значений в множествах S и L . б) Докажите, что эта программа всегда работает правильно в случае одноэлементных множеств S и/или L .
- 1.3. Протестируйте с помощью ручной прокрутки протокол W.C.Lynch при различных ситуациях в каналах передачи данных.
- 1.4. Протестируйте с помощью ручной прокрутки протокол PAR при различных ситуациях в каналах передачи данных. Найдите сценарии, приводящие к некорректному поведению протокола.