

Верификация распределенных алгоритмов и систем



Курс лекций

Лектор: Шошмина Ирина Владимировна

Авторы курса: профессор Карпов Юрий Глебович, Шошмина Ирина Владимировна



План лекции

- Мотивация изучения верификации
- Содержание дисциплины
- Структура преподавания



Автор курса



Ю.Г. Карпов – заслуженный проф. СПбПУ с 2006

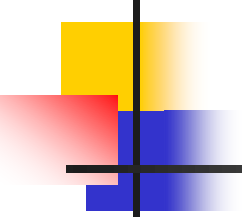
- **Архитектор системы имитационного моделирования AnyLogic**
- **Автор курсов “Математическая логика”, “Теория автоматов и формальных языков”,**

- член Американского Математического общества (с 1975 г.)
- член Association for Computing Machinery (ACM)
- член IEEE Computer Society
- член программных комитетов нескольких международных конференций (ИММ, CoLoS, SEEMAS, PaC и других)
- эксперт Российского фонда фундаментальных исследований (с 2006 г.)



Введение

Мотивация изучения верификации

- 
-
- Название курса
 - Верификация
 - распределенных алгоритмов и систем



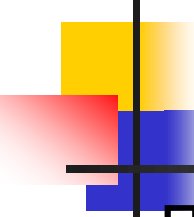
Верификация?

- **Компьютеры – повсюду** *Ubiquitous computing*
(вездесущие вычисления)
ИТ проникли во все сферы жизни современного общества.
- *Раньше компьютер был компьютером, а телефон – телефоном, и любой мог отличить одно от другого. Сейчас и компьютер – не только компьютер, и телефон – не только телефон (A.Tannenbaum)*
- **Эра параллельного программирования**
Последовательные программы имеют очень узкое применение
Параллельные программы все более распространяются.
Многоядерные чипы, встроенные бортовые СУ -> требуются технологии разработки параллельного ПО, чтобы оно выполняло нужные функции
Multicore processors are bringing parallelism to mainstream of computing
- **Параллельные программы полны ошибок**
Параллельные программы непостижимы для человеческого анализа: они очень часто неправильны, содержат ошибки



Примеры последствий программных ошибок

- 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.
- 1985-87 гг: Therac-25 прибор лучевой терапии. Пациенты получили передозировку, шестеро 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 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"



Несет ли профессионал ответственность за человеческие жизни?

Программирование является областью инженерной деятельности,
где критерии качества работы разработчика размыты

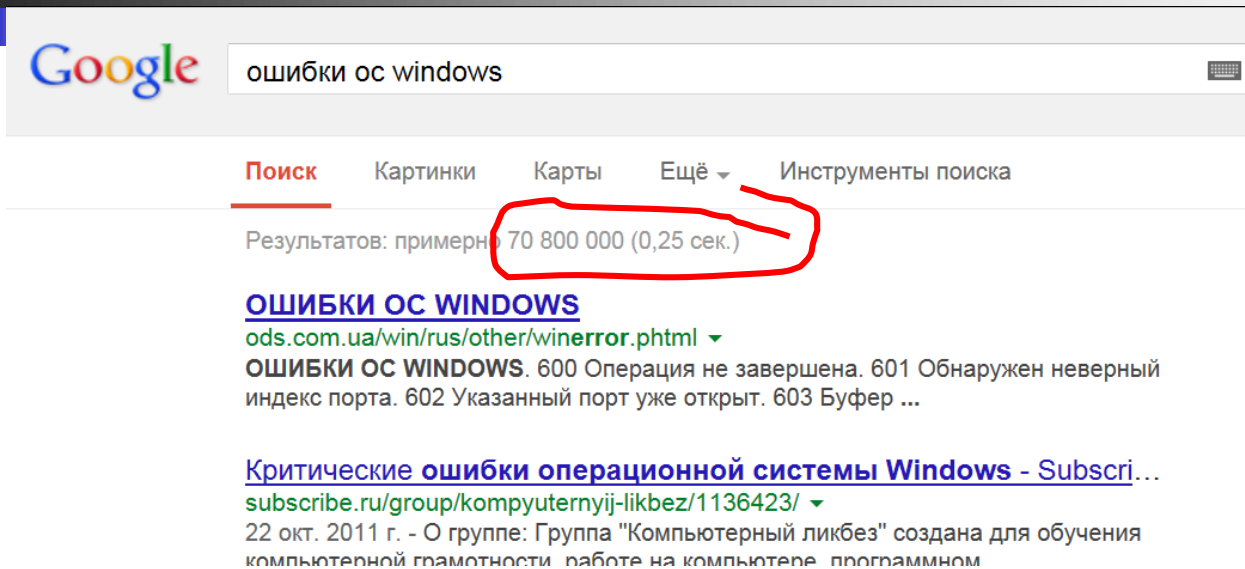
Обычно ПО имеет 10-15 ошибок на 1000 строк кода,
ПО высокого качества – 3 ошибки на 1000 строк кода

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

Ужасы из-за ошибок ПО - в Интернете:

SOFTWARE HORROR STORIES:	http://www.cs.tau.ac.il/~nachumd/horror.html
Collection of Software Bugs:	http://www5.in.tum.de/~huckle/bugse.html
Worst software defects in history:	http://cristianpocovnicu.wordpress.com/2011/03/22/worst-software-defects-in-history/
Википедия:	http://en.wikipedia.org/wiki/List_of_software_bugs

Миллионы сообщений об ошибках в ОС Windows



- По заявлению Майкрософта, в ОС Windows остаются тысячи ошибок!
- Июнь 2013: Microsoft анонсировала программу премирования пользователей, отыскавших уязвимости в предварительной версии операционной системы Windows 8.1. Тем, кто обнаружит ошибки и найдет способы обойти встроенную в ОС защиту, обещают выплатить до \$100 000.

Необходимы методы проверки качества программ – с этим связано понятие **верификации**



Распределенные системы



Параллельные, распределенные системы

- Недетерминизм выполнения (произвольное чередование) операций
 - Высокое влияние атомарности/неатомарности операций
 - Легко привести систему к блокировке процессов
 - Большое количество операций
- || системы обычно работают правильно “**почти всегда**”, они годами могут сохранять тонкие ошибки, проявляющиеся в редких, исключительных ситуациях

Реагирующая система – это коллекция параллельно функционирующих, обычно незавершающихся, взаимодействующих друг с другом и с внешней средой процессов, выполняющих единую задачу

Основные ошибки таких систем связаны с **синхронизации** процессов

Как выполнить анализ их *поведения* (дедлок, голодание и т.п.)?

Модель должна давать возможность описать НЕДЕТЕРМИНИЗМ! (В отличие от функциональных программ, недетерминизм – важнейшая составляющая параллельных процессов!!)



Пример параллельных процессов (недетерминизм, чередование)

$x := 1; x++ \parallel x := 3$

два детерминированных процесса; их параллельное выполнение
недетерминированно (сколько результатов? какие?)

В отличие от функциональных программ, недетерминизм – важнейшая составляющая параллельных процессов!

\parallel - асинхронная композиция, произвольное чередование

Конфуз с голосованием на выборах в Думу 4 декабря 2011. Как так вышло?



58.99%
32.96%
23.74%
19.41%
9.32%
1.46%
0.59%

Σ

146.47
%

Возьмем простой вариант с голосованием на одной из радио станций 24.09.2011.

Параллельная композиция процессов:

За П



```
xП := 0;  
while !Stop  
do  
  wait call_1;  
  xП ++  
od
```

За М



```
xМ := 0;  
while !Stop  
do  
  wait call_2;  
  xМ ++  
od
```

Σ

```
while !Stop  
do  
  xΣ := xП + xМ  
od
```

$\%_П$

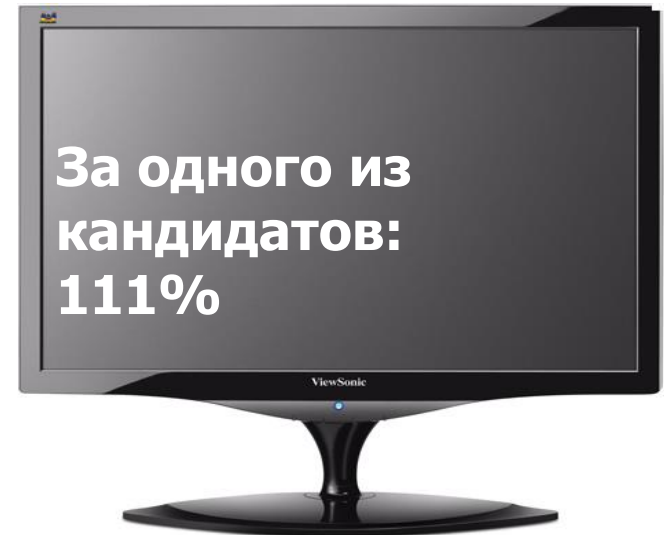
```
while !Stop  
do  
  PП := xП / xΣ  
od
```

$\%_М$

```
while !Stop  
do  
  PМ := xМ / xΣ  
od
```

■ Журналист:

*“Подвисает система голосования...
У меня возникает 111% ...”*



Могут ли быть ошибки в программе из одной строки? (влияние атомарности/неатомарности операций)

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

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

Чему равно x после параллельного выполнения процессов A и B?

Есть ли ошибка?

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. $R_B \rightarrow x$

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 ошибки, аналогичной найденной в бортовом ПО Deep Space1

- Proc Inc = while true do if $x < 200$ then $x := x + 1$ fi od
- Proc Dec = while true do if $x > 0$ then $x := x - 1$ fi od
- Proc Reset = while true do if $x = 200$ then $x := 0$ fi od

Inc || Dec || Reset - будет ли x всегда оставаться в интервале: $0 \leq x \leq 200$?

Пусть $x = 200$ и это обнаружил процесс Dec. Он выполняет $x := x - 1$.

Но из-за интерливинга, ПОСЛЕ того, как Dec это обнаружил, но ДО того, как он выполнил вычитание, процесс Reset тоже это обнаружил, и сбросил x в 0.

После этого Dec выполняет вычитание 1 из переменной $x = 0$, и x становится равным -1

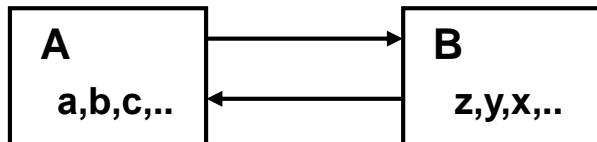
Ответ: не будет

Но **ошибка чрезвычайно редкая**, даже когда переменная x (с очень маленькой вероятностью) стала равной 200, когда (тоже с очень малой вероятностью) именно и Dec, и Reset это обнаружили почти одновременно, друг за другом, именно сначала выполнится Reset, а потом Dec – т.е.

зависит от скоростей процессов, от алгоритма планировщика процессов, от Тестированием подобные ошибки обнаружить не удастся, трасса почти НЕ ПОВТОРЯЕТСЯ

Пример: протокол W.C.Lynch (1968)

Посимвольная полнодуплексная передача по телефонным линиям без потерь с *обнаружением* ошибок



Сообщение: < C, X >

C - управляющий символ

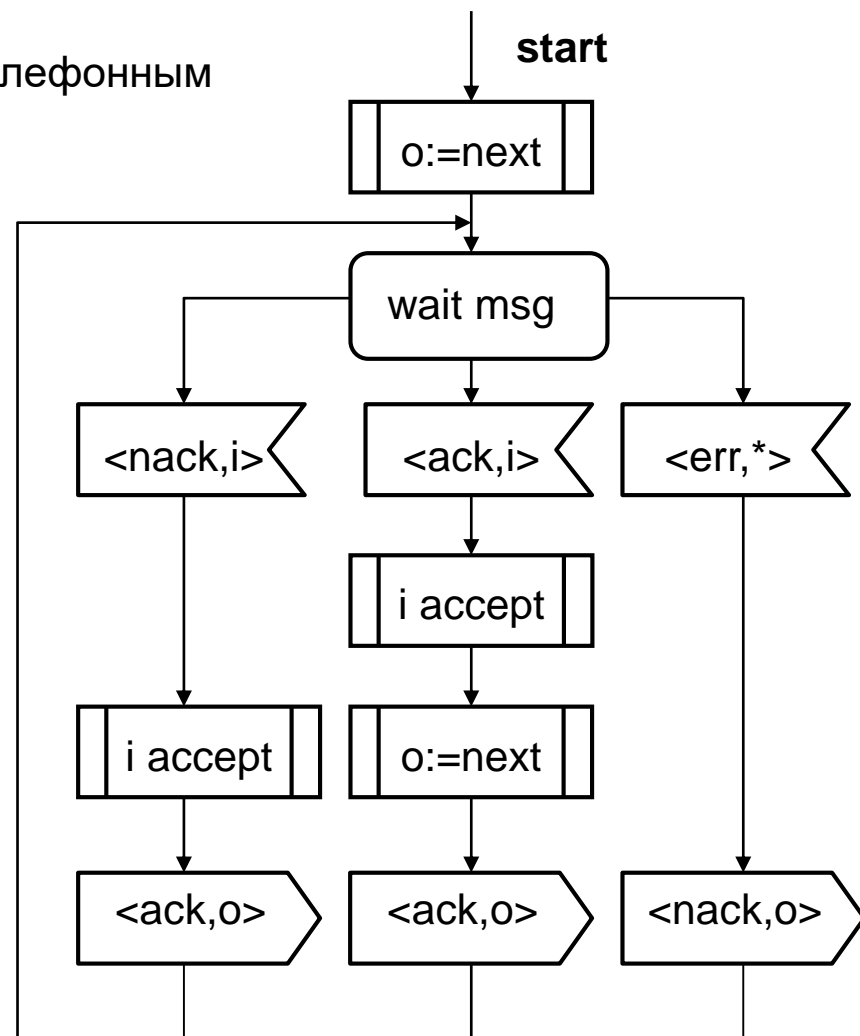
X - информационный символ

Правила протокола:

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

Канал может исказить информацию, но не
может терять, дублировать, вставлять

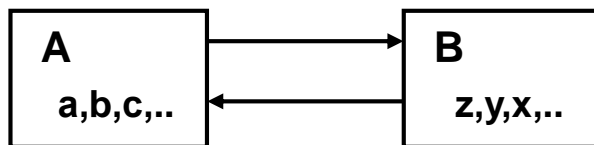
новыеЮ.Г.Карпов



Корректен ли протокол Линча?

Некорректности протокола W.C.Lynch

Правила выглядят логично, но протокол содержит ошибки!



Ошибки протокола Линча:

1. *Останов передачи:*

Если в одном направлении нечего передавать, то и в другом направлении передача останавливается;

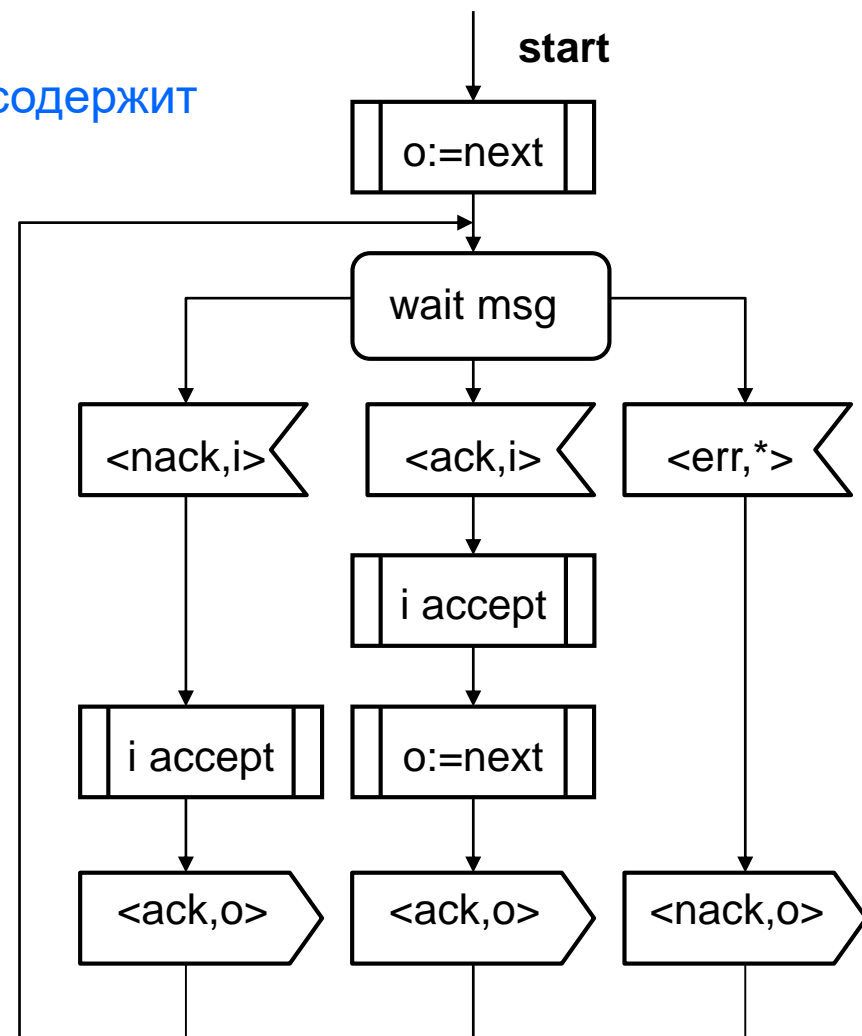
2. *Проблемы инициализации:*

После запуска каждый ждет сообщения от партнера

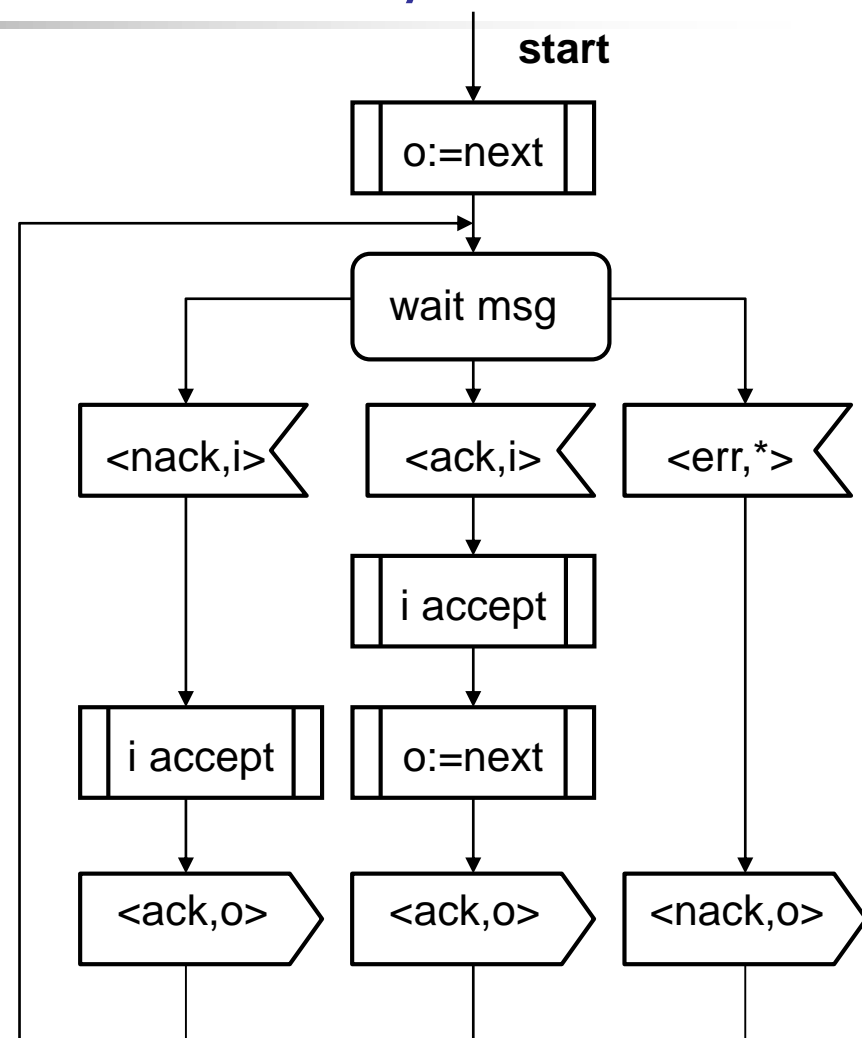
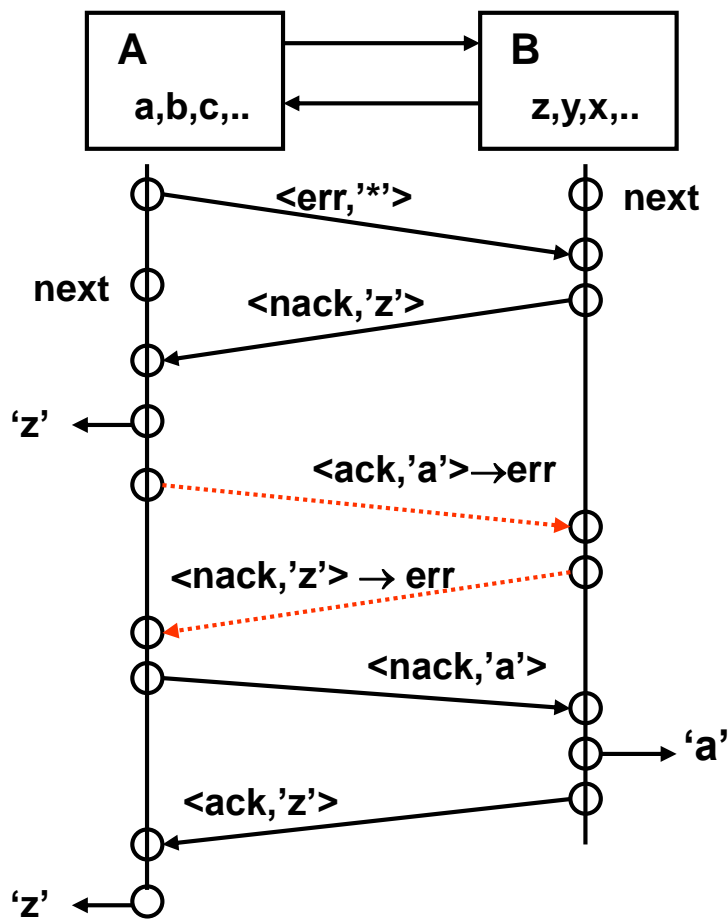
3. *Проблемы с завершением:*

Как завершить сеанс?

4. *Логическая ошибка в организации передачи*



Логическая ошибка в протоколе W.C.Lynch



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



Ошибки в параллельных системах

Нетривиальные параллельные и распределенные системы
трудно понимаются человеком

“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



Ошибки в параллельных, распределенных 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

Важность проблемы обеспечения качества ПО

- 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, требуются все более сложные механизмы обнаружения ошибок



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)
- **Формальная верификация** – набор **формальных** приемов и методов подтверждения того, что разрабатываемая система удовлетворяет **формальным установленным требованиям** (формальной 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 – формальное ДОКАЗАТЕЛЬСТВО правильности: проверка выполнения формально определенных требований на формальной модели системы при любом поведении (наборе данных)



Тестирование vs верификация

■ Тестирование

- тестирование и симуляция (практические методы валидации) позволяют проверить лишь некоторые сценарии поведения систем
- только тривиальные программы могут быть протестированы исчерпывающе (exhaustive):
 - *"In practice exhaustive testing exhaust the testers long before it exhausts the system ..."* (курс по верификации CalTech Uni, 2005)
 - Edsger W. Dijkstra (1969): Program testing can be used to show the presence of bugs, but never to show their absence!

■ 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 больших затрат времени и усилий

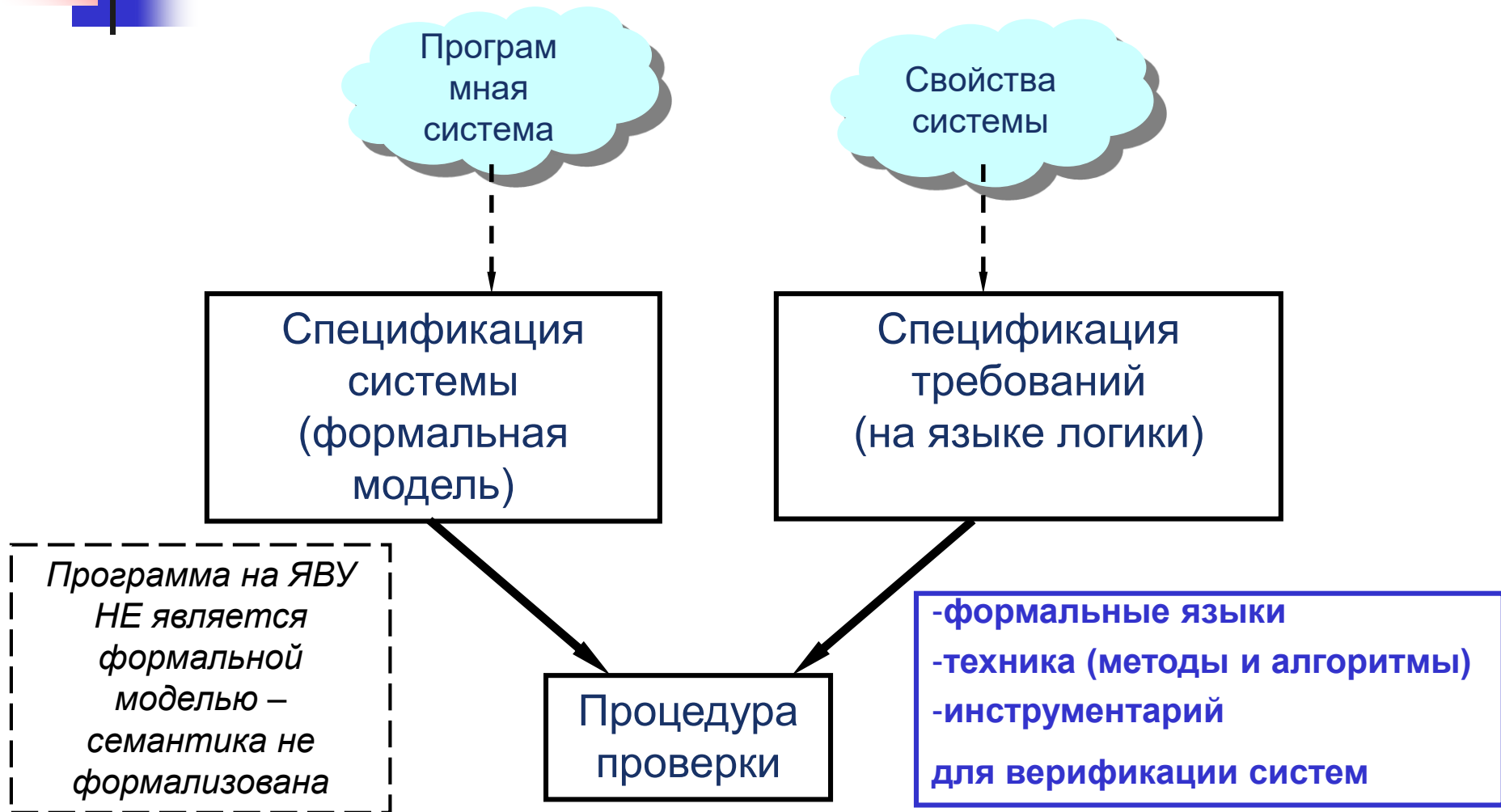


Выводы: Тестирование и верификация взаимодополнительны

- *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

Общий подход к формальной верификации систем



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



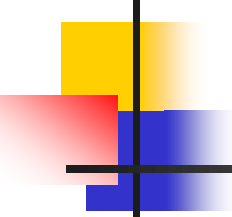
Подходы к верификации ПО

Исследования в области верификации начались с самых ранних лет использования компьютеров

- **Дедуктивная верификация (theorem proving) - с 1967**
 - R.Floyd, C.A.R.Hoare, Edasger W.Dijkstra
- **Алгебры процессов**
 - CCS (Calculus of Communicating Processes – R.Milner)
 - CSP (Communicating Sequential Processes - C.A.R.Hoare)
 - ACP (Process algebra for synchronous Communication - J. A. Bergstra and J. W. Klop)
- **Model checking – началось в 1981 (~30 years)**
 - Пионерские работы в области Model checking были выполнены
E. M. Clarke and E. A. Emerson
J. P. Queille and J. Sifakis
- ...



"You want proof? I'll give you proof!"

- 
- До последнего времени методы верификации могли быть применены для доказательства корректности только “toy systems” а не промышленного ПО, но даже для простых программ требовались огромные усилия
 - С 60 годов 20 века много лет большое число исследователей работало над проблемой доказательства правильности программ, однако, эта проблема еще недавно не имела удовлетворительного решения
 - 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

Algebra of Communicated Processes – ACSR Laws

Table 1: The ACSR bisimulation laws

ITP(1)	$A^u : P = A^{t \leq u} (0, P)$	
ITP(2)	$A^{t \leq u} (P, Q) = 0$	if $P \succ A : Q$
ITP(3)	$A^{t \leq 0} (P, Q) = Q$	
ITP(4)	$A^{t \leq \infty} (P, Q) = A^{t \leq \infty} (P, 0)$	
ITP(5)	$A^{t \leq u} (P, Q) = 0$	if $u < 0$
Choice(1)	$P + 0 = P$	
Choice(2)	$P + P = P$	
Choice(3)	$P + Q = Q + P$	
Choice(4)	$(P + Q) + R = P + (Q + R)$	
Choice(5)	$P + Q = Q$	if $P \prec Q$
Par(1)	$P \parallel Q = Q \parallel P$	
Par(2)	$(P \parallel Q) \parallel R = P \parallel (Q \parallel R)$	
Par(3)	$P \parallel Q = P \parallel Q + P \parallel Q \parallel P$	
LeftM(1)	$e.P \parallel Q = e.(P \parallel Q)$	
LeftM(2)	$(A^{t \leq u} (P, Q)) \parallel R = 0$	if $u > 0$
LeftM(3)	$(P + Q) \parallel R = (P \parallel R) + (Q \parallel R)$	
LeftM(4)	$0 \parallel R = 0$	
Sync(1)	$(a, p).P \parallel (\bar{a}, q).Q = (\tau, p + q).(P \parallel Q)$	
Sync(2)	$(A^{t_1 \leq u} (PI, PC)) \parallel (B^{t_2 \leq v} (QI, QC)) = (A B)^{t \leq w} (RI, RC)$ if $\rho(A) \cap \rho(B) = \emptyset$ and $w = \min(u, v)$ and $RI =$ $PI \uparrow^{t_1} \parallel (QI \uparrow^{t_2} + B^{t_2 \leq v - t_1} (QI \uparrow^{t+t_2/t_2}, QC))$ $+ PI \uparrow^{t_1} \parallel (QI \uparrow^{t_2} + B^{t_2 \leq v - t_1} (QI \uparrow^{t+t_2/t_2}, QC))$ $+ QI \uparrow^{t_2} \parallel (PI \uparrow^{t_1} + A^{t_1 \leq u - t_1} (PI \uparrow^{t+t_1/t_1}, PC))$ $+ QI \uparrow^{t_2} \parallel (PI \uparrow^{t_1} + A^{t_1 \leq u - t_1} (PI \uparrow^{t+t_1/t_1}, PC))$ and $RC =$ $(PI \uparrow^{u/t_1} + A^{t_1 \leq u - w} (PI \uparrow^{w+t_1/t_1}, PC))$ $\parallel (QI \uparrow^{v/t_2} + A^{t_2 \leq v - w} (QI \uparrow^{w+t_2/t_2}, QC))$	
Sync(3)	$e.P \parallel f.Q = 0$	if $l(e) \neq \overline{l(f)}$
Sync(4)	$e.P \parallel A^{t \leq u} (Q, R) = 0$	
Sync(5)	$A^{t_1 \leq u} (P_1, Q_1) \parallel B^{t_2 \leq v} (P_2, Q_2) = 0$ if $u > 0 \wedge v > 0 \wedge \rho(A) \cap \rho(B) \neq \emptyset$	
Sync(6)	$P \mid Q = Q \mid P$	
Sync(7)	$(P + Q) \mid R = P \mid R + Q \mid R$	
Sync(8)	$0 \mid R = 0$	
Rec(1)	$rec X.P = P[rec X.P/X]$	
Rec(2)	If $P = Q[P/X]$ and X is guarded in Q then $P = rec X.Q$	
Rec(3)	$rec X.(P + [X \setminus E]_U) = rec X.(P + [P \setminus E]_U)$	

Table 2: The ACSR bisimulation laws (cont.)

Timeout(1)	$0 \Delta_v Q = 0$	if $v > 0$
Timeout(2)	$(P_1 + P_2) \Delta_v Q = P_1 \Delta_v Q + P_2 \Delta_v Q$	
Timeout(3)	$(A^{t \leq u} (P, Q)) \Delta_v R = A^{t \leq w} (P \Delta_{v-t} R, Q \Delta_{v-w} R)$ if t is not free in R and $0 < w = \min(u, v)$	
Timeout(4)	$e.P \Delta_v Q = e.(P \Delta_v Q)$	if $v > 0$
Timeout(5)	$P \Delta_0 Q = Q$	
Res(1)	$0 \setminus F = 0$	
Res(2)	$(P + Q) \setminus F = (P \setminus F) + (Q \setminus F)$	
Res(3)	$A^{t \leq u} (P, Q) \setminus F = A^{t \leq u} (P \setminus F, Q \setminus F)$	
Res(4)	$((a, n).P) \setminus F = (a, n).(P \setminus F)$	if $a, \bar{a} \notin F$
Res(5)	$((a, n).P) \setminus F = 0$	if $a \in F \vee \bar{a} \in F$
Res(6)	$P \setminus E \setminus F = P \setminus E \cup F$	
Res(7)	$P \setminus \emptyset = P$	
Close(1)	$[0]_U = 0$	
Close(2)	$[P + Q]_U = [P]_U + [Q]_U$	
Close(3)	$[A^u : P]_U = [A]_U^u : [P]_U$	
Close(4)	$[e.P]_U = e.[P]_U$	
Close(5)	$[[P]_U]_V = [P]_{U \cup V}$	
Close(6)	$[P]_\# = P$	
Close(7)	$[P \setminus E]_U = [P]_U \setminus E$	
Except(1)	$0 \dagger Q = Q$	
Except(2)	$P \dagger 0 = P$	
Except(3)	$(P + Q) \dagger R = P \dagger R + Q \dagger R$	
Except(4)	$A^{t \leq u} (P, Q) \dagger R = R + A^{t \leq u} ((P + R), Q \dagger R)$	if t is not free in R
Except(5)	$e.P \dagger Q = Q + e.(P \dagger Q) + (e \mid Q)$	
Except(6)	$(P \dagger Q) \dagger R = P \dagger (Q \dagger R)$	
Except(7)	$P \dagger Q = Q + P \dagger Q$	
Guard(1)	$(a, p) \mid (\bar{a}, q).Q = (\tau, p + q).Q$	
Guard(2)	$e \mid f.Q = 0$	if $l(e) \neq \overline{l(f)}$
Guard(3)	$e \mid A^{t \leq u} (P, Q) = 0$	
Guard(4)	$e \mid (P + Q) = (e \mid P) + (e \mid Q)$	
Guard(5)	$e \mid 0 = 0$	

- Законы бисимуляции ACSR (Algebra of communicated shared resources, 1995)



Метод проверки модели – современный метод

Верифицированные системы, в которых выявлены ошибки

- Cambridge ring protocol
- IEEE Logical Link Control protocol, LLC 802.2
- фрагменты больших протоколов XTP и TCP/IP
- отказоустойчивые системы, протоколы доступа к шинам, протоколы контроля ошибок в аппаратуре,
- криптографические протоколы
- протокол Ethernet collision Avoidance
- **IEEE стандарт** Futurebus cache coherence protocol ($\sim 10^{30}$ состояний): несколько найденных ошибок заставили существенно пересмотреть протокол
- встроенная бортовая система управления (1421 state machines, 10^{476} сост. (J. Staunstrup, Practical verification of embedded software, IEEE Computer, N5, 2000))

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



Deer Space1 (аппарат на Марс) – верификация бортового ПО

- Был верифицирован новый распределенный контроллер с помощью только что созданной системы верификации SPIN (1998)
- Затраты на создание модели
 - 2 сотрудника NASA/Ames Research center
 - 6-8 недель
 - 354 строки кода SPIN
 - 1 неделя на прогоны и документирование результатов верификации (около 300.000 состояний модели, 10 sec/прогон)
- Найдены 4 серьезные ошибки
- Реакция разработчика:

“You’ve found a number of bugs that I am sure would not have been found otherwise. One of the bugs revealed a major design flaw. So I’d say you have had a substantial impact”

K.Havelund, M.Lowry, J.Penix. *Formal Analysis of a Space Craft Controller using SPIN, 1998*

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?

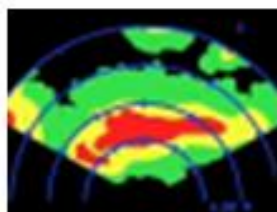
ADVANCED COMPUTING SYSTEMS

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)

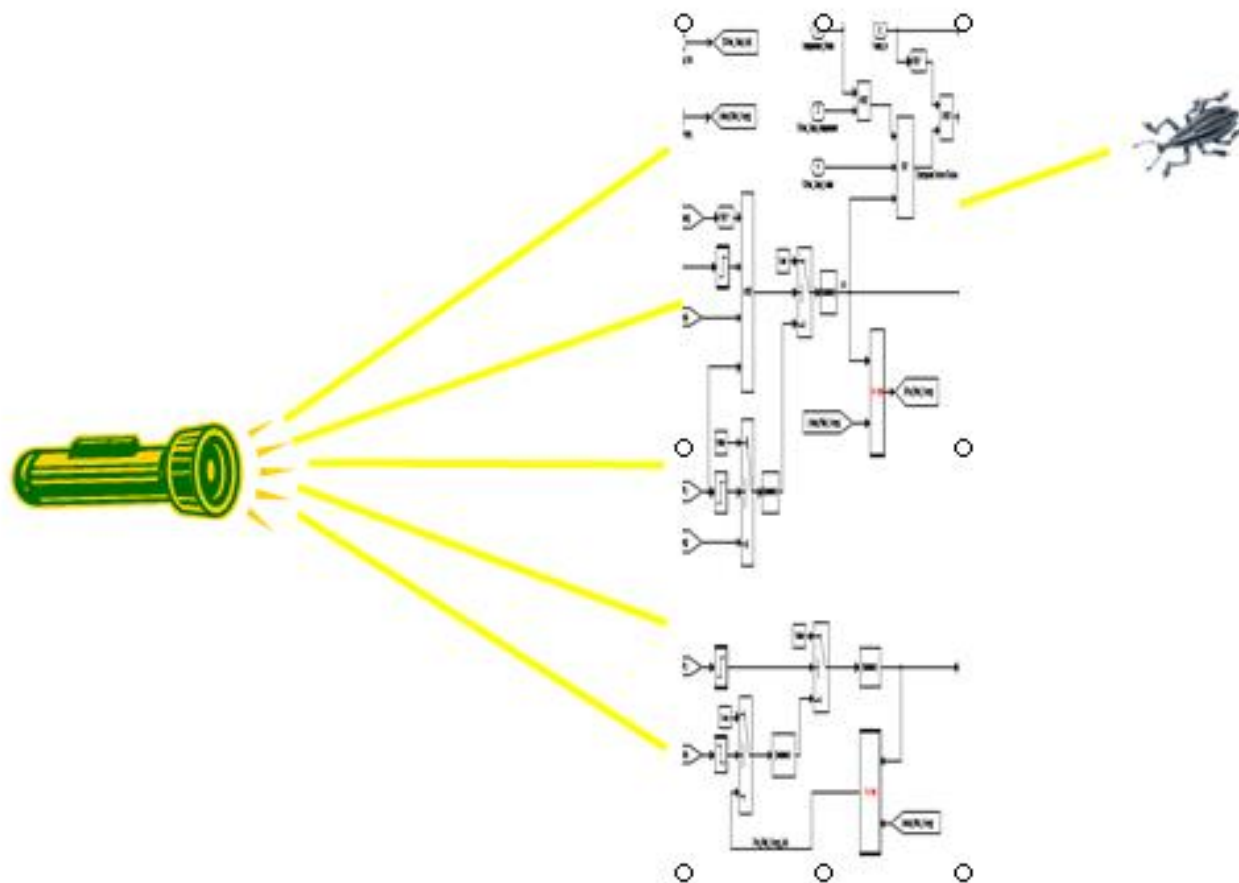


Advantage of Model Checking

ADVANCED COMPUTING SYSTEMS

Testing Checks Only the Values We Select

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

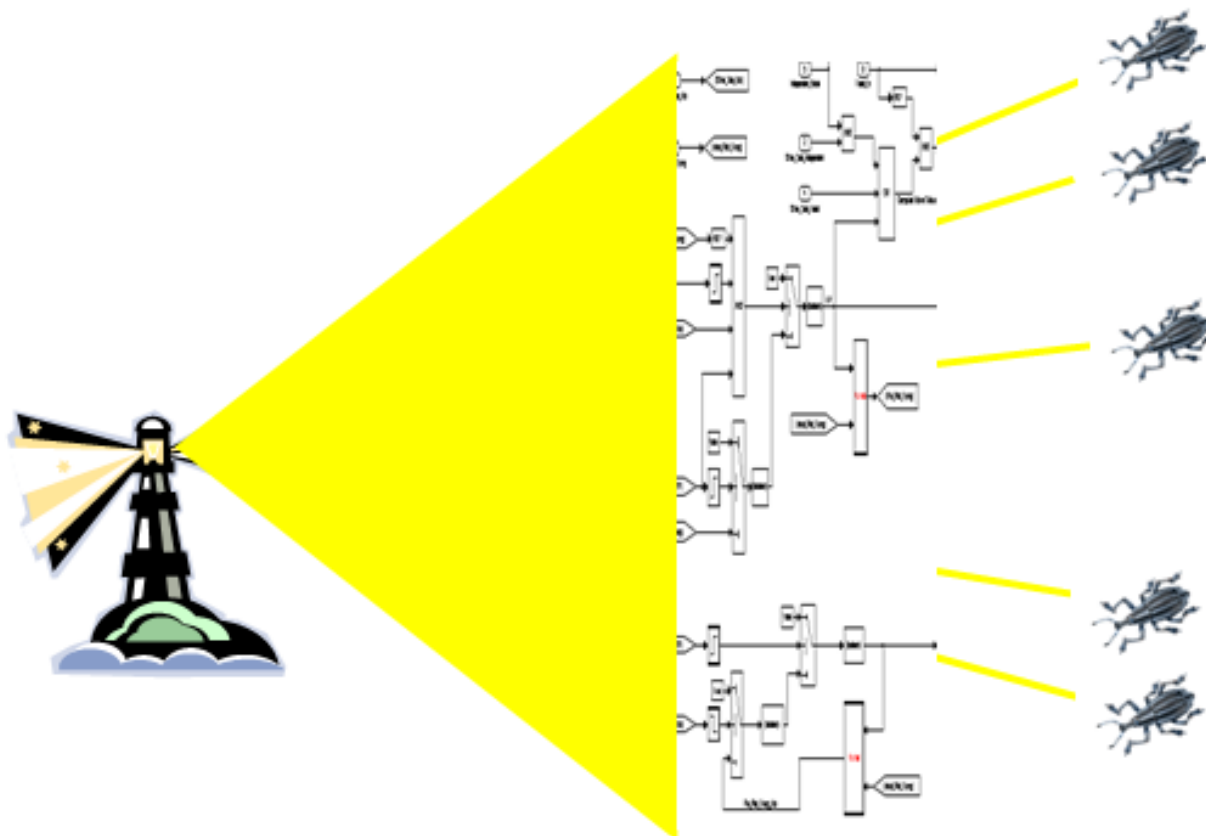




Advantage of Model Checking

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**

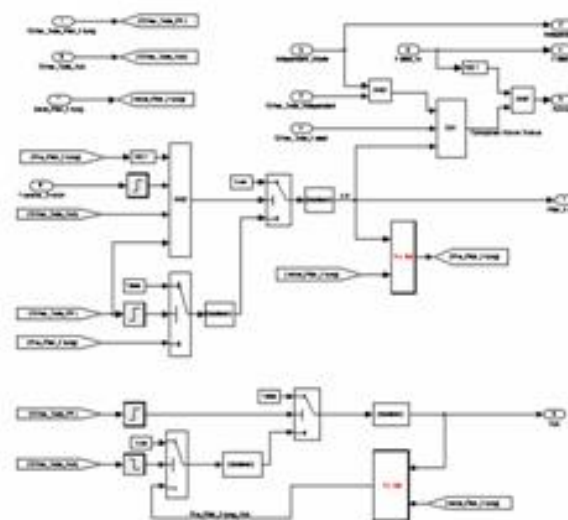
Counterexample Found in 5 Seconds!

**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**



Недостатки и достоинства формальной 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
- Невозможно формально определить, что проверены ВСЕ нужные свойства

■ 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



Методика преподавания курса



Обучение формальным методам

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

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 также необходимые свои 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, SRI International, NASA contractor Report 4673 *“Formal Methods and Their Role in Digital System Validation for Airborne Systems”*



Обучение верификации в нашем курсе

- Курс **Верификация параллельных и распределенных программ** читается более 20 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



Курс даст в руки студентов

теорию

элементы технологии

инструментарий

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

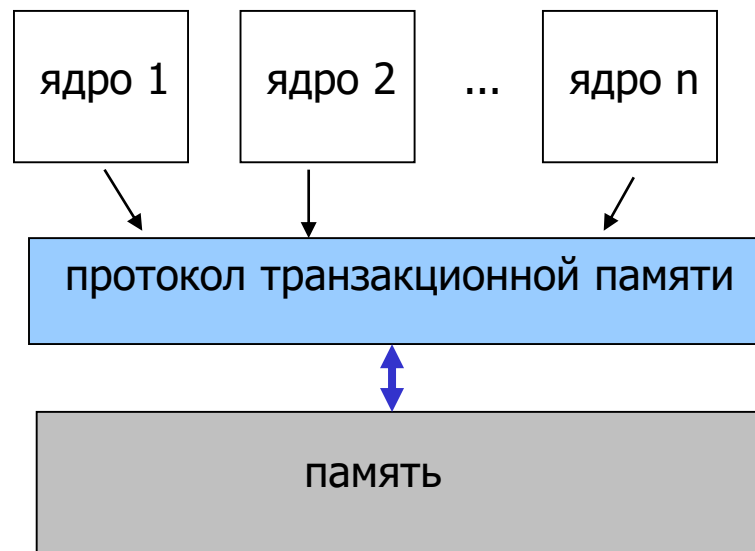
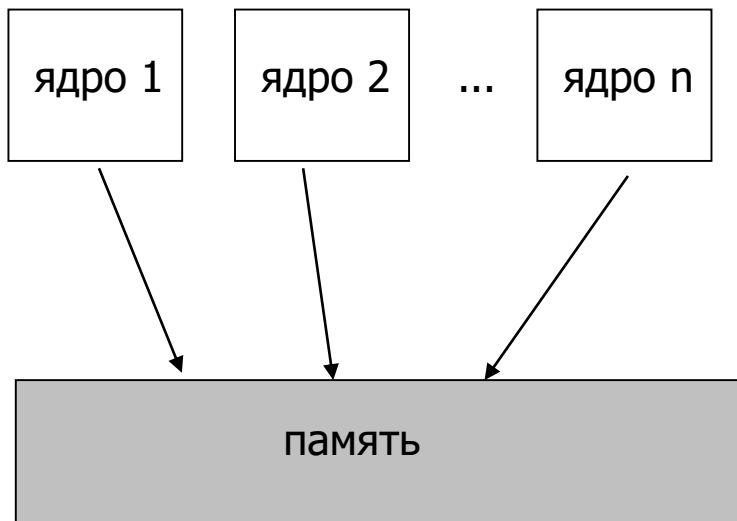
Success Story. Транзакционная память



Многоядерный процессор

- Многоядерные процессоры вывели параллельное программирование и связанные с ним проблемы ошибок в параллельных программах "в народные массы"
- Как решать проблемы ошибок параллельных программ в многоядерных процессорах?

Транзакционная память – одно из решений проблем параллелизма



“Оконный” алгоритм транзакционной памяти



Michel Raynal has been a professor of computer science since 1981. At IRISA (CNRS-INRIA-University joint computing research laboratory located in Rennes), he founded a research group on Distributed Algorithms in 1983.

His research interests include distributed algorithms, distributed computing systems and dependability. His main interest lies in the fundamental principles that underlie the design and the construction of distributed computing systems. He has been Principal Investigator of a number of research grants in these areas, and has been invited by many universities all over the world to give lectures and tutorials on distributed algorithms and fault-tolerant distributed computing systems. He belongs to the editorial board of several international journals.

- Professor Michel Raynal has published **more than 95 papers** in journals (JACM, Acta Informatica, Distributed Computing, etc.); and more than 195 papers in conferences (ACM STOC, ACM PODC, ACM SPAA, IEEE ICDCS, IEEE DSN, DISC, IEEE IPDPS, etc.). He has also written **seven books** devoted to parallelism, distributed algorithms and systems (MIT Press and Wiley).
- Michel Raynal has served in program committees for more than 85 international conferences (including ACM PODC, DISC, ICDCS, DSN, SRDS, etc.) and chaired the program committee of more than 15 international conferences. Michel Raynal got the IEEE ICDCS best paper Award three times in a row: 1999, 2000 and 2001.
- **Michel Raynal has suggested a transaction memory algorithm**



Success Story. Ошибка в протоколе

- В трудах конференции PaCT'09 была напечатана работа Мишеля Райнала по оконному алгоритму транзакционной памяти **и его (аналитическому) доказательству**
- Imbs D.; Raynal M. *Software Transactional Memories: an Approach for Multicore Programming*. In 10th International Conference on Parallel Computing Technologies (PaCT'09), Aug., 2009, Novosibirsk
- Эта работа была отдана студенту 4 курса Алексею Беляеву для проверки корректности протокола в рамках его НИРС
- Студент построил представление протокола на входном языке системы Spin, сформулировал требования к протоколу и обнаружил в нем ошибку. Контрпример позволил понять причину ошибки
- Мы уведомили Мишеля об ошибке, привели ее описание
- В опубликованной в журнале The Journal of Supercomputing Technologies новой статье профессор М.Райнал в разделе Acknowledgements принес благодарность студенту Алексею Беляеву за найденную ошибку в первоначальной версии протокола
- Студент опубликовал результат в журнале Научно-Технические Ведомости

Software Transactional Memories: an Approach for Multicore Programming

Damien IMBS Michel RAYNAL
IRISA, Campus de Beaulieu, 35042 Rennes Cedex, France
{damien.imbs|raynal}@irisa.fr

...

To illustrate these notions, several STM protocols have been presented. All are based on a logical global clock, and on locks. The last protocol has been proved correct. Such a proof constitutes a step in establishing solid theoretical foundations for STM systems.

Acknowledgements

We want like to thank Professor Yuri Karpov and student Alexey Belyaev who find a bug in a previous version of our algorithm when they design a Promela model of our algorithm and check it with the Spin software.



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 (аналитическое 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)

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

Э. М. Кларк, н.а. О. Грамберг
Д. Пелед

ВЕРИФИКАЦИЯ МОДЕЛЕЙ
ПРОГРАММ:
Model Checking



Michael Huth and Mark Ryan

Second Edition / Logic in Computer Science
Modelling and Reasoning about Systems



Ю. Г. Карпов

MODEL CHECKING
Верификация параллельных
и распределенных
программных систем

«Автор будет считать свою задачу выполненной, если его книга будет в каждой библиотеке инженерных вузов и факультетов информатики и электротехники. Сказавшись, однако, что задача об информатике и электротехнике была выполнена, автор был вынужден заметить, что в настоящее время информатика и электротехника являются дисциплинами, которые имеют много общего, но не являются единой наукой. Поэтому автор решил объединить их в одну дисциплину, которую он назвал информатикой».

Mordechai Ben-Ari

Principles of the
Spin Model Checker

Foreword by Gerard J. Holzmann

- **Donald Knuth:** "A person does not really understand something until after teaching it to a computer, i.e. expressing it as an algorithm.. . An attempt to formalize things as algorithms leads to a much deeper understanding than if we simply try to comprehend things in the traditional way"

- Курсовая работа (индивидуальное задание) 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 в Moodle
- Книга
Ю.Г.Карпов. *Model checking*
- Пособие по установке системы Spin и верификации систем с ее помощью будет выложено в Moodle. В пособии определена одна типовая задача из нескольких параллельных процессов управления светофорами на перекрестках. Задача параметризуется топологией перекрестка

Организационные вопросы

- По каждому виду учебной работы выставляются баллы:
 - Курсовая работа (индивидуальная работа) от -5 до + 15 баллов (норма – 1)
 - Письменный экзамен до 70 баллов
- Курсовая работа (индивидуальная работа)
 - 1 получают студенты, представившие стандартное решение и понимающие его
 - Больше 1 (до 15) баллов за курсовую работу получают студенты, представившие оригинальное глубокое решение и полностью его понимающие.
 - Меньше 1 (до -5) баллов получают студенты, плохо понимающие, что они представили в качестве решения
 - Зачета не получают студенты, **не понимающие**, что они представили
- Теоретический курс – письменный **экзамен** (решение задач) 3 часа. Решение каждой задачи оценивается в баллах. Можно максимально получить 70 баллов
- Общая оценка выводится по сумме набранных баллов:
 - | | | | |
|---------|--------|--------|------|
| 1 - 19 | баллов | оценка | – 1; |
| 20 - 39 | | | – 2; |
| 40 - 54 | | | – 3; |
| 55 – 69 | | | – 4; |
| 70 - 85 | | | - 5 |



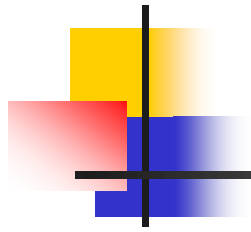
План курса

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



Заключение

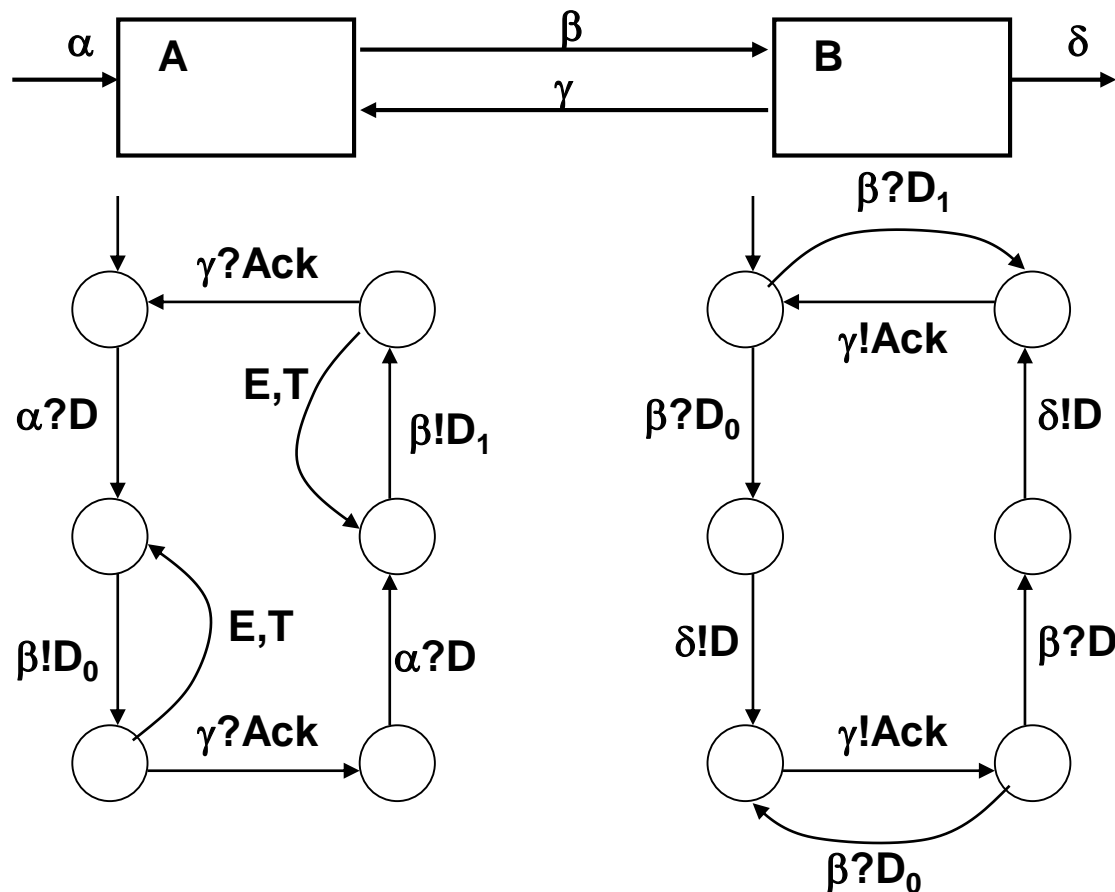
- Необходимость верификации повышается с увеличением сложности программ, распространением параллелизма, увеличением цены ошибки в критических применениях программ
- В области верификации **Model checking** – это технологический прорыв. Зрелость техники Model Checking – АВТОМАТИЗАЦИЯ и внедрение ее в этапы разработки кода
- СЕГОДНЯ верификация – один из этапов разработки ПО.
- Model checking - “*реабилитация формальных методов*”.
- Исследования по применению этих формализмов в верификации и в других областях продолжаются



Спасибо за внимание

Пример ошибки в протоколе: протокол PAR

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



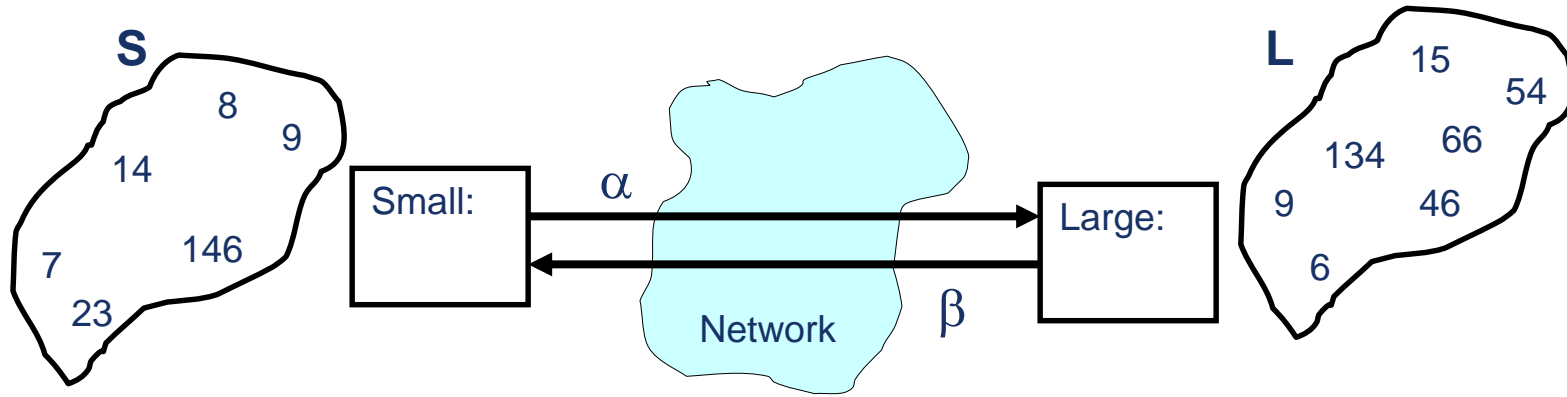
Упражнение: Найти ситуации, при которых происходят ошибки

Если использовать нумерацию не только 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);  
od
```

▪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 );  
od
```