

# 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](mailto:karpov@dcn.infos.ru)



# План курса

---

1. Введение
2. Метод Флойда-Хоара доказательства корректности программ
3. Исчисление взаимодействующих систем (CCS) Р.Милнера
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

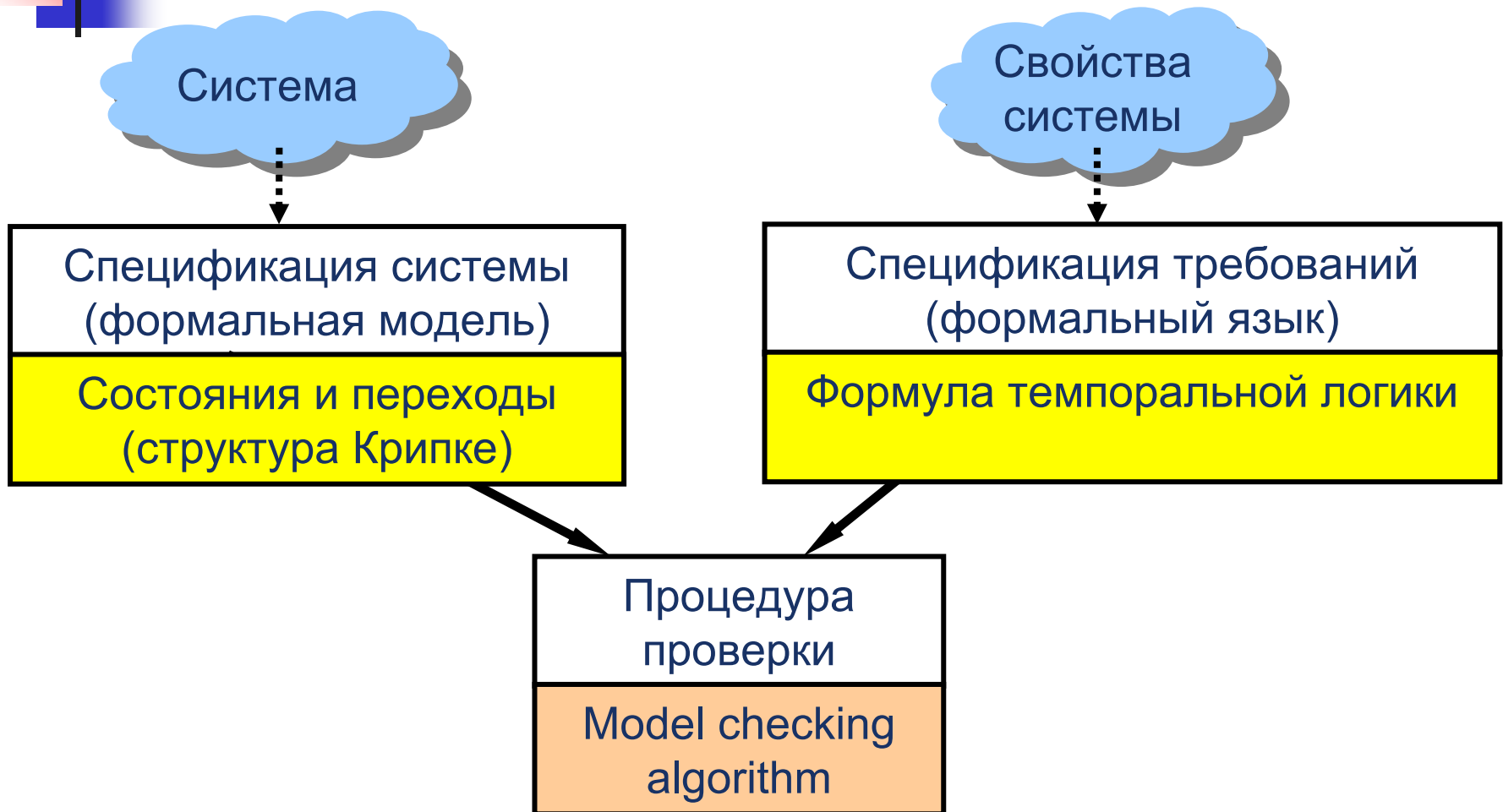


## *Лекция 10*

---

*Применения метода верификации Model checking*

# Model Checking





# Области применения идей Model checking

---

- Оптимизирующие компиляторы
- Алгоритмы на графах
- Бизнес процессы
- Учебные планы
- Системы прав доступа
- Анализ веб-документов и сервисов
- Решение логических задач
- Планирование в ИИ
- Многоагентные системы
- Анализ криптографических протоколов
- Анализ стохастических систем
- ...



# Model checking при разработке компиляторов

- $n: (x:=e) \Rightarrow \varepsilon$  if  $n \models AX(\neg EF \text{ use}(x))$   
Оператор  $(x:=e)$  можно выбросить из программы, если  $x$  не используется во всех следующих состояниях программы на всех ее путях
- $n: (x:=v) \Rightarrow (x:=C)$  if  $n \models A(\neg \text{def}(v) \cup^{-1}(\text{def}(v) \wedge \text{stmtnt}(v:=C))) \wedge \text{const}(C)$   
Оператор  $'x:=v'$  можно заменить на другой оператор  $'x:=C'$ , если  $C$  является литеральной константой, и если на всех путях программы к этому оператору переменная  $v$  не была определена до того места, где она была определена, как константа  $C$  (constant propagation)
- На основе подобных 'условных' правил подстановок в [1] был предложен подход к разработке оптимизирующего компилятора
- Условия, включающие операторы темпоральной логики, могут быть использованы подобным образом при оптимизации кода, поиске 'dead code' и подобных некорректностей в программах

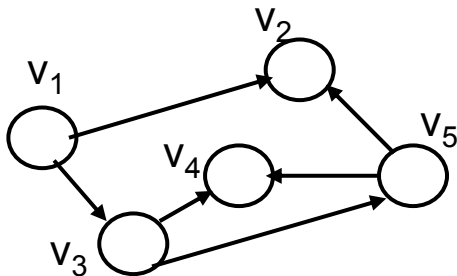
[1] D.Lacey, O. de Moor. Imperative program transformation by rewriting // CC'01 – Proc. of the 10<sup>th</sup> Int Conf on Compiler Construction, 2001, p.52-68

- LACEY, D., AND DE MOOR, O. Imperative program transformation by rewriting. In *CC '01: Proceedings of the 10th International Conference on Compiler Construction* (2001), Springer-Verlag, pp. 52–68.
- Lacey describes a compiler framework in which a compiler is specified in terms of conditional rewrite rules; the conditions can be specified using temporal logic.
- Пример:
- $n : (x := e) \Rightarrow \varepsilon$
- **if**  $n \models AX(\neg EF(use(x)))$
- Смысл этой спецификации  $x := e$  can be eliminated if  $x$  is not used on any path from the assignment. Note that the following simplification of the rule, although more intuitive, is incorrect:
- $n : (x := e) \Rightarrow \varepsilon$
- **if**  $n \models \neg EF(use(x))$
- потому что будущее включает настоящее, и So, this last rule would fail to eliminate a rule such as  $x := x + 1$ , even when  $x$  is never used again, because  $x$  is used in the rule itself.

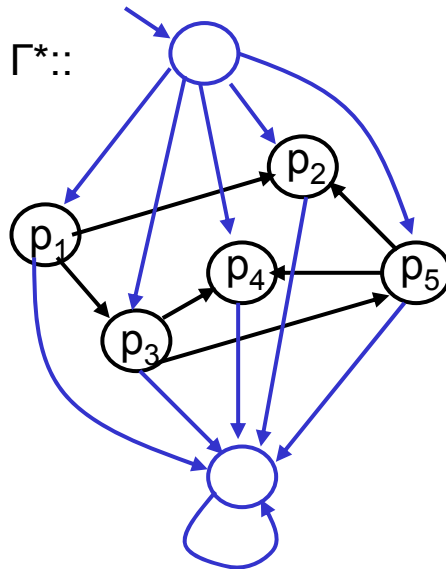
# Алгоритмы на графах

Гамильтонов путь – это путь, проходящий через каждую вершину графа точно по разу.  
NP-трудная проблема

$\Gamma::$



$\Gamma^*::$



Вводим дополнительные вершины, ребра, метки в каждой вершине графа

$$\Phi = E [ Fp_1 \wedge Fp_2 \wedge \dots \wedge Fp_n \wedge G(p_1 \Rightarrow XG \neg p_1) \wedge G(p_2 \Rightarrow XG \neg p_2) \wedge \dots \wedge G(p_n \Rightarrow XG \neg p_n) ]$$

$\Phi$  истинна на расширенном графе  $\Gamma^*$ , если в  $\Gamma$  есть гамильтонов путь

Проверим свойство: на  $\Gamma^*$  выполняется ОТРИЦАНИЕ формулы  $\Phi$ :

$$\neg \Phi = A \neg [ Fp_1 \wedge Fp_2 \wedge \dots \wedge Fp_n \wedge G(p_1 \Rightarrow XG \neg p_1) \wedge G(p_2 \Rightarrow XG \neg p_2) \wedge \dots \wedge G(p_n \Rightarrow XG \neg p_n) ]$$

Если гамильтонов путь существует, то система верификации найдет его и выдаст как опровержение формулы  $\neg \Phi$

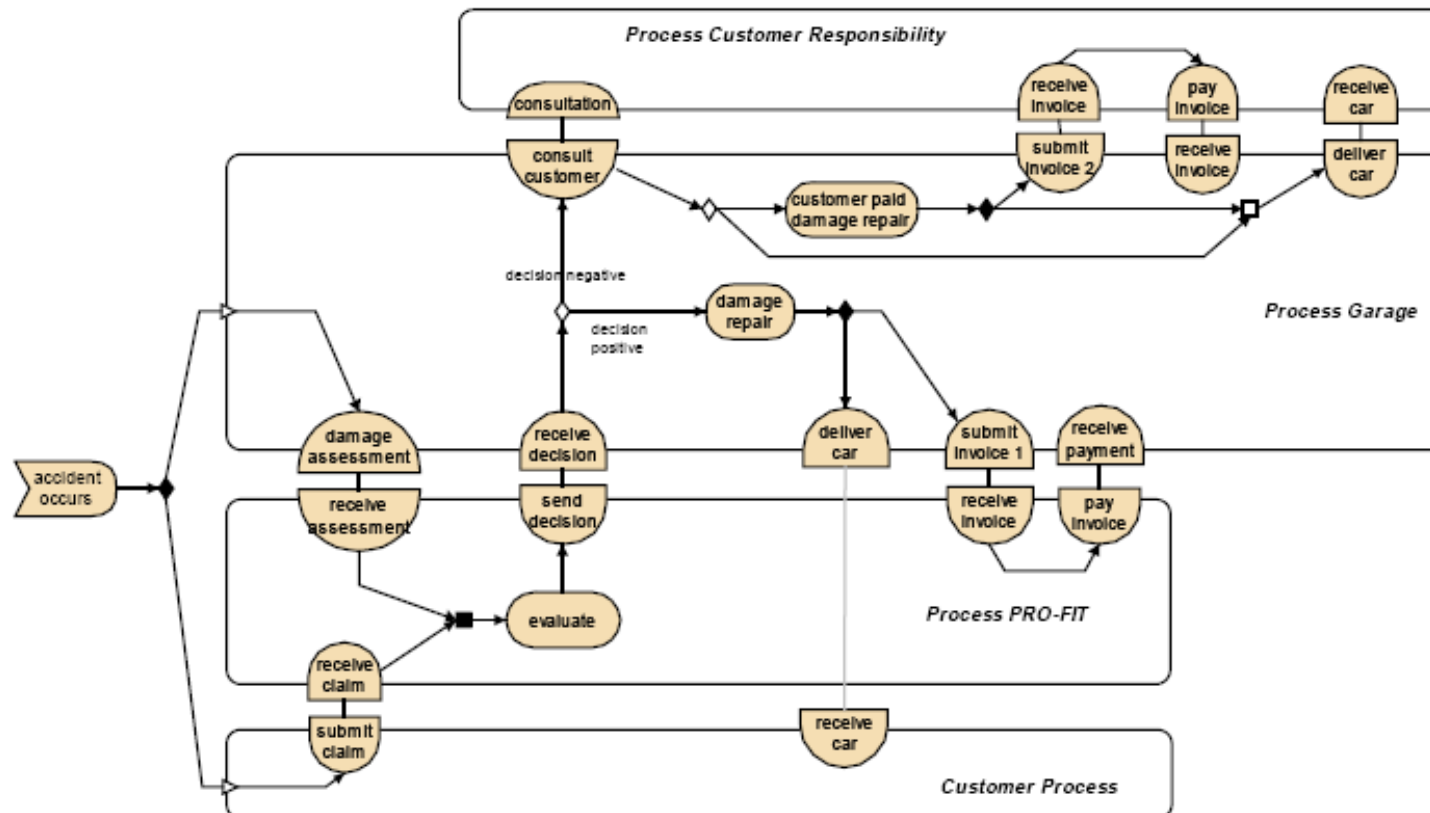


# Анализ бизнес процессов (в частности, в e-commerce)

*Model Checking for Managers. Wil Janssen et.al.(1999).*

Предложен графический язык описания бизнес-процессов, задания требуемых свойств в виде шаблонов и проверка этих свойств. Все транслируется в Spin.

Пример – процессы в страховой компании при возникновении страхового случая при ДТП





## Примеры проверяемых свойств

- Машина ремонтируется только после того, как принято решение об удовлетворения иска (причем это решение может быть и не принято):

$G ( (\neg \text{damage\_repair}) \text{ W } \text{claim\_approved})$

- Иск до 10 тысяч долларов для клиента N4 всегда будет удовлетворен:

$G ( (\text{claim\_below\_10} \wedge \text{customer\_4}) \Rightarrow \text{F claim\_approved})$

- Из гаража машина возвращается владельцу только после ее ремонта:

$G ( (\neg \text{deliver\_car}) \text{ U } \text{damage\_repair})$

- Если иск отвергнут, машину не будут ремонтировать:

$G ( (\text{claim\_rejected} \Rightarrow G \neg \text{damage\_repair}) )$

- Гараж не представит счет за ремонт, если иск отвергнут:

$G ( (\text{claim\_rejected} \Rightarrow G \neg \text{submit\_invoice})$





# Верификация учебных планов

---

M. Baldoni, E. Marengo. Curriculum Model Checking: Declarative Representation and Verification of Properties // LNCS 4753 (2007)

**Из аннотации:** Когда предлагается учебный план, важно проверить по крайней мере три его аспекта:

- (a) что он позволяет учащимся достичь их целей
- (b) что он согласуется с целями, которые объявляет учебное заведение
- (c) что последовательность курсов не имеет разрывов компетенций

Предлагается подход к формальной спецификации целей при разработке учебных планов на основе графического языка, базирующегося на темпоральной логике линейного времени



# Логические задачи: фермер, волк, коза и капуста

**Старинная задача:** Фермеру нужно перевезти через реку **волка, козу и капусту**. Лодка вмещает кроме фермера еще только что-то одно. Но без фермера нельзя волка оставлять с козой, а козу – с капустой

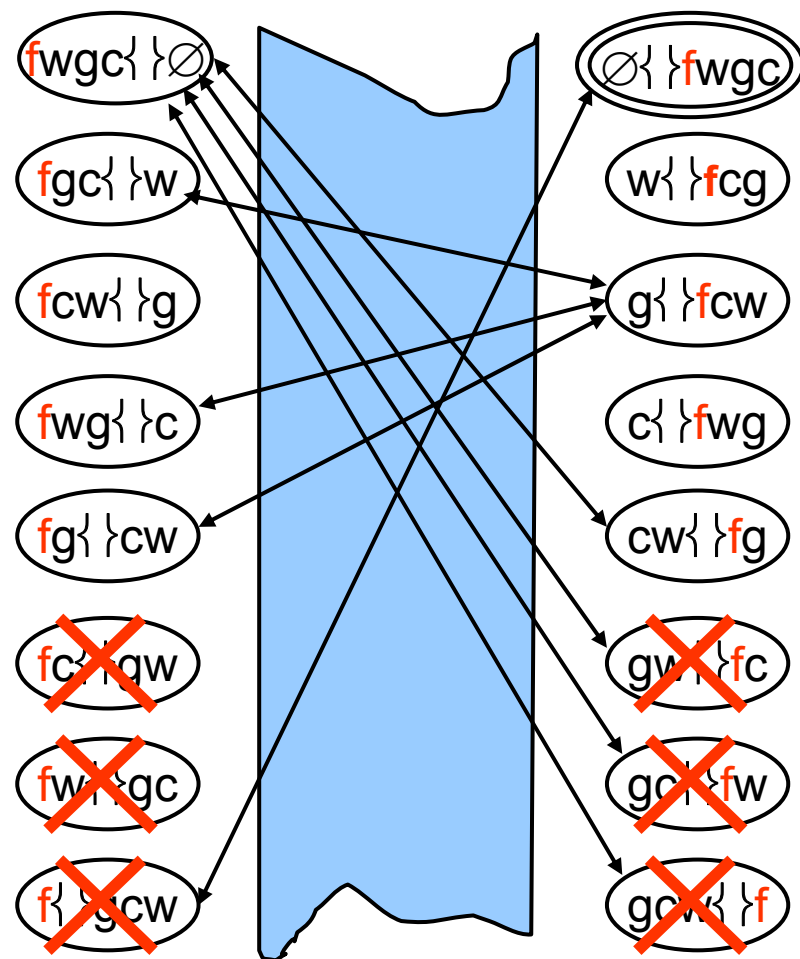


Какую абстракцию построить для решения?

Хотя система не дискретная, для решения задачи можно построить абстрактную модель - систему переходов, и исследовать возможные пути

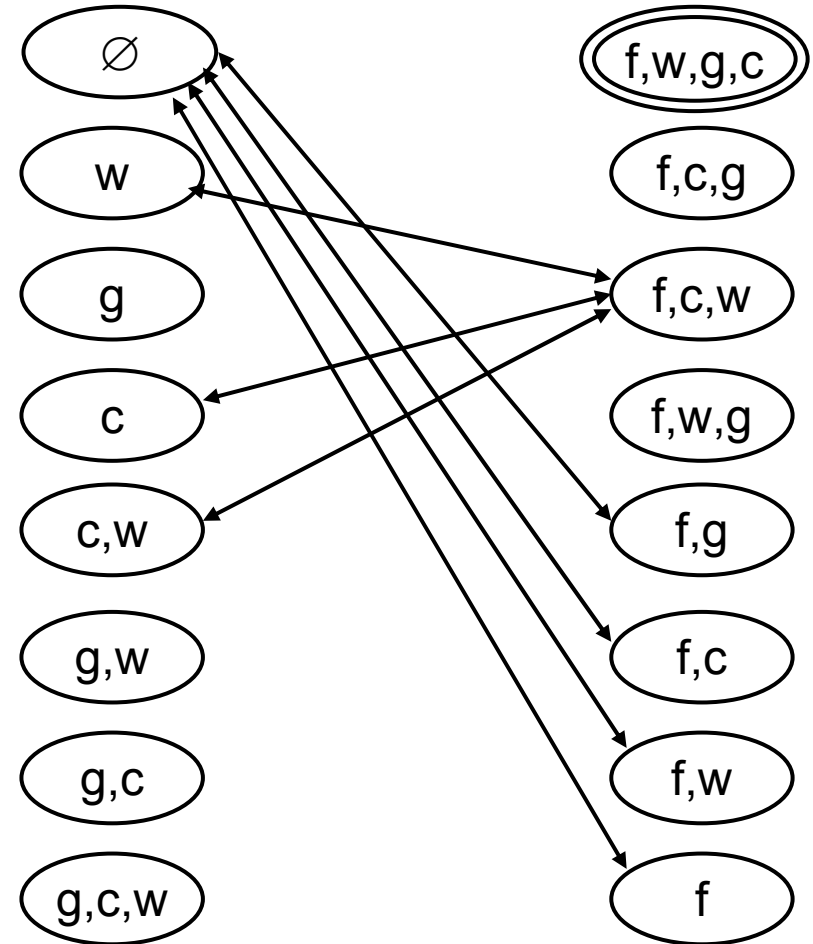
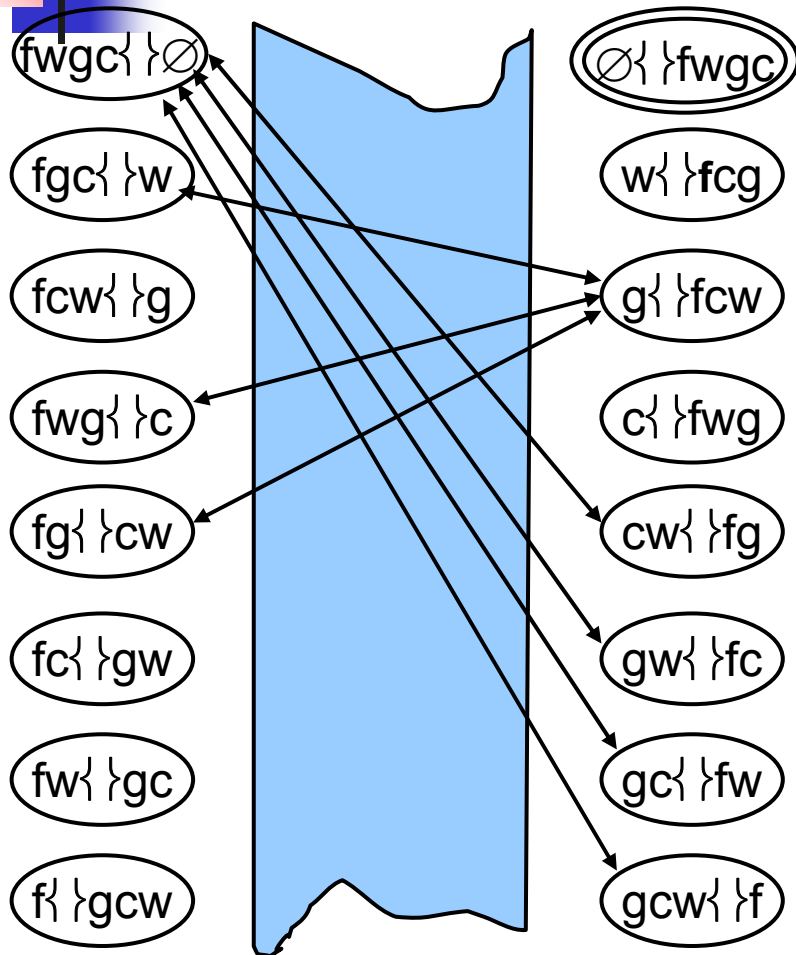
Состояние:

$w \setminus \setminus cgf$



# Model checking для этой проблемы

Структура Крипке:



Атомарные предикаты:

$f=1$  – фермер на правом берегу,  $w=1$  – волк на правом берегу,  
 $c=1$  – капуста на правом берегу,  $g=1$  – коза на правом берегу



# Свойства системы фермер-волк-коза-капуста

1. В заключительное состояние можно попасть

$$EF \text{ fwgc}$$

2. Существует такое решение задачи, при котором, если коза переедет на ту сторону, она там навсегда и останется

$$E[(F \text{ fwgc}) \ \& \ (\neg g \ U \ Gg) ]$$

3. Хотя бы на одном пути, ведущем к цели, ограничения соблюдаются

$$E [ ((g \equiv c) \vee (g \equiv w) \Rightarrow (g \equiv f)) \ U \ \text{fwgc} ]$$

4. При любом варианте решения задачи фермер должен будет возвращаться через реку один

$$A[(F \text{fwgc}) \Rightarrow F[ (f \ \&X \neg f) \wedge (w \ \&X w \vee \neg w \ \&X \neg w) \wedge (g \ \&X g \vee \neg g \ \&X \neg g) \wedge (c \ \&X c \vee \neg c \ \&X \neg c) ] ]$$



# Решение проблемы фермер-волк-коза-капуста с помощью системы Spin

Проверяемое свойство:  $E [ [ (g \equiv w) \vee (g \equiv c) \Rightarrow (g \equiv f) ] U fwgc ]$

существует путь в нужное состояние, на котором в любом состоянии если коза на одном берегу с волком или с капустой, то фермер на том же берегу

Лучше проверить отрицание формулы:

$A \neg [ [ (g \equiv w \vee g \equiv c) \Rightarrow (g \equiv f) ] U fwgc ]$  - хотим проверить, что решения нет

Если Spin найдет опровержение, он выдаст путь – решение задачи

Алгоритм на языке Promela:

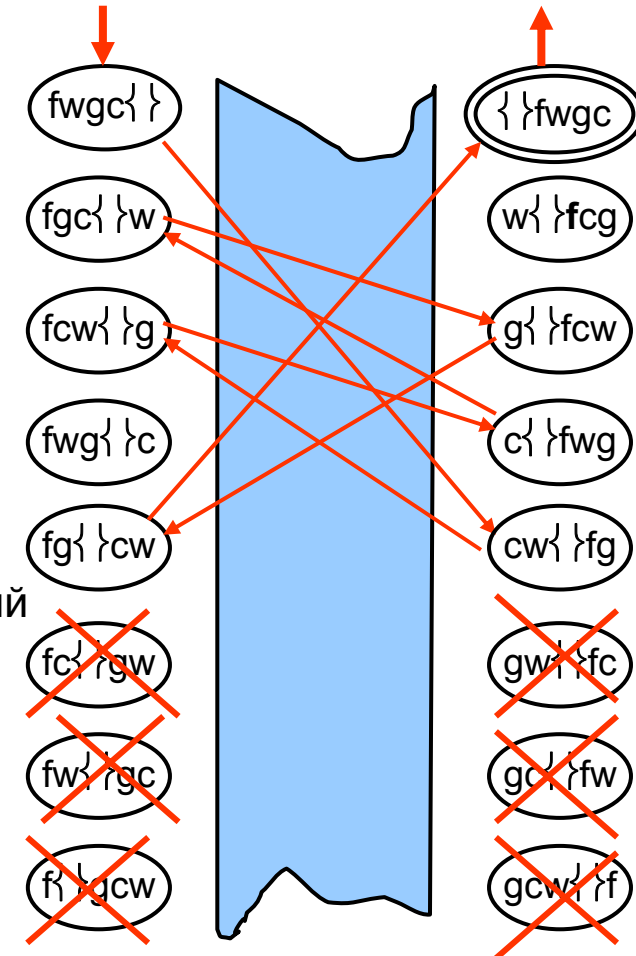
```
define bool f=false, c=false, g=false, w=false;
do
  /* из возможных альтернатив недетерминировано выбирается одна */
  :: True      => f:=!f;          /* фермер может переехать сам */
  :: f = w    => f:=!f; w:=!w;   /* или с волком */
  :: f = c    => f:=!f; c:=!c;   /* или с капустой */
  :: f = g    => f:=!f; g:=!g;   /* или с козой */
  :: f && w && g && c => break;    /* останов, когда все переправятся */
od
```



# Траектория поведения, не удовлетворяющая LTL формуле

## Контрпример!

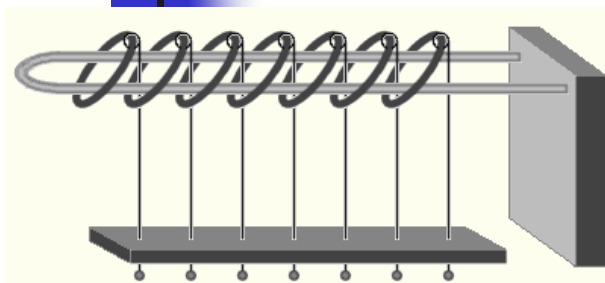
		<b>river:0</b>		
f 0	w 0	g 0	c 0	← Исходное состояние
f 1	w 0	g 1	c 0	← Фермер с козой – на правый
f 0	w 0	g 1	c 0	← Фермер один – на левый
f 1	w 1	g 1	c 0	← Фермер с волком – на правый
f 0	w 1	g 0	c 0	← Фермер с козой – на левый
f 1	w 1	g 0	c 1	← Фермер с капустой – на правый
f 0	w 1	g 0	c 1	← Фермер один – на левый
f 1	w 1	g 1	c 1	← Фермер с козой – на правый
		<b>OK!</b>		
		<b>72</b>		



Решение сгенерировано системой верификации!

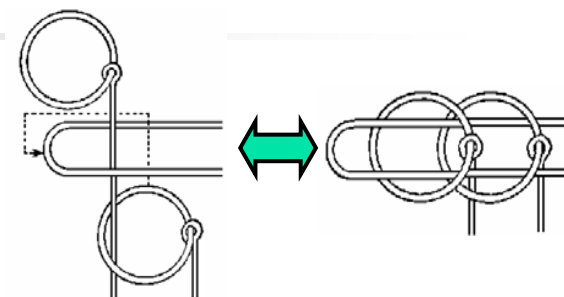
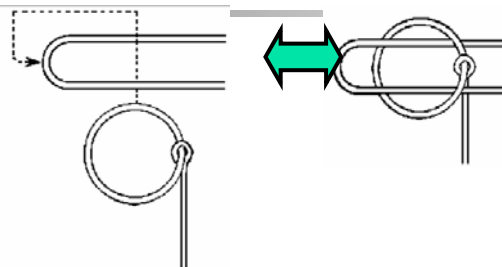


# Головоломка "Китайские кольца"



Головоломка – снять все кольца с петли и/или надеть их

Программа на Promela



1. Первое кольцо всегда можно снять или надеть
2. Если сняты все до N-1, а N – нет, то можно снять либо надеть N+1

```
define bool x1=true,x2=true,x3=true,x4=true,x5=true; /* все надеты */
do
  :: true  $\Rightarrow$  x1=!x1; /* первое можно всегда снять либо надеть */
  :: x1  $\Rightarrow$  x2=!x2; /* 1 надето  $\Rightarrow$  2 можно снять/надеть */
  :: !x1&& x2  $\Rightarrow$  x3=!x3; /* 1-снято, 2-надето  $\Rightarrow$  3 можно снять/надеть */
  :: !x1&&!x2&& x3  $\Rightarrow$  x4=!x4; /* 1,2 сняты, 3-надето  $\Rightarrow$  4 снять/надеть */
  :: !x1&&!x2&&!x3&& x4  $\Rightarrow$  x5=!x5; /* 1,2,3 сняты, 4-надето  $\Rightarrow$  5 снять/надеть */
  :: !x1&&!x2&&!x3&&!x4&& x5  $\Rightarrow$  break; /* останов, когда все сняты */
od
```

Свойство: **A G** ( $\forall_i x_i$ ) на всех путях всегда хотя бы одно кольцо останется

# Планирование как Model Checking

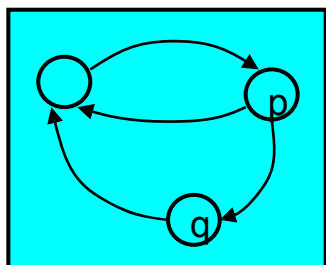
Техника синтеза плана из формальной модели и логической спецификации цели

Структура Крипке –  
система с конечным  
числом состояний

Свойство системы –  
Темпоральная  
формула

Среда – предст.  
системой с конечным  
числом состояний

Цель  
(представляется  
формулой TL)

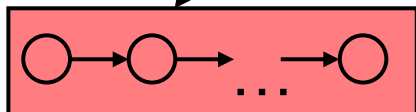


**$AG(p \Rightarrow Fq)$**

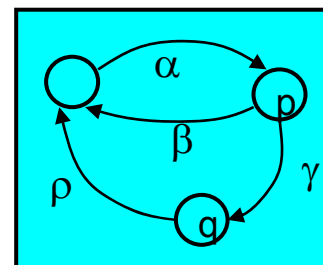
**Model checker**

Нет!

Да!



Контрпример  
Ю.Г.Карпов

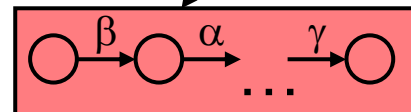


**Достичь g  
нельзя!  
 $AG\neg g$**

**Планировщик**

Есть план!

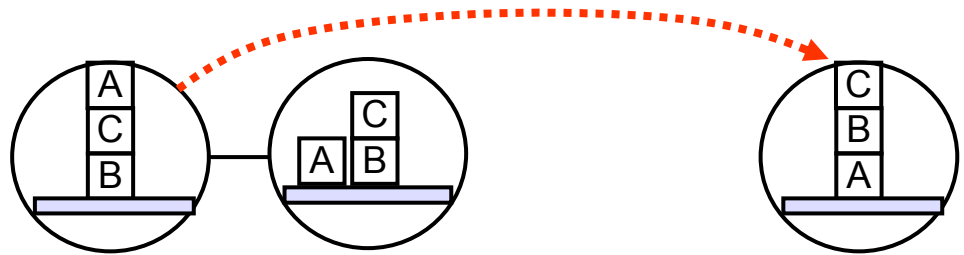
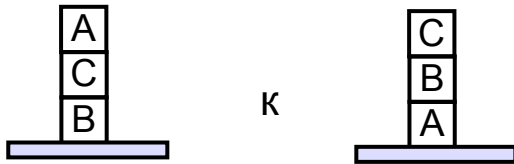
Нет  
плана!



План

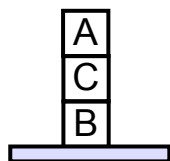
# Классическая задача ИИ: Blocks World

Можно ли и как перейти от

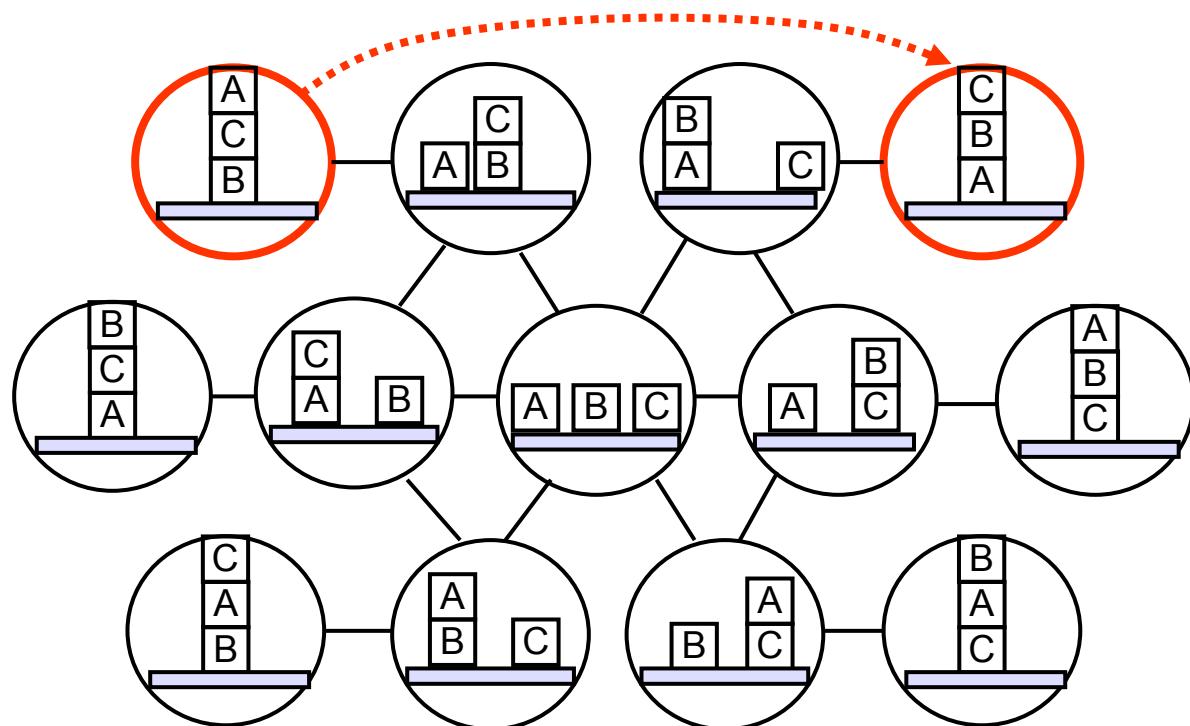
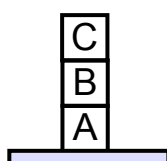


# Классическая задача ИИ: Blocks World

Можно ли и как перейти от



к



Проблема представляется в виде структуры Крипке

ИЛИ

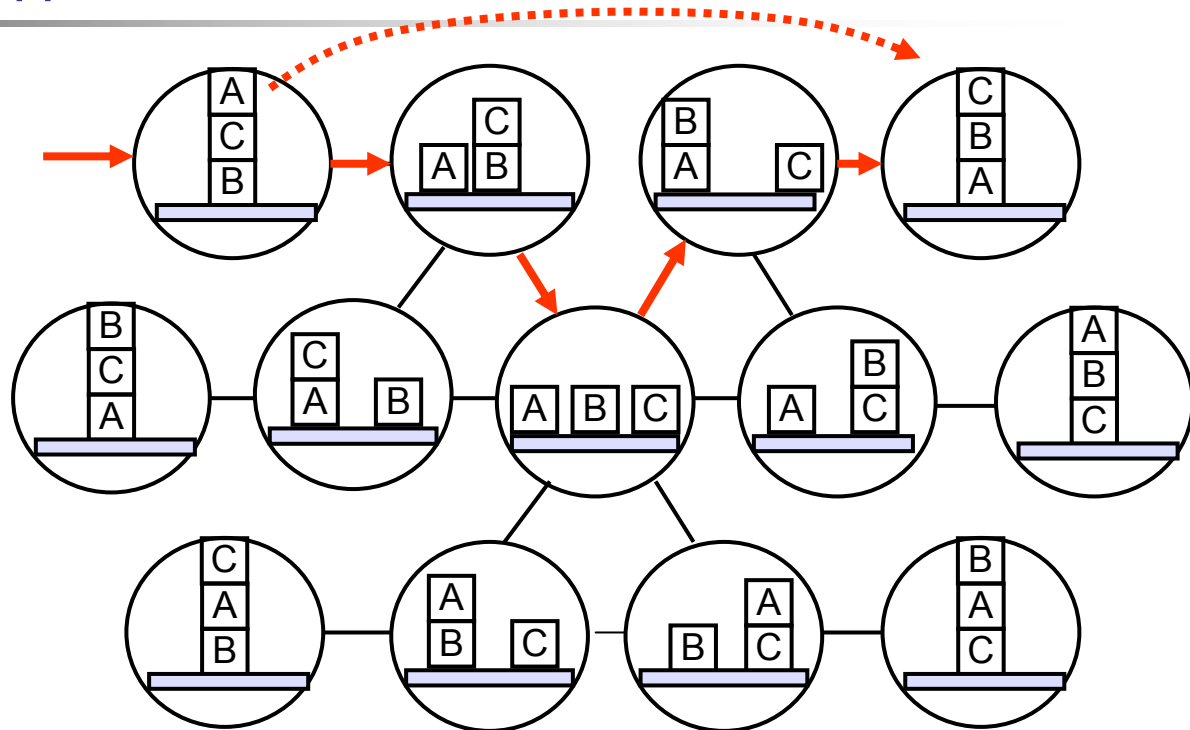
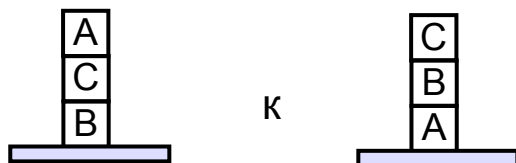
на входном языке спецификации для системы верификации

ИЛИ

на специально разработанном для этой области языке с транслятором во входной язык системы верификации

# Классическая задача ИИ: Blocks World

Можно ли и как перейти от



*Программа на Promela*

```
do
  :: A==0&&B==0&&C==0 -> A=1; B=2 /* Все на столе->B поместить на A */
  :: A==0&&B==1&&C==2 -> A=3      /*A поместить на C, стоящую на B */
  ... /* проверка и выполнение всех 32х возможных вариантов хода */
  :: A==1&&B==2&&C==3 -> break    /* финальное состояние */
od
```

# Ханойская башня

Всего  $3^3=27$  состояний ( $3^N$ )

Каждый из N дисков –  
на любой из 3-х спиц. Для 5  
дисков – 625 состояний

Ограничение: перекладывать по  
одному, только меньший диск  
может лежать на большем

Задача также сводится  
к поиску пути из  
начального состояния  
к цели в пространстве  
состояний среды

Построить алгоритм на языке Promela, который находит решение –  
путь в пространстве состояний к цели



## Программа на Promela

- Задаем три булевых массива  $A[0, N-1]$ ,  $B[0, N-1]$ ,  $C[0, N-1]$  и три целых  $nA$ ,  $nB$ ,  $nC$ , показывающих номера минимальных дисков, находящихся на спицах.
- Начальное состояние  $nA=0$ ,  $nB=N$ ,  $nC=N$ ;
- Искомое состояние  $nA=N$ ,  $nB=N$ ,  $nC=0$ ;
- Переходы:
  - из A: если  $nA < N$ : если  $nA < nB$ , то переход  $A \rightarrow B$ ; если  $nA < nC$ , то переход  $A \rightarrow C$
  - То же из B и из C

```
do
  :: nA < N ->                                     /* на A что-то есть */
    if :: nA < nB -> ...                             /* перенос с A на B */
      :: nA < nC -> ...                             /* перенос с A на C */
      :: else -> break
    fi
  :: nB < N ->                                     /* на B что-то есть */
    ...
  :: nC < N ->                                     /* на C что-то есть */
    ...

  :: nA == N && nB == N && nC == 0 -> break          /* финальное состояние */
od
```

Ю.Г.Карпов



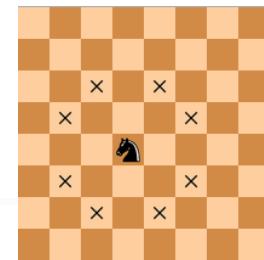
# TL в системах планирования в ИИ

- LTL для выражения свойств
  - $G(on(B,C) \Rightarrow (on(B,C) \cup on(A,B)))$ 
    - *как только достигнем состояния, в котором куб B на кубе C, куб B останется на C до тех пор, пока A не будет помещен на B*
  - $\forall x. (onTable(x) \Rightarrow G onTable(x))$ 
    - *если какой-нибудь объект появится на столе, то он там навсегда останется*
- CTL для описания планов
  - “weak” planning” –  $EFgoal$ 
    - *план, который может достигнуть цели*
  - “strong planning” –  $AFgoal$ 
    - *план, который гарантирует достижение цели*
  - “strong cycling planning” –  $AGEFgoal$ 
    - *итеративная стратегия проб и ошибок*

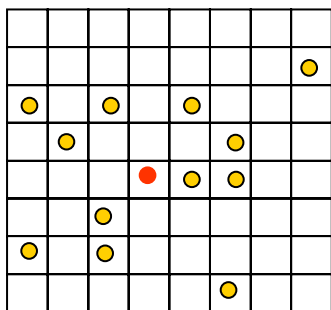
G.Bacchus, F.Kabanza. Using TL to express search control knowledge for planning // AI v. 116(2000)



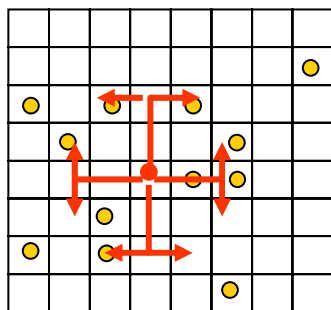
# Обход конем шахматной доски



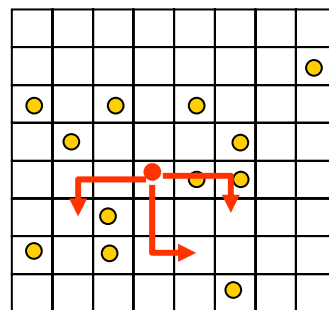
Можно ли обойти шахматную доску размером  $N \times N$  конем из начальной позиции  $(I, J)$ , побывав на каждой клетке только один раз?



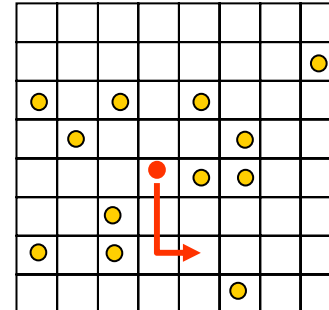
Очередное состояние



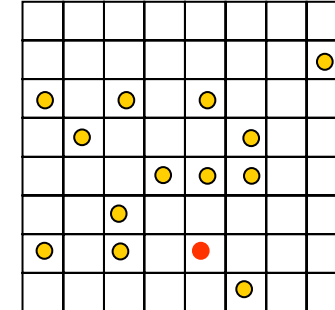
Возможные ходы коня



Где еще не были



Выбираем один ход



Следующее состояние

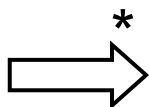
```
define K = 1; A[]=True; A[I,J] = false;
do
  :: I>2 && J<N && A[I-2, J+1] => I=I-2; J=J+1; A[I, J]=False; K++;
  :: I>2 && J>1 && A[I-2, J-1] => I=I-2; J=J-1; A[I, J]=False; K++;
  ... /* проверка и выполнение всех 8 возможных вариантов хода */
  :: K==N*N => break
od
```

Свойство:  $A \ G( K < N*N )$  на всех путях всегда  $K$  не достигнет значения  $N*N$

# Планирование в игре 15 и проверка моделей

При  $N=4$  состояний  $16! \sim 10^{14}$

12	10	3	1
8	9	16	2
5	11	8	7
6	4	14	15



1	2	3	4
5	6	7	8
9	10	11	2
13	14	15	16

12	10	3	1
8	9	16	2
5	11	8	7
6	4	14	15



...



12	10	16	1
8	9	3	2
5	11	8	7
6	4	14	15

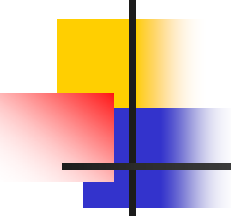
12	10	3	1
8	9	2	16
5	11	8	7
6	4	14	15

Программа на *Promela* для произвольного  $N$

```
do
  /* I, J - координаты пустого места, с номером N*N */
  :: I > 1 ⇒ Swap( a[I, J], a[I-1, J] ); I = I - 1 /* пустая вверх */
  :: I < N ⇒ Swap( a[I, J], a[I+1, J] ); I = I + 1 /* пустая вниз */
  :: J > 1 ⇒ Swap( a[I, J], a[I, J-1] ); J = J - 1 /* пустая влево */
  :: J < N ⇒ Swap( a[I, J], a[I, J+1] ); J = J + 1 /* пустая вправо */
od
```

Свойство:  $A \models \bigwedge_{i,j \in \{1,N\}} (a[i,j] == (i-1)*N + j)$  на всех путях никогда не достигнем состояния, при котором все костяшки выстроятся по порядку





# Предположения в классической теории планирования

- Конечность области – конечное число состояний
- Неявность времени (время не влияет)
- Детерминизм (начальное состояние полностью определено, каждое действие приводит к единственному новому состоянию)
- Наблюдаемость (каждое состояние полностью наблюдаемо)
- Цель определена как множество заключительных состояний, проблема планирования – построить путь из начального в одно из заключительных состояний
- План – последовательность действий - строится один раз, область и ее структура не изменяются в процессе выполнения плана

Основная трудность – для практических задач пространство поиска огромно. Именно эта трудность преодолевается с использованием Model checking и символьным подходом с использованием BDD

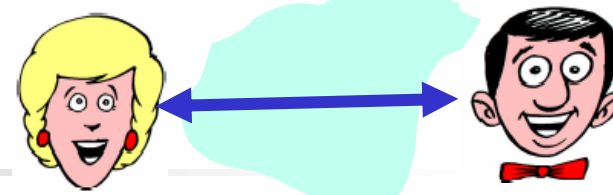


## Успехи этого направления

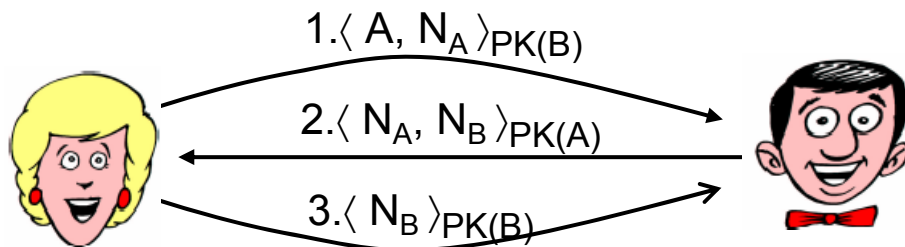
- Несколько ежегодных конференций
- Разработка мощных систем планирования (BDDPlan, CIRCA, MBP, Proplan, ...)
- Практические применения
  - построение контроллеров автоматических систем
  - power supply recovery systems
  - ...
- Int Workshop "Planning via Model Checking" '02:
  - *В настоящее время Model checking является одной из наиболее "горячих" тем информатики. Идея заключается в том, что модель системы проверяется на выполнение для нее логических требований. Изначально эта идея была использована для проверки корректности схем и компьютерных протоколов. Недавно эта же идея была применена для задач планирования в ИИ с замечательным успехом и привела к созданию мощных систем планирования*



# Needham-Schroeder протокол аутентификации



- КАК убедиться в том, что собеседник является тем, за кого он себя выдает?
- Только если он продемонстрирует знание СЕКРЕТА, *известного только ему*
- NS протокол решает эту проблему для открытого канала. После установления соединения и удостоверения, кто есть кто, собеседники могут установить сессию с секретным ключом!! Широко использовался (Kerberos основан на нем)



1. Alice генерирует nonce  $N_A$ , шифрует сообщение публичным ключом  $PK(B)$ , и посылает по несекретному каналу.  $N_A$  - первая половина "секрета",  $A$  - идентификация отправителя

2. Только Bob может прочесть первое сообщение. Он генерирует свой секрет - nonce  $N_B$ , шифрует сообщение ключом  $PK(A)$ , и посылает  $N_A$  и  $N_B$ .

То, что в сообщении присутствует  $N_A$ , убеждает Alice, что это Bob прислал – только он мог дешифровать первое сообщение. Она принимает  $N_A, N_B$  как общий секрет и уверена, что установила связь с B.

3. Alice отправляет  $N_B$ , зашифрованное публичным ключом  $PK(B)$ .

То, что в сообщении присутствует  $N_B$ , убеждает Bob, что это Alice прислала – только она могла дешифровать второе сообщение с его, Боба, nonce.

Поэтому Bob также принимает  $N_A$  и  $N_B$  как общий секрет и уверен, что установил связь с A.

# Атаки в криптографии

- Атака – последовательность действий нечестного участника, ведущих к нарушению свойств корректности криптопротокола
- Цель данного протокола – взаимная идентификация при сохранении конфиденциальности
  - Свойство корректности (“атаки нет”) можно выразить так:
    1. Если А и В завершились успешно, то А верит, что его партнер В iff В верит, что его партнер А
    2. Intruder не должен иметь  $\text{nonce}_A$  или  $\text{nonce}_B$  после того, как В завершился и установил, что его партнер – А

NS протокол опубликован в 1978 с доказательством корректности  
спустя > 10 лет были найдены атаки!

Еще через несколько лет Gavin Lowe (Oxford Uni) – с помощью системы верификации:

- построил модель протокола и действий одного из участников, не подчиняющегося правилам – он выполняет все возможные действия
- выполнил полный поиск в пространстве состояний и нашел несколько атак





# Найдем атаку с помощью Model checking

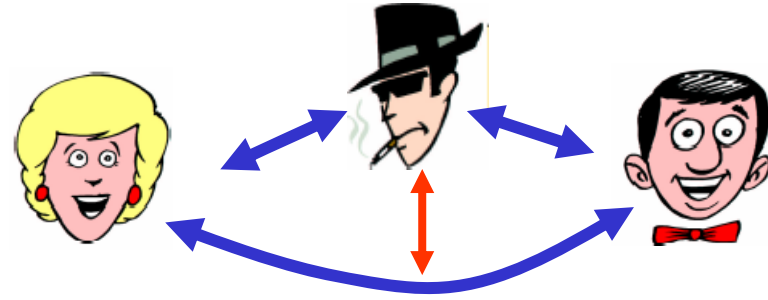
- Абстрагирование (чтобы иметь конечное число состояний)
  - вводится Intruder (I) как третий игрок, используем { A, B, I }
  - *примитивы шифрования считаем стойкими*
- Состояния
  - учитываем всю возможную информацию, которую в принципе может знать участник в каждом состоянии – (ключи, сообщения, данные, nonces)
- Переходы
  - сообщения, полученные участником, влияют на его знание (изменяют состояние участника)
- Используем Spin как средство верификации
  - проверяемое свойство “атаки нет” выразим формально:  
после сессии A верит, что его партнер B iff B верит, что его партнер A  
$$A \ G(\text{statusA=ok} \ \& \ \text{statusB=ok} \ \Rightarrow \ (\text{partnerA=agentB} \ \Leftrightarrow \ \text{partnerB=agentA}) \ )$$
  - после того, как B завершился и установил, что его партнер – A, intruder не имеет nonce A или B,  
$$A \ G \ \neg(\text{statusB=ok} \ \& \ \text{partnerB=agentA} \ \Rightarrow \ \text{I\_knows\_nonceA} \ \vee \ \text{I\_knows\_nonceB})$$



# Злоумышленник

## ■ Что может делать intruder

- выступать как “честный участник” обмена
- перехватывать и запоминать сообщения
- дешифровать сообщения, если ему уже известен ключ
- послать сообщение, которое он мог видеть ранее (даже если он не имеет ключа для его дешифрования)
- создать собственное сообщение, используя известную информацию



## ■ Как найти возможную атаку

- строим недетерминированную программу, в которой возможными альтернативами являются все возможности для Intruder'a
- поскольку Spin исчерпывающе проверяет все пути, которые существуют в модели, и ищет контрпример, то он сгенерирует атаку, если она существует

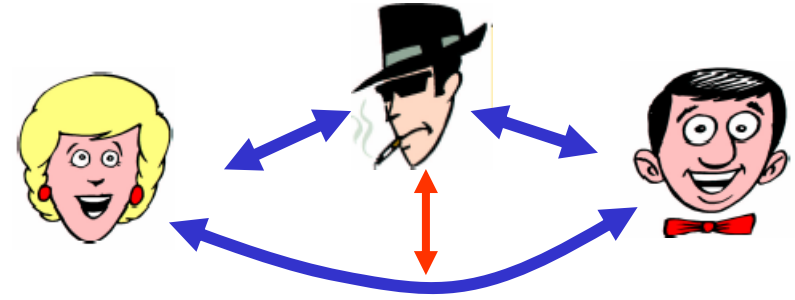


# Анализ NS протокола. Типы данных

## Перечислимый тип:

```
mtype={ok, err, msg1, msg2, msg3,  
      keyA, keyB, keyI,  
      agentA, agentB, agentI,  
      nonceA, nonceB, nonceI};
```

/\* все элементы, которые могут присутствовать в сообщении \*/



Определяем запись, состоящую из посылающего и принимающего агента, ключа и двух данных:

```
typedef mCrypt {sender, receiver, key, info1, info2}
```

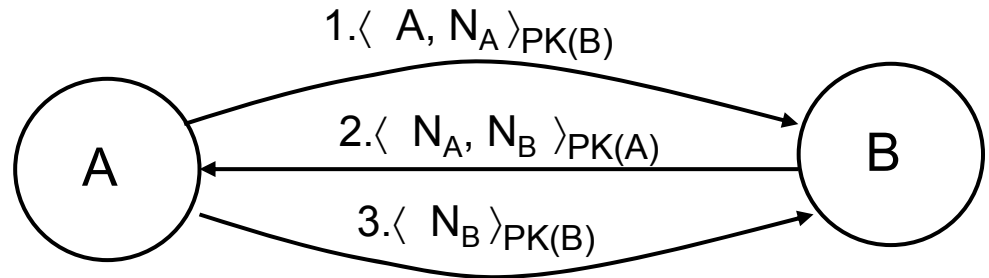
Сеть моделируется как два синхронных канала, разделяемых всеми тремя:

```
chan network[2] = [0] of { mtype, // N сообщения  
                          mCrypt // само сообщение - данные  
                          }
```

## Честный участник: Alice

```
mtype partnerA;  
mtype statusA = err;  
  
active proctype Alice() {  
    mtype pkey, pnonce;  
    mCrypt data;
```

```
// выбор получателя: А может захотеть установить контакт с В или с I  
if  
    :: partnerA = agentB; pkey = keyB;          /* msg от А примет В */  
    :: partnerA = agentI; pkey = keyI;          /* msg от А примет I */  
fi  
network [0]! (msg1, mCrypt {Alice, partnerA, pkey, agentA, nonceA } )  
network [1]? (msg2, data);  
  
    (data.key==keyA)&&(data.info1 == nonceA)  $\Rightarrow$  pnonce = data.info2;  
  
//посылает принятый nonce партнеру  
network [0]! (msg3, mCrypt {Alice, partnerA, pkey, pnonce, 0 } );  
statusA = ok;  
}
```

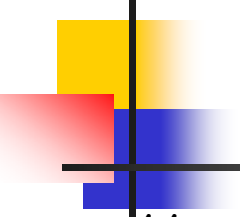




## Программа для Intruder - подслушивание

Описываем все действия, которые intruder может в принципе выбирать, т.е. те, которые возможны в каком-либо состоянии

```
bool I_knows_nonceA = false, I_knows_nonceB = false;
active proctype Intruder() {
  mtype msg, recpt;
  mCrypt data, intercepted;
  do
    // бесконечный цикл подслушивания и посылок сообщений
    :: network [0]? (msg, data) -> //если из канала подслушали сообщение
      // структура сообщения: <N, кто, кому, что>
    if
      :: intercepted = data; // перехваченные данные запоминаем
      :: skip; // либо нет
    fi;
    if
      :: data.key == keyI -> // если в принятом сообщении - наш ключ,
      if // то извлекаем чужой nonce
        :: data.info1==nonceA||data.info2==nonceA->I_knows_nonceA= true;
        :: data.info1==nonceB||data.info2==nonceB->I_knows_nonceB= true;
        :: else -> skip;
      fi
      :: else -> skip;
    fi;
  // затем посылаем свое сообщение
```



# Программа для злоумышленника – посылка своего сообщения

```
:: // посылаем сообщение, случайно собирая его из возможных кусков
if      // произвольно выбираем тип сообщения
:: msg = msg1;
:: ... // то же для msg2 и msg3
fi;
if      // произвольно выбираем получателя, выдавая себя за А, В, или I
:: data.sender=agentA -> data.receiver = agentB; data.key = pkeyB;
...//если выдаем себя за В или I, выбираем произвольно получателей
fi;
if      // собираем свое сообщение data для отправки
:: data = intercepted; // перехваченные данные шлем без изменения
:: if      // ЛИБО собираем содержимое сообщения data
:: data.info1 = agentA;
:: data.info1 = agentB;
:: data.info1 = agentI;
:: I_knows_nonceA -> data.info1 = nonceA;
:: I_knows_nonceB -> data.info1 = nonceB;
:: data.info1 = nonceI;
fi
... // то же для data.info2
fi;
network [1]! (msg, recpt, data);
```

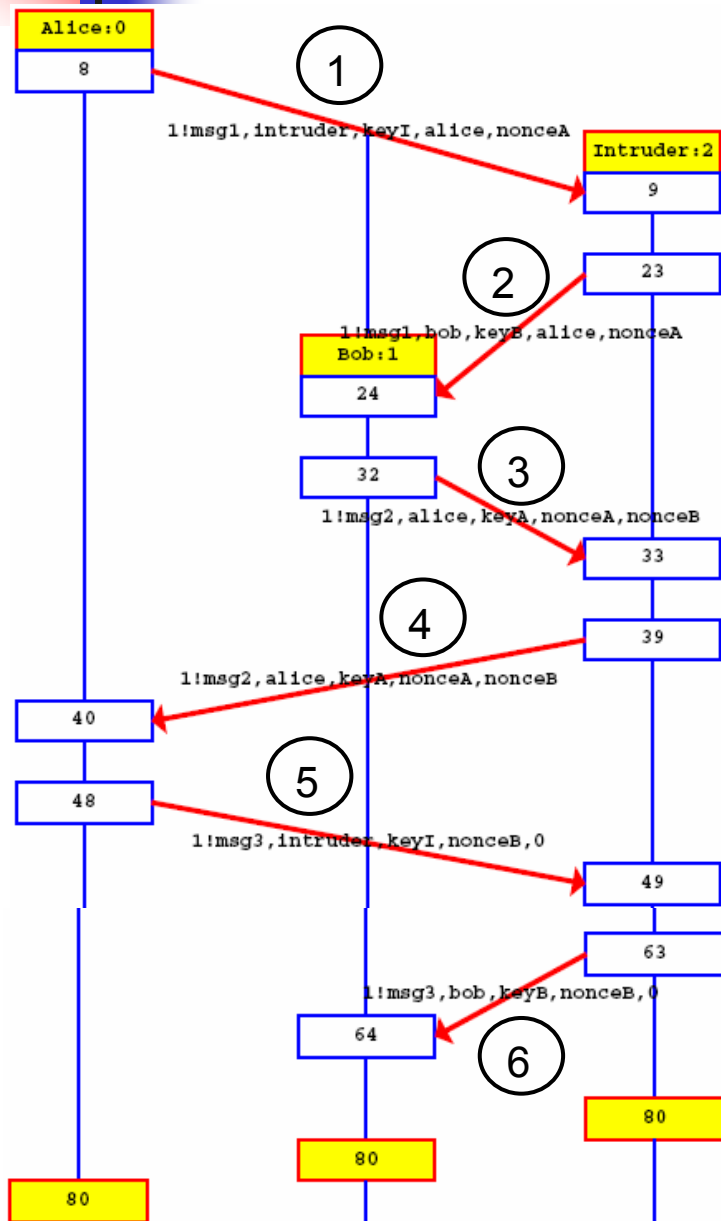


## Идея модели для нахождения атаки

---

- Каждое сообщение, которое в принципе возможно послать, представлено оператором посылки
- Знания, необходимые для конструирования сообщения, присутствуют в защите оператора посылки. Если соответствующие данные получены, то злоумышленник может их вставить в сообщение
- Подавляющее число возможных сообщений злоумышленника некорректны. Они распознаются участниками А и В, поэтому система имеет много дедлоков
- Но нам важно проверить, есть ли путь, на котором возможна атака. Это и делает Spin, проверяя выполнение темпоральной формулы и генерируя контрпример

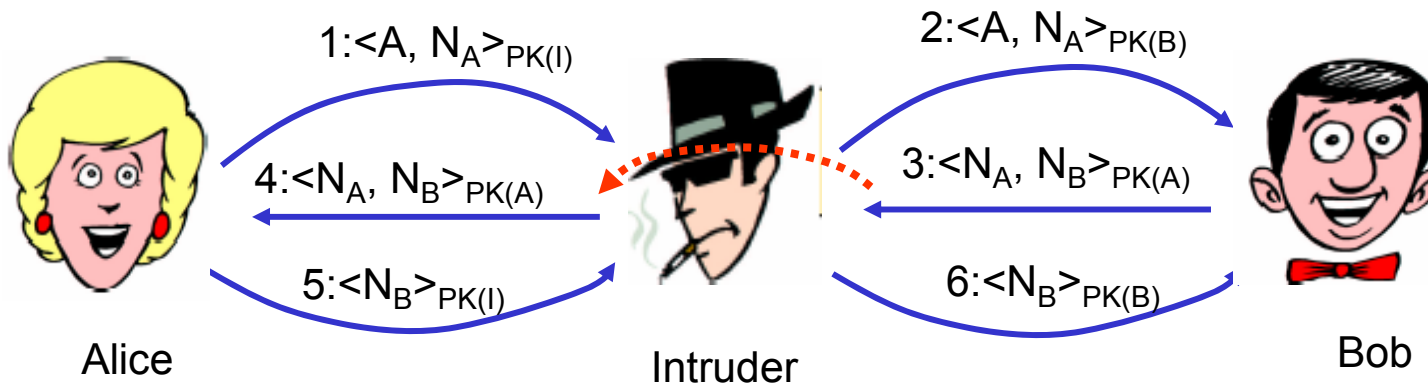
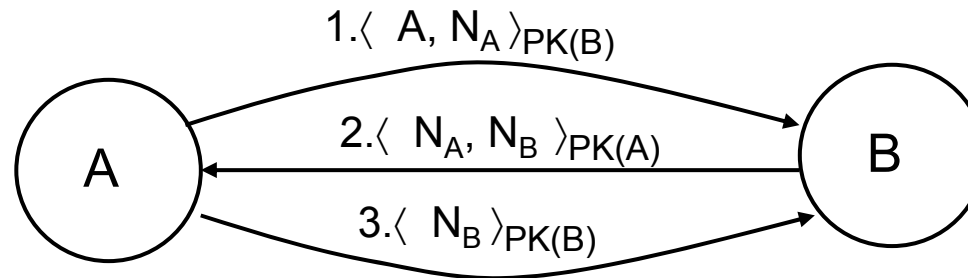
# Найденная атака



1. А начинает взаимодействовать с I как с партнером
2. I посылает сообщение msg1 к В Он маскируется под А, использует nonceA, полученный от А. В считает, что это сообщение получено от А
3. В посылает сообщение msg2, которое зашифровано ключом А. В этом сообщении включены как nonceA, так и nonceB.  
**I перехватывает это сообщение**
- Поскольку оно закодировано открытым ключом А, I **не может** декодировать это сообщение сам и пересылает его А
4. Получив это сообщение, А видит свой nonceA, считает, что все закончилось успешно, и посылает msg3 своему партнеру - агенту I, считая, что установлена связь с В
5. Это сообщение закодировано ключом для I и содержит nonceB. Поэтому агент I может его декодировать, получив nonceB
6. I посылает этот nonceB агенту В, который тоже удовлетворен



## Представление атаки



Bob думает, что он говорит с Alice, а он говорит с Intruder!



# Анализ криптографических протоколов

- Итак, при анализе NS протокола найдена атака. Если сообщение  $\langle N_A, N_B \rangle$  заменить на  $\langle B, N_A, N_B \rangle$ , то верификатор не находит возможной атаки – т.е. если Bob идентифицирует себя в сообщении, атаки нет
- В общем случае, Model checking алгоритм может не выявить некорректность криптопротокола, если его модель на языке Promela слишком грубая
  - G. Lowe доказал, что для этой конкретной проблемы абстракция является достаточной
- Модель с конечным числом состояний может быть неадекватной
  - Атаки произвольной длины
  - Неограниченное число поцес, ключей, сообщений
- Свойства безопасности обычно трудно формализовать:
  - трудно формально выразить такие свойства, как Секретность, Конфиденциальность, Аутентификация, Анонимность





## Заключение: применения Model checking

- ИИ и Model checking - логические задачи и задачи планирования
- Верификация многоагентных систем и разработка логик представления знаний
- Верификация криптографических протоколов и МС - отдельная ветвь криптоанализа
- Расширения для временных и вероятностных моделей (система Prism)
- Другие области:
- **нанотехнологии**
  - Evaluating the Reliability of Defect-Tolerant Architectures for Nanotechnology with Probabilistic Model Checking (2004)
  - Probabilistic Error Model for Unreliable Nano-logic gates (2006)
  - Towards model-checking quantum security protocols. In P.Dini et al, editor, *Proceedings of the First Workshop on Quantum Security: QSec'07*, (2007)
- **системная биология:**
  - N. Chabrier-River. Symbolic Model Checking of Biochemical Systems (INRIA), 2003
  - Validation of Qualitative models of Genetic Regulatory Networks by Model Checking Analysis of Nutritional Stress Response in Escherichia coli. Gregory Batt et al (2005)





## Некоторые ссылки

---

### Преобразования графов

- A.Renzink. Model checking graph transformation. LNCS 3256, 2004

### E-Commerce

- Anderson B.B., ... Model Checking for E-commerce Transactions, CACM, Jan 2006

### Верификация веб-сервисов

- Shin Nakajima. Model checking verification for Reliable Web Service, 2003
- H.Schlingloff. Modeling and Model checking of Web Services. Fraughoffer, 2006
- Bäumlér, S 2006.; Balser, M.; Dunets, A.; Reif, W.; Schmitt, J. *A Verification of Medical Guidelines by Model Checking*. A Case Study, Model Checking Software, SPIN 2006; Vienna, Austria: March 30–April 1. Springer. 2006
- L.Alfaro. Model checking the World Wide Web. Proc of Conf on Comput Aided Verification, 2001

N. Zhang, M. Ryan, D. Guelev. Evaluating Access Control Policies Through Model Checking  
// LNCS 3650 (2005)

- Из аннотации
- Мы представляем model checking алгоритм, который может быть использован для оценки политик управления доступом, а также инструмент, который реализует этот алгоритм.
- Оценка состоит не только в проверке того, что политика дает авторизованному пользователю достаточно возможностей, чтобы выполнить желаемую задачу, но также и проверку того, что политики не позволяют злоумышленникам достичь их целей



## Семантический анализ Web документов

F.Weitl, B.Freitag. Checking semantic integrity constraints on Integrated Web Documents. Proc. Workshop on Model Directed Web Information Integration and Mining, 2004

**Abstract.** Предложена концепция спецификации и верификации ограничений на содержание и структуру документа. В качестве формализма спецификации мы вводим CTLDC - логику CTL, расширенную концепциями логики описаний – Description Concepts. Наш подход позволяет интегрировать онтологии с тем, чтобы достичь интероперабельности, абстрагируясь от реализационных аспектов документов.

- F.Weitl, B.Freitag. Model checking semantic properties of web documents based on temporal description logic. TR Uni of Passau, 2005
- X.Fu, T.Bultan, J Su. Model checking XML Manipulating Software. Santa Barbara Uni 2004



## Другие применения

---

- E. Quintarelli. Model-Checking Based Data Retrieval: An Application to Semistructured and Temporal Data // LNCS 2917, 2004
- E.Kang, N.Day. Formal verification of the A-7E Software Requirements using Template Semantics. Uni Waterloo, 2006
- T.Jeron, P.Morel. Test Generation Derived from Model Checking. LNCS, 1633, 1999





## Заключение

---

- Приемы и методы, разработанные в области model checking начинают применяться в огромном числе других областей, казалось бы, совершенно не связанных с верификацией, программированием, встроенными системами управления
- Эта техника может быть применена в дискретных динамических (развивающихся во времени) системах, там, где есть возможность формализовать эту динамику с помощью систем переходов
- Последнее время множество примеров применения этой техники было продемонстрировано в системной биологии и нанотехнологии



---

Спасибо за внимание