

# Верификация программ на моделях

Лекция №7

Логика линейного времени  
(LTL).

*Константин Савенков (лектор)*

# План лекции

- Логика линейного времени (LTL)
- Свойства, инвариантные к прореживанию
- Связь между LTL и автоматами Бюхи  
/ конструкциями never
- Применение LTL в системе Spin
- Практические приёмы формулирования  
свойств на LTL

# Проверка свойств при помощи автоматов Бюхи (напоминание)

- При помощи автомата Бюхи можно описать наблюдаемое поведение программы и требования к нему,
- Проход автомата соответствует наблюдаемому вычислению (трассе) программы,
- Определение допускаемости прохода позволяет рассуждать о выполнении или нарушении требований (свойств правильности).
- **Задавать свойства правильности при помощи автоматов неудобно.**

# Безопасность и живучесть

(напоминание)

- **Безопасность**

- Любое свойство безопасности можно проверить, исследуя свойства отдельных состояний модели;
- если свойство безопасности нарушено, всегда можно определить достижимое состояние системы, в котором оно нарушается;
- для проверки свойств безопасности требуется генерировать состояния системы и для каждого из них проверять свойство;
- при проверке таких свойств можно обойтись без темпоральных логик и автоматов Бюхи.

- **Живучесть**

- Для проверки свойств живучести необходимо рассматривать последовательности состояний (конечные и бесконечные проходы соотв. автомата Бюхи);
- для проверки свойств используются другие, более сложные алгоритмы;
- свойства удобно описывать при помощи формул темпоральной логики, а проверять – при помощи автоматов Бюхи.

# Примеры темпоральных свойств

- **p всегда** истинно;
- **p рано или поздно** станет всегда ложным;
- **p всегда рано или поздно** станет ложным хотя бы ещё один раз;
- **p всегда** ведёт к  $\neg q$ ;
- **p всегда** ведёт к тому, что **рано или поздно** станет истинным **q**.

# Почему бы не описывать темпоральные свойства на естественном языке?

- нет строгой семантики => возможно множество трактовок
  - в части области проверки: «индикатор не горит» - в начальном состоянии? или это инвариант?
  - в темпоральной части:
    - попробуйте объяснить разницу: «от события А до события Б» и «между событиями А и Б»
    - «деньги выплачиваются, как только работа будет выполнена» -- требуется ли выполнение работы?
- значение зависит от контекста:
  - если нажата кнопка, рано или поздно будет выпущено шасси
  - от взлёта и до посадки, если нажата кнопка, то рано или поздно будет выпущено шасси (область проверки вложенного свойства изменилась)
  - после взлёта, если была нажата кнопка, то до посадки будет выпущено шасси (более строгая формулировка)
- зависит от знания и понимания естественного языка, который сложнее LTL.

# Темпоральная логика LTL

- Ясный, лаконичный и непротиворечивый способ описания требований к программам;
- В явном виде время не присутствует, однако рассуждения ведутся в терминах «никогда», «всегда», «рано или поздно», которые представлены в виде темпоральных операторов.
- Мы рассматриваем темпоральную логику линейного времени – LTL. С её помощью можно описывать свойства, которым должны удовлетворять линейные последовательности наблюдаемых состояний – трассы.
- LTL предложена Амиром Пнуэли (Amir Pnueli) в конце 70-х.

# Темпоральная логика LTL

- Семантика LTL определена на бесконечных проходах автомата Бюхи.

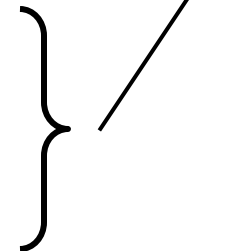
- Пример:

- $\Box ((a \neq b) \rightarrow \Diamond (a = b))$

- `#define p a !=b`

- `#define q a == b`

- `[](p -> <>q)`



Синтаксис Spin

- всегда, когда выполняется ( `a!=b` ), в конце концов становится истинным ( `a == b` ).

- как правило, формула описывает не одну конкретную трассу, а **класс** трасс.



# Формулы LTL

- Могут использоваться для описания как свойств живучести, так и безопасности
- LTL = пропозициональная логика  
( свойства состояний )  
+ темпоральные операторы  
( последовательности состояний )
- Формула LTL  $f ::=$ 
  - $p, q, \dots$  - свойства состояний, включая true и false,
  - $( f )$  – группировка при помощи скобок,
  - $\alpha f$  – унарные операторы,
  - $f_1 \beta f_2$  – бинарные операторы.

# Операторы LTL

- Унарные:

- $\Box$  ( [ ] ) всегда (в будущем),
- $\Diamond$  ( < > ) рано или поздно,
- ( X ) в следующем состоянии,
- $\neg$  ( ! ) логическое отрицание;

*когда-нибудь,  
в конце концов*

- Бинарные:

- U ( U ) пока, *[ до тех пор ]*
- $\wedge$  ( & & ) логическое И,
- $\vee$  ( | | ) логическое ИЛИ,
- $\rightarrow$  ( - > ) логическая импликация,
- $\leftrightarrow$  ( < - > ) логическая эквивалентность.

$\equiv (\neg p \vee q)$

$\equiv (p \rightarrow q) \wedge (q \rightarrow p)$

# Выполнимость формул

- Последовательность состояний прохода  $\sigma$

$$S_0, S_1, S_2, S_3, \dots$$

- Набор пропозициональных формул  $p, q$ :

$$\forall i, i \geq 0, \text{ и } \forall p, \text{ определено } s_i \models p$$

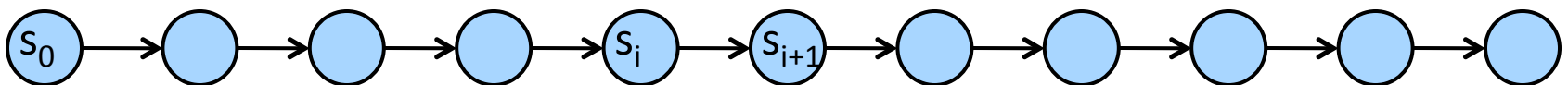
- Семантика LTL:

$$\sigma \models f \iff s_0 \models f$$

$$s_i \models []f \iff \forall j, j \geq i : s_j \models f$$

$$s_i \models \Diamond f \iff \exists j, j \geq i : s_j \models f$$

$$s_i \models Xf \iff s_{i+1} \models f$$



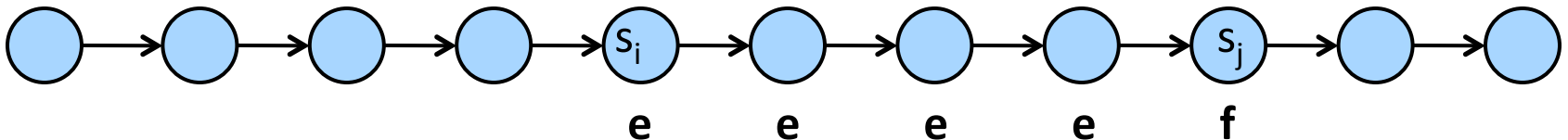
# Слабый и сильный until

слабый:

$$s_i \models eWf \iff s_i \models f \vee (s_i \models e \wedge s_{i+1} \models (eWf))$$

сильный:  
(Spin)

$$s_i \models eUf \iff \begin{array}{l} \exists j, j \geq i : s_j \models f \quad \text{и} \\ \forall k, i \leq k < j : s_k \models e \end{array}$$



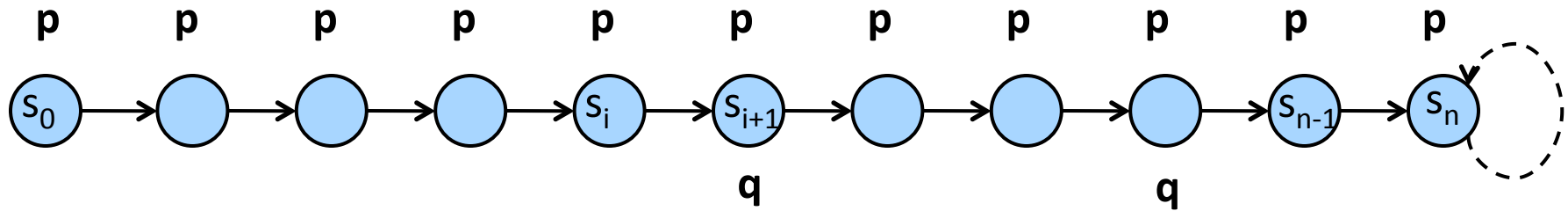
# Практически важные следствия

$$\sigma \models e \ W \ false \iff \sigma \models []e$$

$$\sigma \models true \ U \ f \iff \sigma \models \Diamond f$$

$$\sigma \models e \ W \ f \iff \sigma \models []e \vee e \ U \ f$$

# Примеры



$$\sigma \models []p$$

$$\sigma \models \Diamond p$$

$$\sigma \models []\Diamond p$$

$$\sigma \models []q$$

$$\sigma \models \Diamond q$$

$$\sigma \models []\Diamond q$$

$$\sigma \models pUq$$

$$\sigma \models [](pUq)$$

$$\sigma \models [](pWq)$$

$$\sigma \models qUp$$

$$\sigma \models [](qUp)$$

$$\sigma \models qWp$$

$$\sigma \models []qWp$$

# Цикличность и стабильность

- Свойством цикличности называется любая темпоральная формула, которая представима в виде  $\Box \Diamond p$ , где  $p$  – формула на состоянии;
- Свойством стабильности называется любая темпоральная формула, которая представима в виде  $\Diamond \Box p$ , где  $p$  – формула на состоянии.

# Распространённые LTL-формулы

Формула	Описание	Тип
$[]p$	всегда <b>p</b>	<i>инвариант</i>
$\diamond p$	рано или поздно <b>p</b>	<i>гарантия</i>
$p \rightarrow \diamond q$	если <b>p</b> , то рано или поздно <b>q</b>	<i>отклик</i>
$p \rightarrow qUr$	если <b>p</b> , то <b>q</b> , затем <b>r</b>	<i>приоритет</i>
$[]\diamond p$	всегда рано или поздно будет <b>p</b>	<i>цикличность (прогресс)</i>
$\diamond[]p$	рано или поздно всегда будет <b>p</b>	<i>стабильность (бездействие)</i>
$\diamond p \rightarrow \diamond q$	если рано или поздно <b>p</b> , то рано или поздно <b>q</b>	<i>корреляция</i>



# Эквивалентные преобразования

$$![]p \quad \Leftrightarrow \quad \Diamond!p$$

$$!\Diamond p \quad \Leftrightarrow \quad []!p$$

---

$$!(pUq) \quad \Leftrightarrow \quad !qW(!p \wedge !q)$$

$$!(pWq) \quad \Leftrightarrow \quad !qU(!p \wedge !q)$$

---

$$[](p \wedge q) \quad \Leftrightarrow \quad []p \wedge []q$$

$$\Diamond(p \wedge q) \quad \Leftrightarrow \quad \Diamond p \wedge \Diamond q$$

---

$$pW(q \vee r) \quad \Leftrightarrow \quad (pWq) \vee (pWr)$$

$$(p \wedge q)Wr \quad \Leftrightarrow \quad (pWr) \wedge (qWr)$$

---

$$pU(q \vee r) \quad \Leftrightarrow \quad (pUq) \vee (pUr)$$

$$(p \wedge q)Ur \quad \Leftrightarrow \quad (pUr) \wedge (qUr)$$

---

$$[]\Diamond(p \vee q) \quad \Leftrightarrow \quad []\Diamond p \vee []\Diamond q$$

$$\Diamond[](p \wedge q) \quad \Leftrightarrow \quad \Diamond[]p \wedge \Diamond[]q$$

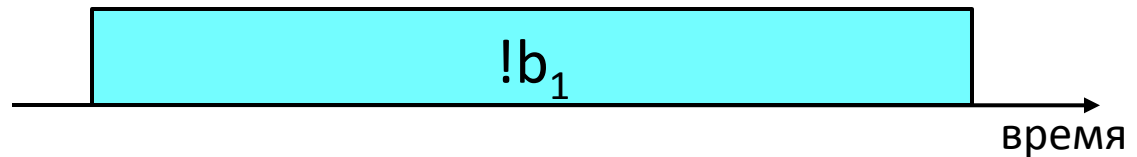
# Примеры темпоральных свойств

- **p** всегда истинно;  $\boxed{[]p}$
- **p** рано или поздно станет всегда ложным;  $\boxed{<>[]!p}$
- **p** всегда рано или поздно станет ложным хотя бы ещё один раз;
- **p** всегда ведёт к  $\neg q$ ; —  $\boxed{[] (p \rightarrow !q)}$   $\boxed{[] <> !p}$
- **p** всегда ведёт к тому, что рано или поздно станет истинным **q**.  $\boxed{[] (p \rightarrow <> q)}$

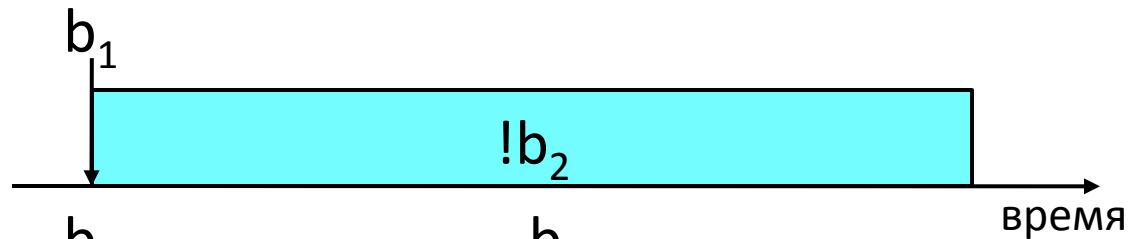
# Правильная интерпретация формул LTL

$$LTL: (\langle \rangle (b_1 \wedge (!b_2 U b_2))) \rightarrow [] !a_3$$

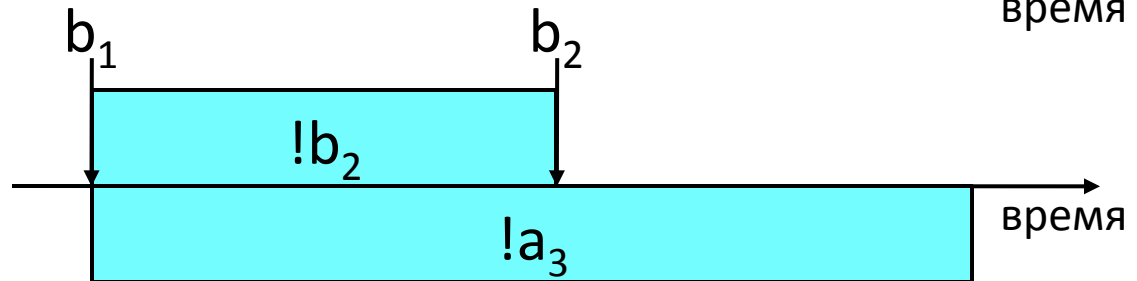
1. Пусть  $b_1$  всегда ложно,  
 $p \rightarrow q$  означает, что  $!p \vee q$ ;  
**формула выполняется.**



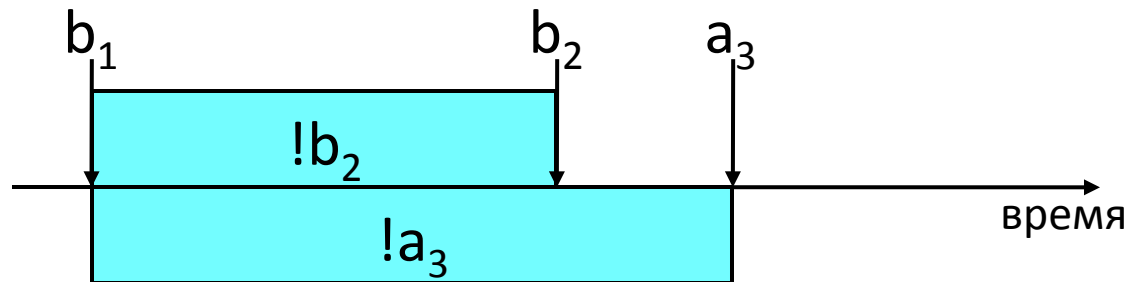
2. Пусть  $b_1$  стало истинно,  
но  $b_2$  всегда ложно;  
**формула выполняется.**



3. Пусть  $b_1$  стало истинно,  
затем —  $b_2$ , однако  $a_3$   
всегда ложно; **формула**  
**выполняется.**



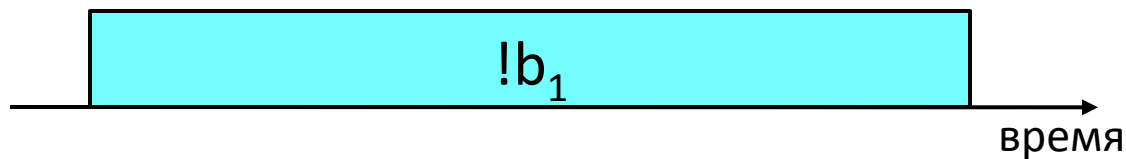
4. Пусть  $b_1$  стало истинно,  
затем —  $b_2$ , затем —  $a_3$ ;  
**формула не**  
**выполняется.**



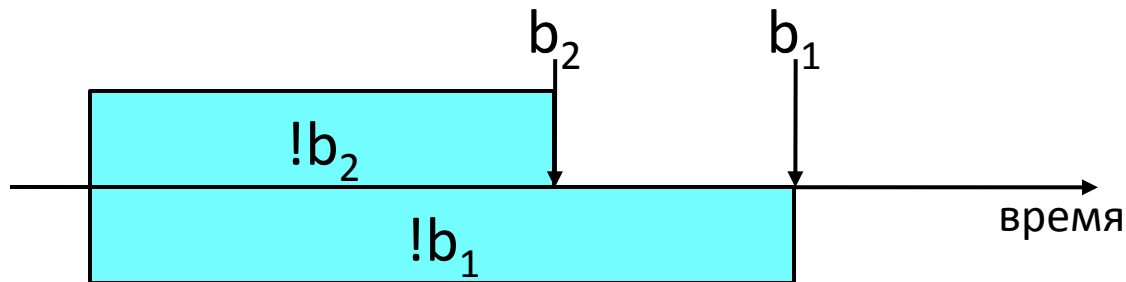
# Правильная интерпретация формул LTL

$$LTL: (\langle \rangle b_1) \rightarrow (\langle \rangle b_2)$$

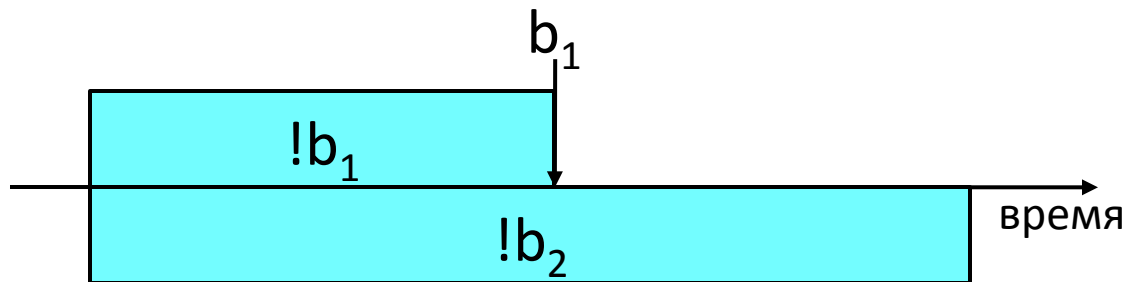
1. Пусть  $b_1$  и  $b_2$  всегда ложно; **формула выполняется.**



2. Пусть и  $b_1$ , и  $b_2$  становятся истинными; **формула выполняется.**



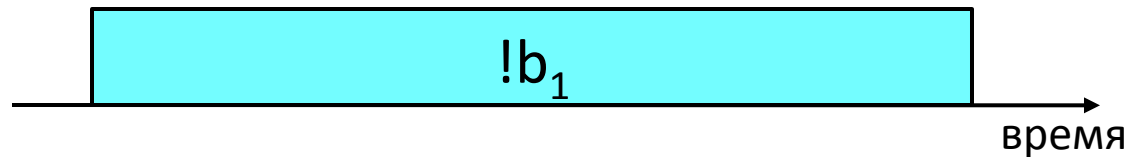
3. Пусть  $b_1$  становится истинным, но  $b_2$  всегда ложно; **формула не выполняется.**



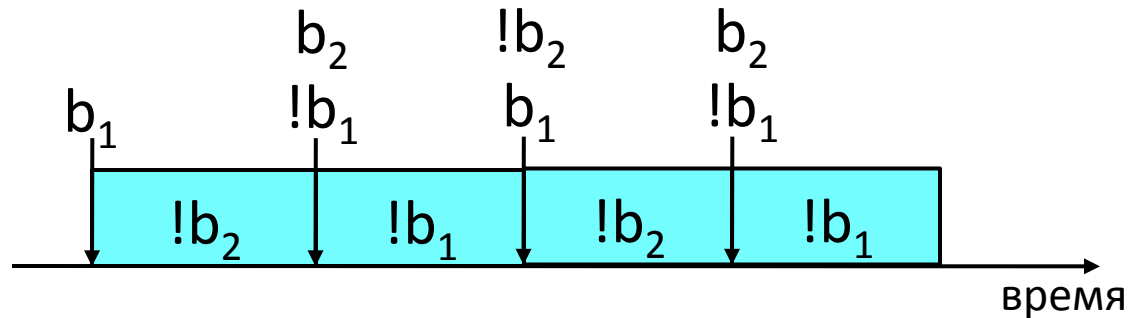
# Правильная интерпретация формул LTL

$$LTL: \quad []((\langle \rangle b_1) \rightarrow (\langle \rangle b_2))$$

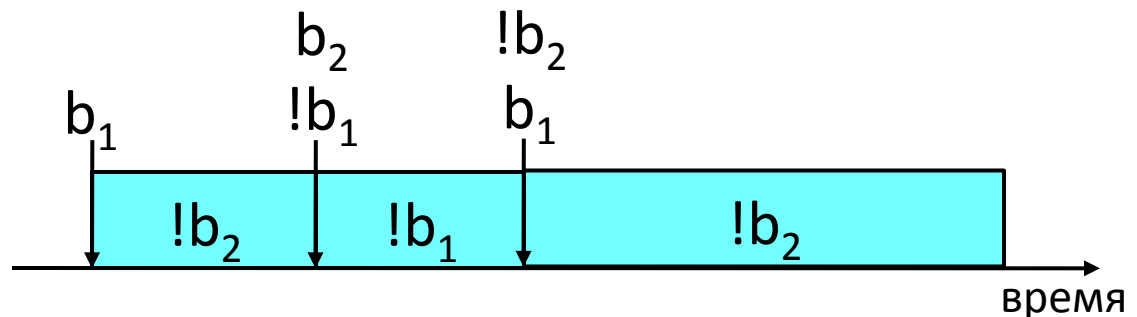
1. Пусть  $b_1$  и  $b_2$  всегда ложно; **формула выполняется.**



2. Пусть  $b_1$  и  $b_2$  бесконечно чередуются; **формула выполняется.**



3. Пусть  $b_2$  становится истинным только один раз; **формула не выполняется.**



# Описание требований при помощи LTL

*“ $p$  приведёт к  $q$ ”*

- $p \rightarrow q$ 
  - нет темпоральных операторов, т.е. применяется только к первому состоянию;
  - выполняется только если  $p \vee q$  в первом состоянии, остальная трасса не рассматривается;
  - не подходит;
  - **нужно использовать темпоральные операторы.**

# Описание требований при помощи LTL

*“ $p$  приведёт к  $q$ ”*

- $[] p \rightarrow q$ 
  - **правила приоритета!**  $[]$  применяется только к  $p$ ;
  - означает  $([]p) \rightarrow q$ ;
  - **не подходит;**
  - **нужно расставить скобки.**

# Описание требований при помощи LTL

*“**p** приведёт к **q**”*

- **[ ] (p -> q)**
  - проверяем условие во всех состояниях, но причинно-следственная связь между **p** и **q** отсутствует;
  - выполняется, только если **! p ∨ q** во всех состояниях;
  - **не подходит;**
  - **нужно описать, что p является причиной q.**



# Описание требований при помощи LTL

*“ $p$  приведёт к  $q$ ”*

- $[] (p \rightarrow \langle \rangle q)$ 
  - уже лучше;
  - тем не менее, формула выполнима, если  $q$  становится истинным в том же состоянии, что и  $p$  – причинно-следственная связь отсутствует;
  - **не подходит;**
  - **нужно описать, что  $q$  не может произойти раньше следующего шага после  $p$ .**

# Описание требований при помощи LTL

*“ $p$  приведёт к  $q$ ”*

- $[] (p \rightarrow X(\leftrightarrow q))$ 
  - практически то, что нужно;
  - формула выполнима, если  $p$  всегда ложно;
  - **возможно, не подходит;**
  - **нужно описать, что  $p$  обязательно произойдёт и приведёт к  $q$ .**

# Описание требований при помощи LTL

*“ $p$  приведёт к  $q$ ”*

- $[] (p \rightarrow X(<>q)) \ \&\& \ (<>p)$ 
  - скорее всего, мы имели ввиду именно это;
  - несколько отличается от первоначального  $p \rightarrow q$ ;
  - LTL позволяет выразить множество различных оттенков свойства;
  - **подойдёт ли такое свойство для модели параллельной программы?**

# Оператор neXt

- Оператор **X** нужно использовать аккуратно:
  - с его помощью делается утверждение о выполнимости формулы на непосредственных потомках текущего состояния;
  - в распределённых системах значение оператора **X** неочевидно;
  - поскольку алгоритм планирования процессов неизвестен, не стоит формулировать спецификацию в предположении о том, какое состояние будет следующим;
  - стоит ограничиться предположением о справедливости планирования.

# Свойства, инвариантные к прореживанию

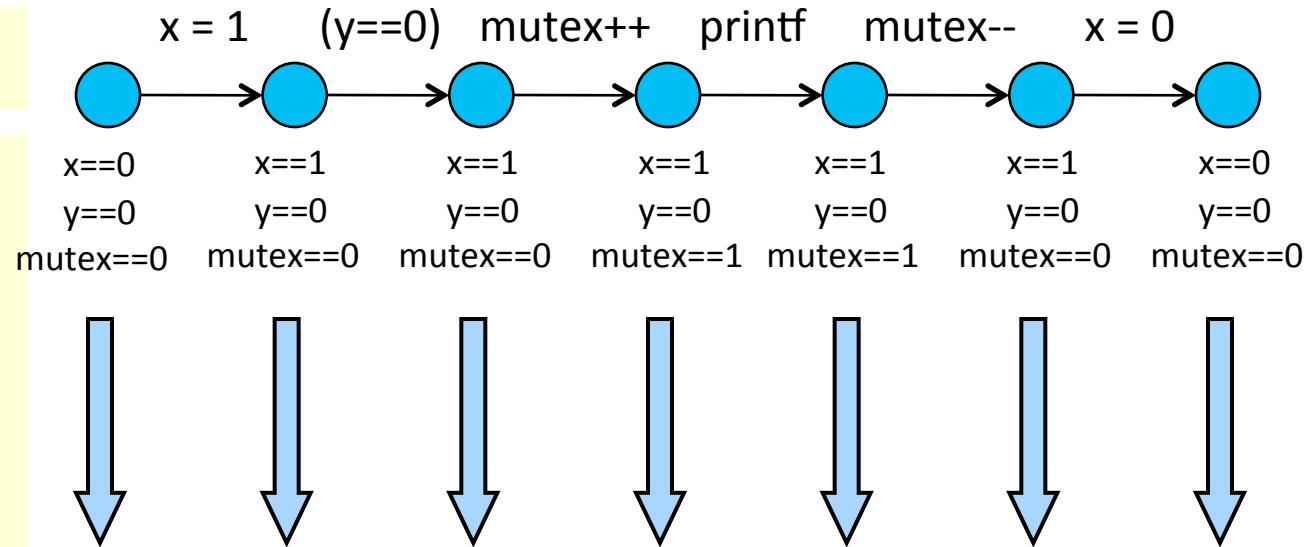
- Пусть  $\varphi$  – трасса некоторого вычисления над пропозициональными формулами  $P$ ,
  - по трассе можно определить, выполняется ли на ней темпоральная формула,
  - трассу можно записать в виде  $\varphi = \varphi_1^{n1} \varphi_2^{n2} \varphi_3^{n3} \dots$ , где значения пропозициональных формул на каждом интервале совпадают.
- Обозначим  $E(\varphi)$  набор всех трасс, отличающихся лишь значениями  $n1, n2, n3$  (т.е. длиной интервалов)
  - $E(\varphi)$  называется **расширением прореживания**  $\varphi$ .

# Расширение прореживания

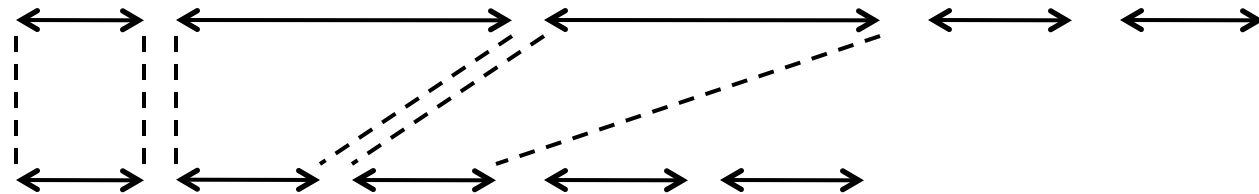
```
p: (x == mutex)
q: (x != y)
```

```
bit x,y;
byte mutex;
active proctype A()
{
    x = 1;
    (y == 0) ->
    mutex++;
    printf("%d\n", _pid);
    mutex--;
    x = 0;
}
```

трасса  $\varphi$



p	!p	!p	p	p	!p	p
!q	q	q	q	q	q	!q



трасса  $\varphi_1 \in E(\varphi)$

p	!p	p	!p	p
!q	q	q	q	!q

# Свойства, инвариантные к прореживанию

- Свойство  $\varphi$ , инвариантное к прореживанию, либо истинно для всех трасс из  $E(\varphi)$ , либо ни для одной из них:

$$\varphi \models f \rightarrow \forall v \in E(\varphi), v \models f$$

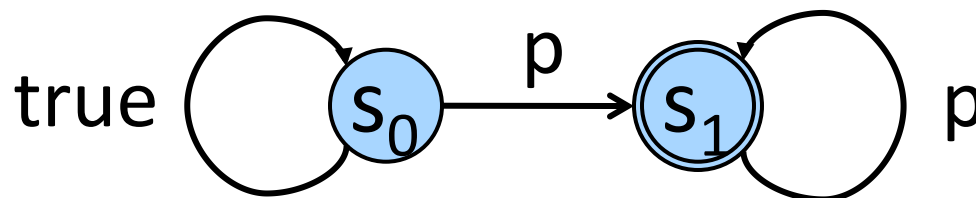
- истинность такого свойства зависит от порядка, в котором пропозициональные формулы меняют свои значения, и не зависит от длины трассы;
- Теорема:** все формулы LTL без оператора X инвариантны к прореживанию.
- Более того, в рамках LTL без X можно описать **все** свойства, инвариантные к прореживанию.

Связь между LTL  
и  
автоматами Бюхи  
(конструкциями never)



# Связь LTL с автоматами Бюхи

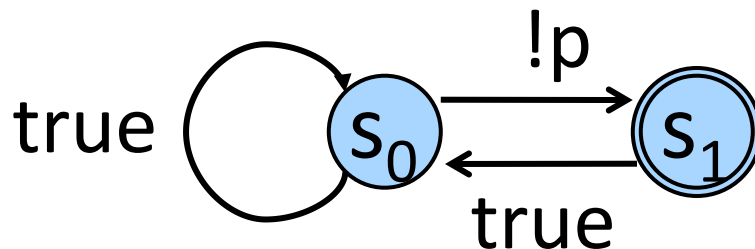
- Удобно проверять допустимость трасс для некоторого автомата Бюхи;
- Удобно описывать свойства правильности при помощи темпоральных формул;
- **Что дальше?**
- **Теорема:** Для всякой формулы LTL  $f$  существует автомат Бюхи, который допускает те и только те прогоны, которым соответствуют трассы, на которых выполнима  $f$ .
- **Пример:** формуле  $\langle \rangle [] p$  соответствует автомат



# Переход от LTL к автоматам

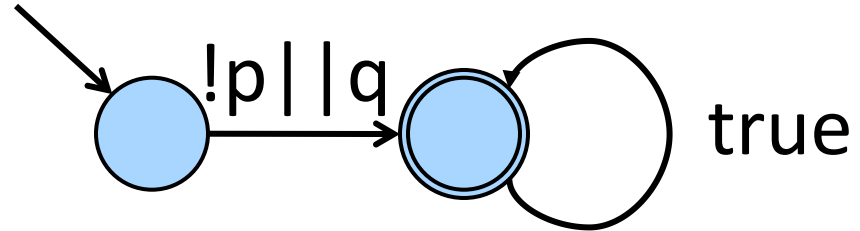
- Привести свойство правильности LTL к форме never языка Promela достаточно просто: нужно построить отрицание формулы LTL  $f$  и поместить его в тело never:
  - $f$  выполняется на всех вычислениях  $\iff$
  - $\neg f$  не выполняется ни на одном вычислении  $\iff$
  - автомат  $\{\neg f\}$  не допускает ни одного прохода, полученного в результате синхронного выполнения с системой.

$$\neg \Diamond [] p \equiv [] \neg \Diamond p \equiv [] \Diamond \neg p$$

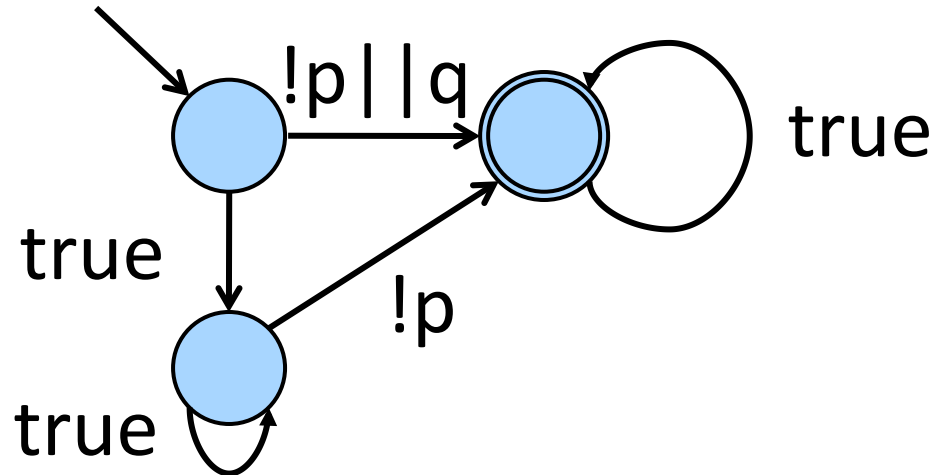


# Примеры

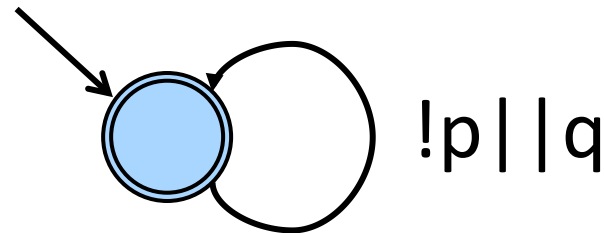
- $p \rightarrow q$



- $[]p \rightarrow q$

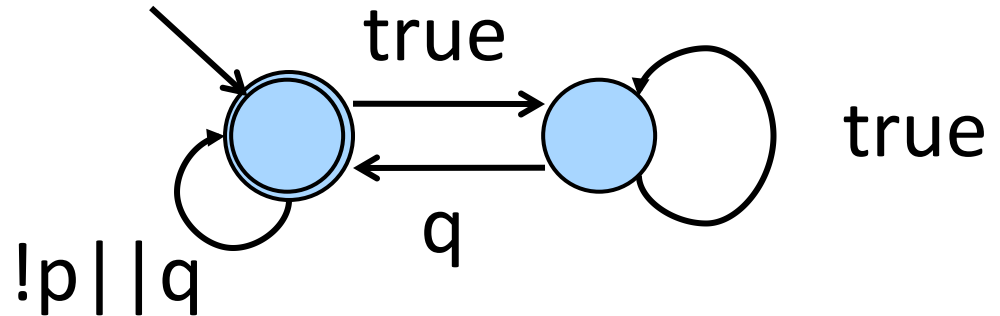


- $[](p \rightarrow q)$

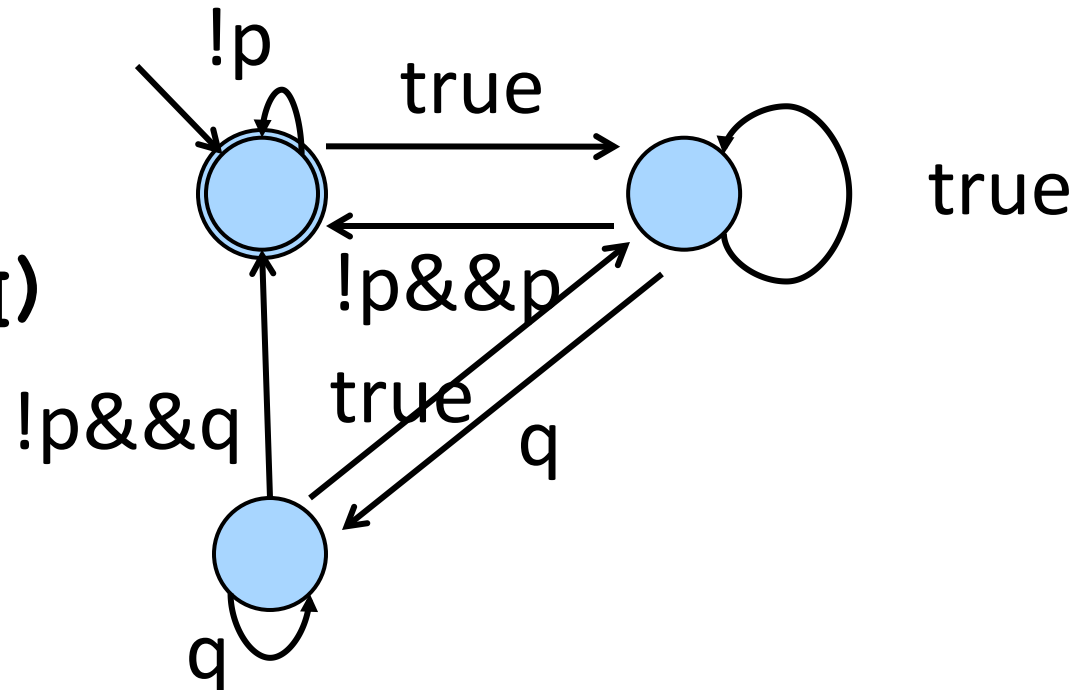


# Примеры

- $[] (p \rightarrow \langle \rangle q)$

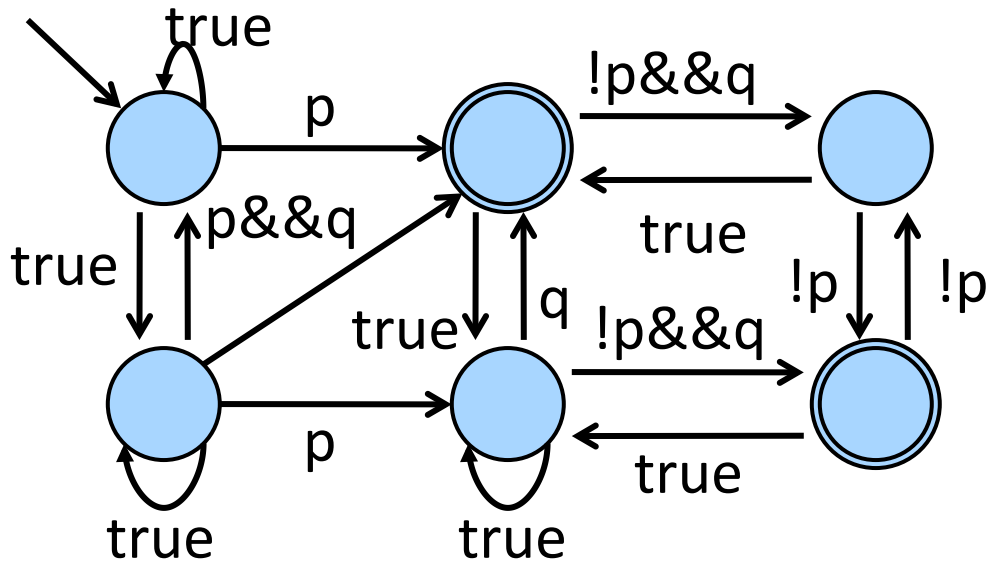


- $[] (p \rightarrow X \langle \rangle q)$



# И, наконец...

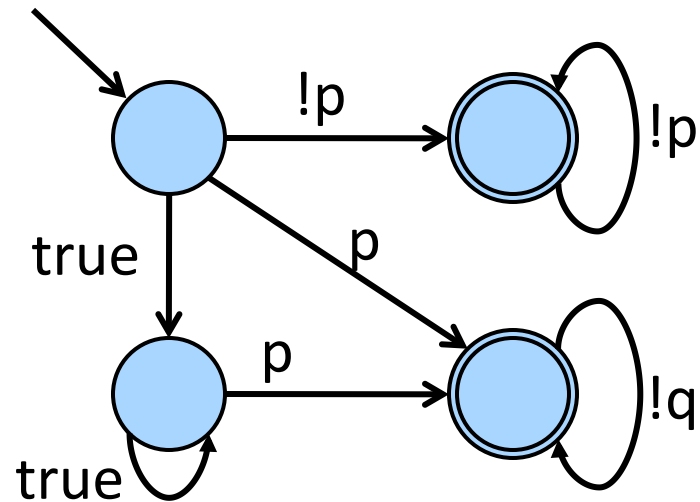
- $[ ] (p \rightarrow X \langle \rangle q) \ \&\& \ (\langle \rangle p)$



*Здесь формула LTL явно  
понятнее автомата...*

*Однако мы собирались  
построить автомат  
для отрицания формулы*

- $! [ ] (p \rightarrow X \langle \rangle q) \ \&\& \ (\langle \rangle p)$



# Отрицания формул LTL

Формулу LTL всегда можно переписать в таком виде, что отрицания будут появляться только перед пропозициональными символами

- $\neg [] (p \rightarrow \Diamond q)$
- $\neg [] (\neg p \vee \Diamond q)$  – по определению логической импликации
- $\Diamond \neg (\neg p \vee \Diamond q)$  –  $\neg [] p$  эквивалентно  $\Diamond \neg p$
- $\Diamond (p \wedge \neg \Diamond q)$  – По правилу ДеМоргана
- $\Diamond (p \wedge [] \neg q)$  –  $\neg \Diamond q$  эквивалентно  $[] \neg q$

# И, наконец...

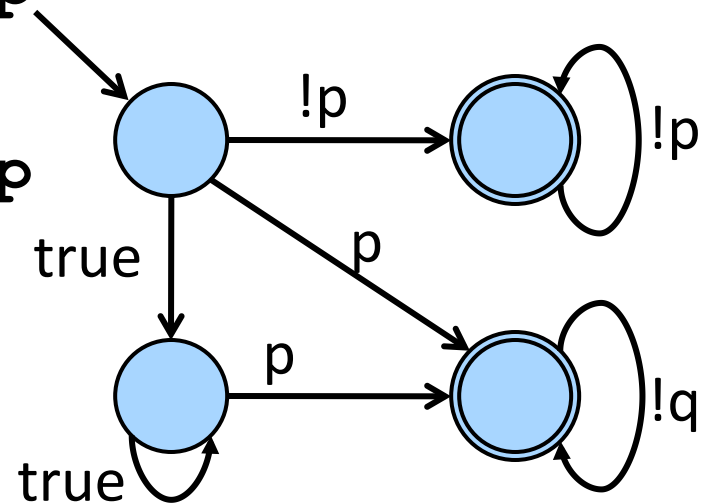
- $! ([ ] (p \rightarrow X \langle \rangle q) \ \&\& \ (\langle \rangle p))$

–  $! [ ] (!p \ || \ (X \langle \rangle q)) \ || \ ! \langle \rangle p$

–  $\langle \rangle ! (!p \ || \ (X \langle \rangle q)) \ || \ [ ] !p$

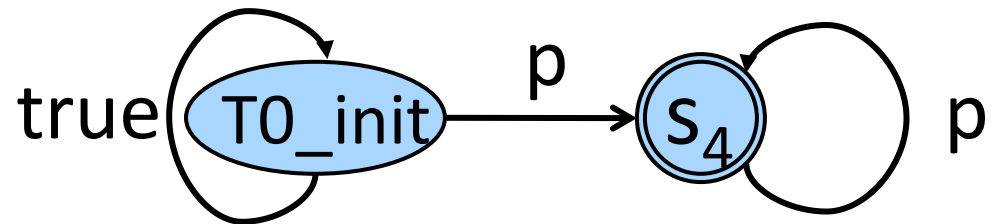
–  $\langle \rangle (p \ \&\& \ ! (X \langle \rangle q)) \ || \ [ ] !p$

–  $\langle \rangle (p \ \&\& \ (X [ ] !q)) \ || \ [ ] !p$

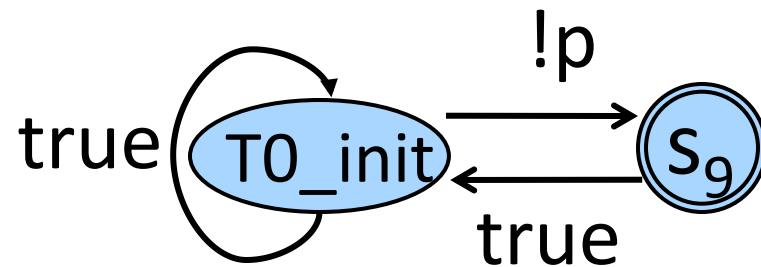


# Используем Spin для трансформации формул LTL

```
> ./spin -f '<>[]p'
never {      /* <>[]p */
T0_init:
    if
    :: ((p)) -> goto accept_S4
    :: (1) -> goto T0_init
    fi;
accept_S4:
    if
    :: ((p)) -> goto accept_S4
    fi;
}
```



```
> ./spin -f '!<>[]p'
never {      /* !<>[]p */
T0_init:
    if
    :: (! ((p))) -> goto
accept_S9
    :: (1) -> goto T0_init
    fi;
accept_S9:
    if
    :: (1) -> goto T0_init
    fi;
}
```





# Правила синтаксиса

```
> ./spin -f `[[]p -> <> (a+b <= c) ]'
```

```
#define q (a+b <= c)
```

Вводим строчные макроопределения  
для всех булевых подформул

```
> ./spin -f `[[] (p -> <> q) '
never {      /* [[] (p -> <> q) */
T0_init:
  if
  :: (((! ((p))) || ((q)))) -> goto accept_S20
  :: (1) -> goto T0_S27
  fi;
accept_S20:
  if
  :: (((! ((p))) || ((q)))) -> goto T0_init
  :: (1) -> goto T0_S27
  fi;
accept_S27:
  if
  :: ((q)) -> goto T0_init
  :: (1) -> goto T0_S27
  fi;
T0_S27:
  if
  :: ((q)) -> goto accept_S20
  :: (1) -> goto T0_S27
  :: ((q)) -> goto accept_S27
  fi;
```

Следим за приоритетом  
операторов

# LTL для проверки правильности при помощи Spin

```
int x = 100;

active proctype A()
{
    do
        :: x%2 -> x = 3*x + 1
    od
}

active proctype B()
{
    do
        :: !x%2 -> x = x/2
    od
}
```

Ограничено ли значение x?

```
> ./spin -f `[ ] (x > 0 && x <= 100) '
tl_spin: expected ')', saw '>'
tl_spin: [ ] (x > 0 && x <= 100)
-----^
>
```

Синтаксическая ошибка!

# LTL для проверки правильности при помощи Spin

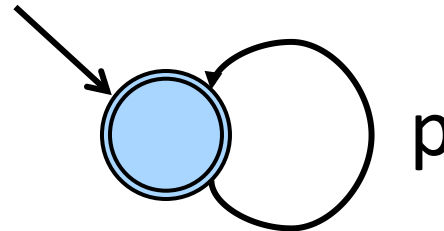
```
int x = 100;

active proctype A()
{
    do
        :: x%2 -> x = 3*x + 1
    od
}

active proctype B()
{
    do
        :: !x%2 -> x = x/2
    od
}
```

```
#define p (x > 0 && x <= 100)
```

```
> ./spin -f `[ ] p'
never {      /* [ ]p */
accept_init:
T0_init:
    if
    :: ((p)) -> goto T0_init
    fi;
}
```



Формула синтаксически корректна, но мы же будем проверять её невыполнимость!

# LTL для проверки правильности при помощи Spin

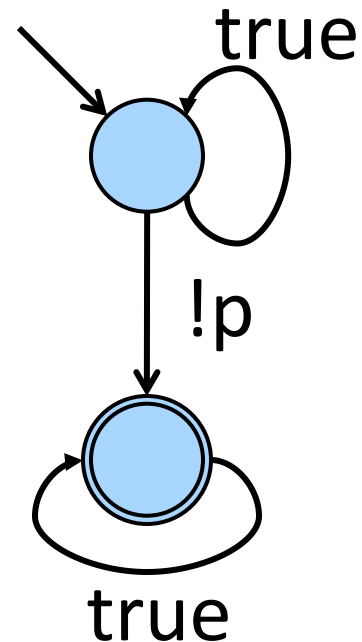
```
int x = 100;

active proctype A()
{
    do
        :: x%2 -> x = 3*x + 1
    od
}

active proctype B()
{
    do
        :: !x%2 -> x = x/2
    od
}
```

```
#define p (x > 0 && x <= 100)
```

```
> ./spin -f `![] p'
never {      /* ![]p */
T0_init:
    if
        :: (! ((p))) -> goto
accept_all
        :: (1) -> goto T0_init
    fi;
accept_all:
    skip
}
```



То, что  $![]p$  не может быть выполнено, означает, что  $[]p$  не может быть нарушено (нет ни одного контрпримера)

# LTL для проверки правильности при помощи Spin

```
int x = 100;
#define p (x > 0 && x <= 100)
active proctype A()
{
    do
        :: x%2 -> x = 3*x + 1
    od
}
active proctype B()
{
    do
        :: !x%2 -> x = x/2
    od
}
never {      /* ![!p */
T0_init:
    if
        :: (! ((p))) -> goto accept_all
        :: (1) -> goto T0_init
    fi;
accept_all:
    skip
}
```

```
> ./spin -a lt11.pml
> gcc -o pan pan.c
> ./pan -a
(Spin Version 5.1.4 -- 27 January 2008)
      + Partial Order Reduction
Full statespace search for:
never claim                +
assertion violations        + (if within scope of claim)
acceptance cycles          + (fairness disabled)
invalid end states         - (disabled by never claim)
...
```

# Ещё один пример

```
int x = 100;

active proctype A()
{
    do
        :: x%2 -> x = 3*x + 1
    od
}

active proctype B()
{
    do
        :: !x%2 -> x = x/2
    od
}
```

Пусть

**#define p (x == 1)**

Выполнимо ли свойство **[]<>p**  
?

Даже в таком простом случае  
метод пристального взгляда  
перестает работать

Для проверки неизбежности **[]<>p**  
проверяем невозможность

**![]<>p**

**<>!(<>p)**

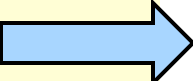
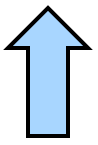
**<>[]!p**

# LTL для проверки правильности при помощи Spin

```
int x = 100;
#define p (x == 1)
active proctype A()
{
    do
        :: x%2 -> x = 3*x + 1
    od
}
active proctype B()
{
    do
        :: !x%2 -> x = x/2
    od
}
never {      /* ![]<>p */
T0_init:
```

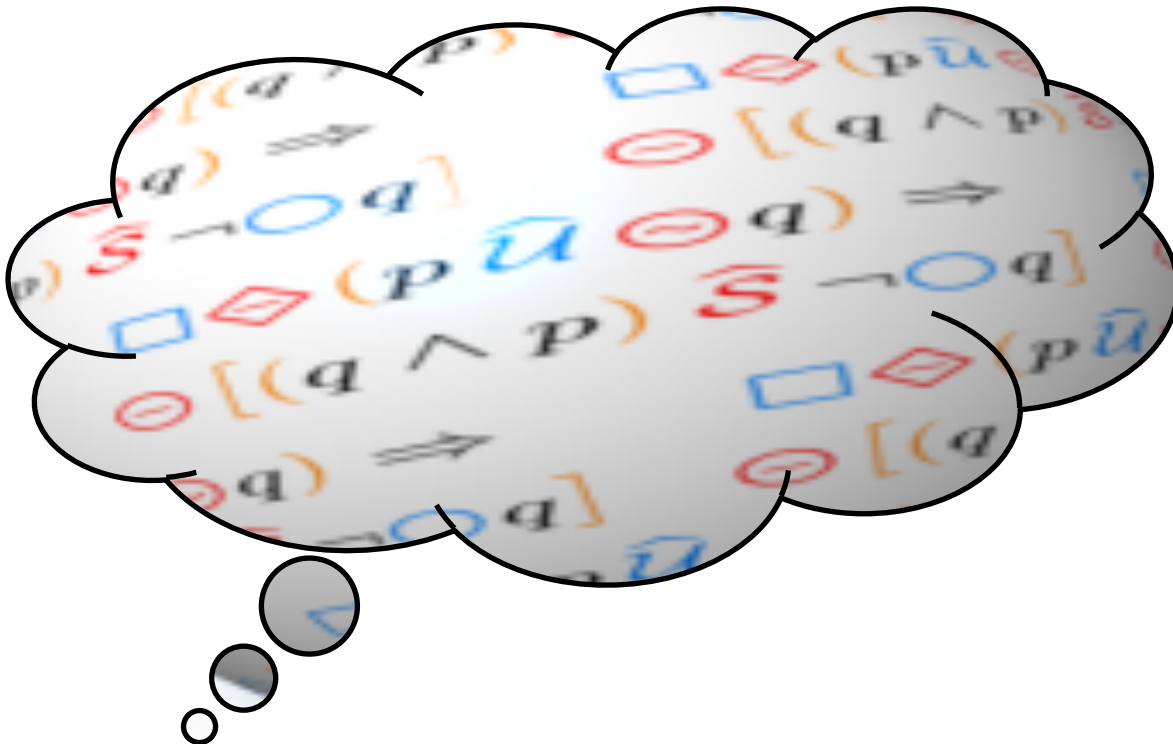
```
    if
        :: (! ((p))) -> goto accept_S4
        :: (1) -> goto T0_init
    fi;
accept_S4:
    if
        :: (! ((p))) -> goto accept_S4
    fi;
```

```
> ./spin -t -c lt12.pml
proc 0 = A
proc 1 = B
Never claim moves to line 18      [(!((x==1)))]
Never claim moves to line 23      [(!((x==1)))]
<<<<<START OF CYCLE>>>>>
spin: trail ends after 3 steps
-----
final state:
-----
#processes: 2
x = 100
3:      proc 1 (B) line 11 "lt12.pml" (state 3)
3:      proc 0 (A) line 5 "lt12.pml" (state 3)
3:      proc - (:never:) line 22 "lt12.pml" (state 9)
2 processes created
```



```
> ./spin -a lt11.pml
> gcc -a pan pan.c
> ./pan -a
pan: acceptance cycle (at depth 2)
pan: wrote lt12.pml.trail
```

# Практические приёмы описания свойств на LTL





# Практические приёмы описания свойств на LTL

- Выполнимость формулы LTL проверяется только для первого состояния в трассе
- Темпоральные операторы управляют проверкой выполнимости своих аргументов
- Сложное свойство можно (и нужно!) строить как суперпозицию простых
- Суперпозиция темпоральных операторов не ограничивает диапазон действия «вложенного» оператора.

# Выполнимость формул LTL

- Выполнимость формулы LTL определена и проверяется для одного (первого) состояния трассы { выполнимость подформул  
– уже нет }
- Распространение свойств на другие состояния управляется темпоральными операторами { для этого они и нужны }
- Единственный оператор, который может ограничить сверху проверку выполнимости формулы – Until { описать свойство, выполняющееся  
для участка программы (одной  
итерации, одного вызова функции)  
без Until не получится }

# Суперпозиция формул LTL

- Составлять сложные формулы LTL нужно методом суперпозиции простых формул
- Внешняя формула задаёт, на каких участках вычислений будет проверяться подформула
- Подформула задаёт свойства, проверяемые для участков вычислений
- Суперпозиция темпоральных операторов не ограничивает диапазон действия «вложенного» оператора.

# Пример

- В ходе одного вызова функции  $F()$   $x$  всегда не превышает 3.

где проверяем ( $g$ )

что проверяем( $f$ )

- $\varphi = g(f)$

- Что проверяем?**

- $f = x \leq 3$  – в синтаксисе LTL нет таких символов!
- `#define a x <= 3` – определяем пропозициональный символ
- $f = []a$  – вроде бы точно описывает свойство  $f$

# Пример

- В ходе одного вызова функции  $F()$   $x$  всегда не превышает 3.

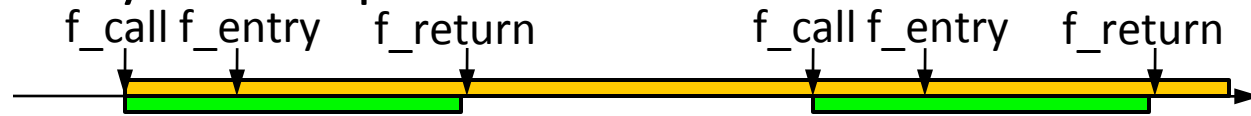
где проверяем ( $g$ )

что проверяем( $f$ )

- $\varphi = g(f)$

- Где проверяем?**

- `#define lab_f P@f_call` – поставим метку на вызов  $f$
- `g = [] (lab_f -> f)` – не то, проверяем для всего участка пути за первым вызовом  $F$



- `#define f_b P@f_entry` – нужно проверять свойство
- `#define f_e P@f_return` внутри тела функции
- `g = [] (f_b -> (f U f_e))` – похоже на правду, проверяем от входа в функцию до выхода из неё

# Пример

- В ходе одного вызова функции  $F()$   $x$  всегда не превышает 3.

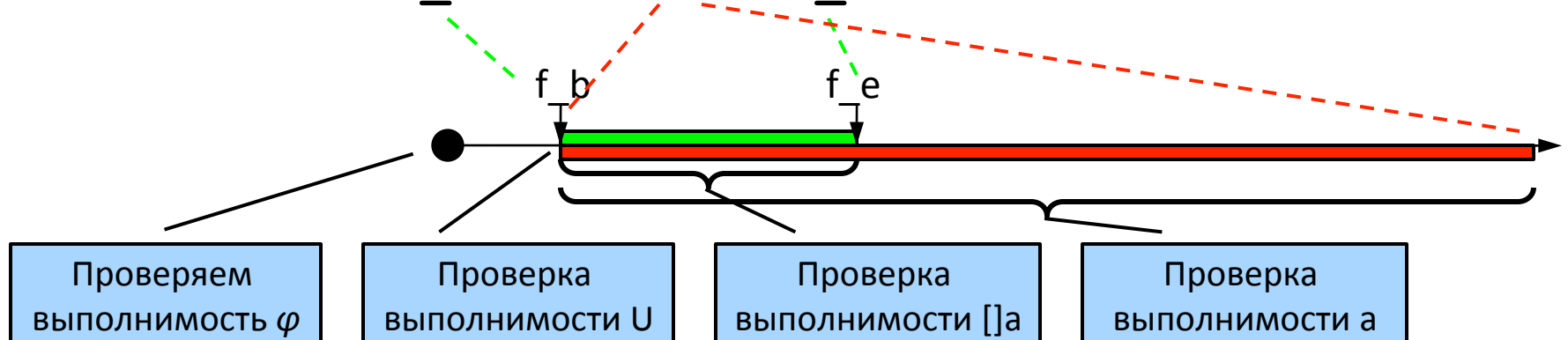
где проверяем ( $g$ )

что проверяем ( $f$ )

- $\varphi = g(f)$

- Подставляем одно в другое**

$$- \varphi = [] (f\_b \rightarrow ([] a \cup f\_e))$$



- зона действия  $[]$  по определению не ограничивается **Until**!

$$- \varphi = [] (f\_b \rightarrow (a \cup f\_e))$$

– правильная формула

# Помощь в формулировке свойств

The Patterns - Iceweasel

Файл Правка Вид Журнал Закладки Инструменты Справка

http://patterns.projects.cis.ksu.edu/documentation/patterns.shtml

Getting Started Latest Headlines

## SPEC PATTERNS

SANTOS laboratory

### OVERVIEW

- [ABOUT](#)
- [PEOPLE](#)
- [FUNDING](#)
- [RELATED PROJECTS](#)

### DOCUMENTATION

- [THE PATTERNS](#)
- [PROPERTY SPECIFICATIONS](#)

### COLLABORATIONS

- [PAPERS](#)

### The Patterns

The information in the patterns can be presented in a variety of ways. One organization, illustrated below, is based on classifying the patterns in terms of the kinds of system behaviors they describe.

```
graph TD;
    PP[Property Patterns] --> Occ[Occurrence];
    PP --> Ord[Order];
    Occ --> Abs[Absence];
    Occ --> Uni[Universality];
    Occ --> Ex[Existence];
    Ord --> Pre[Precedence];
    Ord --> Res[Response];
    Ord --> CP[Chain Precedence];
    Ord --> CR[Chain Response];
```

- **Occurrence Patterns** talk about the occurrence of a given event/state during system execution.
- **Order Patterns** talk about relative order in which multiple events/states occur during system execution.
- While not themselves patterns, **Pattern Notes** discuss common ways to vary the existing patterns to suite your needs.

An alternative organization for this information is to group pattern to formalism mappings by specification formalism. The supported formalisms are listed below. Clicking on the formalism will bring you to pages with mappings for each property pattern in that formalisms. We supply the mappings on these formalism-specific pages and you are referred to the complete patterns for information about relationships and example uses.

- **Linear Temporal Logic** (LTL)
- **Computation Tree Logic** (CTL)
- **Graphical Interval Logic** (GIL)
- **Quantified Regular Expressions** (QRE)
- **INCA Queries**

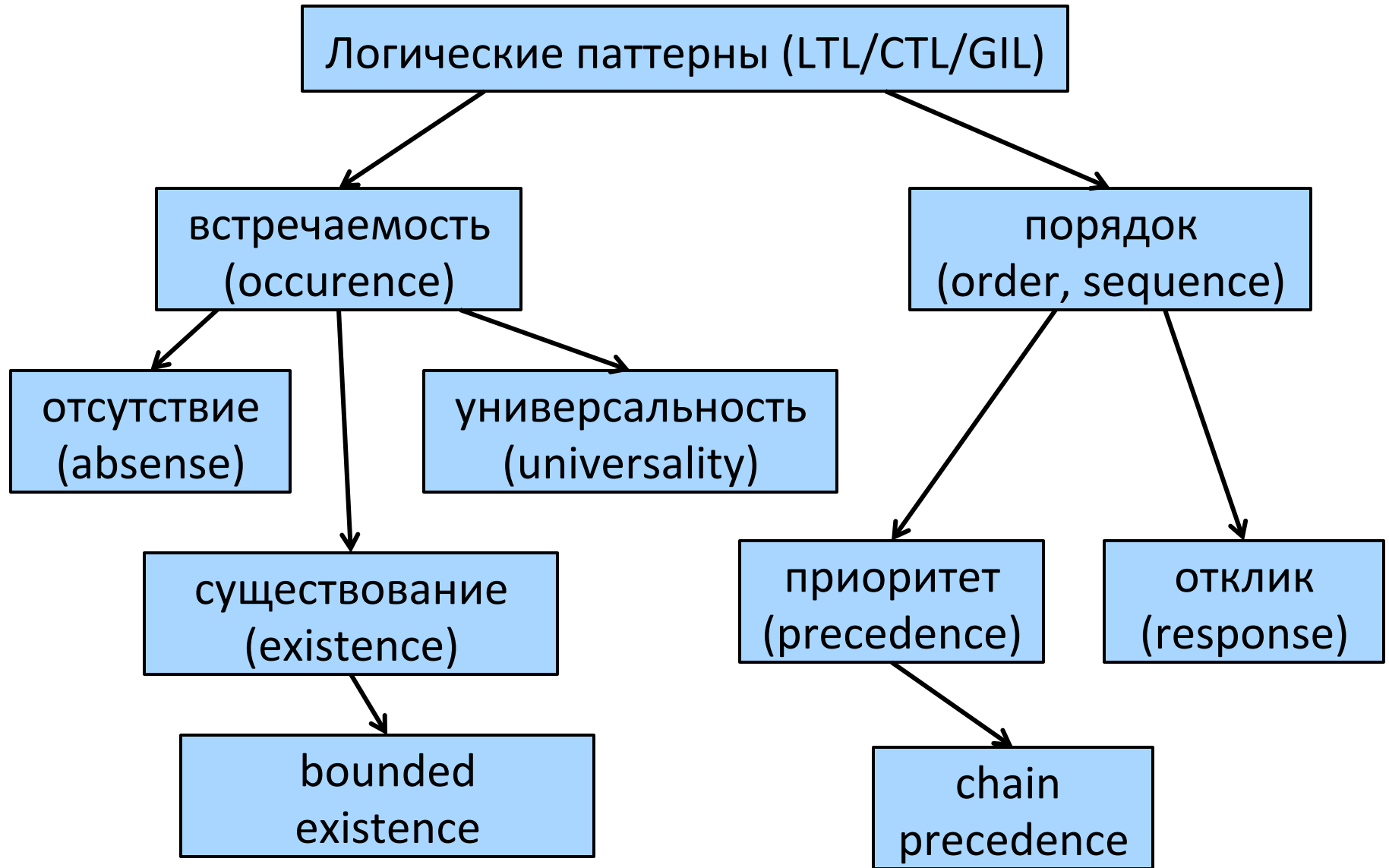
Mappings for additional formalisms have also been developed (by other researchers)

- **Action Computation Tree Logic** (ACTL)
- **Regular Alternation-Free Mu-Calculus**

Готово

# База шаблонов темпоральной логики

<http://patterns.projects.cis.ksu.edu>





# База шаблонов темпоральной логики

<http://patterns.projects.cis.ksu.edu>

- Для каждого шаблона – пять вариантов формул:

имя

пример для “absense” и LTL

всегда

$[\ ] (!p)$

перед r

$\langle \rangle r \rightarrow (!p \cup r)$

после q

$[\ ] (q \rightarrow [\ ] (!p))$

между r и q

$[\ ] ((r \ \&\& \ !q \ \&\& \ \langle \rangle q) \rightarrow (!p \cup q))$

после r до q

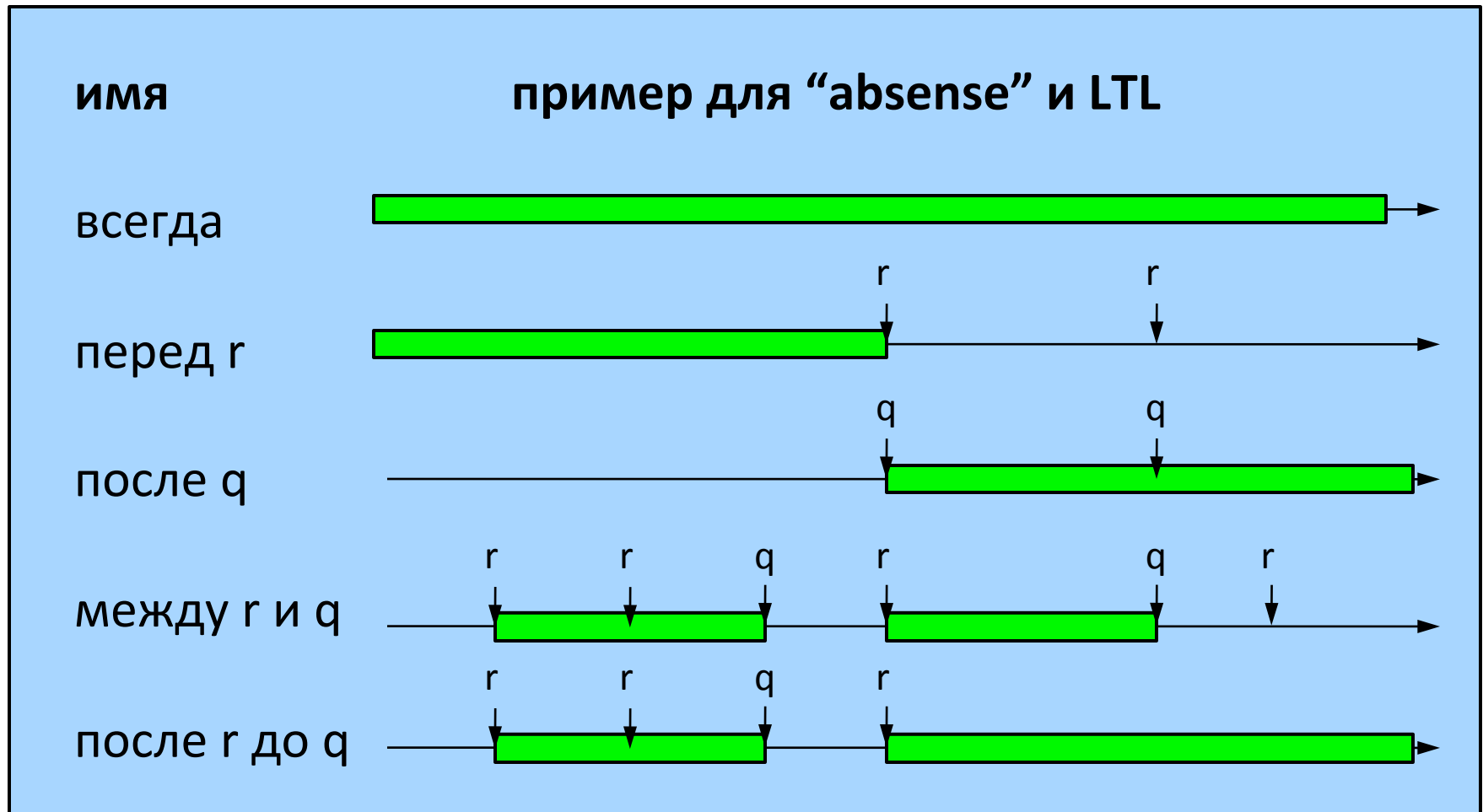
$[\ ] ((r \ \&\& \ !q) \rightarrow ((!p \cup q) \ || \ [\ ] !p))$

В чём разница?

# База шаблонов темпоральной логики

<http://patterns.projects.cis.ksu.edu>

- Для каждого шаблона – пять вариантов формул:



Спасибо за внимание!  
Вопросы?

