

# НАДЁЖНОСТЬ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ

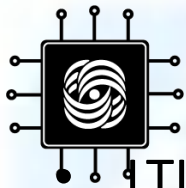
## Лекция 6: Логика TCTL *Средство верификации UPPAAL*

ВМиК МГУ им. М.В. Ломоносова,  
Кафедра АСВК, Лаборатория Вычислительных Комплексов  
Ассистент Волканов Д.Ю.



# План лекции

- Логика CTL, CTL\* и TCTL
- Общее описание средства UPPAAL
- Модуль описания
  - Пример системы реального времени
  - Описание сети автоматов
  - Синтаксис выражений
- Модуль симуляции
- Модуль верификации
  - Общее описание
  - Язык запросов



# Сравнение LTL с другими логиками

- LTL-формула описывает свойство, которое должно выполняться на **всех** вычислениях, начинающихся из исходного состояния системы

## Операторы:

|    |                         |
|----|-------------------------|
| !  | логическое отрицание    |
| && | логическое И            |
|    | логическое ИЛИ          |
| X  | в следующем             |
| U  | состоянии сильный until |
| U  | слабый until            |
| <> | рано или поздно         |
| [] | всегда                  |

Серым отмечены операторы, которые можно вывести из других:

|                                 |    |                                                       |
|---------------------------------|----|-------------------------------------------------------|
| $\varphi_1 \parallel \varphi_2$ | == | $!(\neg \varphi_1 \&\& \neg \varphi_2)$               |
| $\langle \rangle \varphi$       | == | $\text{true} \cup \varphi$                            |
| $[\ ] \varphi$                  | == | $\neg \langle \rangle \neg \varphi$                   |
| $\varphi_1 \cup \varphi_2$      | == | $[\ ] \varphi_1 \parallel (\varphi_1 \cup \varphi_2)$ |

## грамматика:

### пропозициональные формулы:

p  
!f  
(f)  
f && f  
f || f

### темпоральные формулы:

f  
!  $\varphi$   
( $\varphi$ )  
 $\varphi \&\& \varphi$   
 $\varphi \parallel \varphi$   
X  $\varphi$   
 $\varphi \cup \varphi$   
 $\langle \rangle \varphi$   
 $[\ ] \varphi$

p – некоторый пропозициональный символ

f – некоторая пропозициональная формула

$\varphi$  – некоторая темпоральная формула



# Логика STL\*

Логика ветвящегося времени:

- использует кванторы  $\forall$  и  $\exists$ ,
- использует F вместо  $\langle \rangle$  и G вместо  $[]$ .

## Операторы

|    |                        |
|----|------------------------|
| !  | логическое отрицание   |
| && | логическое И           |
|    | логическое ИЛИ         |
| E  | существует путь        |
| A  | для всех путей         |
| X  | в следующем            |
| U  | состоянии until        |
| F  | (сильный)              |
| G  | рано или поздно всегда |

Серым отмечены операторы, которые можно вывести из других:

$$\begin{aligned} \varphi_1 || \varphi_2 &= \neg (!\varphi_1 \&\& \neg \varphi_2) \\ &= A \varphi &= \text{true } U \varphi \\ &= \neg F \neg \varphi \end{aligned}$$

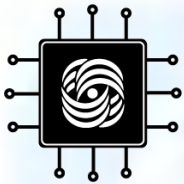
## формулы состояния: p

!f  
(f)  
f && f  
f || f  
A  $\varphi$   
E  $\varphi$

## формулы пути:

f  
!  $\Phi$   
( $\varphi$ )  
 $\varphi \&\& \varphi$   
 $\varphi || \varphi$   
X  $\varphi$   
 $\varphi U \varphi$   
F  $\varphi$   
G  $\varphi$

p – некоторый пропозициональный символ  
f – некоторая формула состояния  
 $\varphi$  – некоторая формула пути



# Логика CTL

- Логика CTL – фрагмент логики CTL\*, в котором под управлением квантора пути (E или A) может находиться не более одного оператора X или U.

Корректная CTL формула:

$p$   
 $\neg \varphi$   
 $\varphi \ \&\& \ \varphi$   
 $\varphi \ || \ \varphi$   
 $E \ X \ \varphi$   
 $E \ (\varphi \ U \ \varphi)$   
 $A \ (\varphi \ U \ \varphi)$

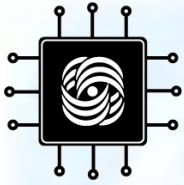
$p$  – некоторый пропозициональный символ

$f$  – некоторая формула состояния

$\varphi$  – некоторая формула пути

Можно вывести:

|      |    |             |
|------|----|-------------|
| EF f | == | E(true U f) |
| AF f | == | A(true U f) |
| EG f | == | !AF !f      |
| AG f | == | !EF !f      |
| AX f | == | !EX !f      |



# Пример

В LTL  $\langle \rangle p$  означает:

**$A\langle \rangle p$**                       **для всех** вычислений, начинающихся  
в исходном состоянии  $s_0$ , выполняется  $\langle \rangle p$

В CTL можно  
выразить:

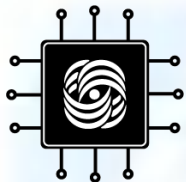
**$EF(p)$**                       **существует** вычисление, для  
которого выполняется  $\langle \rangle p$

**$AF(p)$**                       **для всех** вычислений выполняется  $\langle \rangle p$

**$AG(p)$**                       **для всех** вычислений  $p$  – инвариант

**$EG(p)$**                       **существует** вычисление, для которого  
выполняется  $p$  – инвариант

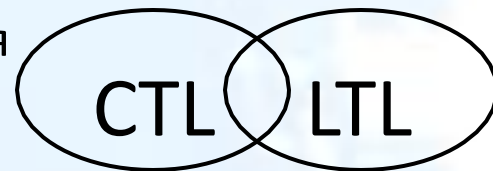
ИТД.



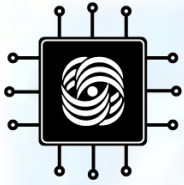
# Выразительные возможности CTL\* и

- CTL\* и CTL описывают подмножества  $\omega$ -регулярных автоматов над *деревьями*

- автоматы над деревьями более выразительны, чем автоматы над словами (CTL-формула выполнима на дереве трасс, а не на одной трассе);
- CTL и LTL являются подмножествами CTL\*;
- CTL и LTL не сравнимы по выразительной мощности (пересекаются, но не включают);
- на LTL можно описать свойства, не выразимые на CTL:
  - CTL не позволяет описать свойства вида  $[\ ]\langle \rangle(p)$ ,
  - при помощи  $[\ ]\langle \rangle(p)$  в LTL задаются ограничения справедливости;



- на CTL можно описать свойства, не выразимые на LTL:
  - на LTL нельзя описать свойства вида  $AGEF(p)$ ,
  - $AGEF(p)$  используется для описания свойства reset: из любого состояния система может перейти в нормальное.



# Выразительная мощность

Модальное  $\mu$ -исчисление,  
Автоматы над  $\omega$ -деревьями

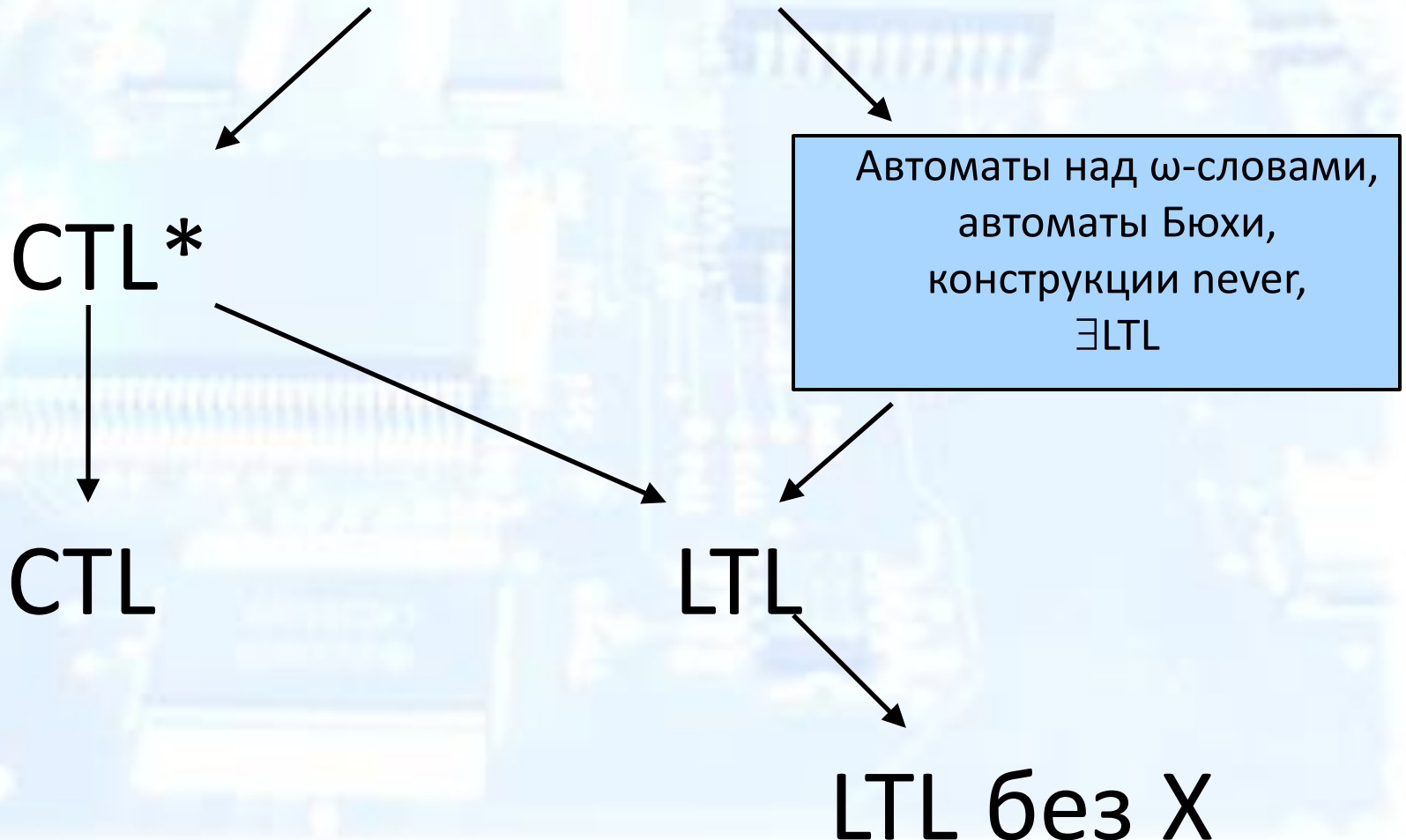
CTL\*

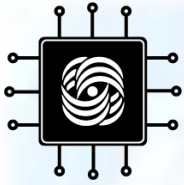
CTL

Автоматы над  $\omega$ -словами,  
автоматы Бюхи,  
конструкции never,  
 $\exists$ LTL

LTL

LTL без X





# Примеры формализаций высказываний

- *Джейн вышла замуж и родила ребенка*

$P( \text{Джейн\_выходит\_замуж} \wedge F \text{ Джейн\_рожает\_ребенка} )$

- *Джейн родила ребенка и вышла замуж*

$P( \text{Джейн\_рожает\_ребенка} \wedge F \text{ Джейн\_выходит\_замуж} )$

- *Джон умер и его похоронили*

$P( \text{Джон\_умирает} \wedge XF \text{ Джона\_хоронят} )$

- *Если я видел ее раньше, то я ее узнаю при встрече*

$G( P \text{ Увидел} \Rightarrow G( \text{Встретил} \Rightarrow X \text{ Узнал} ) )$

- *Ленин – жил, Ленин – жив, Ленин – будет жить* (В.В.Маяковский)

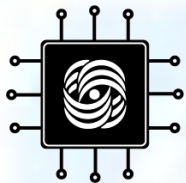
$PG \text{ Ленин\_жив}$

- *Любое посланное сообщение будет получено*

$G( \text{Послано}(m) \Rightarrow F \text{ Получено}(m) )$

- *Вчера он сказал, что придет завтра, значит, он придет сегодня*

$X^{-1}X \text{ Приходит} \Rightarrow \text{Приходит} \quad (\text{истинно})$



# LTL и CTL – подклассы CTL\*

Алгоритмы проверки выполнения формул CTL\* сложны  $\Rightarrow$  нужны подклассы

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

В CTL каждый темпоральный оператор предваряется квантором пути A или E

Формулы LTL:

$AG(p \Rightarrow Fq)$

$A(\neg a \vee Gb \ \& \ (aU \neg c))$

$A(aU \neg b)$

Формулы CTL:

$AG(p \ \& \ \neg EF(q \Rightarrow r))$

$EF(a \ \& \ E(aU \neg c))$

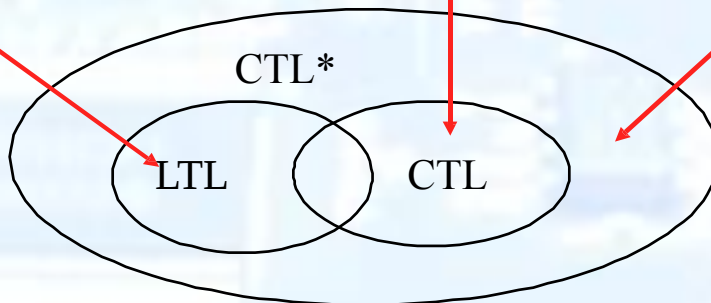
$A(aU \neg b)$

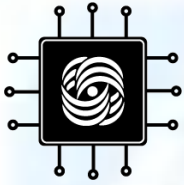
Формулы CTL\*:

$E(\neg p \ \& \ XAFq)$

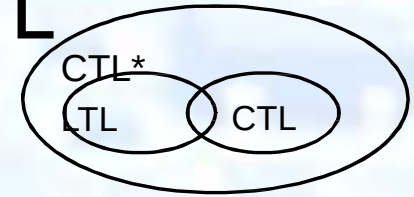
$EX(a \ \& \ AX(bUc))$

$A(aU \neg (Fb))$

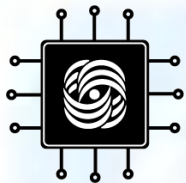




# Сравнение логик LTL и CTL



- Формулы этих двух логик характеризуют свойства разных объектов
  - LTL – формулы пути, CTL – формулы состояний
- Выражают свойства вычислений, которые представлены по-разному
  - LTL – множество поведений, CTL – деревья поведений
- Интерпретируются по-разному
  - формулы LTL - на бесконечном множестве поведений
  - формулы CTL – на конечном множестве состояний
- Методы анализа - алгоритмы model checking - совершенно разные
- Выразительная мощь несравнима: есть формулы CTL, невыразимые в LTL, и наоборот
  - например,  $FG\phi$  не может быть выражена в CTL (с некоторого времени в будущем  $\phi$  будет все время истинным)  $AFAGp$  сильнее,  $AFEGp$  – слабее
  - например,  $AGEF\phi$  в CTL не может быть выражена в LTL

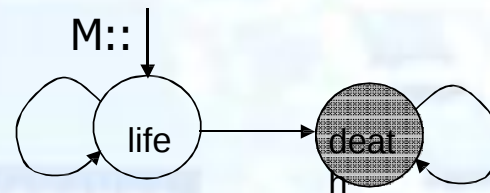


# Пример формализации в CTL\*

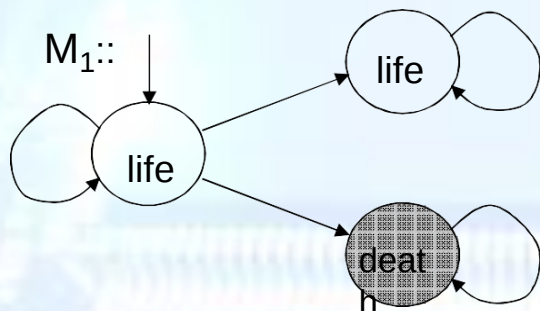
*“Летят за днями дни, и каждый час уносит  
Частичку бытия, а мы с тобой вдвоем  
Предполагаем жить, и глядь — как раз - умрем”*

*А.С.Пушкин*

$$\varphi_{br} = \mathbf{A} [(\mathbf{G} \text{ life}) \Rightarrow (\mathbf{GEX} \text{ death})]$$

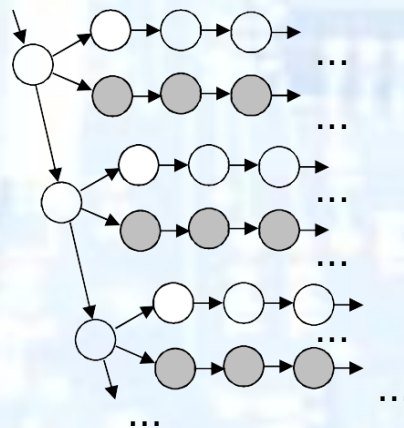


$$M \models \varphi_{br}$$

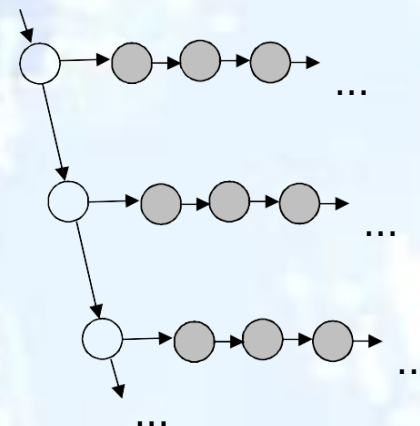


$$M_1 \not\models \varphi_{br}$$

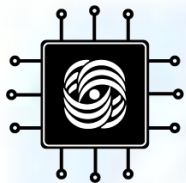
Развертка M1:



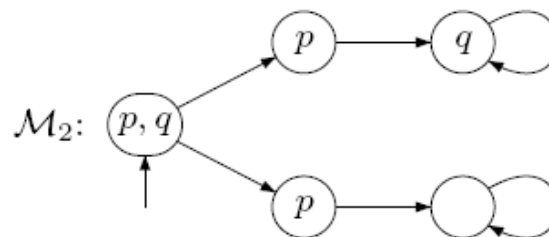
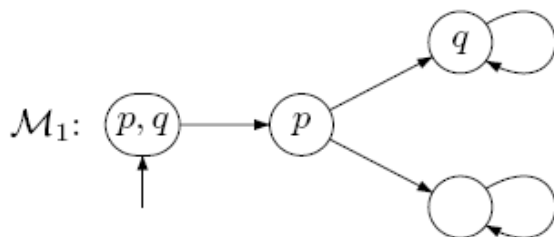
Развертка M:



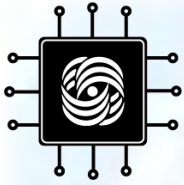
На развёртке M1 формула  $\varphi_{br}$  не выполняется: ее развертка имеет траектории “вечной жизни” без возможного ответвления на состояния, помеченные *death*



# Различия LTL и CTL\*



- Эти две модели не различаются LTL, но различаются CTL\*:
  - $AG(p \rightarrow EFq)$ . В  $\mathcal{M}_1$  эта формула выполняется, в  $\mathcal{M}_2$  она не выполняется
  - В LTL обе модели представляются одним и тем же множеством из двух вычислений, поэтому LTL их не различает



# TCTL

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

Грамматика TCTL (= CTL + Time):

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

$p$  – атомарный предикат

$\alpha$  - ограничение на таймеры и формульные часы  $z$  – формульные часы

$z \text{ in } \phi$  - введение новых часов в формулу  $\phi$

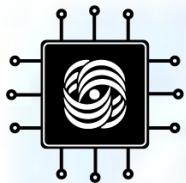
$E[\phi U \phi], A[\phi U \phi]$  – как в CTL

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

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

Формулы TCTL включают  $E[\phi U_{\sim k}\psi], A[\phi U_{\sim k}\psi], EF_{\sim k}\phi, EG_{\sim k}\phi, AF_{\sim k}\phi, AG_{\sim k}\phi$

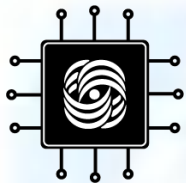
где  $\sim$  - любой символ из  $\{<, \leq, =, \geq, >\}$  и  $k$  – рациональное число



# Примеры свойств реального времени

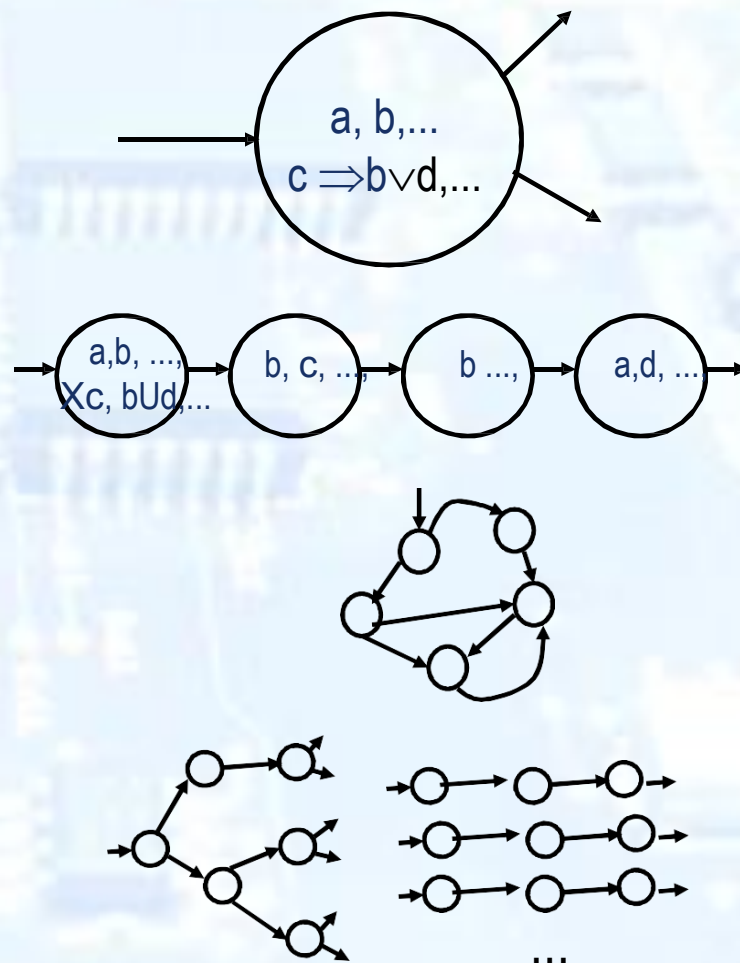
Model checking временных автоматов относительно выполнения заданной формулы TCTL сводится к model checking его регионального графа относительно выполнения формулы CTL

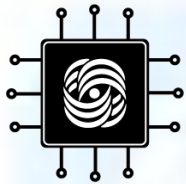
1.  $[p \text{ U}_{<2} q]$  –  $p$  истинно непрерывно до тех пор, пока не станет истинно  $q$ , и истинность  $q$  наступит не позднее, чем через 2 единицы времени
2.  $\text{AG}(\text{problem} \Rightarrow \text{AG}_{\geq 5} \text{ alarm})$  – как только проблема возникла, сигнал alarm зазвучит сразу и будет звучать не менее 5
3.  $\text{AG}(\neg \text{far} \Rightarrow \text{AF}_{<7} \text{ far})$  – поезд покинет область контроля не позже, чем через 7
4.  $\text{AG} [\text{send}(m) \Rightarrow \text{AF}_{<5} \text{ receive}(r_m)]$  – подтверждение приходит в пределах 5
5.  $\text{EG} [\text{send}(m) \Rightarrow \text{AF}_{>4} \text{ receive}(r_m)]$  - подтверждение может быть получено более, чем за 4
6.  $\text{AG} [\text{AF}_{=15} \text{ tick}]$  – тики следуют периодически точно через 15 е.в. (но, кроме того, могут быть и в промежутках)
7.  $\text{AG} (x \leq y)$  - таймер  $x$  всегда не больше таймера  $y$
8.  $\text{A} [\text{off} \text{ U } x \geq 3]$  - включен до тех пор, пока таймер  $x$  не будет иметь значение  $\geq 3$    
if off U  $x \geq 3$  to on будет



# Общие идеи в TL

- Логика высказываний строится введением
- В логике линейного времени LTL кроме атомарных утверждений и операций логики высказываний вводятся темпоральные операторы  $\{U, X\}$  (кроме них удобно использовать еще  $F$  и  $G$ )
- По конкретной цепочке состояний (миров) в каждом состоянии можем вычислить истинностные значения любой формулы темпоральной логики LTL
- В логике CTL\* добавляются кванторы пути, позволяющие различать свойства различных путей
- Формула CTL\* определена для конкретной интерпретации (структуры Крипке) и всех возможных ее вычислений
- CTL является подмножеством CTL\* - в формулах CTL **каждый** темпоральный оператор предваряется квантором пути

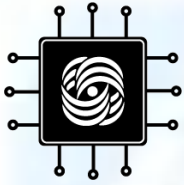




# Особенности URRAAL

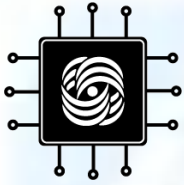
URRAAL — это весьма популярное средство верификации систем реального времени

- Простое в использовании
- Бесплатное для научных исследований
- Имеет в основе простую, но при этом довольно мощную математическую модель
- Малопригодно для разработки больших систем
- Не поддерживает описание иерархии вложенности компонентов систем



# Структура

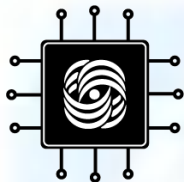
- Модуль описания
  - входная модель – расширение модели **сети временных автоматов**
  - возможность описания параметризованных шаблонов
  - наличие данных различной степени локальности
- Модуль симуляции
  - генерация и визуализации трассы сети
  - различные типы генерации (случайная, из файла)
- Модуль верификации
  - проверка темпоральных свойств
  - предоставление трассы-контрпримера
  - воспроизведение контрпримера в модуле симуляции



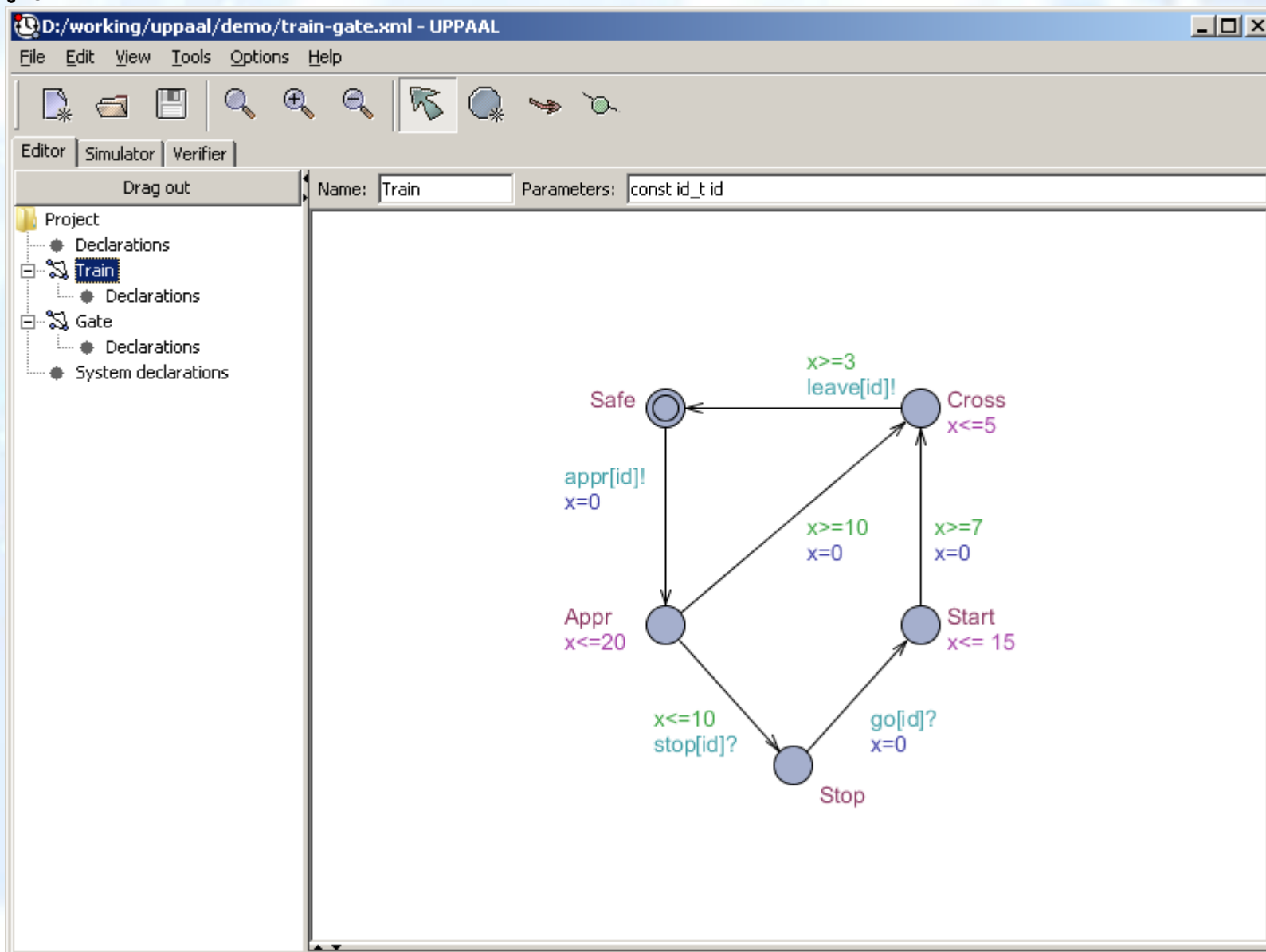
# Пример

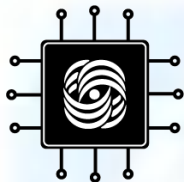
## Train Gate

- Система обеспечения доступа к критическому ресурсу
- Требуется обеспечить проход нескольких поездов по одному мосту
- В наличии устройство управления поездами
- Остановка и разгон поезда происходят не мгновенно

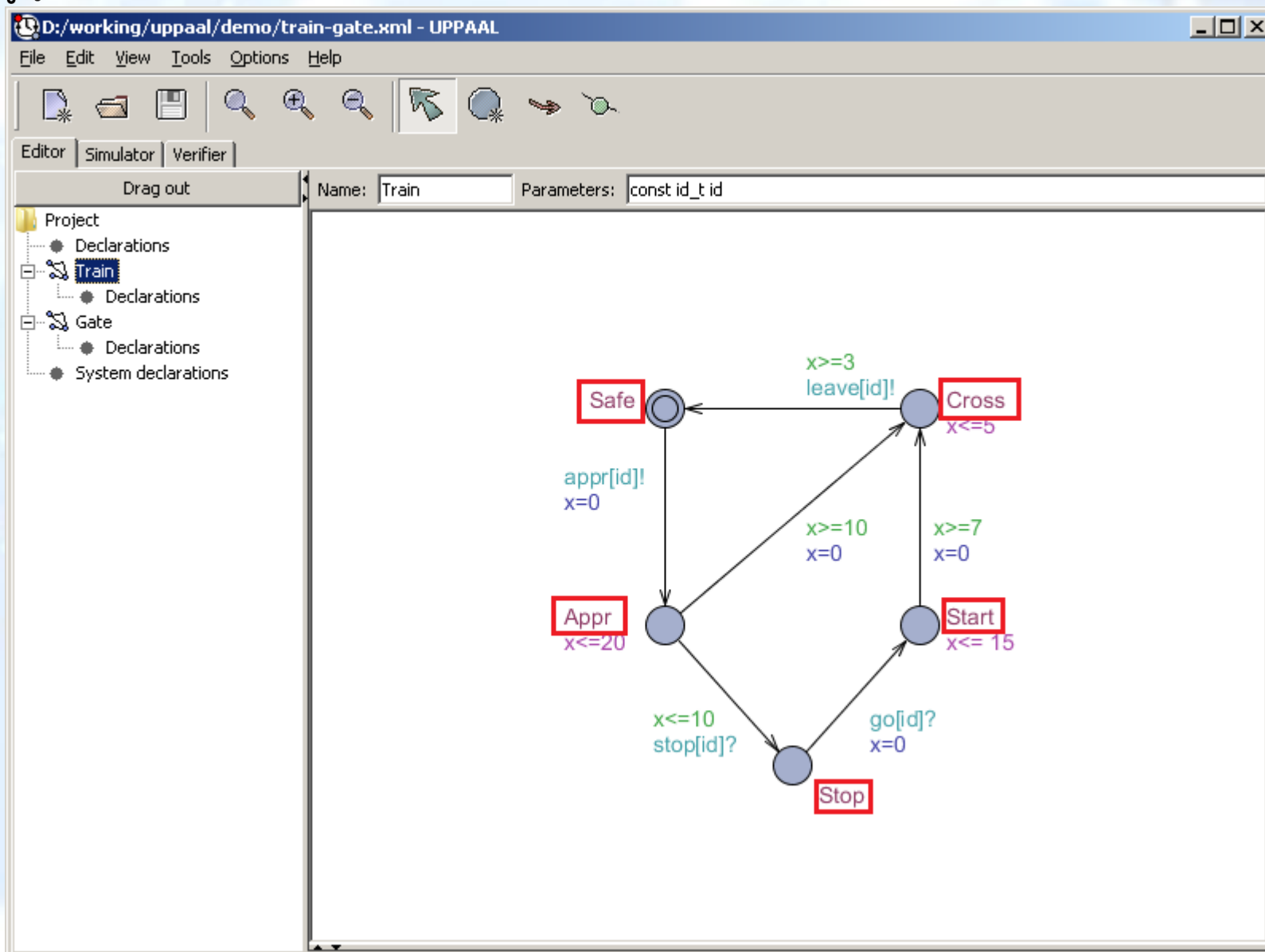


# Модуль описания



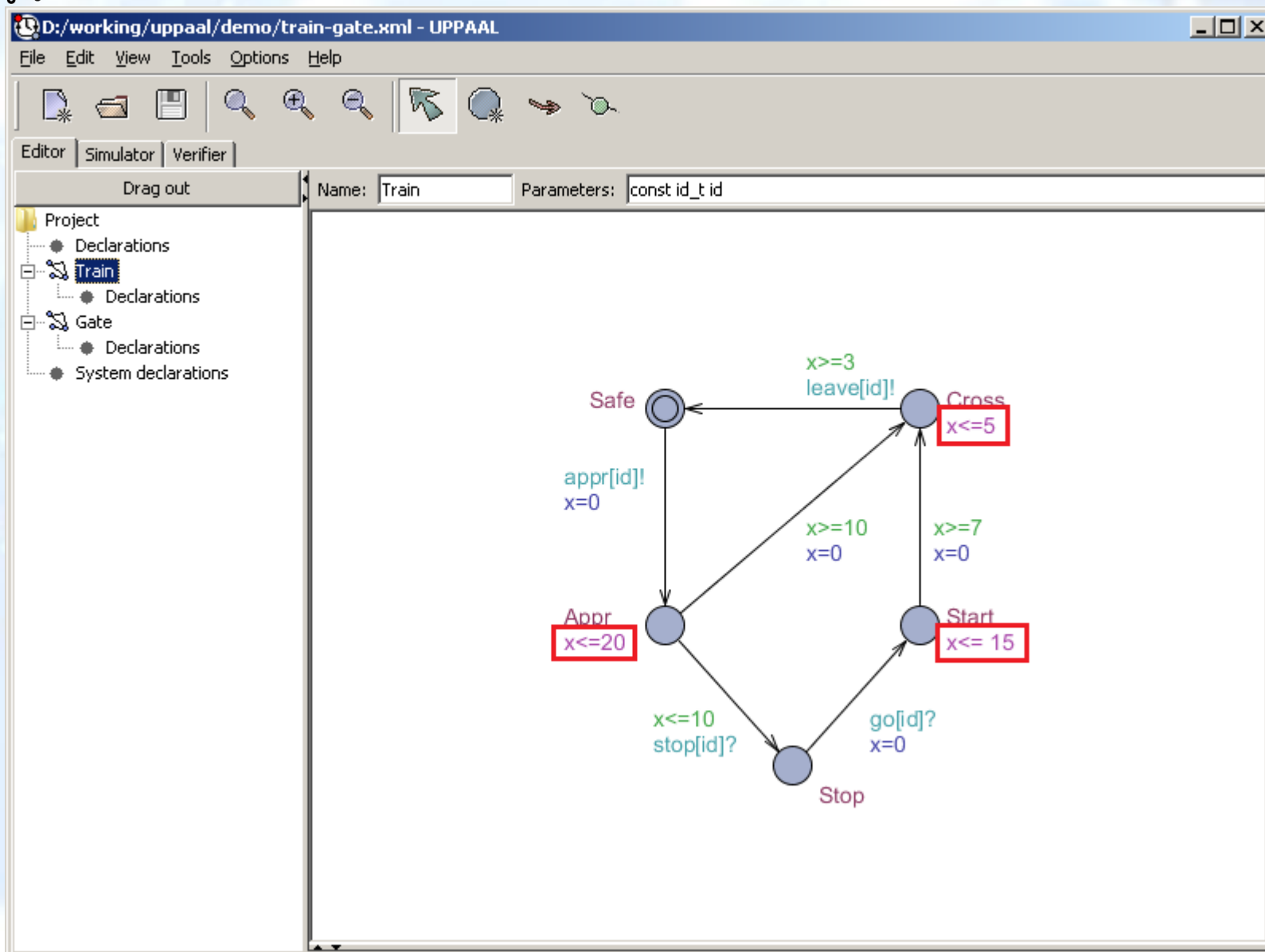


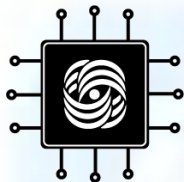
# Имена состояний



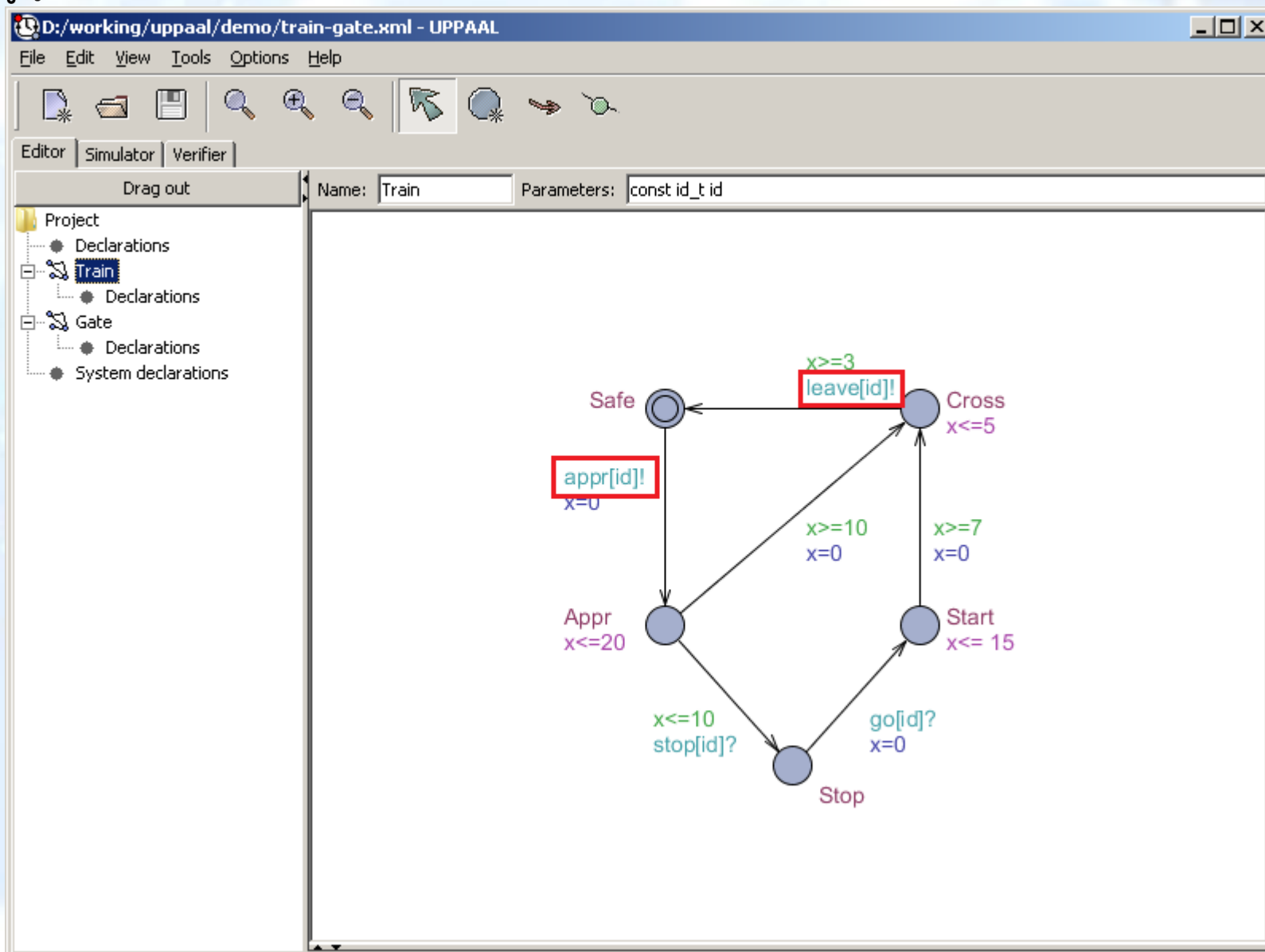


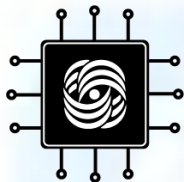
# Инварианты



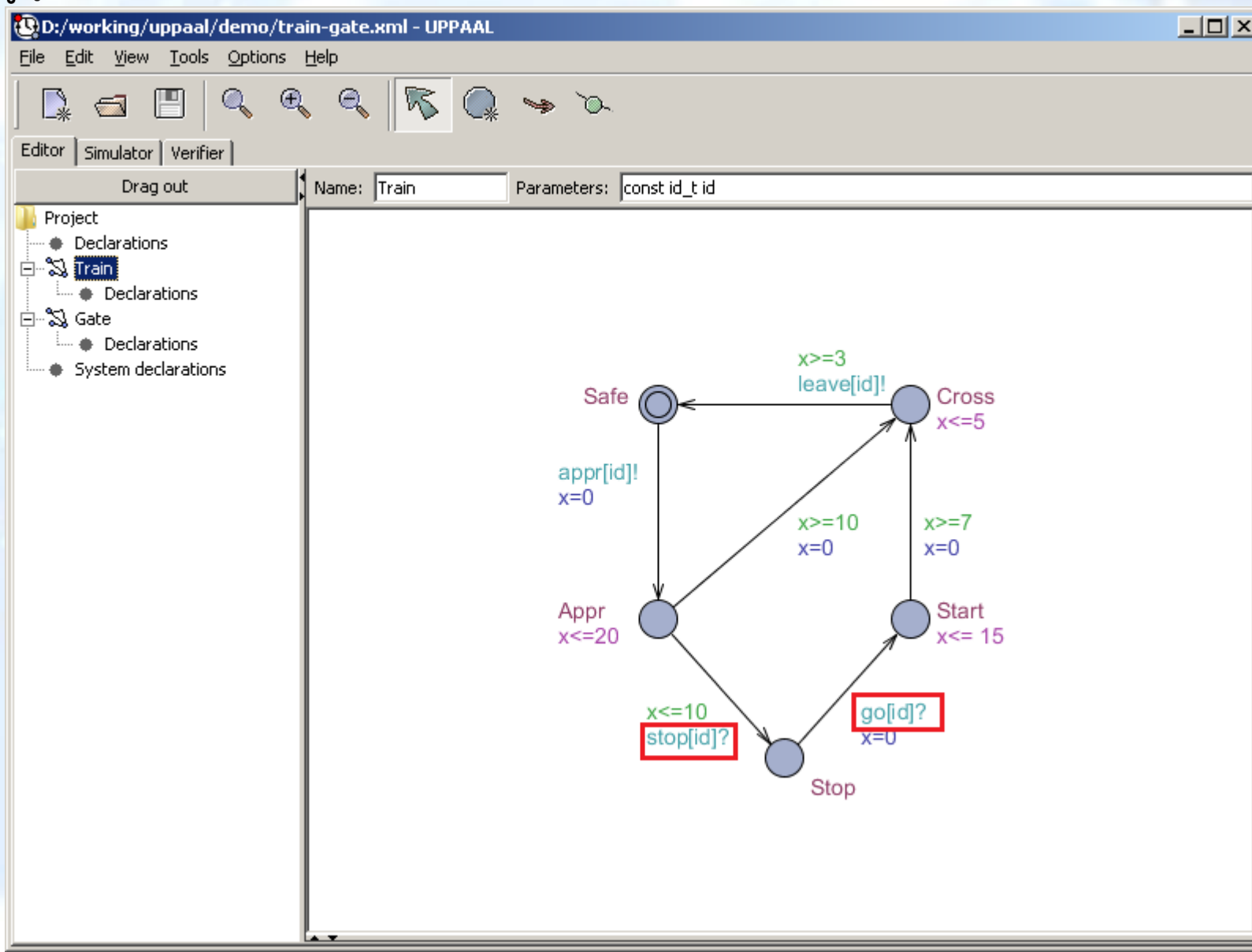


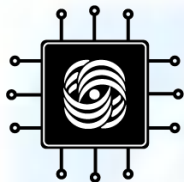
# Посылка сигналов



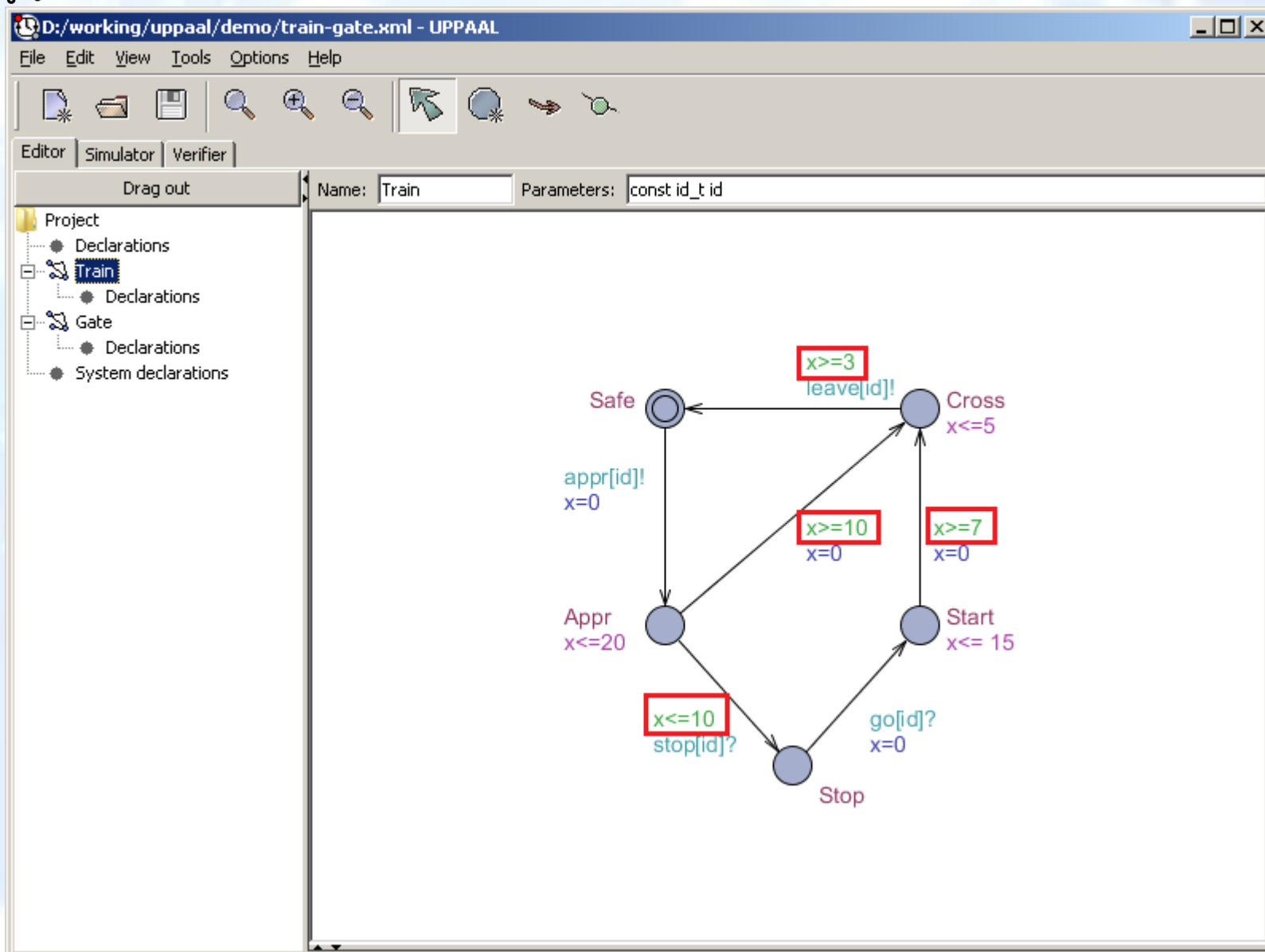


# Прием сигналов



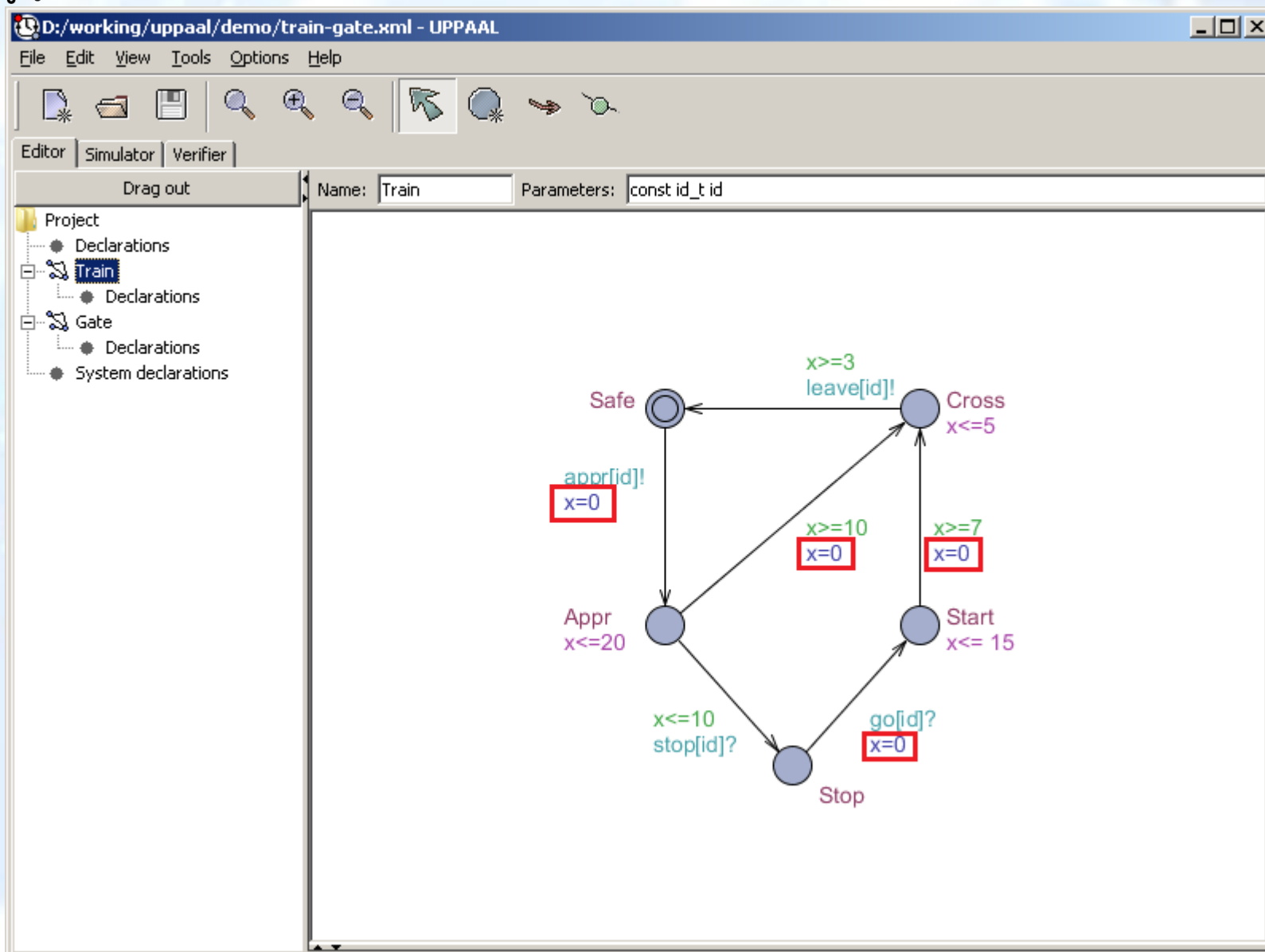


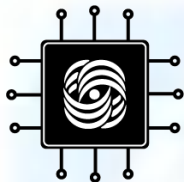
# Предусловия



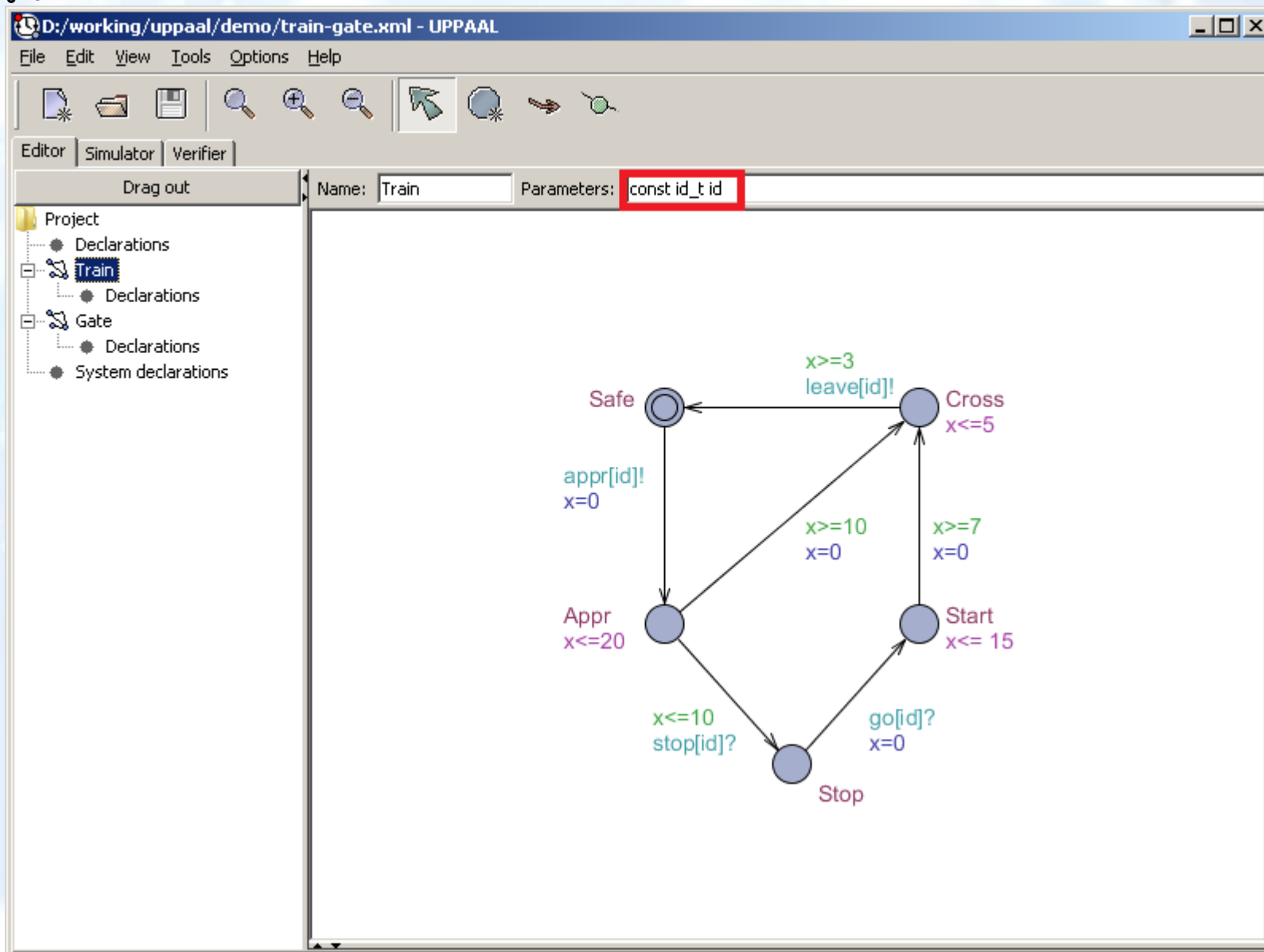


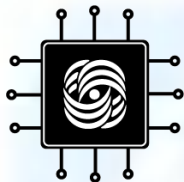
# Присваивания



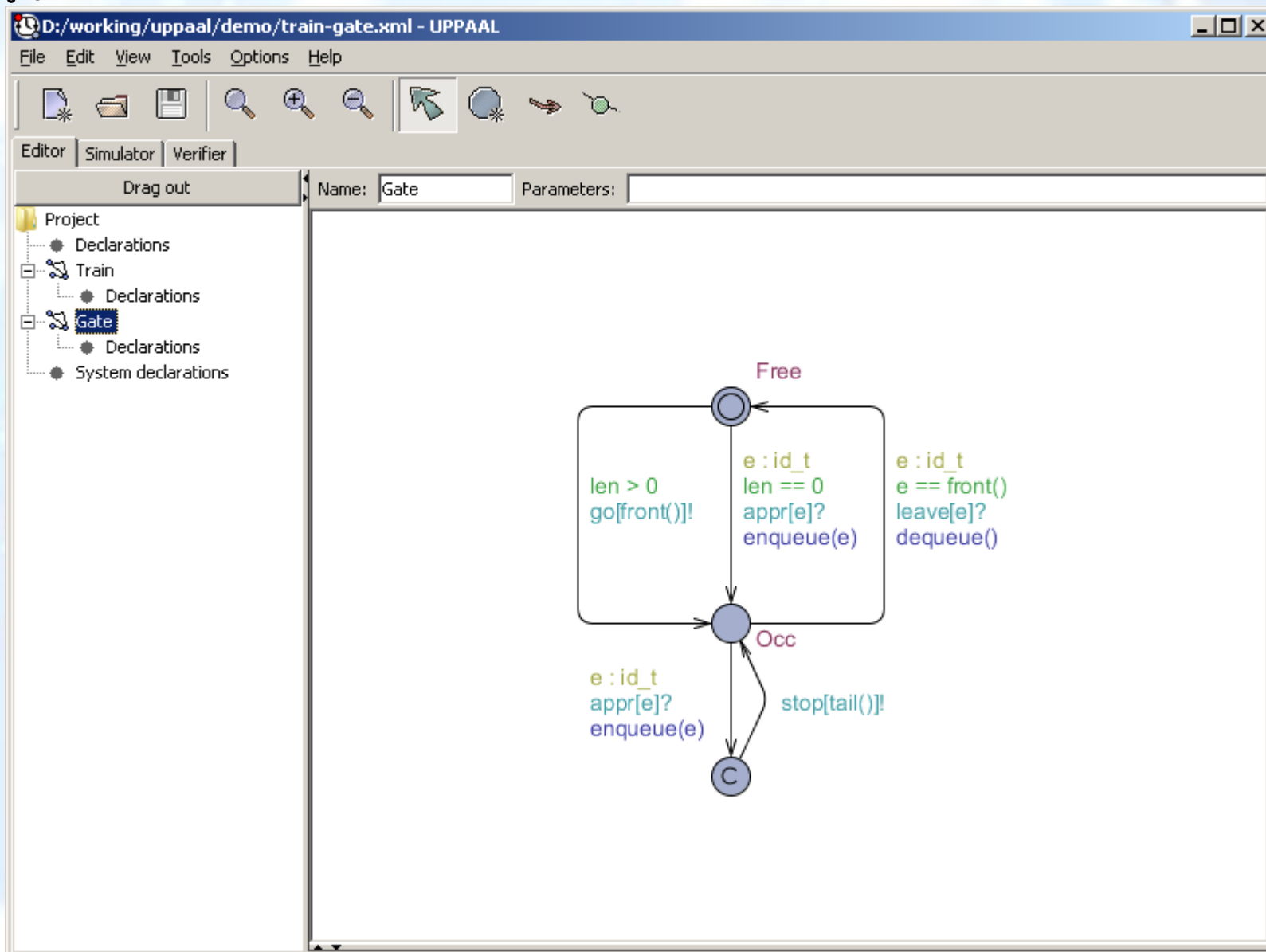


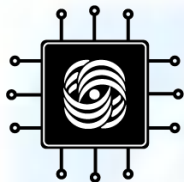
# Параметры



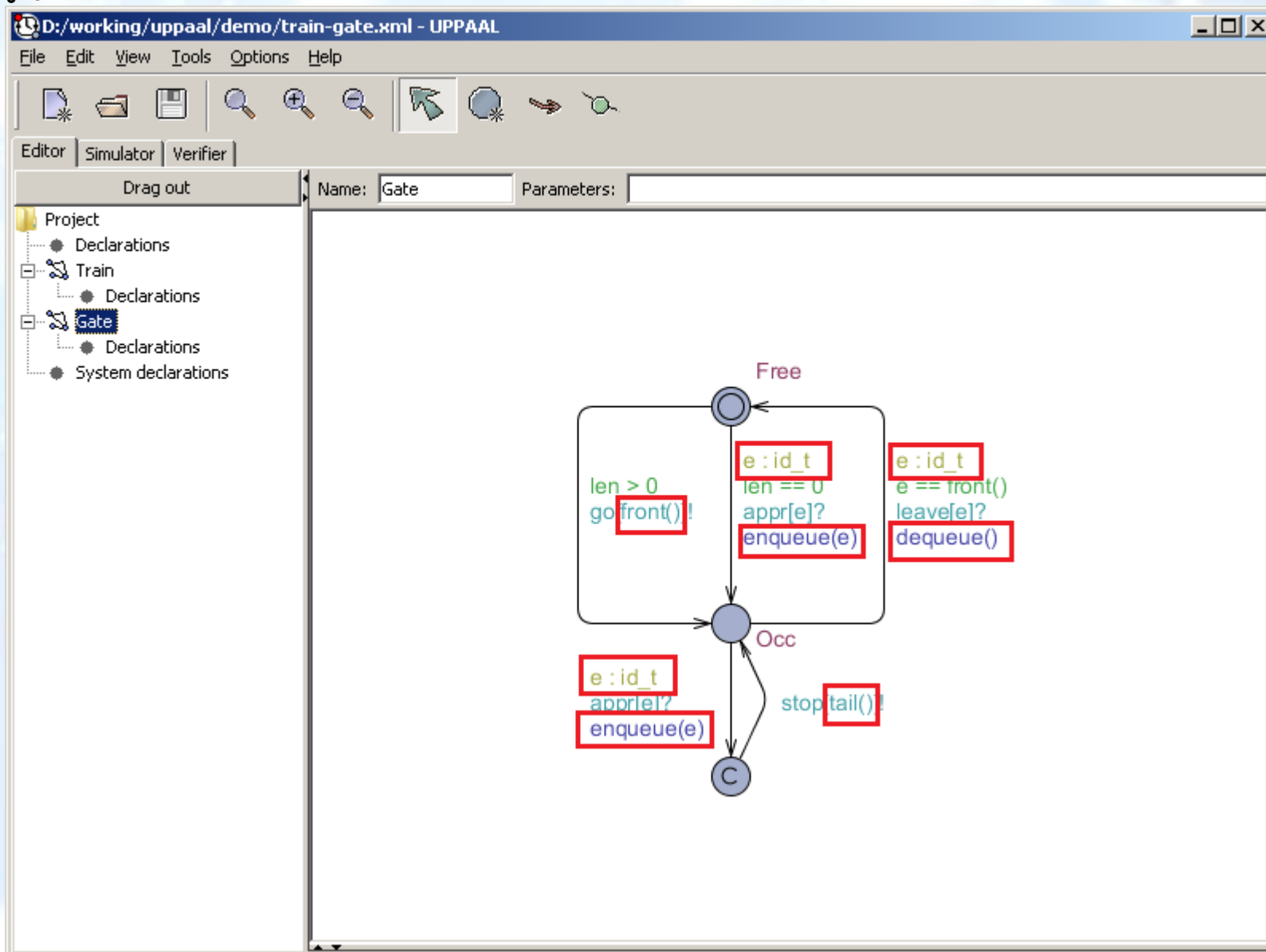


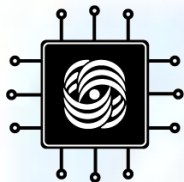
# Модуль описания



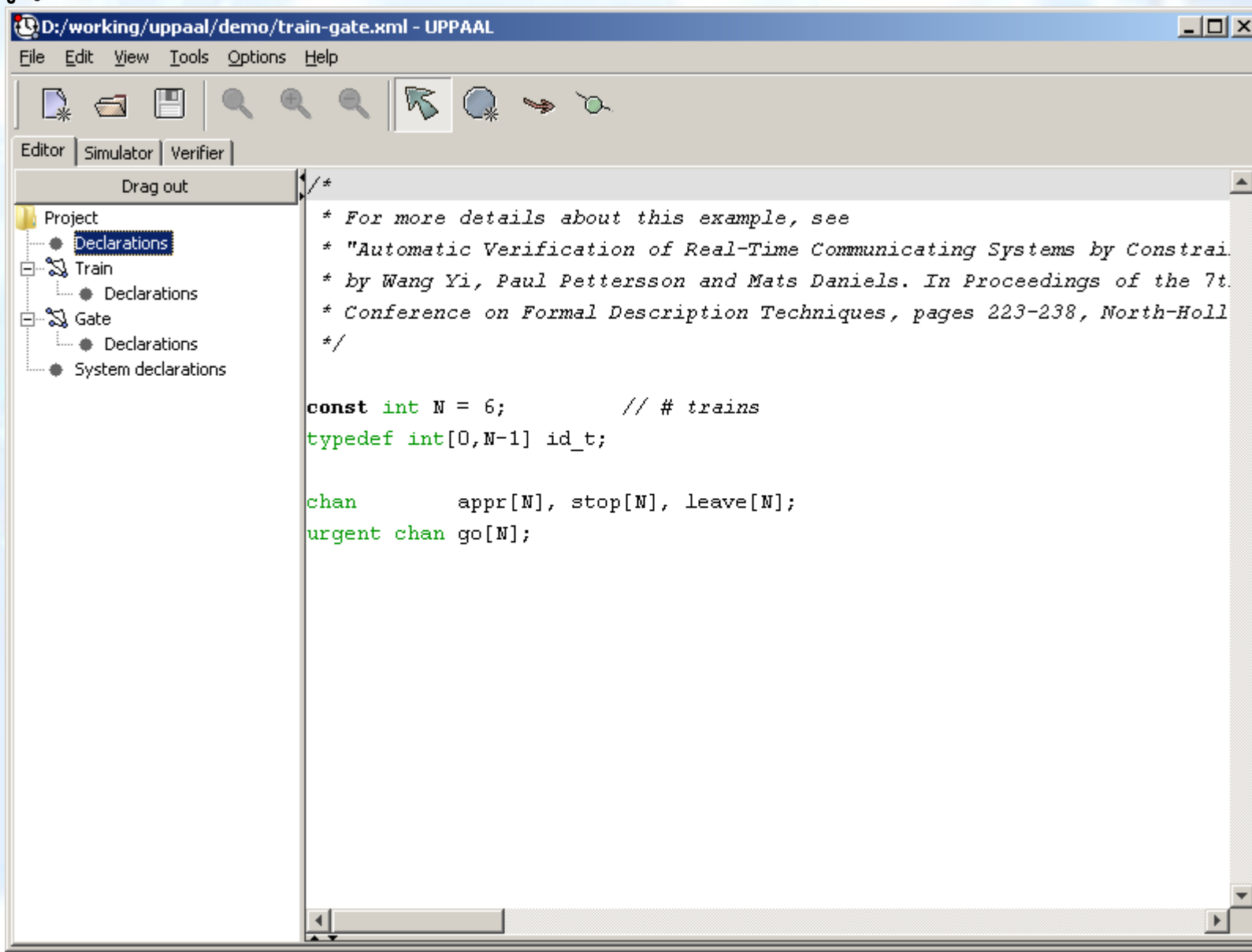


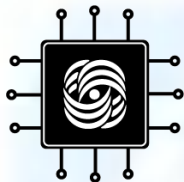
# Дополнительные возможности



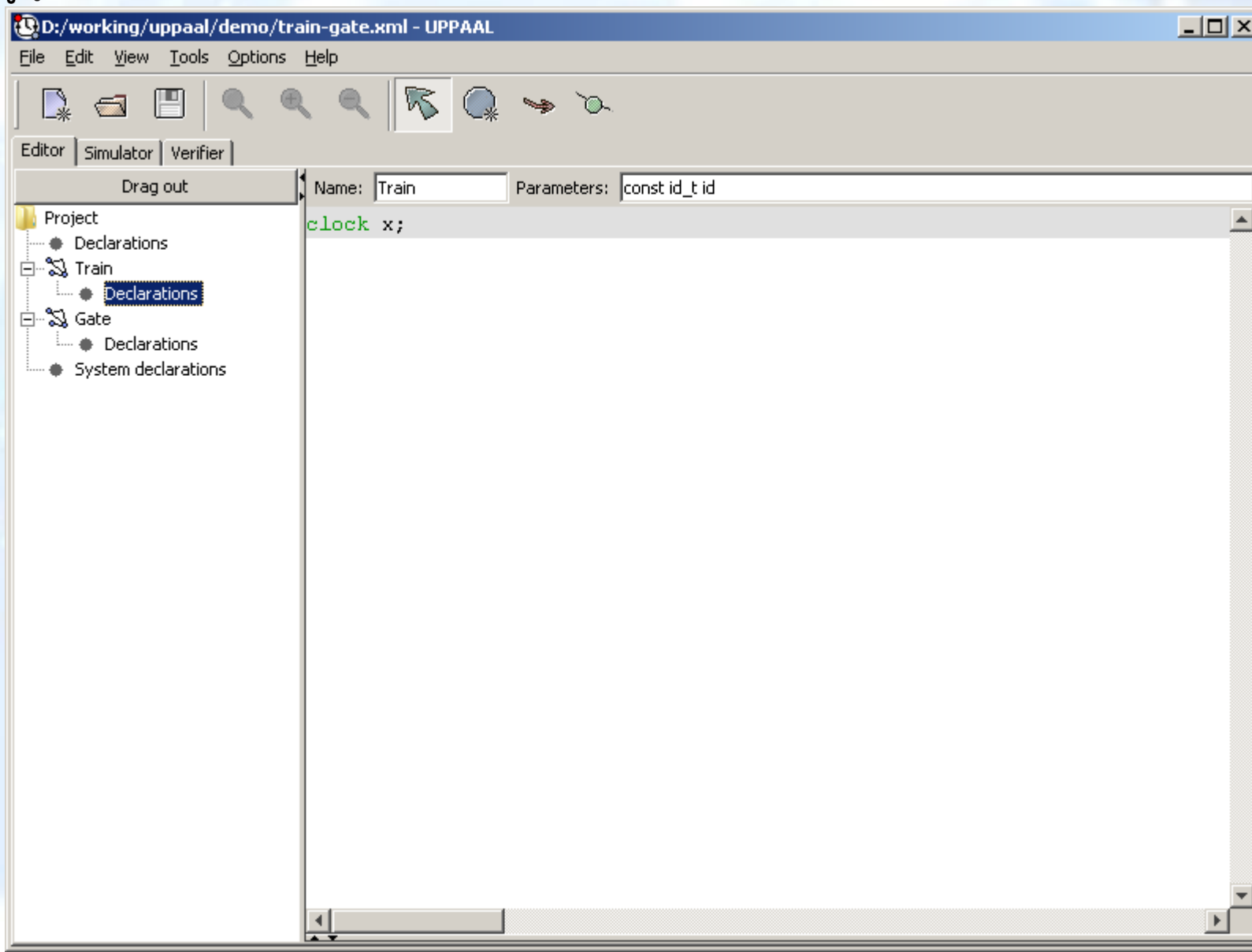


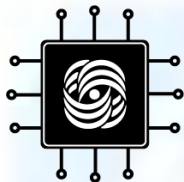
# Глобальные описания





# Локальные описания





# Локальные описания

D:/working/uppaal/demo/train-gate.xml - UPPAAL

File Edit View Tools Options Help

Editor Simulator Verifier

Drag out

Project

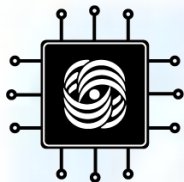
- Declarations
- Train
  - Declarations
- Gate
  - Declarations**
- System declarations

Name: Gate Parameters:

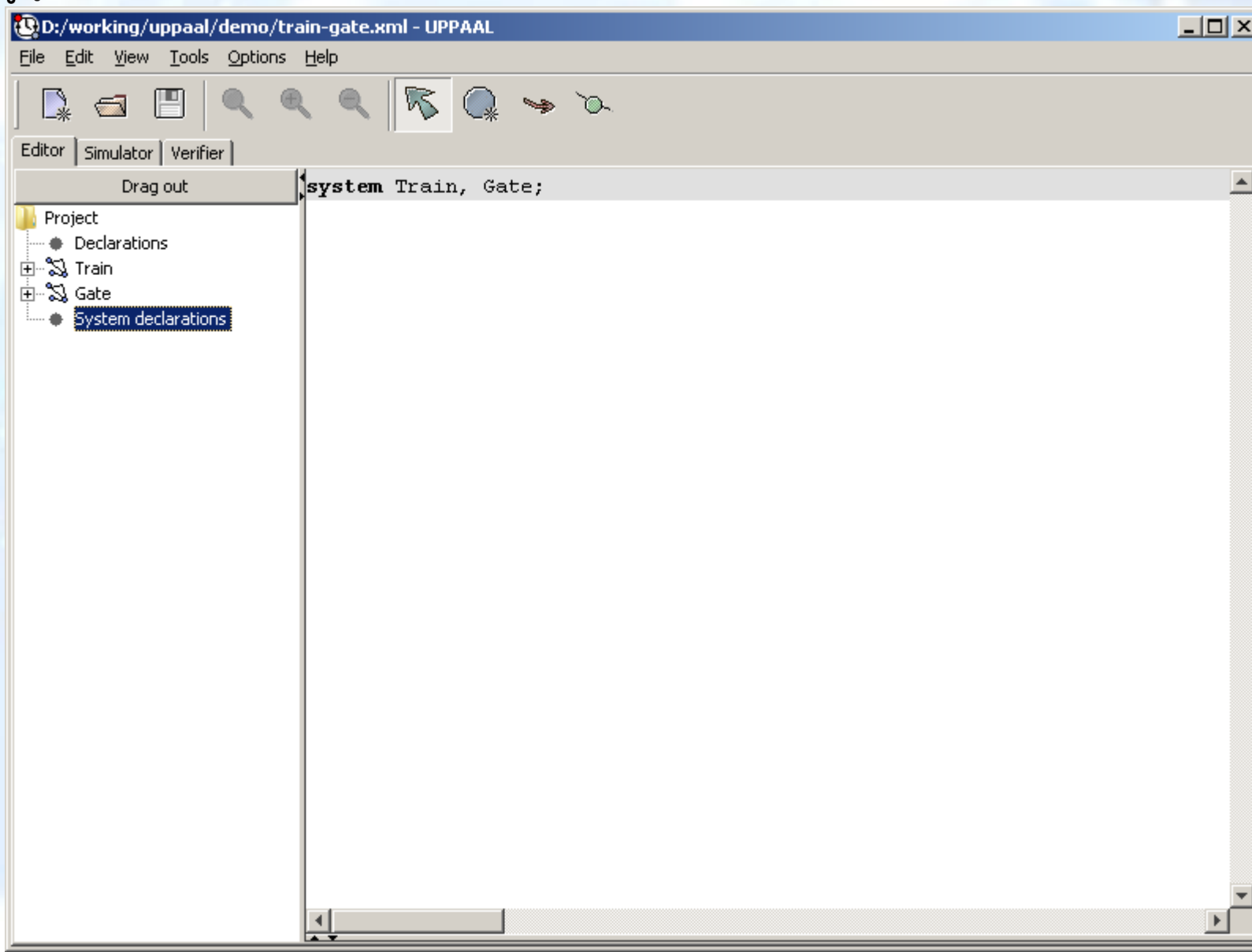
```
id_t list[N+1];
int[0,N] len;

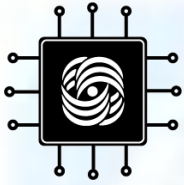
// Put an element at the end of the queue
void enqueue(id_t element)
{
    list[len++] = element;
}

// Remove the front element of the queue
void dequeue()
{
    int i = 0;
    len -= 1;
    while (i < len)
    {
        list[i] = list[i + 1];
        i++;
    }
    list[i] = 0;
}
```



# Описание системы

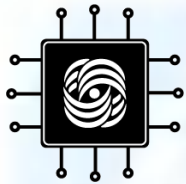




# Синтаксис выражений

- Выражение (в стиле языка C)

```
Expr ::= ID | NAT |  
        Expr[Expr] | (Expr) |  
        Expr++ | ++Expr | Expr-- | --Expr |  
        Expr AOp Expr | UOp Expr | Expr BOp Expr |  
        Expr?Expr:Expr | Expr.ID  
UPop ::= - | ! | not  
BOp ::= < | <= | == | != | >= | > | + | - |  
        * | / | % | & | | | ^ | << | >> | && |  
        || | <? | >? | and | or | imply  
AOp ::= := | += | -= | *= | /= | %= | |= |  
        &= | ^= | <<= | >>=
```



# Синтаксис выражений

## Предусловие

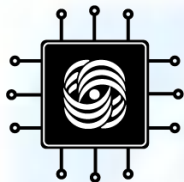
- Булево выражение
- Таймеры ( $t, t'$ ) могут встретиться только в  
 $t \text{ rel } \text{Exp} \quad t - t' \text{ rel } \text{Exp}$
- Указанные выше сравнения должны идти в предусловии подряд через конъюнкцию

## Присваивания

- Разделенные запятой выражения вида  
 $x \text{ AOp } \text{Exp}$

## Инвариант

- Булево выражение без отрицаний
- Таймеры ( $t$ ) могут встретиться только в выражениях вида  
 $t \text{ rel } \text{Exp}$



# Модуль симуляции

D:\working\uppaal\demo\train-gate.xml - UPPAAL

File Edit View Tools Options Help

Editor Simulator Verifier

Drag out

Enabled Transitions

appr[0]: Train(0) --> Gate  
appr[1]: Train(1) --> Gate  
appr[2]: Train(2) --> Gate  
appr[3]: Train(3) --> Gate  
appr[4]: Train(4) --> Gate  
appr[5]: Train(5) --> Gate

Next Reset

Simulation Trace

(Safe, Safe, Safe, Safe, Safe, Safe, Free)

Trace File:

Prev Next Replay  
Open Save Auto

Slow Fast

Drag out

Gate.list[0] = 0  
Gate.list[1] = 0  
Gate.list[2] = 0  
Gate.list[3] = 0  
Gate.list[4] = 0  
Gate.list[5] = 0  
Gate.list[6] = 0  
Gate.len = 0  
Train(0).x >= 0  
Train(1).x >= 0  
Train(2).x >= 0  
Train(3).x >= 0  
Train(4).x >= 0  
Train(5).x >= 0  
Train(0).x = Train(1).x  
Train(1).x = Train(2).x  
Train(2).x = Train(3).x  
Train(3).x = Train(4).x  
Train(4).x = Train(5).x  
Train(5).x = Train(0).x

Train(0)

Safe (red circle) → Appr (blue circle) → Stop (blue circle) → Cross (blue circle) → Safe (red circle)

Transitions for Train(0):  
Safe → Appr: appr[0]! x=0  
Appr → Stop: x<=10 stop[0]?  
Stop → Cross: go[0]? x=0  
Cross → Safe: x>=3 leave[0]!  
Cross → Appr: x>=10 x=0  
Appr → Cross: x>=7 x=0  
Cross → Cross: x<=5

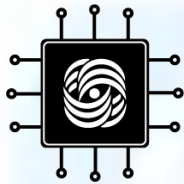
Train(1)

Safe (red circle) → Cross (blue circle) → Safe (red circle)

Transitions for Train(1):  
Safe → Cross: x>=3 leave[1]!  
Cross → Safe: x<=5

Train(0) Train(1) Train(2) Train(3) Train(4) Train(5)

Safe Safe Safe Safe Safe Safe



# Визуализация активных состояний

D:/working/uppaal/demo/train-gate.xml - UPPAAL

File Edit View Tools Options Help

Editor Simulator Verifier

Drag out

Enabled Transitions

- appr[0]: Train(0) --> Gate
- appr[1]: Train(1) --> Gate
- appr[2]: Train(2) --> Gate
- appr[3]: Train(3) --> Gate
- appr[4]: Train(4) --> Gate

Next Reset

Simulation Trace

(Safe, Safe, Safe, Safe, Safe, Safe, Free)

Trace File:

Prev Next Replay

Open Save Auto

Slow Fast

Drag out

```
Gate.list[0] = 0
Gate.list[1] = 0
Gate.list[2] = 0
Gate.list[3] = 0
Gate.list[4] = 0
Gate.list[5] = 0
Gate.list[6] = 0
Gate.len = 0
Train(0).x >= 0
Train(1).x >= 0
Train(2).x >= 0
Train(3).x >= 0
Train(4).x >= 0
Train(5).x >= 0
Train(0).x = Train(1).x
Train(1).x = Train(2).x
Train(2).x = Train(3).x
Train(3).x = Train(4).x
Train(4).x = Train(5).x
Train(5).x = Train(0).x
```

Safe

Appr  
x <= 20

Cross  
x <= 5

Start  
x <= 15

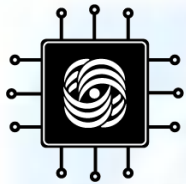
Stop

Gate

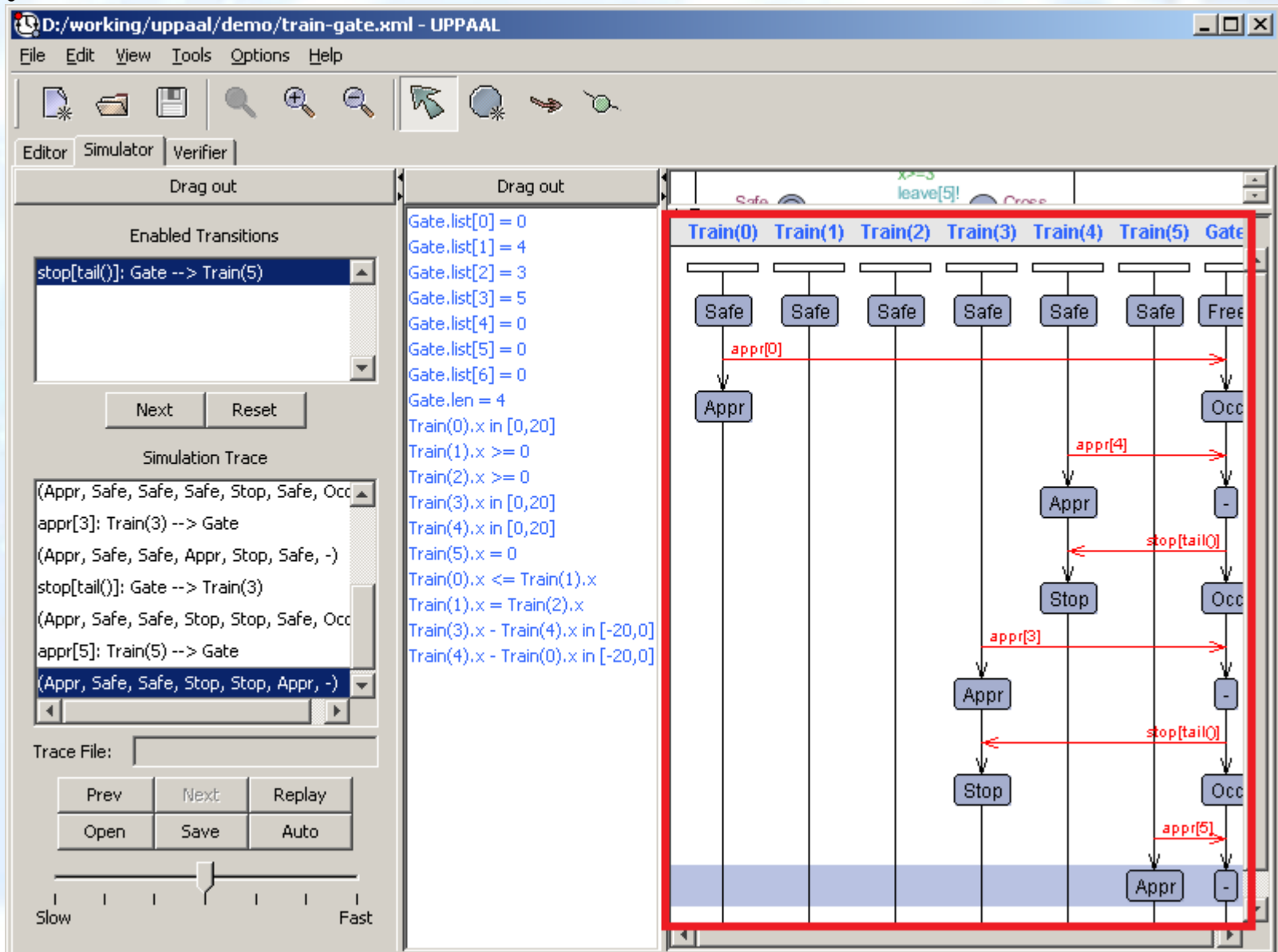
Free

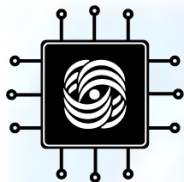
Occ

Train(0) Train(1) Train(2) Train(3) Train(4) Train(5)



# Визуализация трассы





# Окно текущих значений данных

D:\working\uppaal\demo\train-gate.xml - UPPAAL

File Edit View Tools Options Help

Editor Simulator Verifier

Drag out

Enabled Transitions

stop[tail()]: Gate --> Train(5)

Next Reset

Simulation Trace

(Appr, Safe, Safe, Safe, Stop, Safe, Occ  
appr[3]: Train(3) --> Gate  
(Appr, Safe, Safe, Appr, Stop, Safe, -)  
stop[tail()]: Gate --> Train(3)  
(Appr, Safe, Safe, Stop, Stop, Safe, Occ  
appr[5]: Train(5) --> Gate  
(Appr, Safe, Safe, Stop, Stop, Appr, -)

Trace File:

Prev Next Replay  
Open Save Auto

Slow Fast

Drag out

Gate.list[0] = 0  
Gate.list[1] = 4  
Gate.list[2] = 3  
Gate.list[3] = 5  
Gate.list[4] = 0  
Gate.list[5] = 0  
Gate.list[6] = 0  
Gate.len = 4  
Train(0).x in [0,20]  
Train(1).x >= 0  
Train(2).x >= 0  
Train(3).x in [0,20]  
Train(4).x in [0,20]  
Train(5).x = 0  
Train(0).x <= Train(1).x  
Train(1).x = Train(2).x  
Train(3).x - Train(4).x in [-20,0]  
Train(4).x - Train(0).x in [-20,0]

Train(0) Train(1) Train(2) Train(3) Train(4) Train(5) Gate

Safe Safe Safe Safe Safe Safe Free

appr[0]

Appr

appr[4]

Appr

stop[tail()

Occ

appr[3]

Appr

stop[tail()

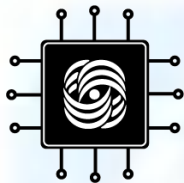
Stop

Occ

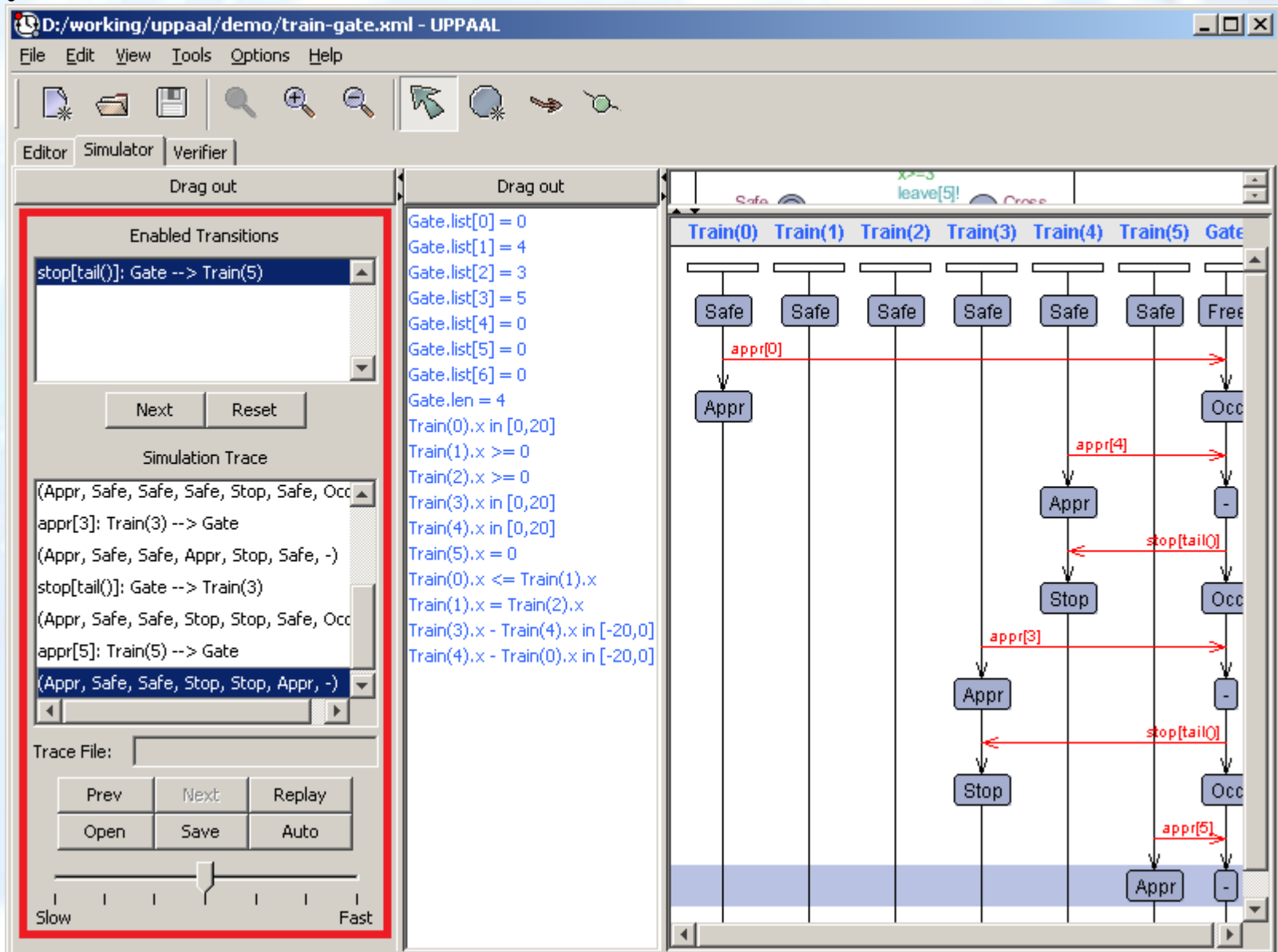
appr[5]

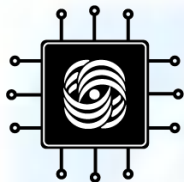
Appr

Occ



# Окно генерации трассы





# Модуль верификации

D:\working\uppaal\demo\train-gate.xml - UPPAAL

File Edit View Tools Options Help

Editor Simulator Verifier

Overview

|                    |   |
|--------------------|---|
| A[] Gate.Free      | ● |
| E<> Gate.Occ       | ● |
| E<> Train(0).Cross | ● |
| E<> Train(1).Cross | ● |

Check  
Insert  
Remove  
Comments

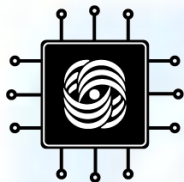
Query

A[] Gate.Free

Comment

Status

Property is satisfied.  
Train(4).Appr --> Train(4).Cross  
Property is satisfied.  
Train(5).Appr --> Train(5).Cross  
Property is satisfied.  
A[] not deadlock  
Property is satisfied.  
A[] Gate.Free  
Property is not satisfied.



# Окно результатов проверки

D:/working/uppaal/demo/train-gate.xml - UPPAAL

File Edit View Tools Options Help

Editor Simulator Verifier

Overview

|                    |             |                                                                                                                                                                   |
|--------------------|-------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| A[] Gate.Free      | Red light   | <input type="button" value="Check"/><br><input type="button" value="Insert"/><br><input type="button" value="Remove"/><br><input type="button" value="Comments"/> |
| E<> Gate.Occ       | Grey light  |                                                                                                                                                                   |
| E<> Train(0).Cross | Green light |                                                                                                                                                                   |
| E<> Train(1).Cross | Green light |                                                                                                                                                                   |

Query

A[] Gate.Free

Comment

Status

Property is satisfied.

Train(4).Appr --> Train(4).Cross

Property is satisfied.

Train(5).Appr --> Train(5).Cross

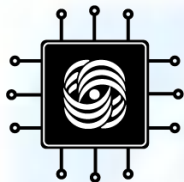
Property is satisfied.

A[] not deadlock

Property is satisfied.

A[] Gate.Free

Property is not satisfied.



# Окно текущего запроса

D:/working/uppaal/demo/train-gate.xml - UPPAAL

File Edit View Tools Options Help

Editor Simulator Verifier

Overview

|                    |   |
|--------------------|---|
| A[] Gate.Free      | ● |
| E<> Gate.Occ       | ● |
| E<> Train(0).Cross | ● |
| E<> Train(1).Cross | ● |

Check  
Insert  
Remove  
Comments

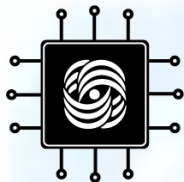
Query

A[] Gate.Free

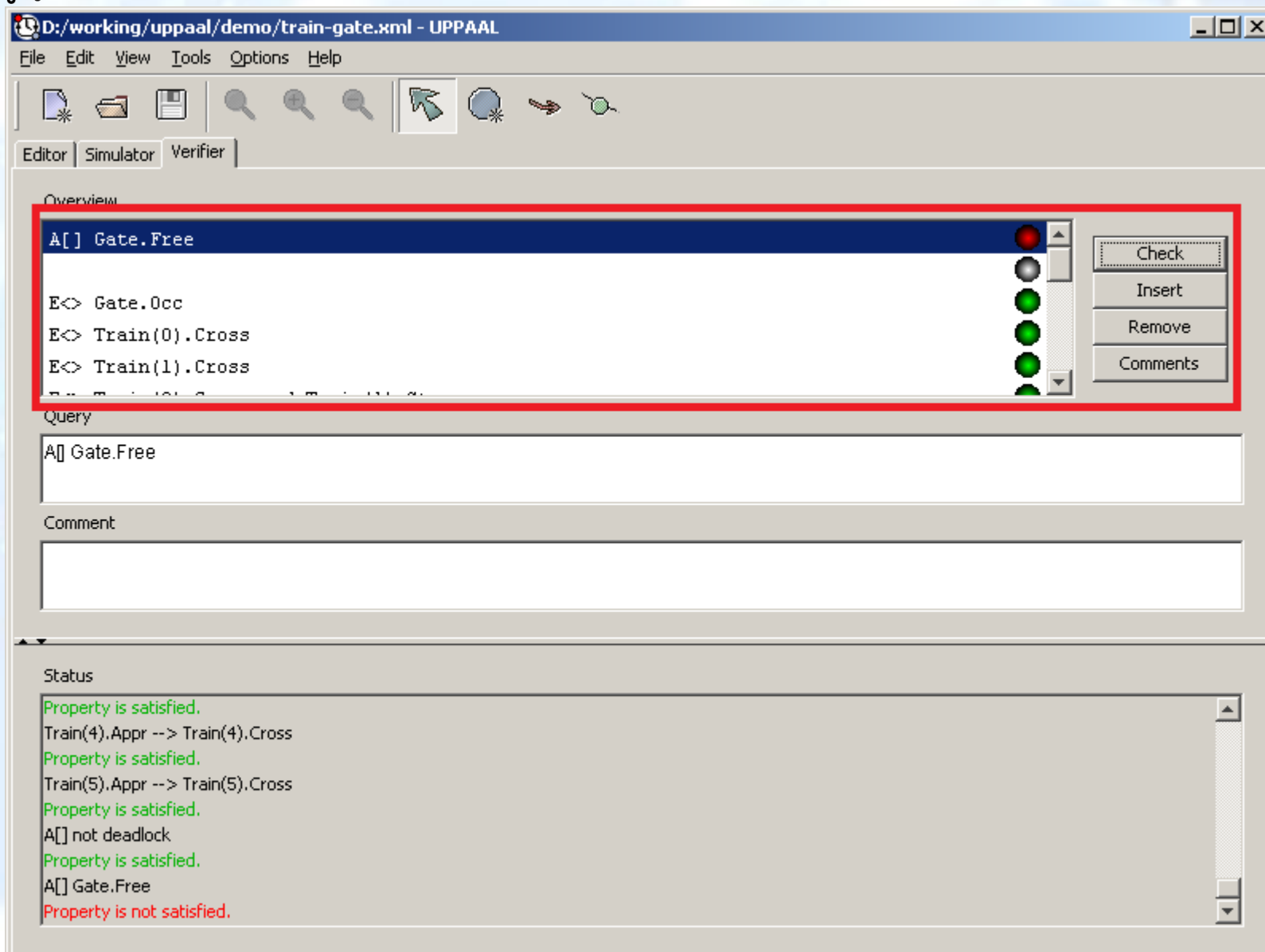
Comment

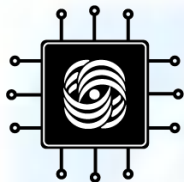
Status

Property is satisfied.  
Train(4).Appr --> Train(4).Cross  
Property is satisfied.  
Train(5).Appr --> Train(5).Cross  
Property is satisfied.  
A[] not deadlock  
Property is satisfied.  
A[] Gate.Free  
Property is not satisfied.



# Окно списка запросов





# Примеры запросов

D:/working/uppaal/demo/train-gate.xml - UPPAAL

File Edit View Tools Options Help

Editor Simulator Verifier

Overview

```
E<> Train(0).Cross and Train(1).Stop
E<> Train(0).Cross and (forall (i : id_t) i != 0 imply Train(i).Stop)

A[] forall (i : id_t) forall (j : id_t) Train(i).Cross && Train(j).Cross imply i == j
A[] Gate.list[N] == 0
```

Check  
Insert  
Remove  
Comments

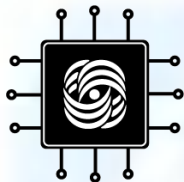
Query

A[] Gate.Free

Comment

Status

```
Property is satisfied.
Train(4).Appr --> Train(4).Cross
Property is satisfied.
Train(5).Appr --> Train(5).Cross
Property is satisfied.
A[] not deadlock
Property is satisfied.
A[] Gate.Free
Property is not satisfied.
```



# Примеры запросов

D:/working/uppaal/demo/train-gate.xml - UPPAAL

File Edit View Tools Options Help

Editor Simulator Verifier

Overview

|                                  |                                     |
|----------------------------------|-------------------------------------|
| Train(0).Appr --> Train(0).Cross | <input checked="" type="checkbox"/> |
| Train(1).Appr --> Train(1).Cross | <input checked="" type="checkbox"/> |
| Train(2).Appr --> Train(2).Cross | <input checked="" type="checkbox"/> |
| Train(3).Appr --> Train(3).Cross | <input checked="" type="checkbox"/> |
| Train(4).Appr --> Train(4).Cross | <input checked="" type="checkbox"/> |
| Train(5).Appr --> Train(5).Cross | <input checked="" type="checkbox"/> |

Check  
Insert  
Remove  
Comments

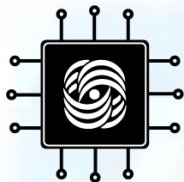
Query

A[] Gate.Free

Comment

Status

Property is satisfied.  
Train(4).Appr --> Train(4).Cross  
Property is satisfied.  
Train(5).Appr --> Train(5).Cross  
Property is satisfied.  
A[] not deadlock  
Property is satisfied.  
A[] Gate.Free  
Property is not satisfied.



# Примеры запросов

D:/working/uppaal/demo/train-gate.xml - UPPAAL

File Edit View Tools Options Help

Editor Simulator Verifier

Overview

|                                  |   |
|----------------------------------|---|
| Train(2).Appr --> Train(2).Cross | ● |
| Train(3).Appr --> Train(3).Cross | ● |
| Train(4).Appr --> Train(4).Cross | ● |
| Train(5).Appr --> Train(5).Cross | ● |
| A[] not deadlock                 | ● |

Check  
Insert  
Remove  
Comments

Query

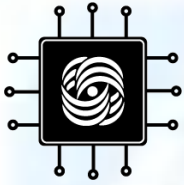
A[] not deadlock

Comment

The system is deadlock-free.

Status

Property is satisfied.  
Train(4).Appr --> Train(4).Cross  
Property is satisfied.  
Train(5).Appr --> Train(5).Cross  
Property is satisfied.  
A[] not deadlock  
Property is satisfied.  
A[] Gate.Free  
Property is not satisfied.



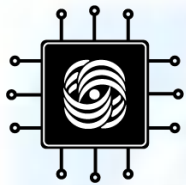
# Язык запросов

- Допустимая формула языка запросов

$\Phi ::= A[] \varphi \mid A\langle \rangle \varphi \mid E[] \varphi \mid E\langle \rangle \varphi$   
 $\mid \varphi \text{ --> } \varphi$

$\varphi ::= \text{Exp} \mid A.l \mid \text{deadlock}$   
 $\mid \varphi \text{ or } \varphi \mid \varphi \text{ and } \varphi \mid \text{not } \varphi \mid (\varphi)$

- `deadlock` можно использовать только с `A[]`, `E<>`
- Семантика определяется на основе семантики формул логики CTL
  - $A.l \Leftrightarrow$  автомат  $A$  находится в состоянии  $l$
  - $\text{deadlock} \Leftrightarrow$  вычисление после достижения текущей конфигурации невозможно продолжить
  - $\varphi_1 \text{ --> } \varphi_2 \Leftrightarrow \text{AG}(\neg\varphi_1 \vee \text{AF}(\varphi_2))$
  - Выполнимость выражений определяется естественным образом



**Спасибо за внимание**