Formal methods for software and hardware verification

LECTURES:

Vladimir Anatolyevich Zakharov,
Vladislav Vasilyevich Podymov

http://mk.cs.msu.ru/

Lecture 4.

Properties of program computations

Classification of computational properties

Fairness constraints

Computational Tree Logic CTL*

Temporal Logics CTL and LTL

Properties of program computations

Reactive systems are such information processing systems (circuits, protocols, embedded systems, controllers, etc.) that operate in interaction with the environment by receiving requests (control signals, stimulos, etc) and outputting responses (actions, instructions, etc.)

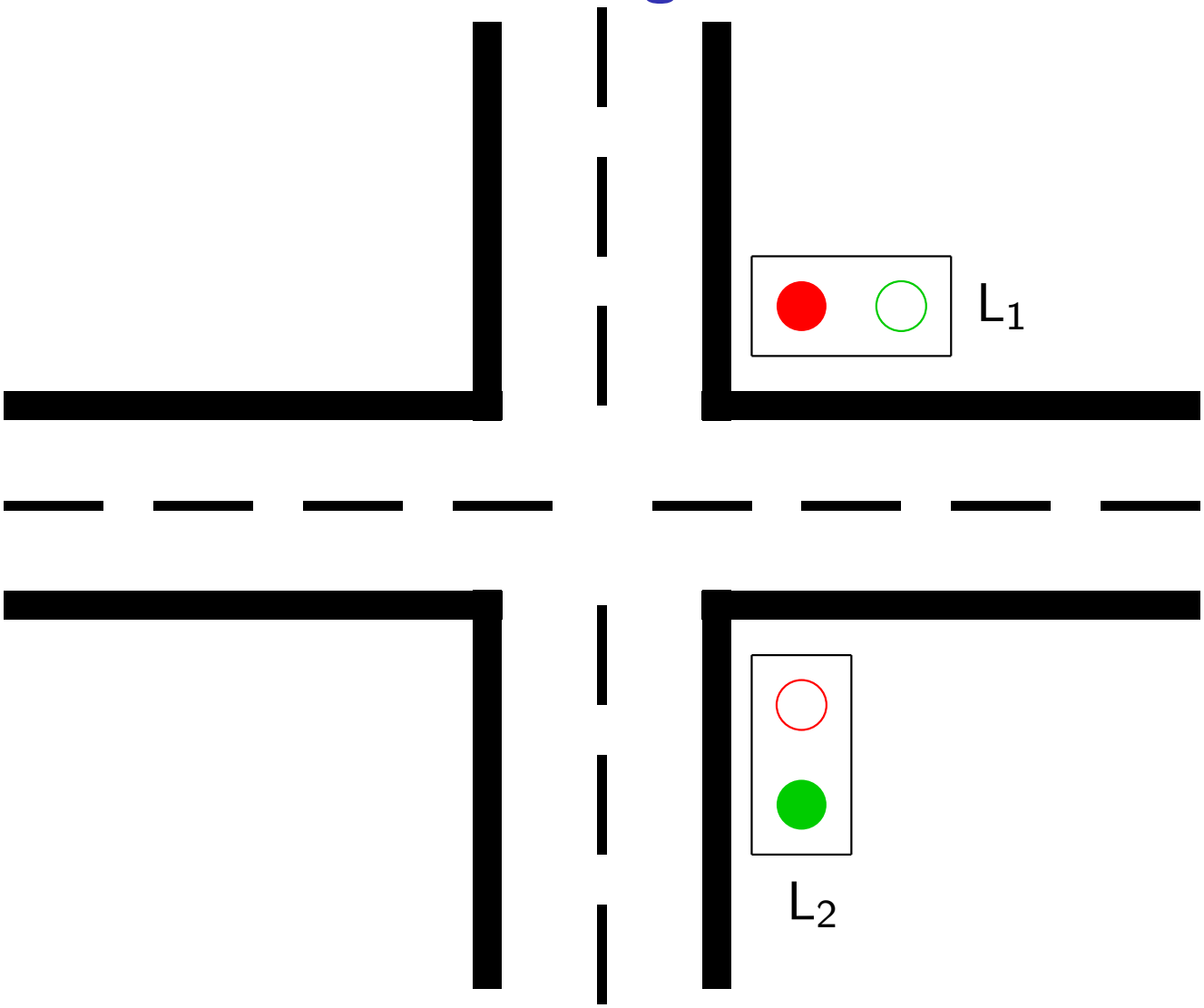
A behavior of a reactive system displays itself in the events that occur in the course of its computation.

Many reactive systems never terminate.

Therefore, formal languages intended to specify behavior of reactive systems must be able to describe infinite sequences of events.

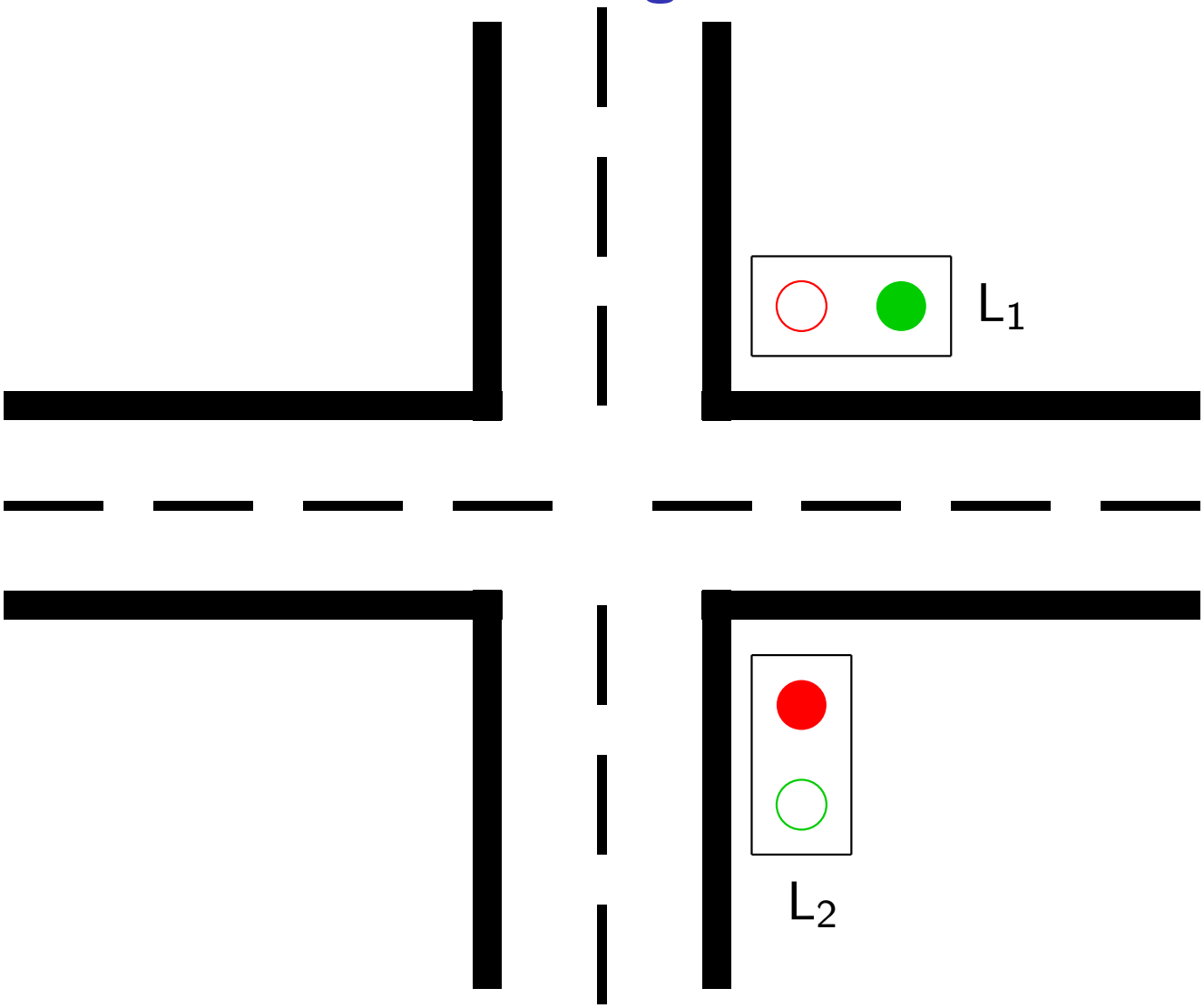
Properties of program computations

Crossroad with trafic lights



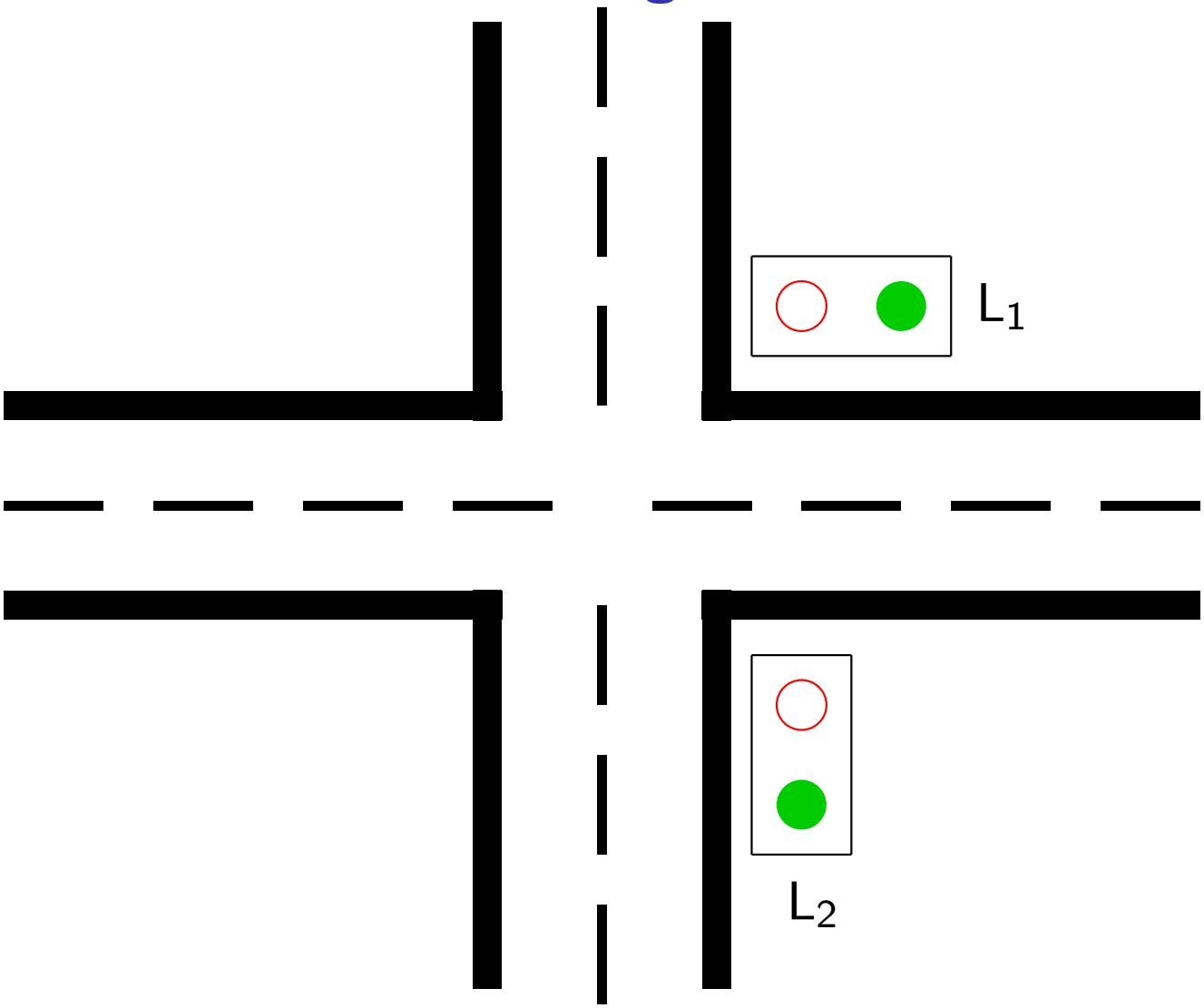
Properties of program computations

Crossroad with trafic lights



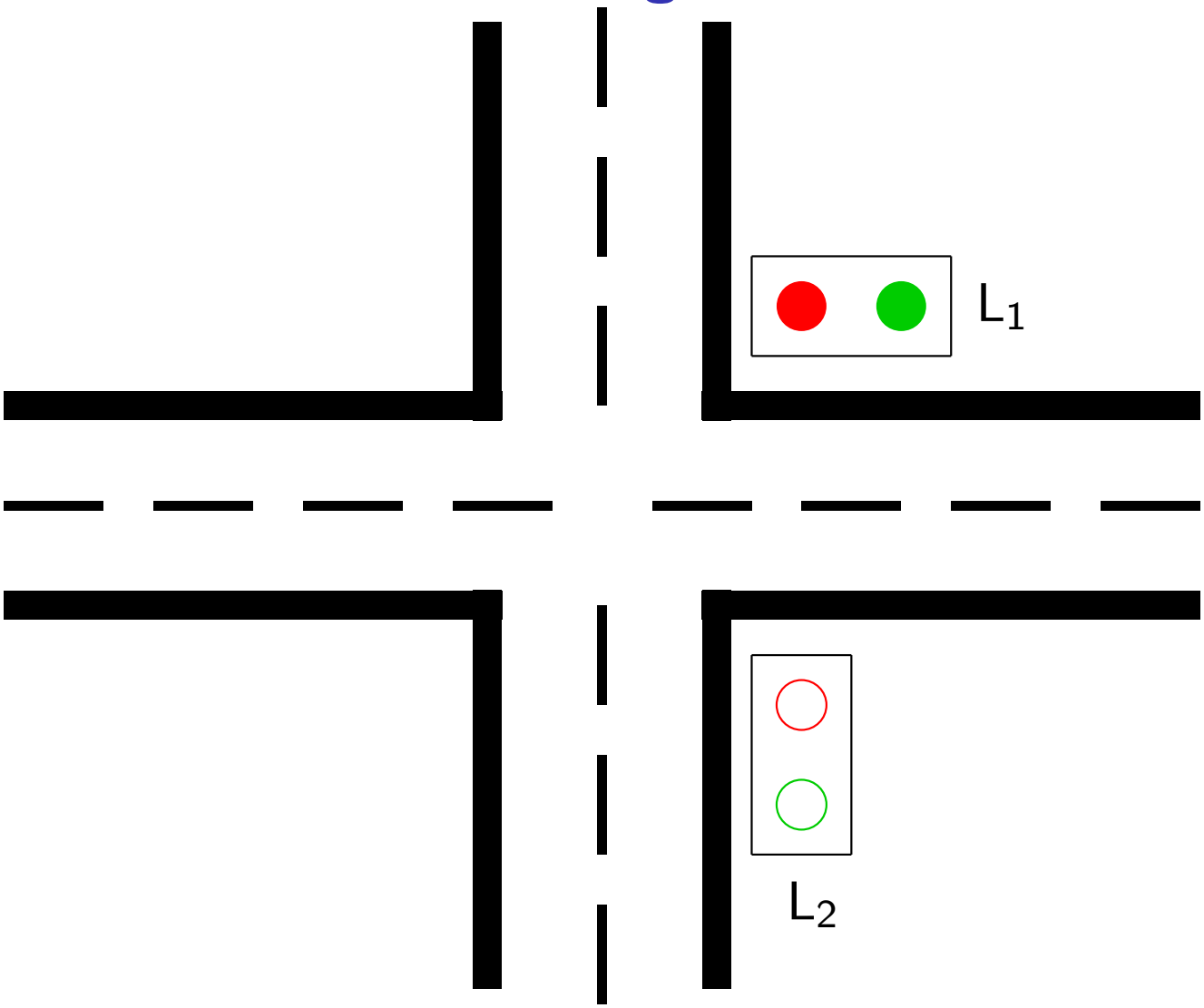
Properties of program computations

Crossroad with trafic lights



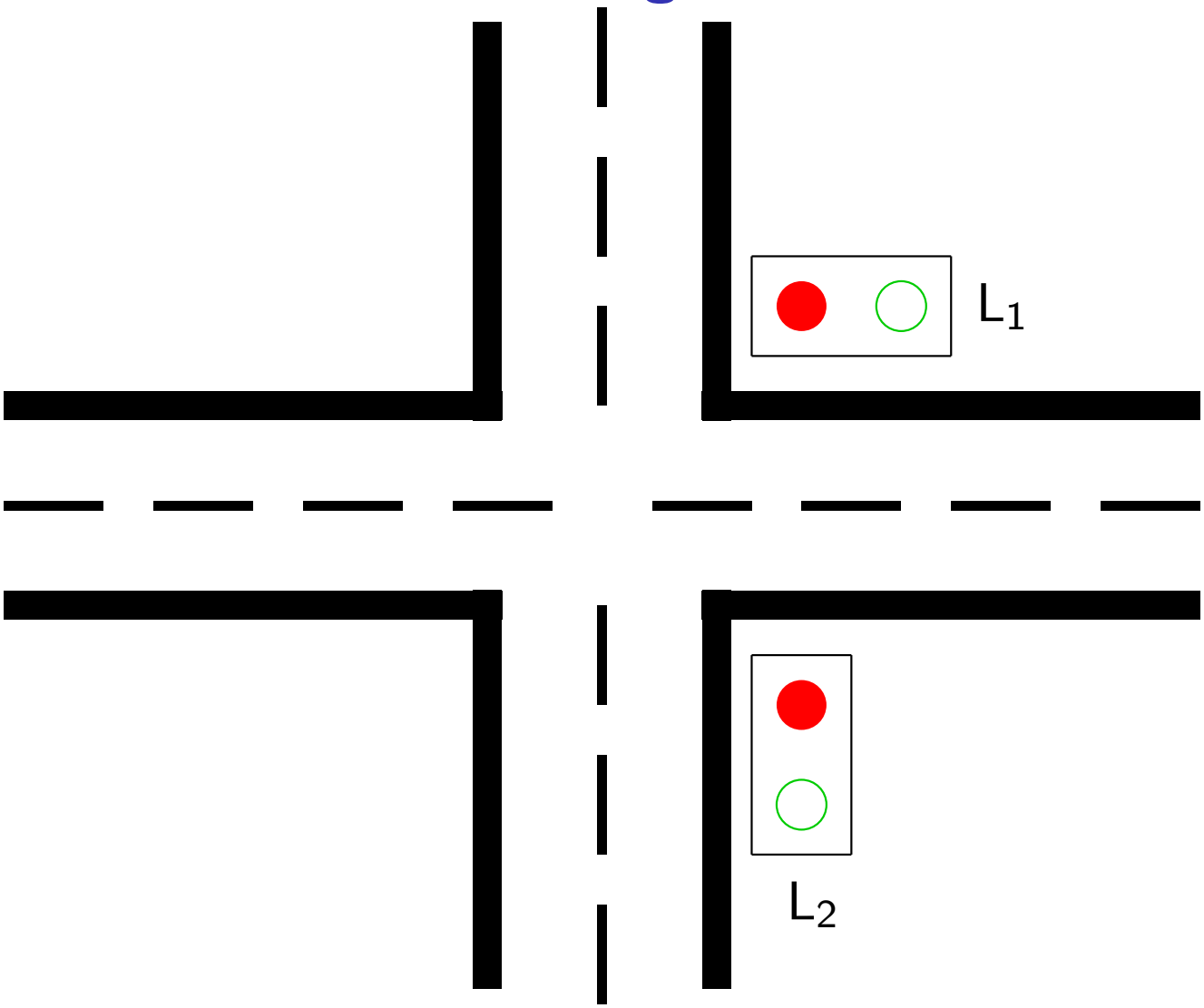
Properties of program computations

Crossroad with trafic lights



Properties of program computations

Crossroad with trafic lights



Properties of program computations

Every state of a traffic control system is characterized by a visible event — a set of active colours (colours of those lights that are turned on) of traffic lights.

Properties of program computations

Every state of a traffic control system is characterized by a visible event — a set of active colours (colours of those lights that are turned on) of traffic lights.

Every computation of a system is characterized by a trace which is a sequence of events.

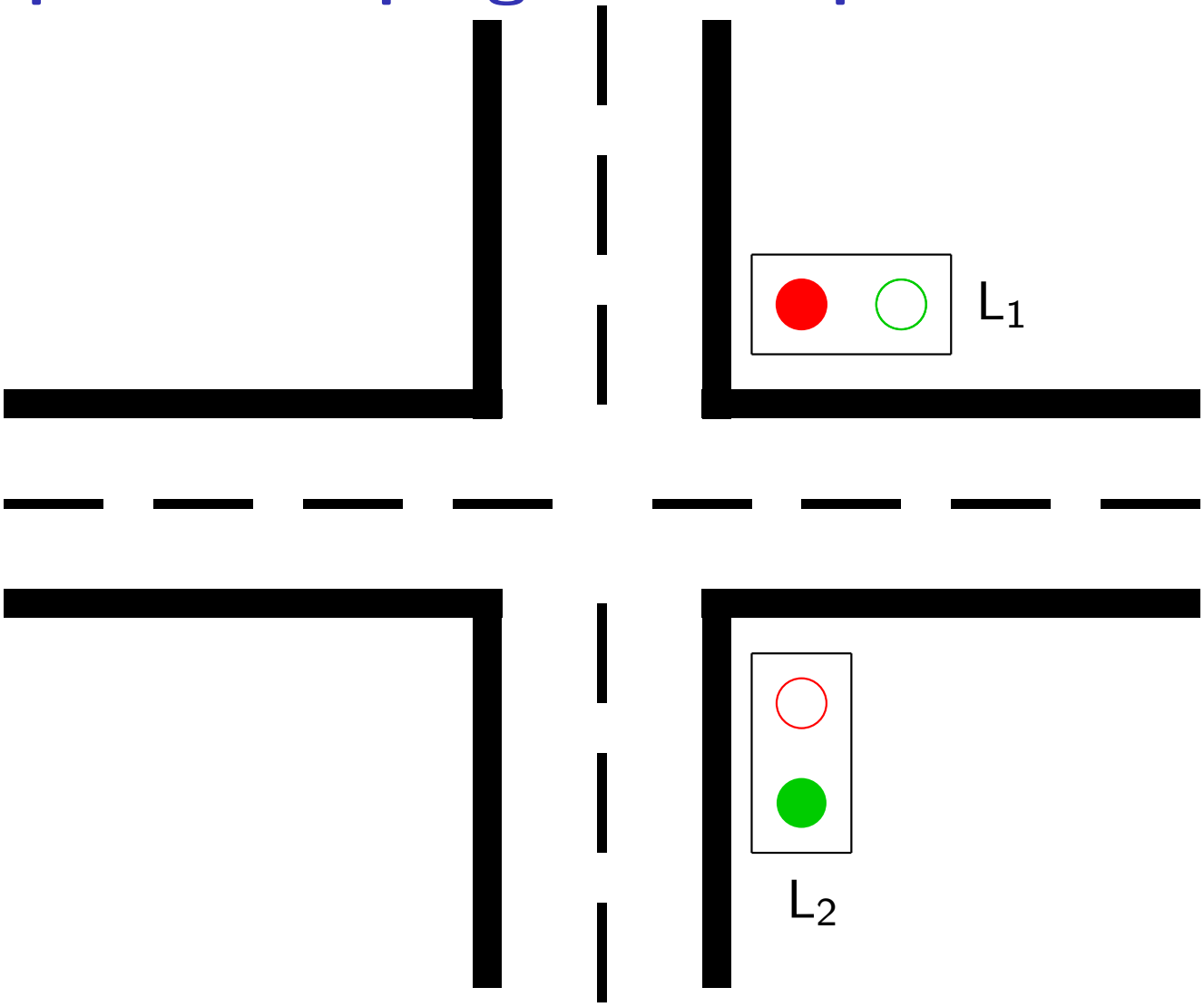
Properties of program computations

Every state of a traffic control system is characterized by a visible event — a set of active colours (colours of those lights that are turned on) of traffic lights.

Every computation of a system is characterized by a trace which is a sequence of events.

A behavior of a system is characterized by a property which is a set of traces.

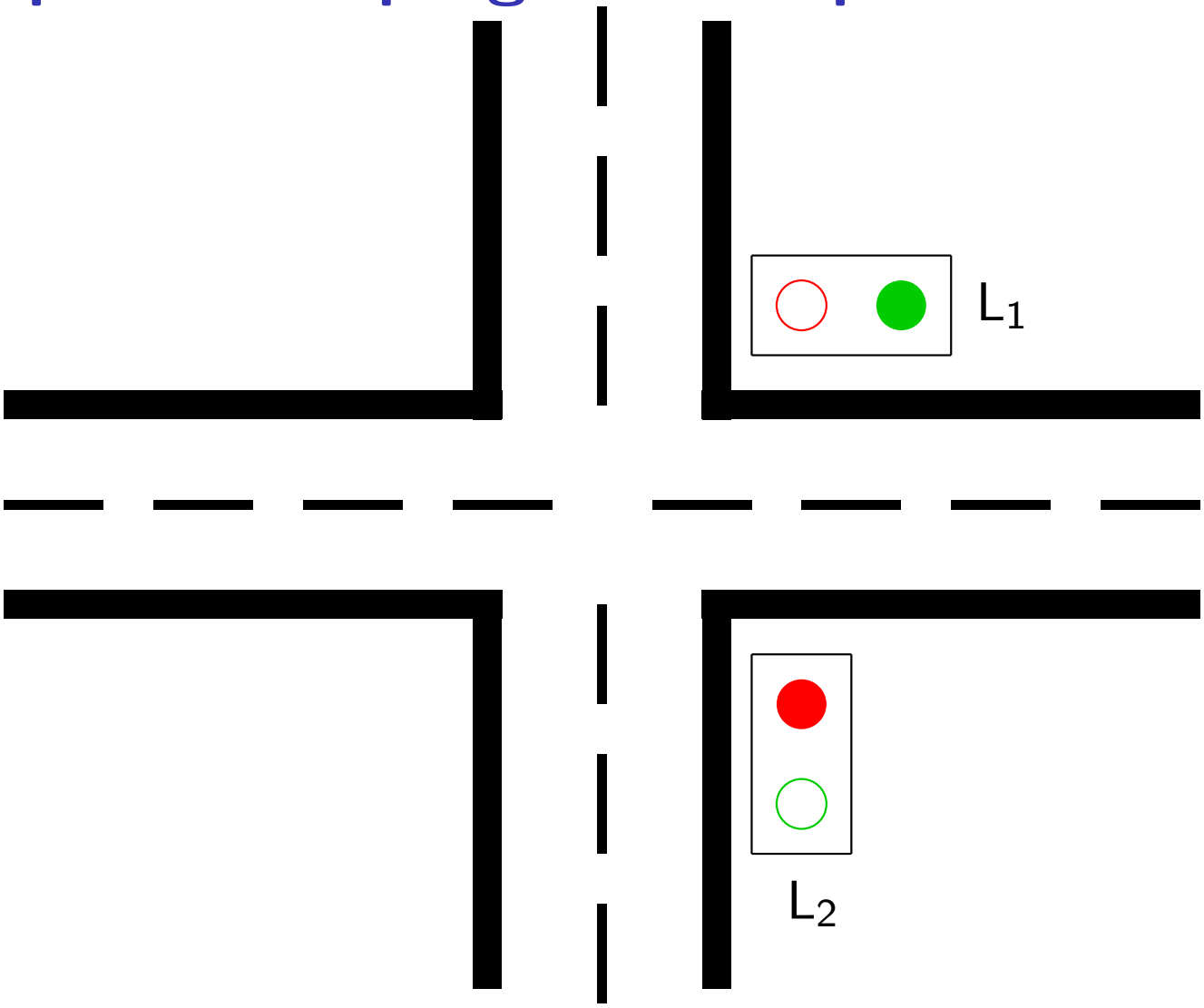
Properties of program computations



Trace:

$\{red_1, green_2\},$

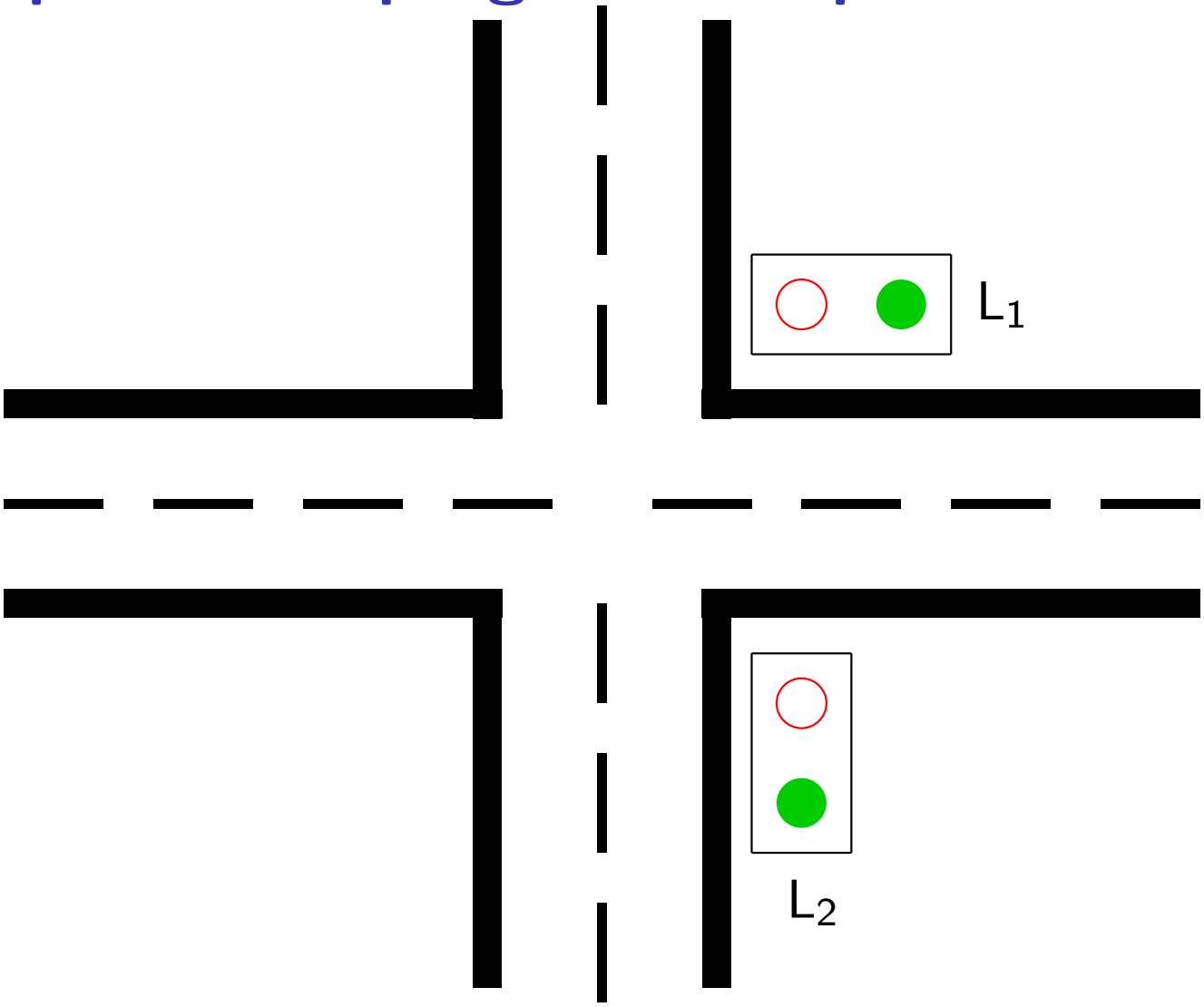
Properties of program computations



Trace:

$\{red_1, green_2\}, \{green_1, red_2\},$

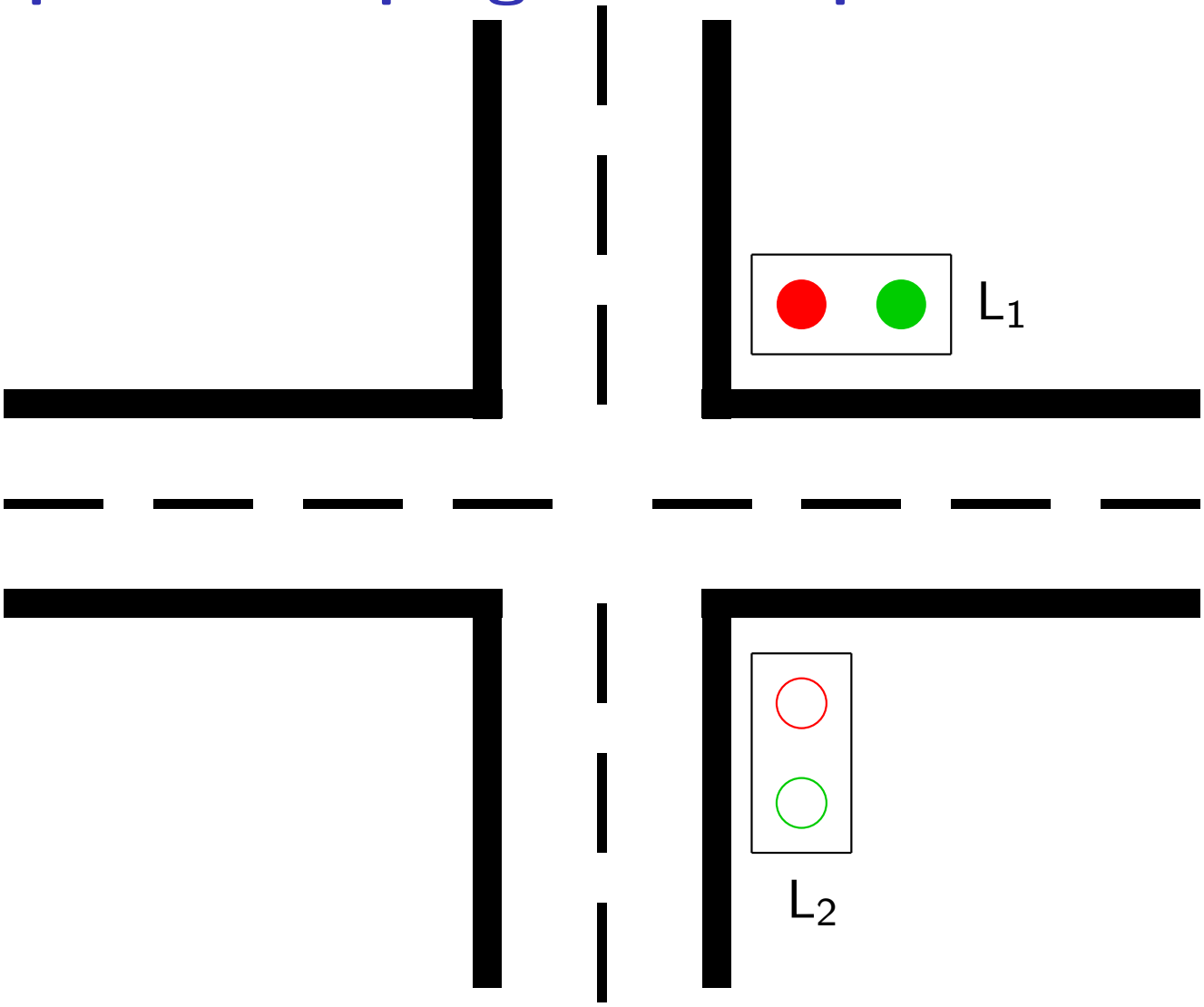
Properties of program computations



Trace:

$$\{red_1, green_2\}, \{green_1, red_2\}, \{green_1, green_2\}$$

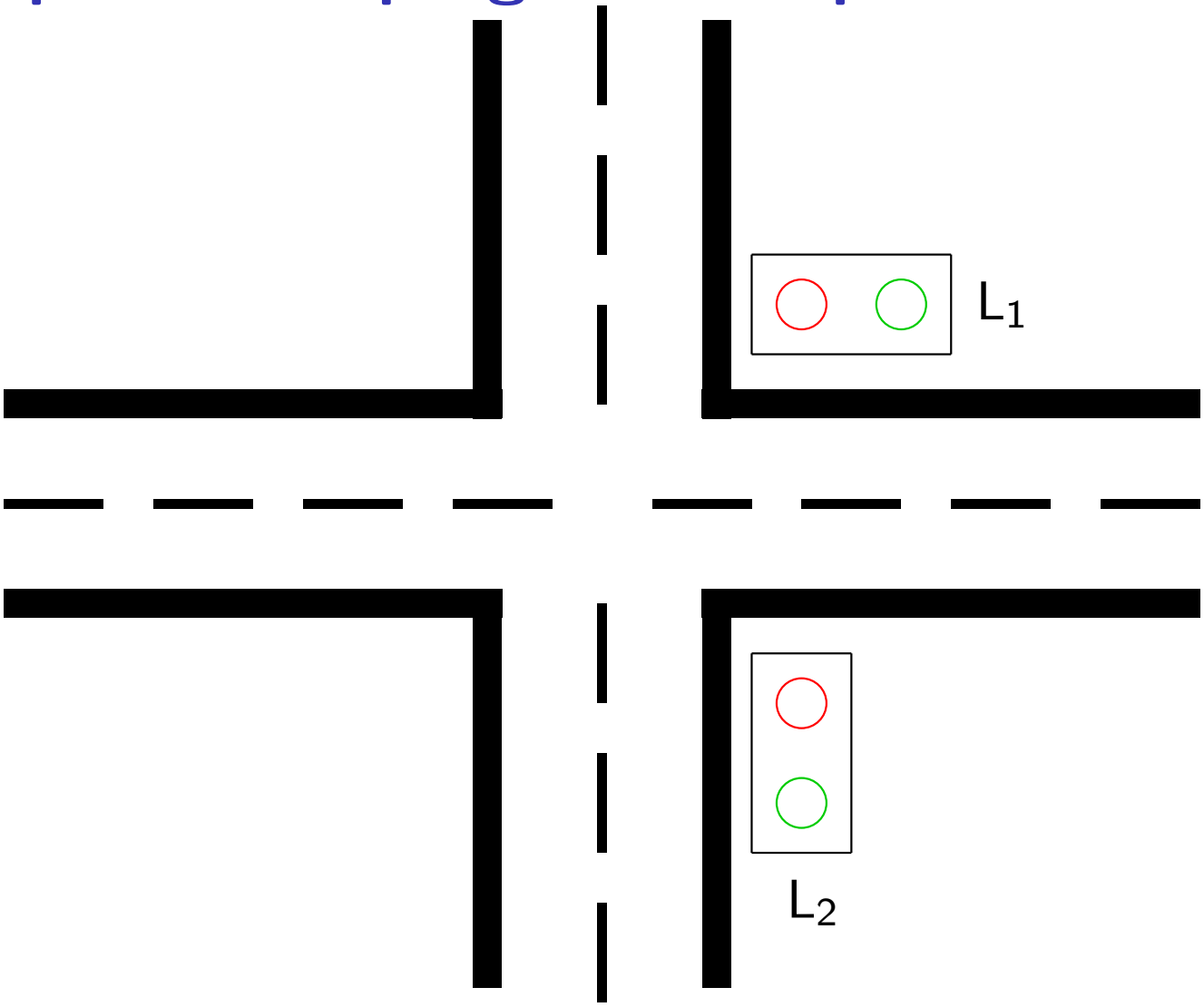
Properties of program computations



Trace:

$\{red_1, green_2\}, \{green_1, red_2\}, \{green_1, green_2\}\{red_1, green_1\},$

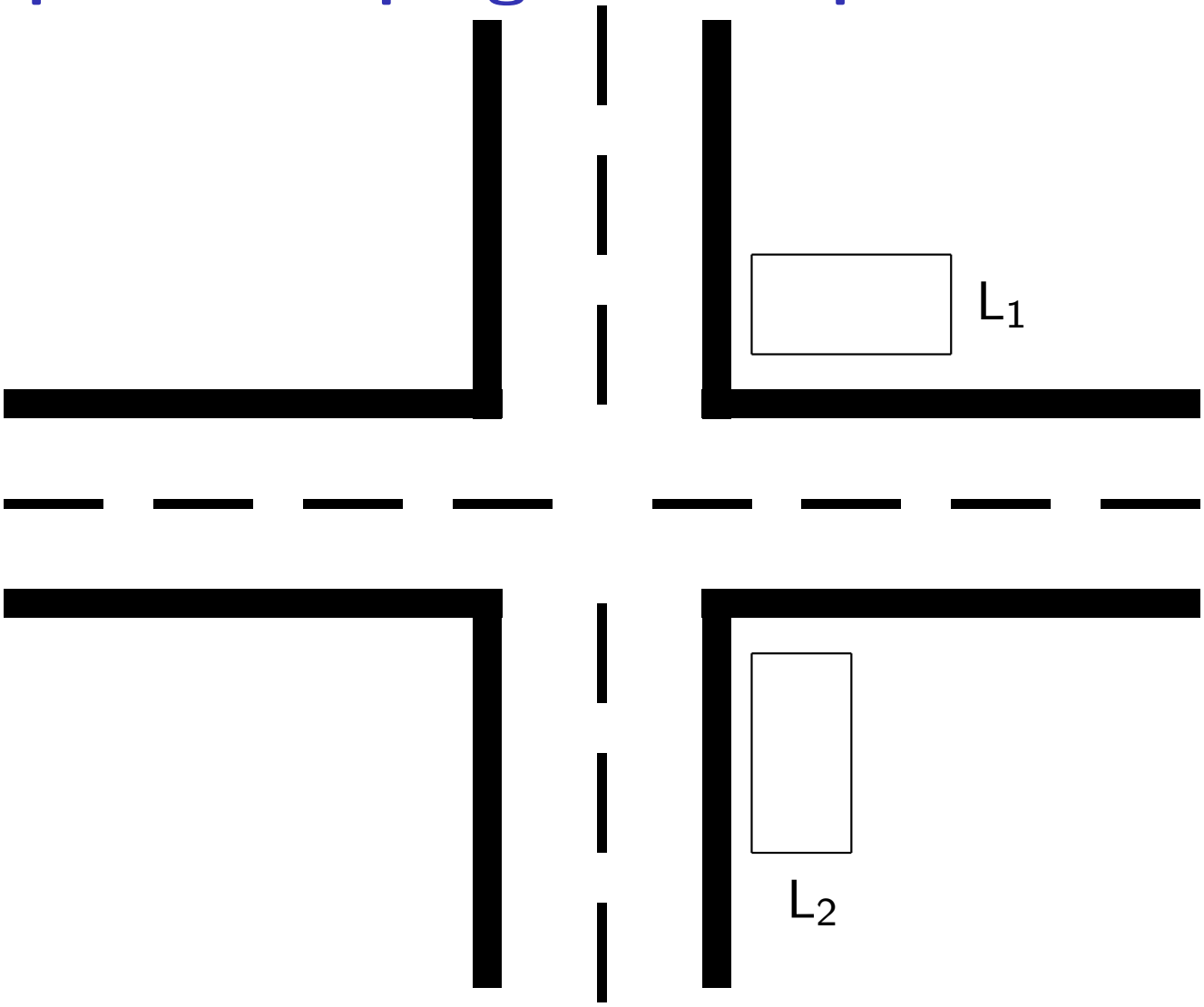
Properties of program computations



Trace:

$\{red_1, green_2\}, \{green_1, red_2\}, \{green_1, green_2\}\{red_1, green_1\}, ???$

Properties of program computations



Trace:

$\{red_1, green_2\}, \{green_1, red_2\}, \{green_1, green_2\}\{red_1, green_1\}, \emptyset,$

Properties of program computations

Properties of the traffic control system:

Properties of program computations

Properties of the traffic control system:

- Every traffic light always has exactly one active colour;

Properties of program computations

Properties of the traffic control system:

- Every traffic light always has exactly one active colour;
- Every traffic light eventually turns green;

Properties of program computations

Properties of the traffic control system:

- Every traffic light always has exactly one active colour;
- Every traffic light eventually turns green;
- Both traffic lights never turn green at the same time;

Properties of program computations

Properties of the traffic control system:

- Every traffic light always has exactly one active colour;
- Every traffic light eventually turns green;
- Both traffic lights never turn green at the same time;
- Every traffic lights changes colors infinitely often.

Properties of program computations

More formally, these notions are defined as follows.

Let AP be a set of atomic propositions.

Then

- an event is any set of atomic propositions, $E,\ E \subseteq AP$;
- a trace is any infinite sequence of events, $\alpha,\ \alpha \in (2^{AP})^{\omega}$,

$$\alpha = E_0, E_1, E_2, \ldots ;$$

- a computational property is any set of traces, $P,\ P \subseteq (2^{AP})^{\omega}$.

Properties of program computations

Let $M = (S, S_0, R, L)$ be a Kripke model and let $\pi = s_0, s_1, s_2, \ldots$ be a path from an initial state s_0, $s_0 \in S_0$, in this model. Then a trace of the path π is the sequence

$$\alpha(\pi) = L(s_0), L(s_1), L(s_2), \ldots$$

of sets of those atomic propositions that are true in the states of this path.

Denote by $Tr(M)$ the set of all traces of a Kripke model M .

Properties of program computations

Let $M = (S, S_0, R, L)$ be a Kripke model and let $\pi = s_0, s_1, s_2, \ldots$ be a path from an initial state s_0, $s_0 \in S_0$, in this model. Then a trace of the path π is the sequence

$$\alpha(\pi) = L(s_0), L(s_1), L(s_2), \ldots$$

of sets of those atomic propositions that are true in the states of this path.

Denote by $Tr(M)$ the set of all traces of a Kripke model M .

A model M satisfies a property P (in symbols: $M \models P$) if $Tr(M) \subseteq P$ holds.

Properties of program computations

A Kripke model M_1 is called a refinement of a model M_2 if $Tr(M_1) \subseteq Tr(M_2)$ holds.

Proposition 1.

If a Kripke model satisfies some property then every its refinement satisfies the same property.

Proof.

Hometask .

Classification of computational properties

The most widely used properties of program computations are divided into the following classes:

- **safety properties** ,
- **liveness properties** ,
- **fairness constraints** .

Classification of computational properties

Safety property: «**nothing bad ever happens**».

A computational property P is a safety property if it meets the following requirement:

- every trace α, $\alpha \in (2^{AP})^{\omega} \setminus P$, has such a finite prefix β that

$$\beta\alpha' \notin P.$$

holds for every trace α' .

Classification of computational properties

Safety property: «**nothing bad ever happens**».

A computational property P is a safety property if it meets the following requirement:

- every trace α, $\alpha \in (2^{AP})^{\omega} \setminus P$, has such a finite prefix β that

$$\beta\alpha' \notin P.$$

holds for every trace α' .

Unsafe behavior means that «if error occurs it can not be fixed later».

Classification of computational properties

Examples of safety property:

- A traffic light turns red only after it turns yellow.
- A printer is unavailable for other devices until it finishes printing.
- Two processes can not access the same memory page at the same time.
- Sliding window protocol never loses the first transmitted data packet.

Classification of computational properties

Liveness property: «something good will eventually happens».

A computational property P is a liveness property if it meets the following requirement:

- for every finite sequence $\beta,\ \beta \in (2^{AP})^*$, the inclusion

$$\beta\alpha \in P.$$

holds for some trace $\alpha,\ \alpha \in (2^{AP})^\omega$.

Classification of computational properties

Liveness property: «something good will eventually happens».

A computational property P is a liveness property if it meets the following requirement:

- for every finite sequence $\beta,\ \beta \in (2^{AP})^*$, the inclusion

$$\beta\alpha \in P.$$

holds for some trace $\alpha,\ \alpha \in (2^{AP})^\omega$.

Liveness means that «the goal is achievable no matter what happened earlier».

Classification of computational properties

Examples of liveness properties:

- A traffic light will eventually turn green.
- After printing is complete a printer clears the content of its buffer.
- A process can access a certain memory page infinitely often.
- Sliding window protocol always delivers the first data packet to its destination.

Classification of computational properties

Proposition 2.

If P is both a liveness property and a safety property then $P = (2^{AP})^{\omega}$.

Proof.

Hometask .

Classification of computational properties

Proposition 2.

If P is both a liveness property and a safety property then $P = (2^{AP})^{\omega}$.

Proof.

Hometask .

Proposition 3.

For every property P there exist such a liveness property P_{live} and such a safety property P_{safe} that

$$P = P_{live} \cap P_{safe}$$

Доказательство.

Hometask [Hard problem! Will be highly appreciated!] .

Classification of computational properties

What is the type of a property:

«After opening a file a process can read from it infinitely often»?

Classification of computational properties

What is the type of a property:

«After opening a file a process can read from it infinitely often»?

It is neither safety, nor liveness property.

Fairness constraints

When we deal with an information processing system composed of several parallel processes then we use the (interleaving assumption) to model in sequential way a behavior of such a system:

computation of parallel runs of several processes

$$\begin{array}{lcl} run_1 & = & a_1, a_2, a_3, \ldots \\ \| & & \\ run_2 & = & b_1, b_2, b_3, \ldots \end{array}$$

is represented by the set of **interleaved** sequences $comp_1$ and $comp_2$ — all possible shuffles that preserve the order of actions from the same run.

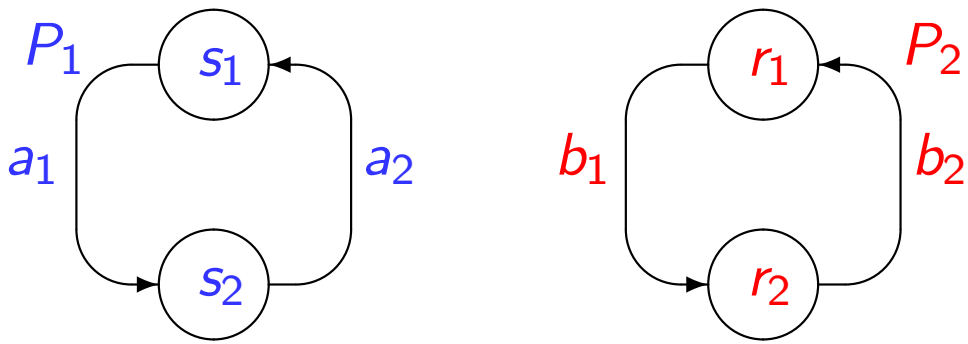
An example of an interleaving: $a_1, b_1, b_2, a_2, b_3, a_3, a_4, \ldots$.

Another example: $b_1, a_1, a_2, b_2, a_3, a_4, b_3, \ldots$

Fairness constraints

However, the interleaving assumption does not fully capture the actual behavior of parallel compositions.

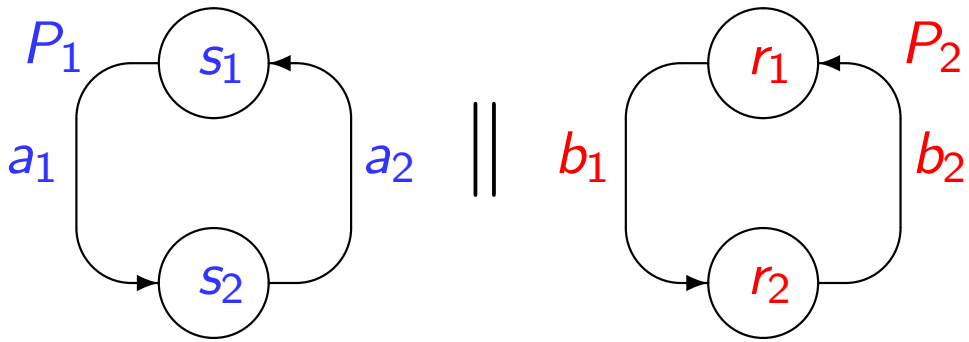
Consider a pair of Kripke models for processes P_1 and P_2.



Fairness constraints

However, the interleaving assumption does not fully capture the actual behavior of parallel compositions.

Consider a pair of Kripke models for processes P_1 and P_2.



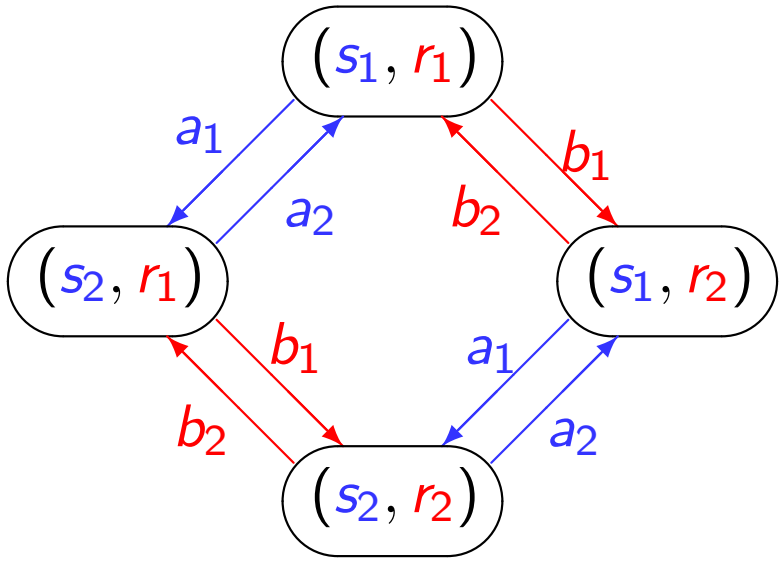
Parallel asynchronous composition of these processes does not have such a run as

$$comp(P_1 \parallel P_2) = a_1, a_2, a_1, a_2, \ldots \textbf{etc.}$$

since it can not be represented as an interleaving of any two runs of these processes.

Fairness constraints

But if we build a Kripke model for the parallel asynchronous composition of processes $P_1 \parallel P_2$,



then could find in it a path

$$\pi = a_1, a_2, a_1, a_2, \ldots \textbf{etc.}$$

Hence, we need some additional means to avoid these «unrealistic», «unjust» paths in Kripke models.

Fairness constraints

Fairness constraints are additional requirements for distinguishing interleaving paths among all paths in Kripke models.

Each fairness constraint refers to some action (transition) in the model and requires this action to be performed on each computation, depending on how often the conditions for its execution are met.

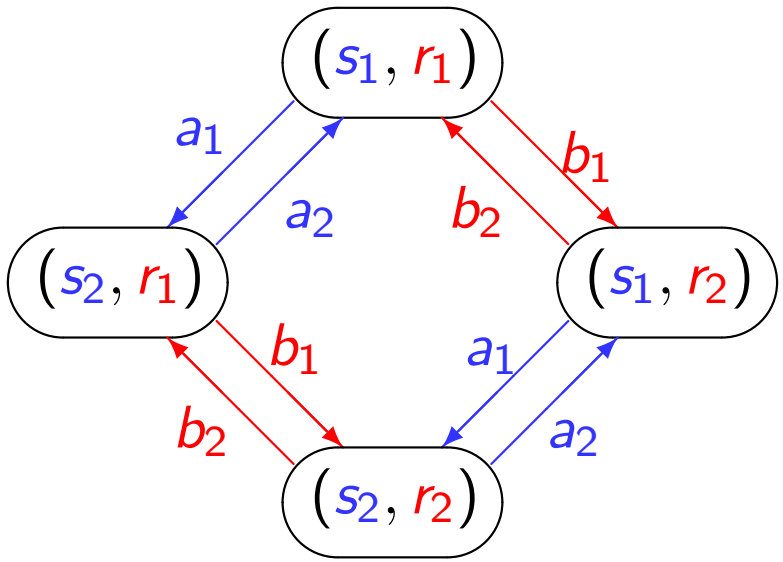
Fairness constraints

Two types of fairness constraints are distinguished.

- Weak fairness : if a path **almost always** passes via a state that enables an action *act* then the action *act* must be performed **infinitely often**.
- Strong fairness : if a path **infinitely often** passes via a state that enables an action *act* then the action *act* must be performed **infinitely often**.

Fairness constraints

Thus, a Kripke model for parallel asynchronous composition $P_1 \parallel P_2$



has a path

$$\pi = a_1, a_2, a_1, a_2, \ldots \textbf{и.т.д.}$$

which does not satisfies weak fairness constraint on the action b_1 , and, therefore, it does not model any actual run.

Fairness constraints

A fair Kripke model is a Labeled Transition System $M = (S, S_0, R, L, Act)$ supplied with Fairness Constraints $Fair = (WeakFair, StrongFair)$, where

- Act is a set of actions,
- R is a transition relation, $R \subseteq S \times Act \times S$
- WeakFair is a subset of actions subjected to the weak fairness constraints, $WeakFair \subseteq Act$;
- StrongFair is a subset of actions subjected to the strong fairness constraints, $StrongFair \subseteq Act$.

Fairness constraints

A fair Kripke model is a Labeled Transition System $M = (S, S_0, R, L, Act)$ supplied with Fairness Constraints $Fair = (WeakFair, StrongFair)$, where

- Act is a set of actions,
- R is a transition relation, $R \subseteq S \times Act \times S$
- WeakFair is a subset of actions subjected to the weak fairness constraints, $WeakFair \subseteq Act$;
- StrongFair is a subset of actions subjected to the strong fairness constraints, $StrongFair \subseteq Act$.

Denote by $Tr(M, Fair)$ the set of all those initial paths in a Kripke model M that satisfy fairness constraints $Fair$.

Then Model Checking Problem is that of checking the inclusion

$$Tr(M, Fair) \subseteq P,$$

which is denoted as $M, Fair \models P$.

Temporal Logics

Temporal Logics are intended for the specification of computational properties of reactive systems, i.e. sets of traces in the Labeled Transition Systems (Kripke structures).

Temporal Logics do not explicitly refer to time; instead they only allow one to speak and reason about a relative order in which the events occur. Temporal formulas make it possible, for example, to assert without mentioning any specific moments of time that certain critical states will be eventually passed or that some erroneous states will be never achieved while some guard conditions are met, and so on

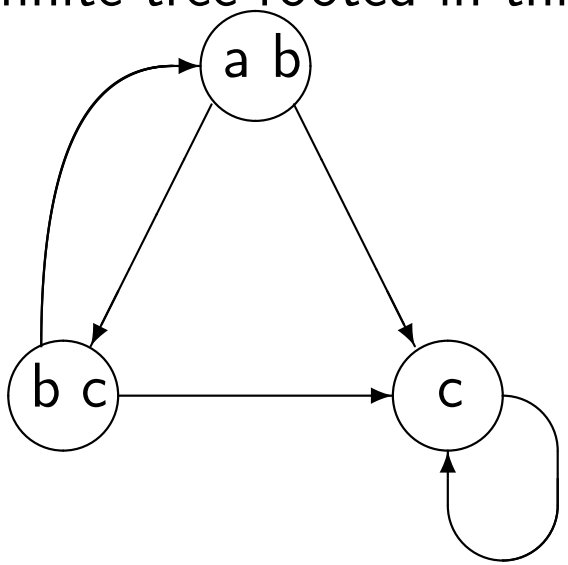
To this end, special **temporal operators** are used.

Temporal logics differ in the set of used temporal operators and the semantics of these operators.

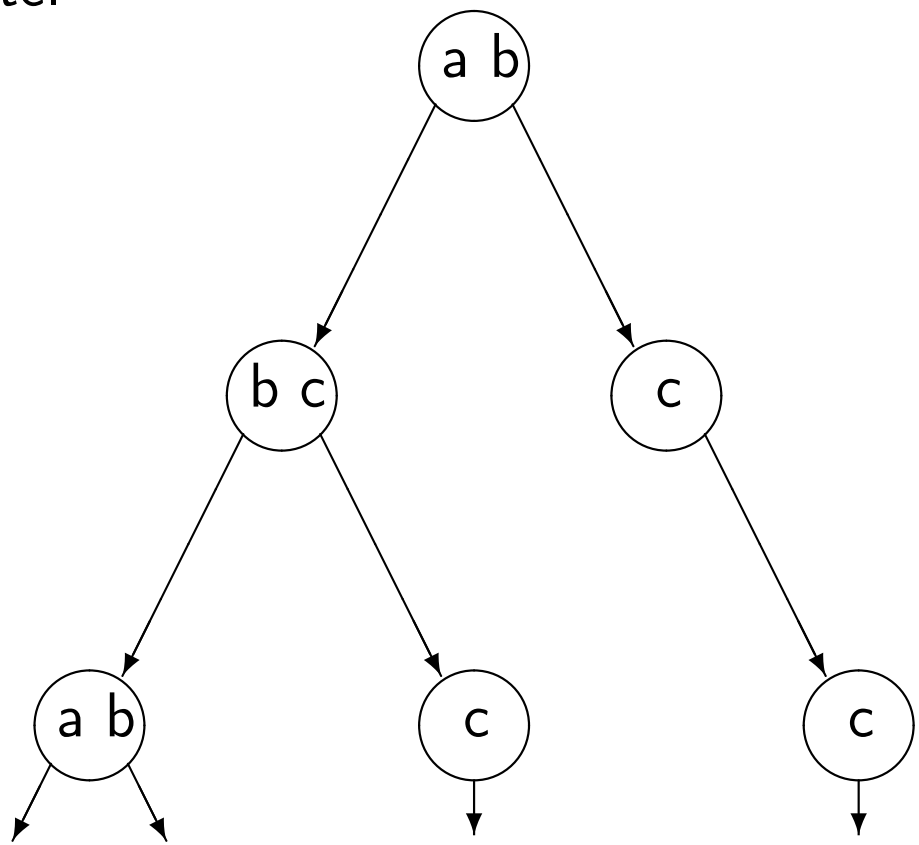
We will focus on the very expressive temporal logic called CTL*.

Computational Tree Logic CTL*

Formulas of CTL* describe the properties of computational trees. Such a tree is formed by choosing some state in a Kripke structure as the initial state and by unfolding the transitions system into an infinite tree rooted in this state.



Transition system, or, Kripke structure



Unfolding of the transition system in the infinite tree

Computational Tree Logic CTL*

Formulas of CTL* are built of path quantifiers and temporal operators.

Path quantifiers **A** («for every computational path») and **E** («for some computational path») are used to specify a branching structure in the computational tree. These quantifiers are used when one asserts that all computations (paths) or some computations (paths) that begin in certain states have the prescribed property.

Temporal operators are used to describe the properties of a path in a tree.

Computational Tree Logic CTL*

There are three unary temporal operators.

- Nexttime operator **X** («at the next moment of time»): **X** P asserts that the property P has to hold at the next state of the computational path.
- Eventuality operator **F** («sometimes in future»): **F** P asserts that the property P has to hold at some state of the subsequent computational path.
- Globality operator **G** («always»): **G** P asserts that the property P holds at every state of the subsequent computational path.

Computational Tree Logic CTL*

There are also two binary temporal operators.

- Conditional awaiting **U** (Until): P_1 **U** P_2 asserts that the property P_1 has to hold at least until P_2 becomes true, which must hold at the current or a future moment of time.
- Emancipation **R** (Release): P_1 **R** P_2 asserts that the property P_1 has to be true until and including the moment of time where P_2 first becomes true; if P_2 never becomes true, P_1 must remain true forever.

Computational Tree Logic CTL*

There are formulas of two types in CTL*: state formulas (which are interpreted at certain states of a model) and path formulas (which are interpreted on a certain computational path of a model).

Computational Tree Logic CTL*

There are formulas of two types in CTL*: state formulas (which are interpreted at certain states of a model) and path formulas (which are interpreted on a certain computational path of a model).
State formulas are defined as follows:

- If $p \in AP$, then p is a state formula.
- If f_1 and f_2 are state formulas, then $\neg f_1$, $f_1 \wedge f_2$ and $f_1 \vee f_2$ are state formulas.
- If f is a path formula, then $\mathbf{E}\, f$ and $\mathbf{A}\, f$ are state formulas.

Computational Tree Logic CTL*

There are formulas of two types in CTL*: state formulas (which are interpreted at certain states of a model) and path formulas (which are interpreted on a certain computational path of a model).
State formulas are defined as follows:

- If $p \in AP$, then p is a state formula.
- If f_1 and f_2 are state formulas, then $\neg f_1$, $f_1 \wedge f_2$ and $f_1 \vee f_2$ are state formulas.
- If f is a path formula, then $\mathbf{E}\, f$ and $\mathbf{A}\, f$ are state formulas.

Path formulas are defined as follows:

- If f is a state formuls, then f is a path formula.
- If g_1 and g_2 are path formulas, then $\neg g_1$, $g_1 \wedge g_2$, $g_1 \vee g_2$, $\mathbf{X}\, g_1$, $\mathbf{F}\, g_1$, $\mathbf{G}\, g_1$, $g_1\, \mathbf{U}\, g_2$, and $g_1\, \mathbf{R}\, g_2$ are path formulas.

Computational Tree Logic CTL*

There are formulas of two types in CTL*: state formulas (which are interpreted at certain states of a model) and path formulas (which are interpreted on a certain computational path of a model).
State formulas are defined as follows:

- If $p \in AP$, then p is a state formula.
- If f_1 and f_2 are state formulas, then $\neg f_1$, $f_1 \wedge f_2$ and $f_1 \vee f_2$ are state formulas.
- If f is a path formula, then $\mathbf{E}\, f$ and $\mathbf{A}\, f$ are state formulas.

Path formulas are defined as follows:

- If f is a state formuls, then f is a path formula.
- If g_1 and g_2 are path formulas, then $\neg g_1$, $g_1 \wedge g_2$, $g_1 \vee g_2$, $\mathbf{X}\, g_1$, $\mathbf{F}\, g_1$, $\mathbf{G}\, g_1$, $g_1\, \mathbf{U}\, g_2$, and $g_1\, \mathbf{R}\, g_2$ are path formulas.

Formulas of CTL* are all state formulas built according to the above rules.

Computational Tree Logic CTL*

Formulas of CTL* are interpreted on Kripke structures $M = (S, S_0, R, L)$.

A path in M is any such infinite sequence of states $\pi = s_0, s_1, \ldots$, that $(s_i, s_{i+1}) \in R$ holds for every $i \geq 0$.
A path is just an infinite branch in the computational tree of a model M.

We will write π^i to denote a suffix of π which begins with the state s_i.

If f is a state formula, then by writing $M, s \models f$ we indicate that f is satisfied in a state s of a Kripke structure M. Analogously, if g is a path formula, then $M, \pi \models g$ means that g is satisfied on a path π in a Kripke model M.

Computational Tree Logic CTL*

Semantics for state formulas

1). $M, s \models p \Leftrightarrow p \in L(s)$;

2). $M, s \models \neg f_1 \Leftrightarrow M, s \not\models f_1$;

3). $M, s \models f_1 \vee f_2 \Leftrightarrow M, s \models f_1$ or $M, s \models f_2$;

4). $M, s \models f_1 \wedge f_2 \Leftrightarrow M, s \models f_1$ and $M, s \models f_2$;

5). $M, s \models \mathbf{E}\, f \Leftrightarrow$ there exists such a path π from a state s in a model M that $M, \pi \models f$;

6). $M, s \models \mathbf{A}\, f \Leftrightarrow M, \pi \models f$ for every path π from a state s in a model M.

Computational Tree Logic CTL*

Semantics for path formulas

7). $M, \pi \models g_1 \Leftrightarrow M, s \models g_1$ holds for the first state s on the path π in the model M;

8). $M, \pi \models \neg g_1 \Leftrightarrow M, \pi \not\models g_1$;

9). $M, \pi \models g_1 \vee g_2 \Leftrightarrow M, \pi \models g_1$ or $M, \pi \models g_2$;

10). $M, \pi \models g_1 \wedge g_2 \Leftrightarrow M, \pi \models g_1$ and $M, \pi \models g_2$;

11). $M, \pi \models \mathbf{X}\, g_1 \Leftrightarrow M, \pi^1 \models g_1$;

12). $M, \pi \models \mathbf{F}\, g_1 \Leftrightarrow$ there exists such $k \geqslant 0$ that $M, \pi^k \models g_1$;

13). $M, \pi \models \mathbf{G}\, g_1 \Leftrightarrow M, \pi^k \models g_1$ holds for every $k \geqslant 0$;

14). $M, \pi \models g_1\, \mathbf{U}\, g_2 \Leftrightarrow$ there exists such $k \geqslant 0$ that $M, \pi^k \models g_2$ and $M, \pi^j \models g_1$ holds for every $0 \leqslant j < k$;

15). $M, \pi \models g_1\, \mathbf{R}\, g_2 \Leftrightarrow$ for every $j \geqslant 0$, if $M, \pi^i \not\models g_1$ holds for every $i < j$, then $M, \pi^j \models g_2$.

Computational Tree Logic CTL*

It is easy to see that every assertion expressible by a formula of CTL* can be as well expressed by means of the set of Boolean connectives $\vee$, $\neg$, and temporal operators $\mathbf{X}$, $\mathbf{U}$, $\mathbf{E}$:

- $f \wedge g \equiv \neg(\neg f \vee \neg g)$,
- $f \,\mathbf{R}\, g \equiv \neg(\neg f \,\mathbf{U}\, \neg g)$,
- $\mathbf{F}\, f \equiv \mathit{True}\ \mathbf{U}\, f$,
- $\mathbf{G}\, f \equiv \neg\, \mathbf{F}\, \neg f$,
- $\mathbf{A}(f) \equiv \neg\, \mathbf{E}(\neg f)$.

CTL and LTL

Two subsets of CTL* are commonly used: one of them is called branching time logic, and the other — linear time logic.

The difference between these logics is in the way they relate to branching in the computation trees.

In the branching time logic every temporal operator immediately follows some path quantifier.

Formulas of linear time logic are used to specify properties of paths in computational trees.

CTL and LTL

Computational Tree Logic (CTL) is such a fragment of CTL* in which every temporal operator $\mathbf{X}$, $\mathbf{F}$, $\mathbf{G}$, $\mathbf{U}$ and $\mathbf{R}$ must immediately follow some path quantifier. More formally, CTL is a subset of CTL* in which path formulas are defined as follows.

- If f and g are state formulas, then $\mathbf{X}\, f$, $\mathbf{F}\, f$, $\mathbf{G}\, f$, $f\, \mathbf{U}\, g$, and $f\, \mathbf{R}\, g$ are path formulas.

Examples.

$\mathbf{A}\, \mathbf{G}\, p_1 \rightarrow (p_2\ \mathbf{E}\, \mathbf{U}\, (\mathbf{A}\, \mathbf{F}\, p_3))$,

$\mathbf{E}\, \mathbf{F}\, p \vee \mathbf{A}\, \mathbf{G}(p \wedge q)$.

CTL and LTL

Linear Temporal Logic (LTL) includes only such formulas $\mathbf{A}\, f$, where f is a quantifier-free path formula. More formally, the definition of LTL path formulas is as follows.

- If $p \in AP$, then f is a path formula.
- If f and g are path formulas, then $\neg f$, $f \wedge g$, $f \vee g$, $\mathbf{X}\, f$, $\mathbf{F}\, f$, $\mathbf{G}\, f$, $f\, \mathbf{U}\, g$, $f\, \mathbf{R}\, g$ are path formulas as well.

Example.

$\mathbf{A}(\mathbf{F}\, \mathbf{G}\, \mathit{enabled} \rightarrow \mathbf{G}\, \mathbf{F}\, \mathit{fired})$ — weak fairness constraint;

$\mathbf{A}\, \mathbf{G}(\mathit{passive} \rightarrow (\mathit{active}\, \mathbf{U}\, \mathbf{X}\, \mathit{passive}))$.

CTL and LTL

These three logics CTL*, CTL, and LTL have different expressive power.

Proposition 4.

There does not exist such a CTL-formula which is equivalent to LTL-formula $\mathbf{A}\,(\mathrm{FG}\,\mathit{stable})$.

This formula specifies a liveness requirement: in all possible computations the system will sooner or later stabilize and will continue to remain stable.

Proof.

Hometask [Hard problem! Will be highly appreciated!] .

CTL and LTL

Proposition 5.

There does not exist such a LTL-formula which is equivalent to CTL-formula **AG** (**EF** *restart*) .

This formula specifies the following property: in every computation the system is always capable to restart.

Proof.

Hometask [Hard problem! Will be highly appreciated!] .

CTL and LTL

Proposition 5.

There does not exist such a LTL-formula which is equivalent to CTL-formula $\mathbf{AG}\ (\mathbf{EF}\ \mathit{restart})$.

This formula specifies the following property: in every computation the system is always capable to restart.

Proof.

Hometask [Hard problem! Will be highly appreciated!] .

A disjunction of these two formulas $\mathbf{A}\ (\mathrm{FG}\, p) \vee \mathbf{AG}\ (\mathbf{EF}\, p)$ is an example of a computational property which can not be expressed neither in CTL, nor in LTL.

CTL

Now we confine ourselves with the consideration of CTL.

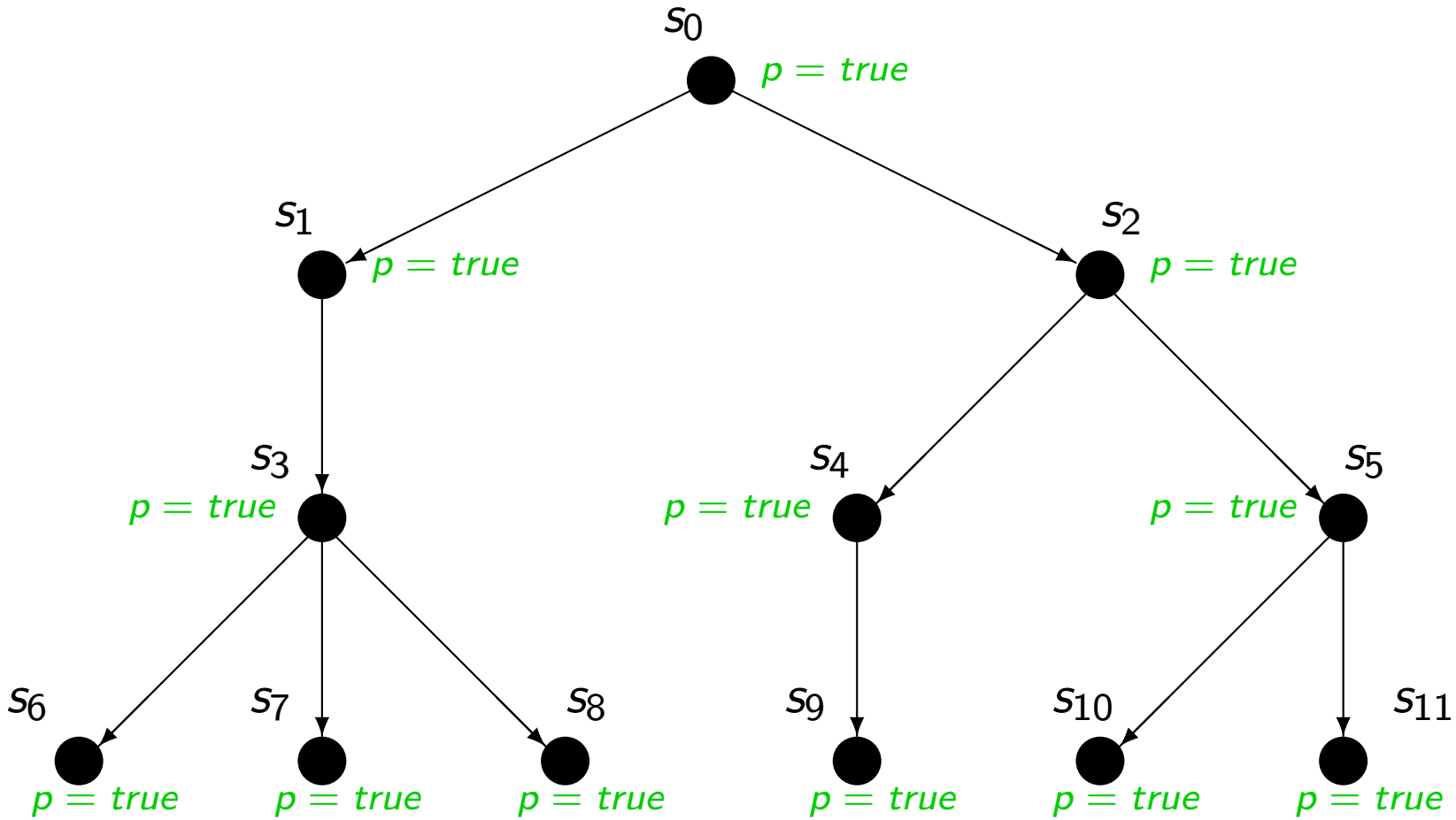
There are 10 basic operators in CTL:

- AX и EX,
- AF и EF,
- AG и EG,
- AU и EU,
- AR и ER.

CTL

Computational Tree Logic CTL

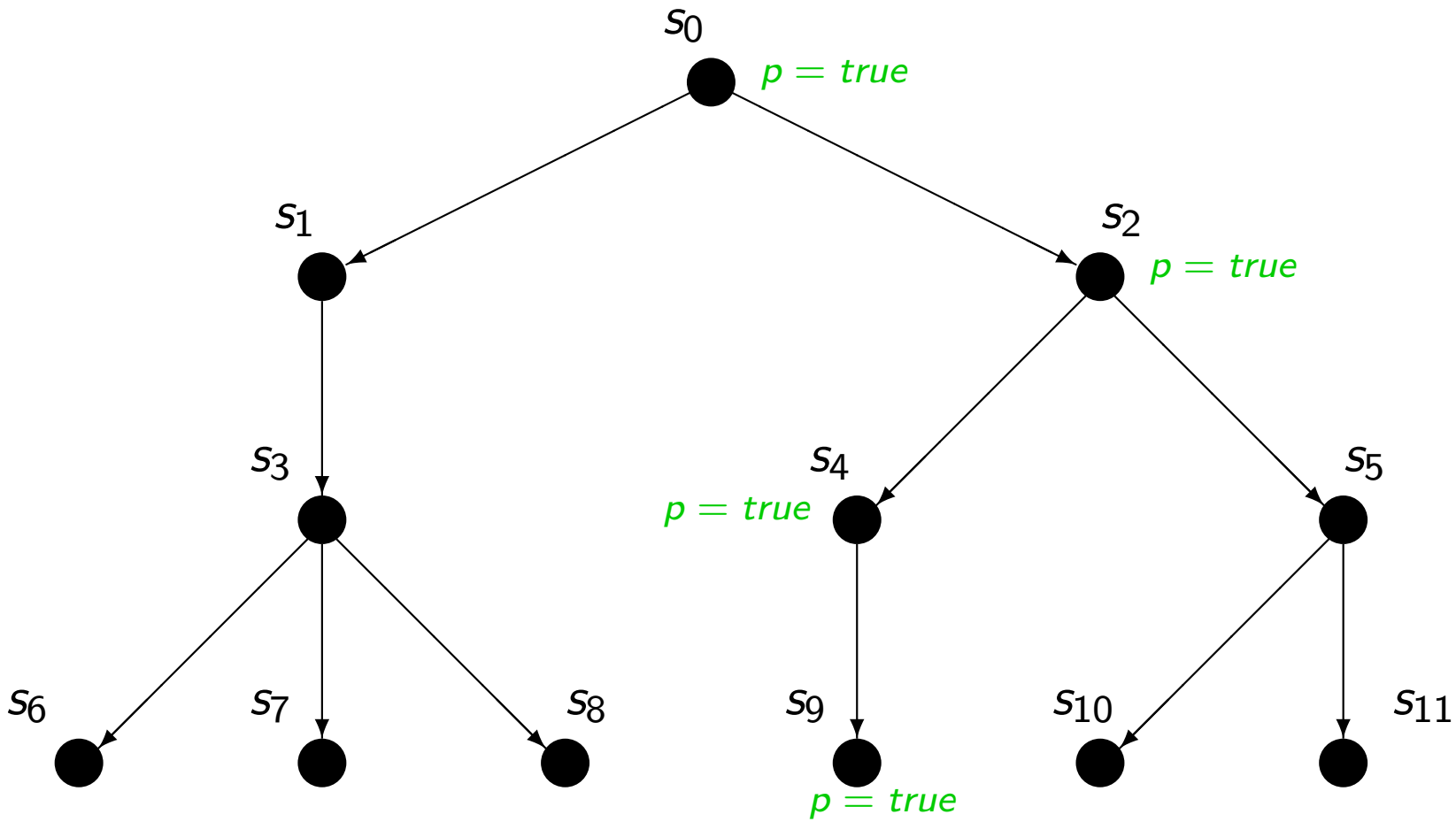
$$I, s_0 \models \mathrm{AG}p$$



CTL

Computational Tree Logic CTL

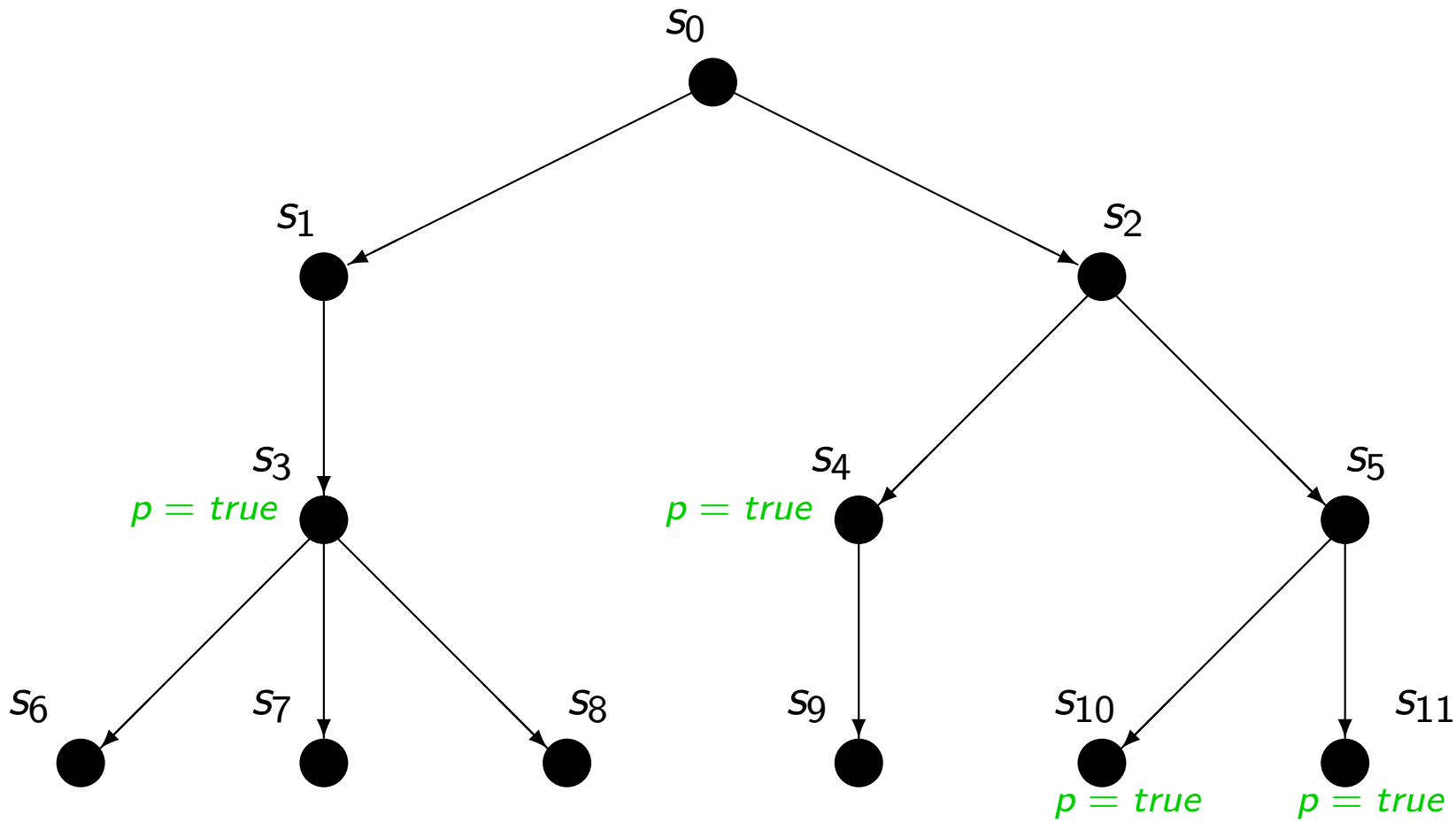
$$I, s_0 \models \mathrm{EG}p$$



CTL

Computational Tree Logic CTL

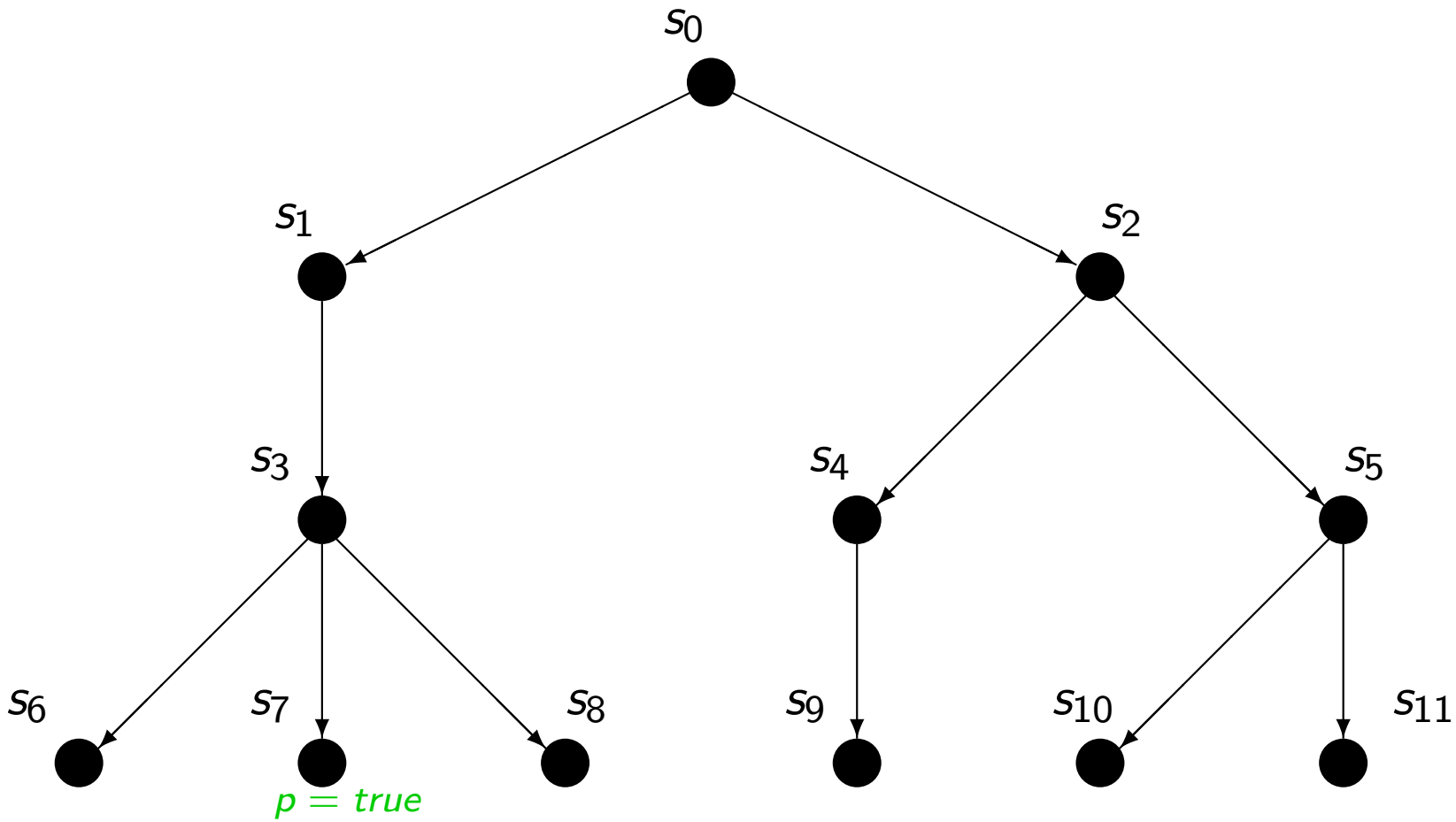
$$I, s_0 \models \mathrm{AF}p$$



CTL

Computational Tree Logic CTL

$$I, s_0 \models \mathrm{EF}\, p$$



Computational Tree Logic CTL

But every temporal operator in CTL can be expressed by means of only three operator EX , EG и EU :

- $\mathbf{AX}\, f \equiv \neg\, \mathbf{EX}(\neg f)$;
- $\mathbf{EF}\, f \equiv \mathbf{E}[\mathit{True}\, \mathbf{U}\, f]$;
- $\mathbf{AG}\, f \equiv \neg\, \mathbf{EF}(\neg f)$;
- $\mathbf{AF}\, f \equiv \neg\, \mathbf{EG}(\neg f)$;
- $\mathbf{A}[f\ \mathbf{U}\ g] \equiv \neg\, \mathbf{E}[\neg g\ \mathbf{U}\ (\neg f \wedge \neg g)] \wedge \neg\, \mathbf{EG}\, \neg g$;
- $\mathbf{A}[f\ \mathbf{R}\ g] \equiv \neg\, \mathbf{E}[\neg f\ \mathbf{U}\ \neg g]$;
- $\mathbf{E}[f\ \mathbf{R}\ g] \equiv \neg\, \mathbf{A}[\neg f\ \mathbf{U}\ \neg g]$.

Computational Tree Logic CTL

Some typical CTL formulas that arise in verification of computing systems with finitely many states are given below:

Computational Tree Logic CTL

Some typical CTL formulas that arise in verification of computing systems with finitely many states are given below:

- **EF** $(Start \wedge \neg Ready)$: it is possible to reach such a computational state when the condition *Start* is satisfied, whereas the condition *Ready* is not;

Computational Tree Logic CTL

Some typical CTL formulas that arise in verification of computing systems with finitely many states are given below:

- **EF** $(Start \wedge \neg Ready)$: it is possible to reach such a computational state when the condition *Start* is satisfied, whereas the condition *Ready* is not;
- **AG** $(Req \rightarrow \mathbf{AF}\, Ack)$: whenever a request is received sooner or later it will be confirmed;

Computational Tree Logic CTL

Some typical CTL formulas that arise in verification of computing systems with finitely many states are given below:

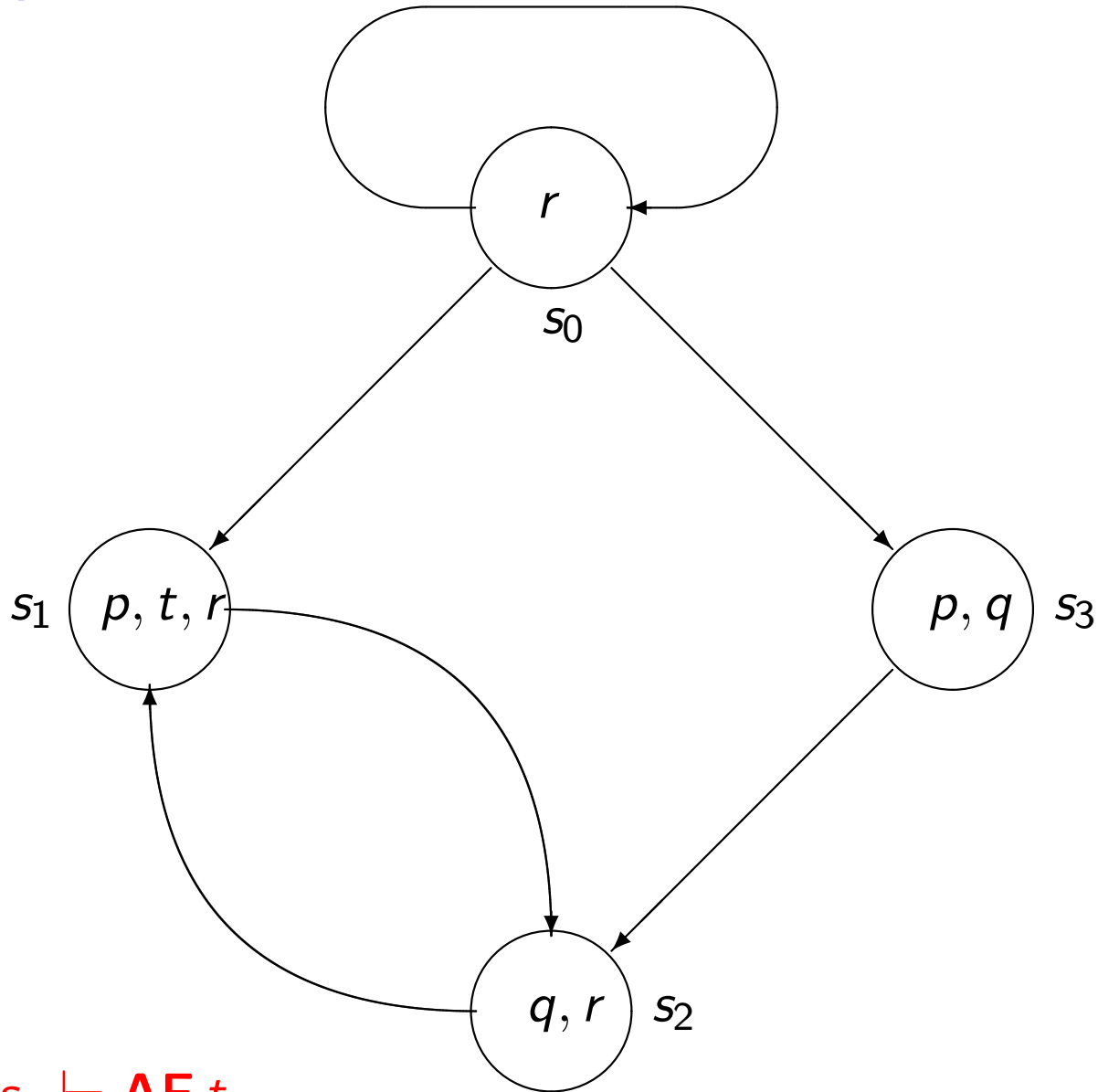
- **EF** $(Start \wedge \neg Ready)$: it is possible to reach such a computational state when the condition *Start* is satisfied, whereas the condition *Ready* is not;
- **AG** $(Req \rightarrow \mathbf{AF}\, Ack)$: whenever a request is received sooner or later it will be confirmed;
- **AG** $(\mathbf{AF}\, DeviceEnabled)$: condition *DeviceEnabled* is satisfied infinitely often in every computation;

Computational Tree Logic CTL

Some typical CTL formulas that arise in verification of computing systems with finitely many states are given below:

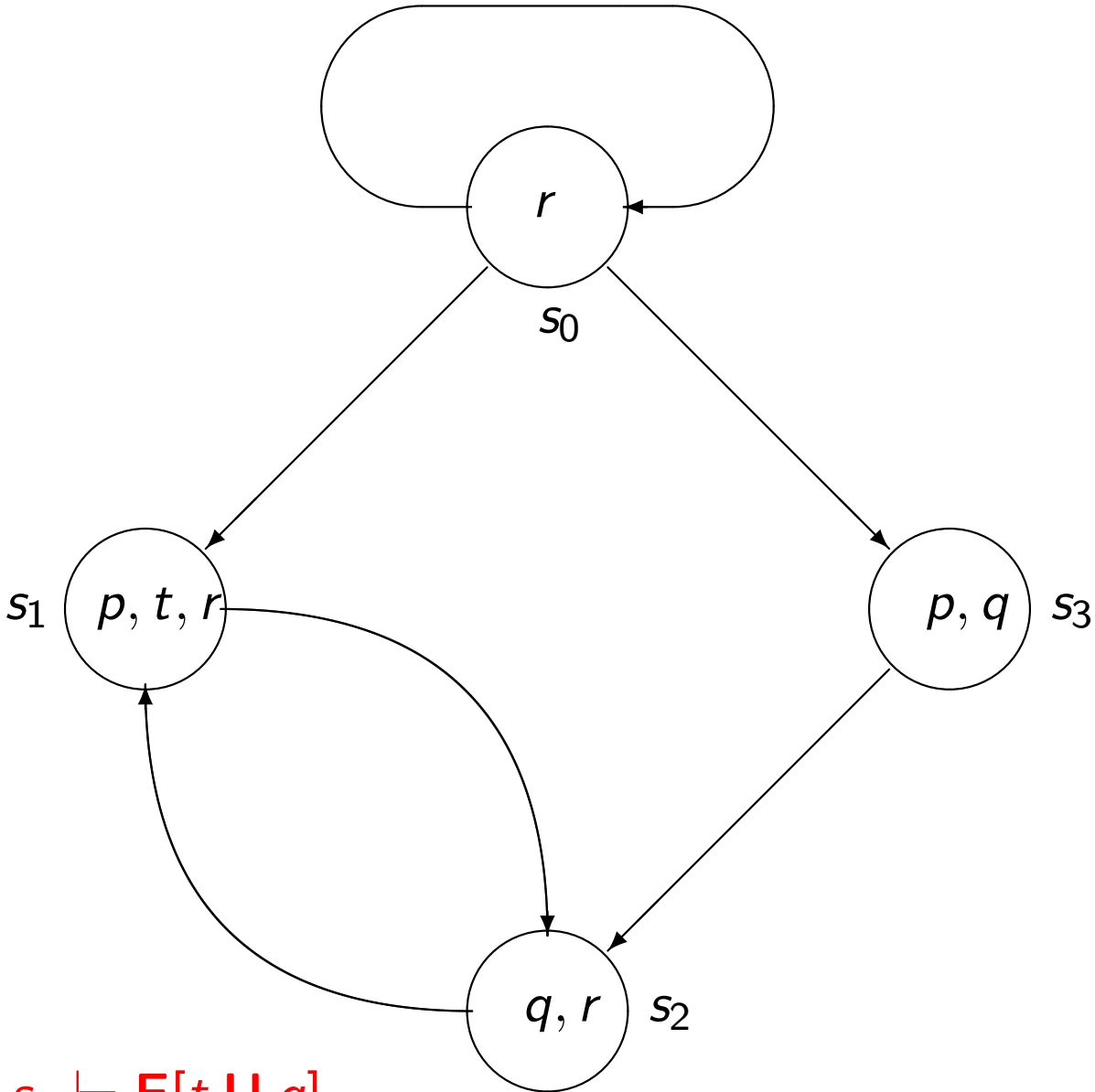
- **EF** $(Start \wedge \neg Ready)$: it is possible to reach such a computational state when the condition *Start* is satisfied, whereas the condition *Ready* is not;
- **AG** $(Req \rightarrow \mathbf{AF}\, Ack)$: whenever a request is received sooner or later it will be confirmed;
- **AG** $(\mathbf{AF}\, DeviceEnabled)$: condition *DeviceEnabled* is satisfied infinitely often in every computation;
- **AG** $(\mathbf{EF}\, Restart)$: the *Restart* state is reachable from every state in every computation.

Check the satisfiability of CTL formulas in the given Kripke structure..



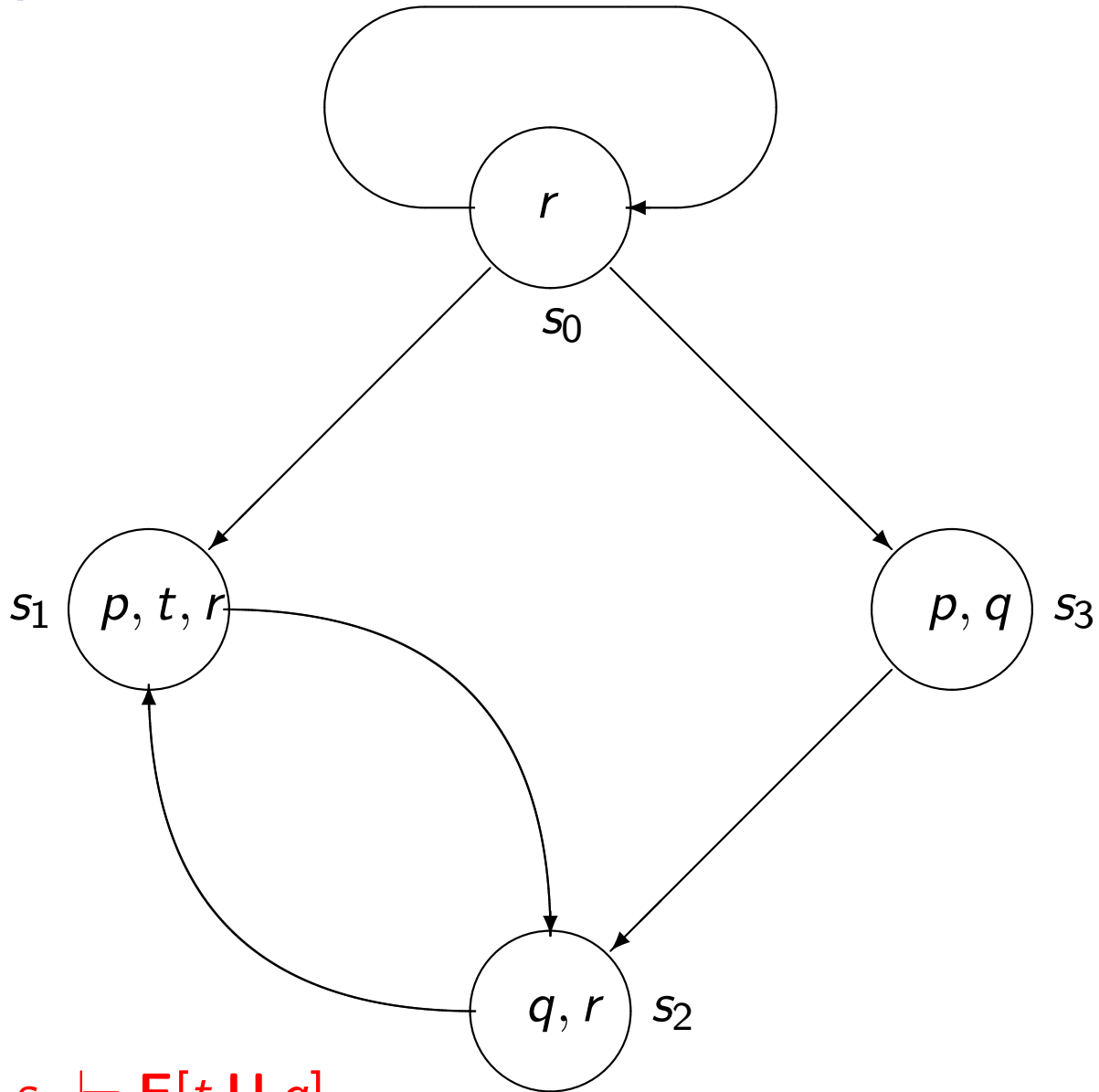
1. $\mathcal{M}_1, s_0 \models \mathbf{AF}\, t$

Check the satisfiability of CTL formulas in the given Kripke structure..



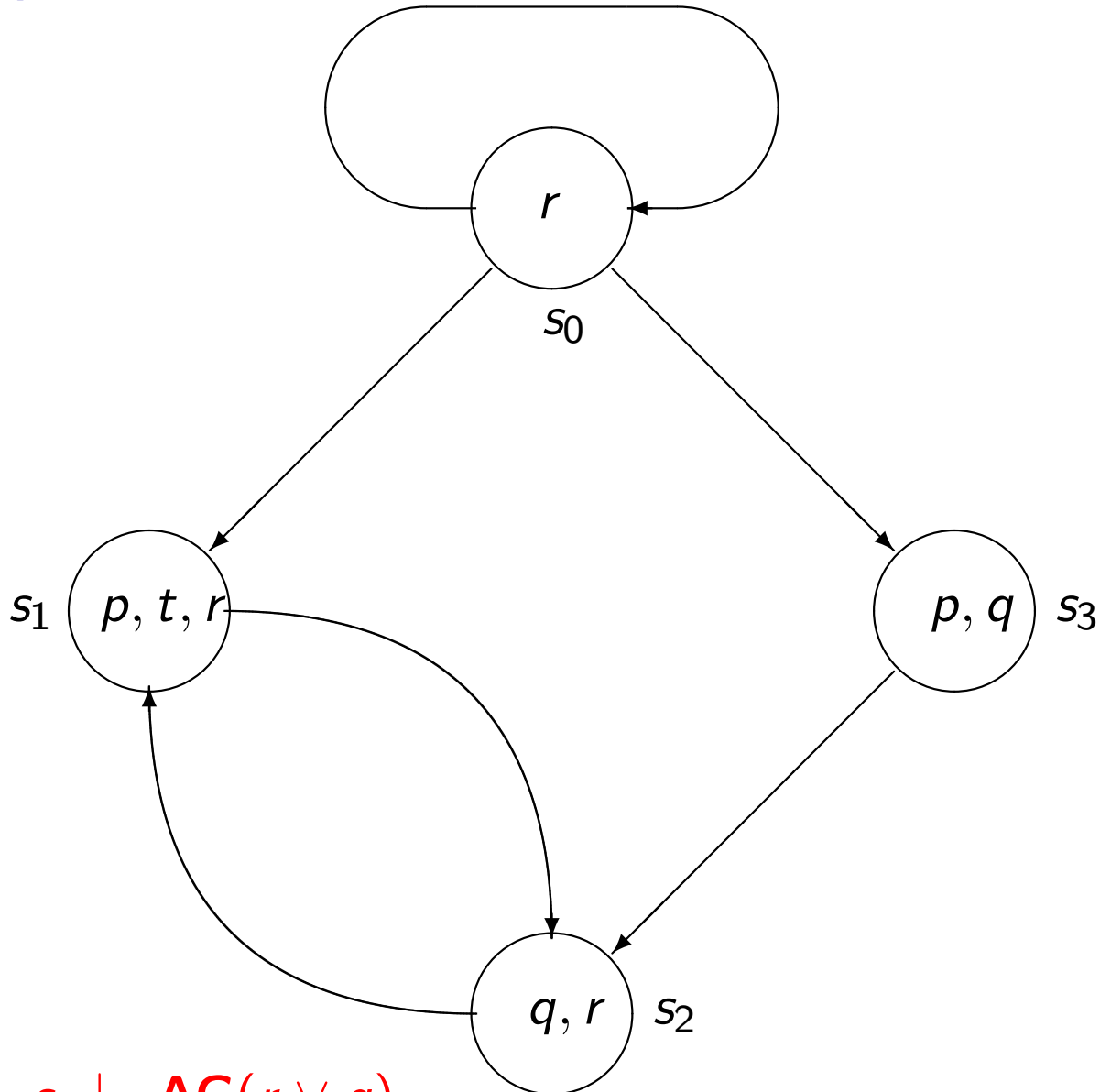
2. $\mathcal{M}_1, s_0 \models \mathbf{E}[t\ \mathbf{U}\ q]$

Check the satisfiability of CTL formulas in the given Kripke structure..



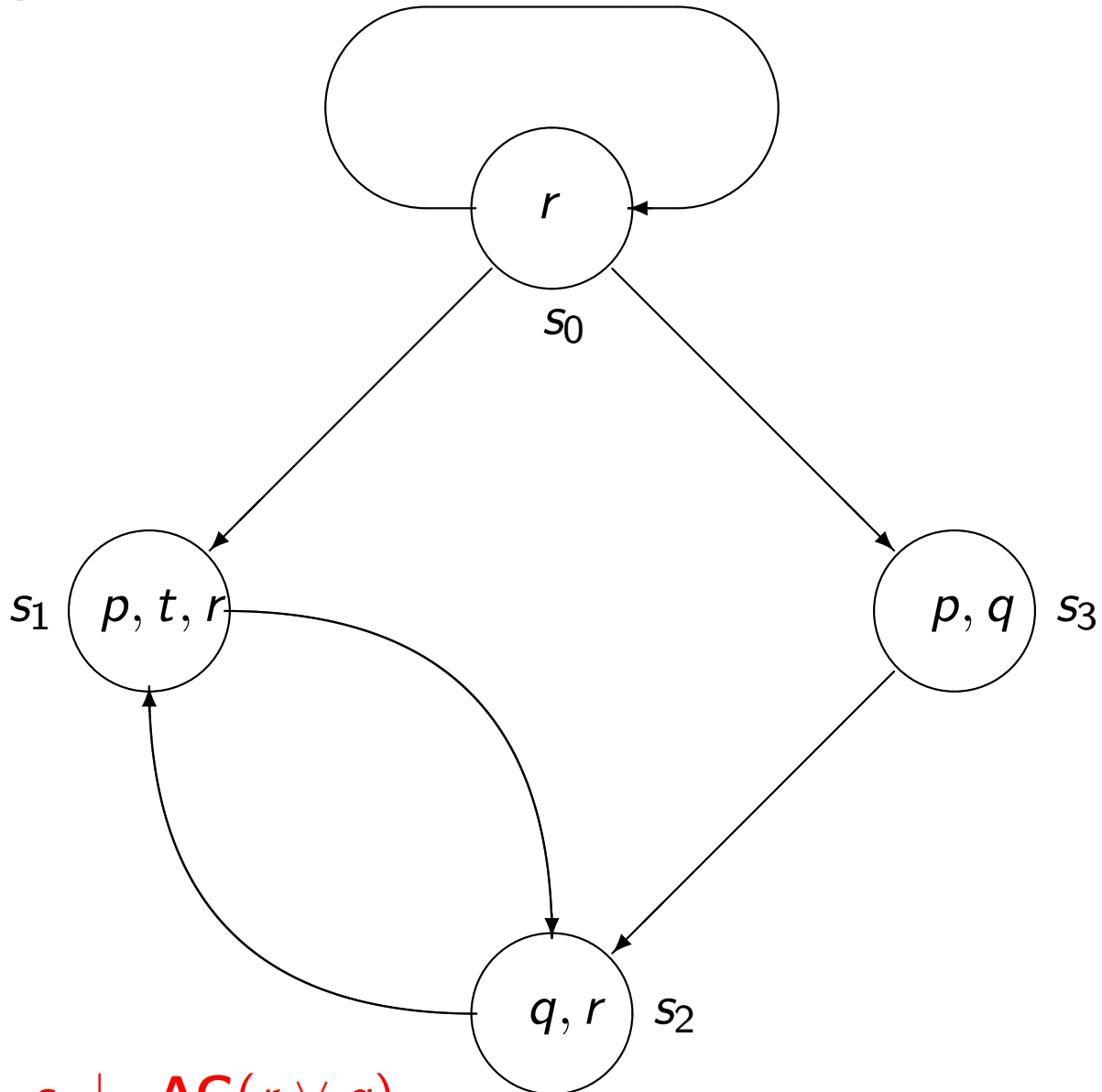
3. $\mathcal{M}_1, s_2 \models \mathbf{E}[t\ \mathbf{U}\ q]$

Check the satisfiability of CTL formulas in the given Kripke structure..



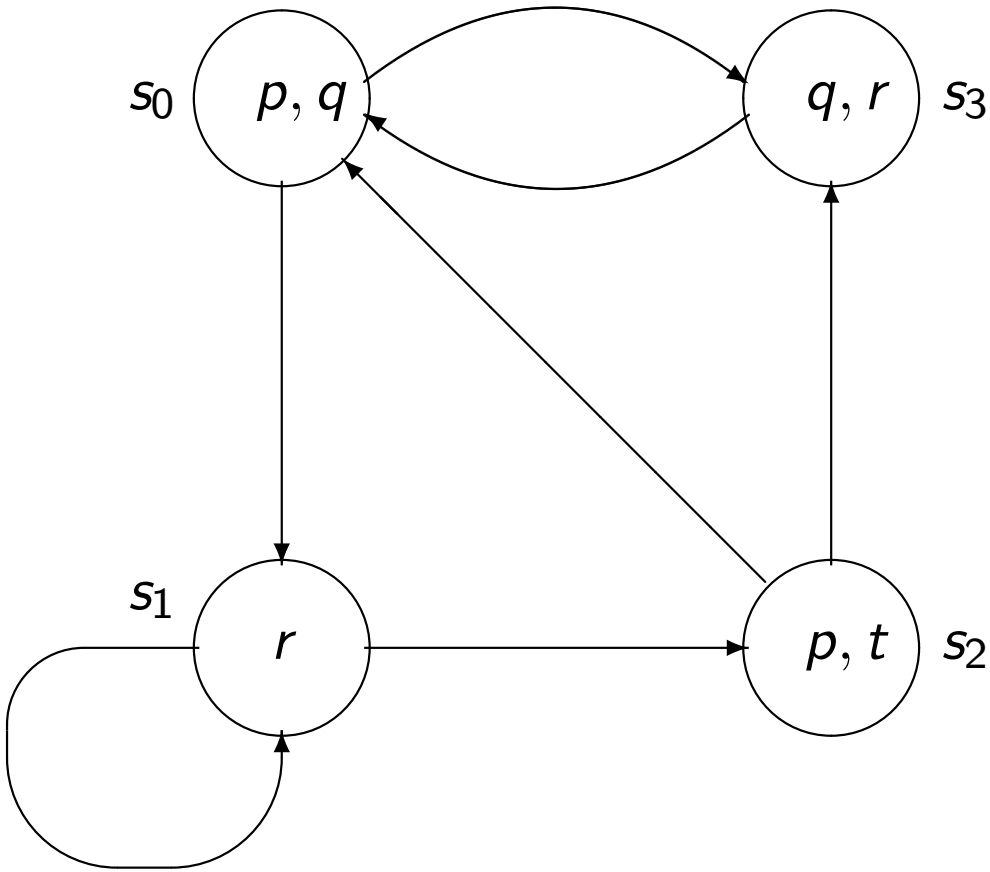
4. $\mathcal{M}_1, s_0 \models \mathbf{AG}(r \vee q)$

Check the satisfiability of CTL formulas in the given Kripke structure..



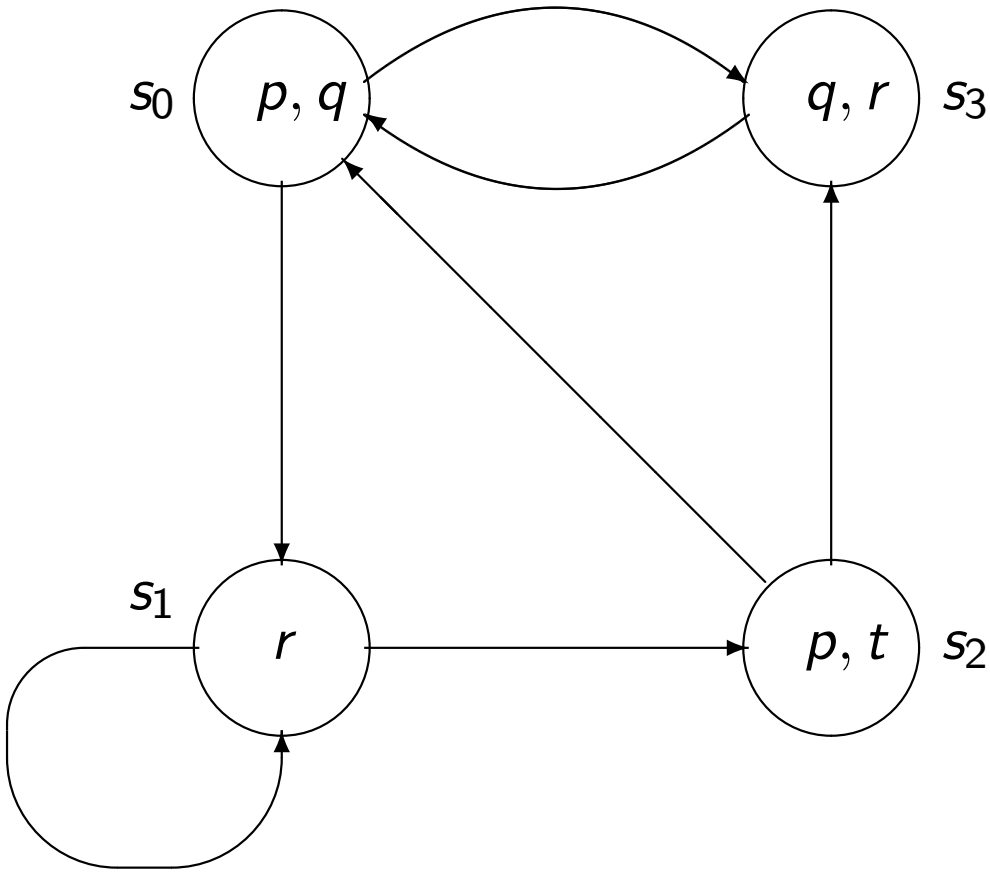
5. $\mathcal{M}_1, s_2 \models \mathbf{AG}(r \vee q)$

Check the satisfiability of CTL formulas in the given Kripke structure.



6. $\mathcal{M}_1, s_0 \models \mathbf{AG}(\mathbf{AF}(p \vee r))$

Check the satisfiability of CTL formulas in the given Kripke structure.



7. $\mathcal{M}_1, s_2 \models \mathbf{AG}(\mathbf{AF}(p \vee r))$

Write down temporal formulas that adequately express the following propositions.

- A failed elevator will remain faulty unless it is repaired.
- Always after receiving a request, sooner or later a response will be issued, unless the request is canceled.
- If an event q occurs after an event p, then an event r will not occur until an event t occurs.
- In all computations an event q precedes events p and r.
- In all computations an event r never occurs between events p and q.
- In all computations an event p occurs at most twice.
- In all computations an event p occurs finitely often.

Which pairs of formulas are equivalent?

- $\mathbf{EF}\,\varphi$ и $\mathbf{EG}\,\varphi$;
- $\mathbf{EF}\,\varphi \vee \mathbf{EF}\,\psi$ и $\mathbf{EF}(\varphi \vee \psi)$;
- $\mathbf{AF}\,\varphi \vee \mathbf{AF}\,\psi$ и $\mathbf{AF}(\varphi \vee \psi)$;
- $AF\neg\varphi$ и $\neg\,\mathbf{EG}\,\varphi$;
- $EF\neg\varphi$ и $\neg\,\mathbf{AF}\,\varphi$;
- $\mathbf{A}[\varphi_1\,\mathbf{U}\,\mathbf{A}[\varphi_2\,\mathbf{U}\,\varphi_3]]$ и $\mathbf{A}[\mathbf{A}[\varphi_1\,\mathbf{U}\,\varphi_2]\,\mathbf{U}\,\varphi_3]$;
- *true* и $\mathbf{AF}\,\varphi \;\rightarrow\; \mathbf{EG}\,\varphi$.

END OF LECTURE 4.