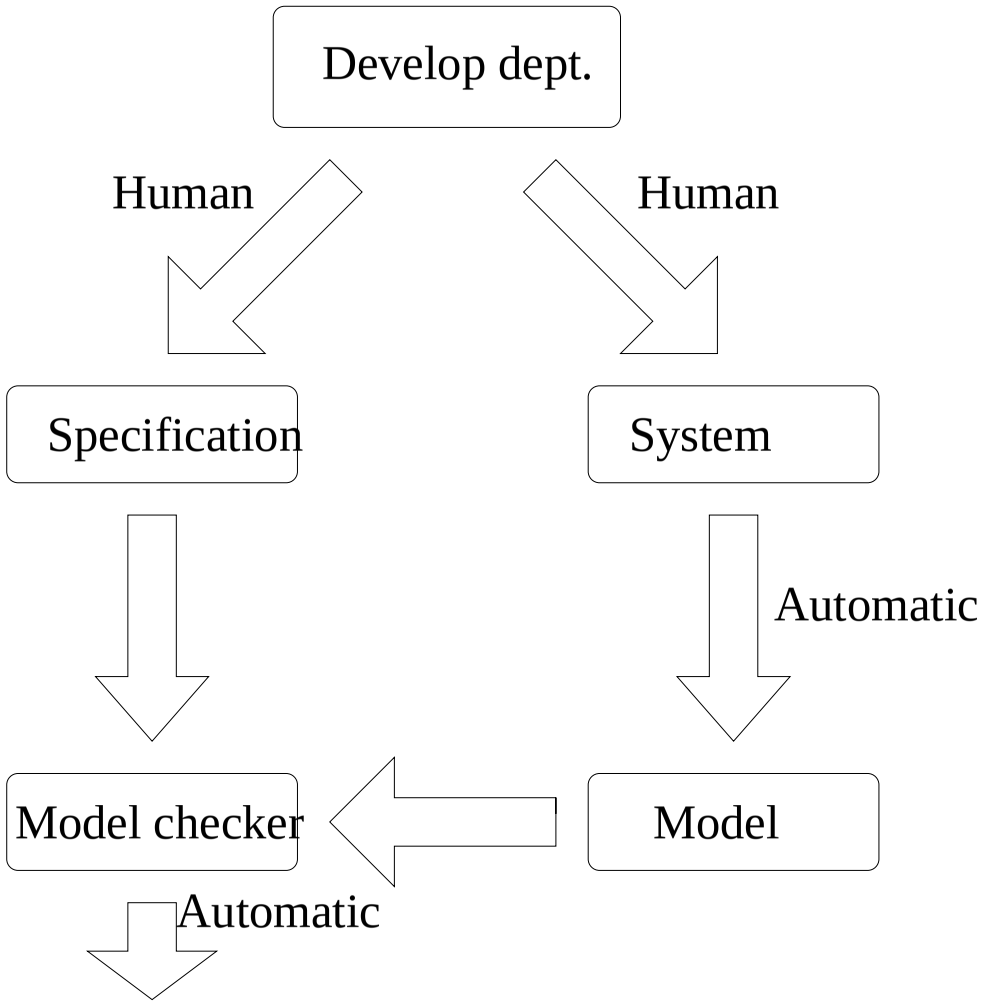
Basic definitions

Doron Bustan
email: doron_b@cs.rice.edu

Model-checking-general flow



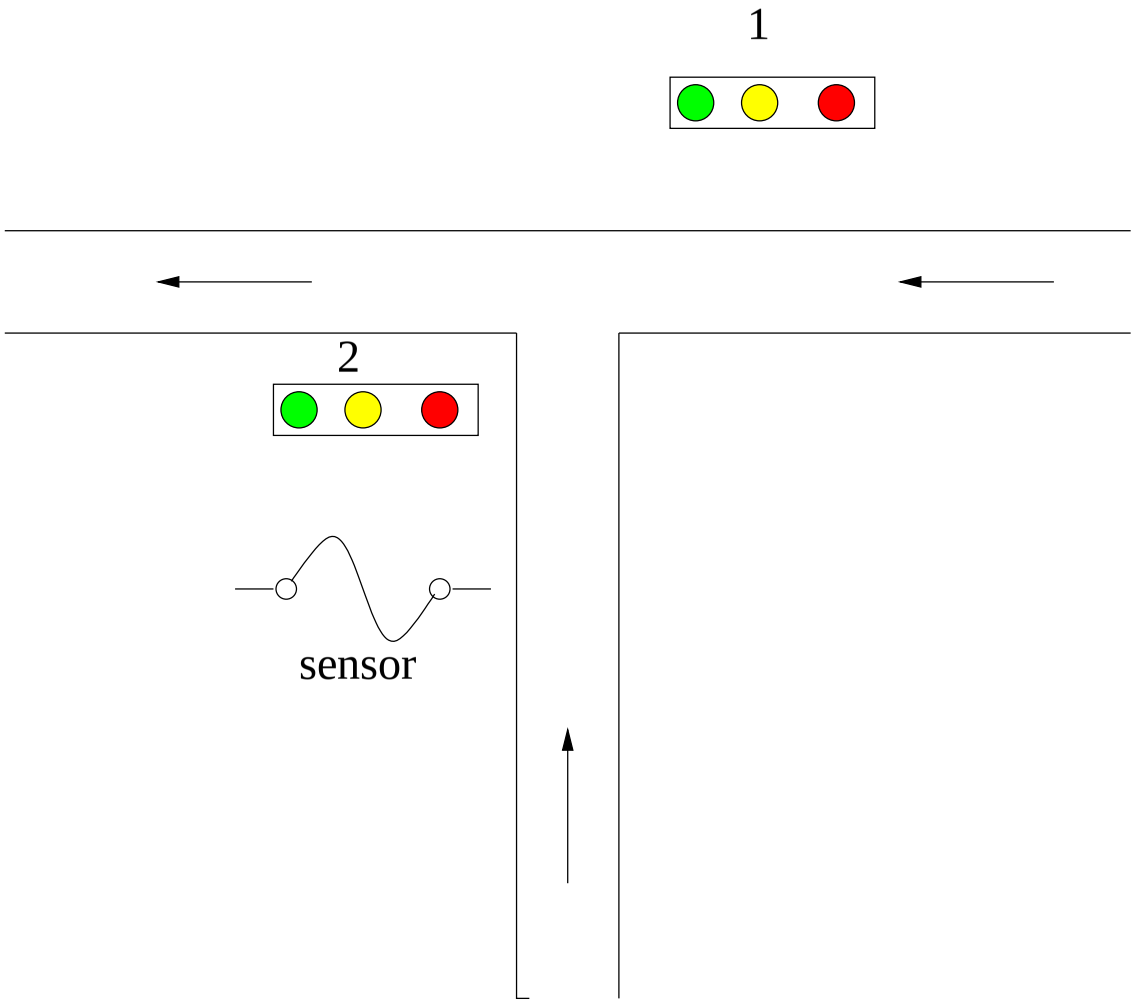
Kripke structure - modeling a system as a graph

A Kripke structure is a four tuple $M =< S, I, R, L >$ where :

- S is a set of states. Represents the set of all configuration of the system, represented by the vertexes of the graph.
- $I \subseteq S$ is a set of initial states - represents the initial configurations of the system.
- R is a transition relation. A transition implies that the system can move from one configuration to another, a transition is represented by an edge in the graph.
- $L : S \rightarrow 2^{A_t}$ is a labeling function, which labels each state the atoms true in the configuration it represents.

An example

Assume that we want to model the following system:



An example (cont.)

The system includes two traffic lights t_1, t_2, and a sensor $sens$. Where $t_1, t_2 \in \{G, Y, R\}$ and $sens \in \{T, F\}$.
Note that the system cannot determine the value of $sens$.

An example (cont.)

We suggest the following program:

`initially:` $t_1 = G, t_2 = R$;

`while(T){`

$t_1 = G \wedge sens = T \quad \rightarrow \quad t_1 = Y$

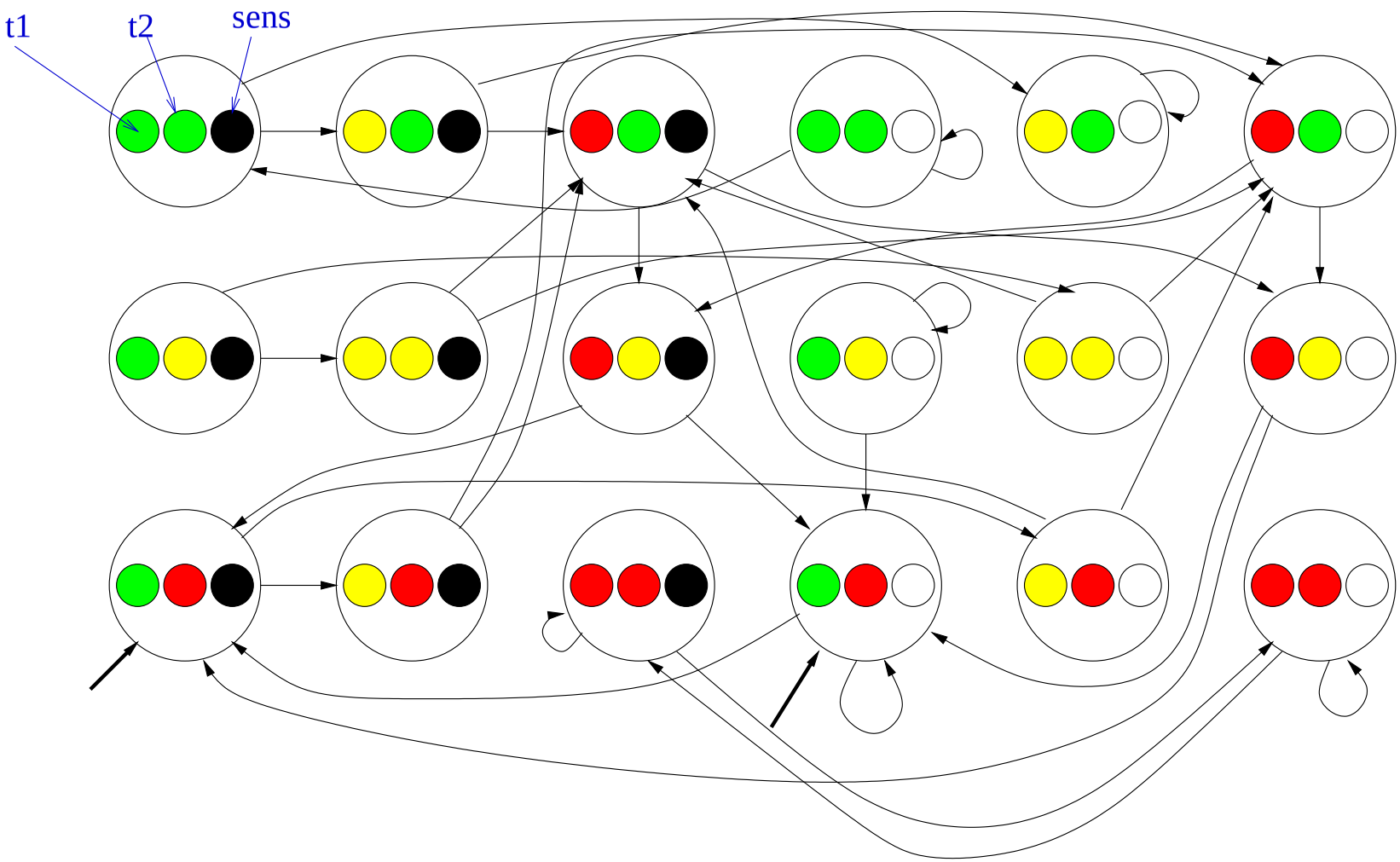
$t_1 = Y \quad \rightarrow \quad t_1 = R,\ t_2 = G$

$t_2 = G \quad \rightarrow \quad t_2 = Y$

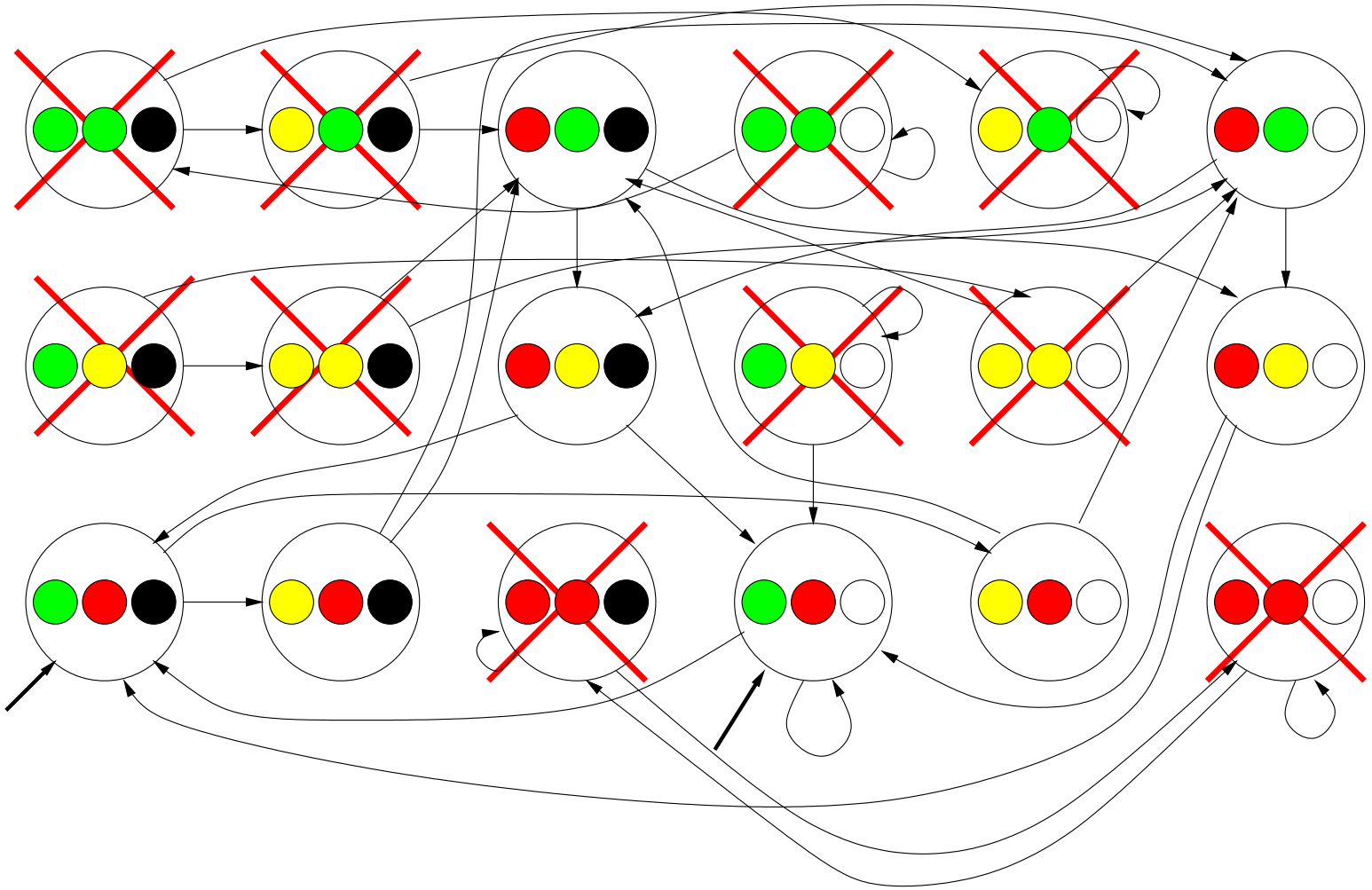
$t_2 = Y \quad \rightarrow \quad t_1 = G,\ t_2 = R$

`}`

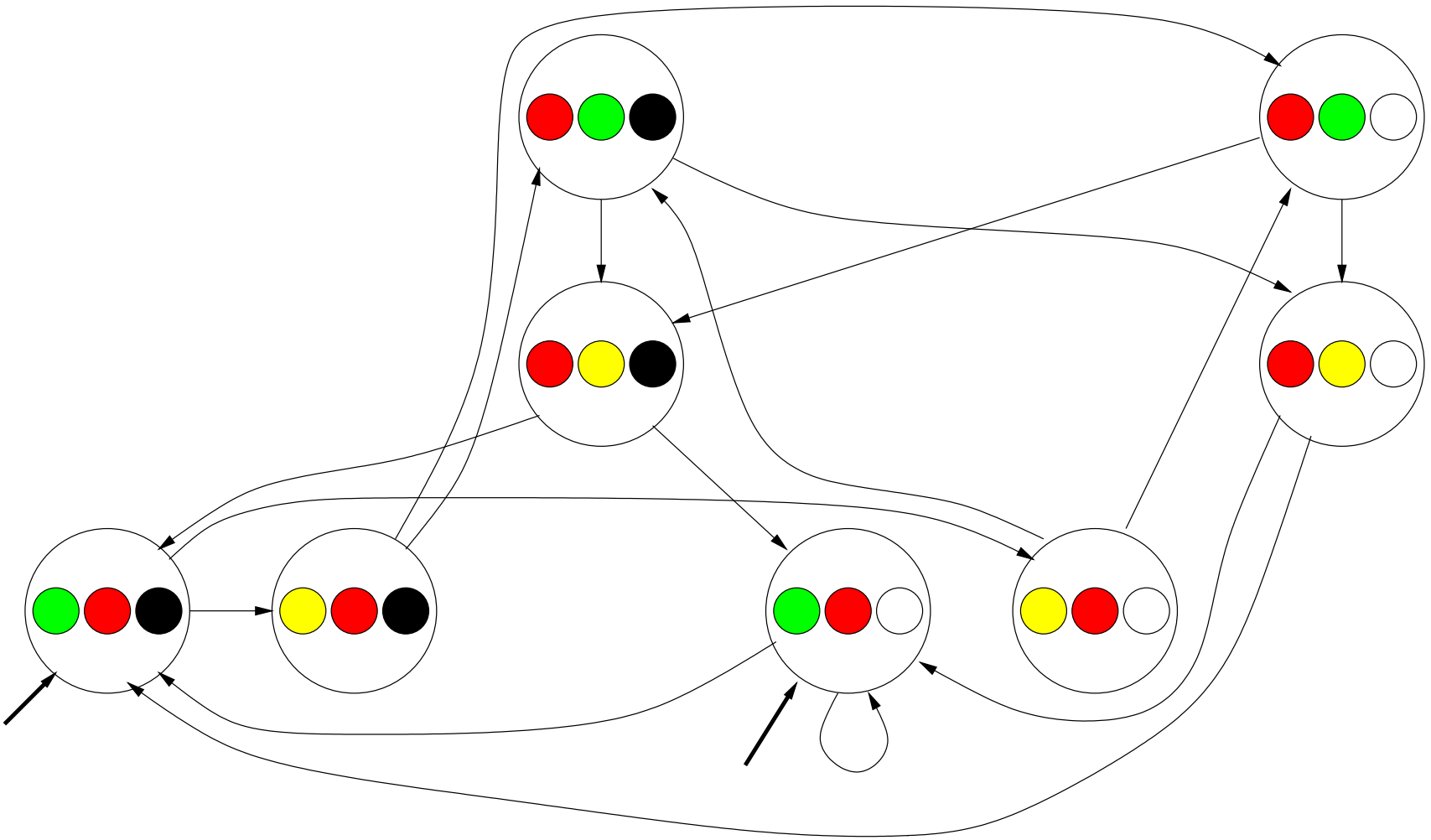
The Kripke structure



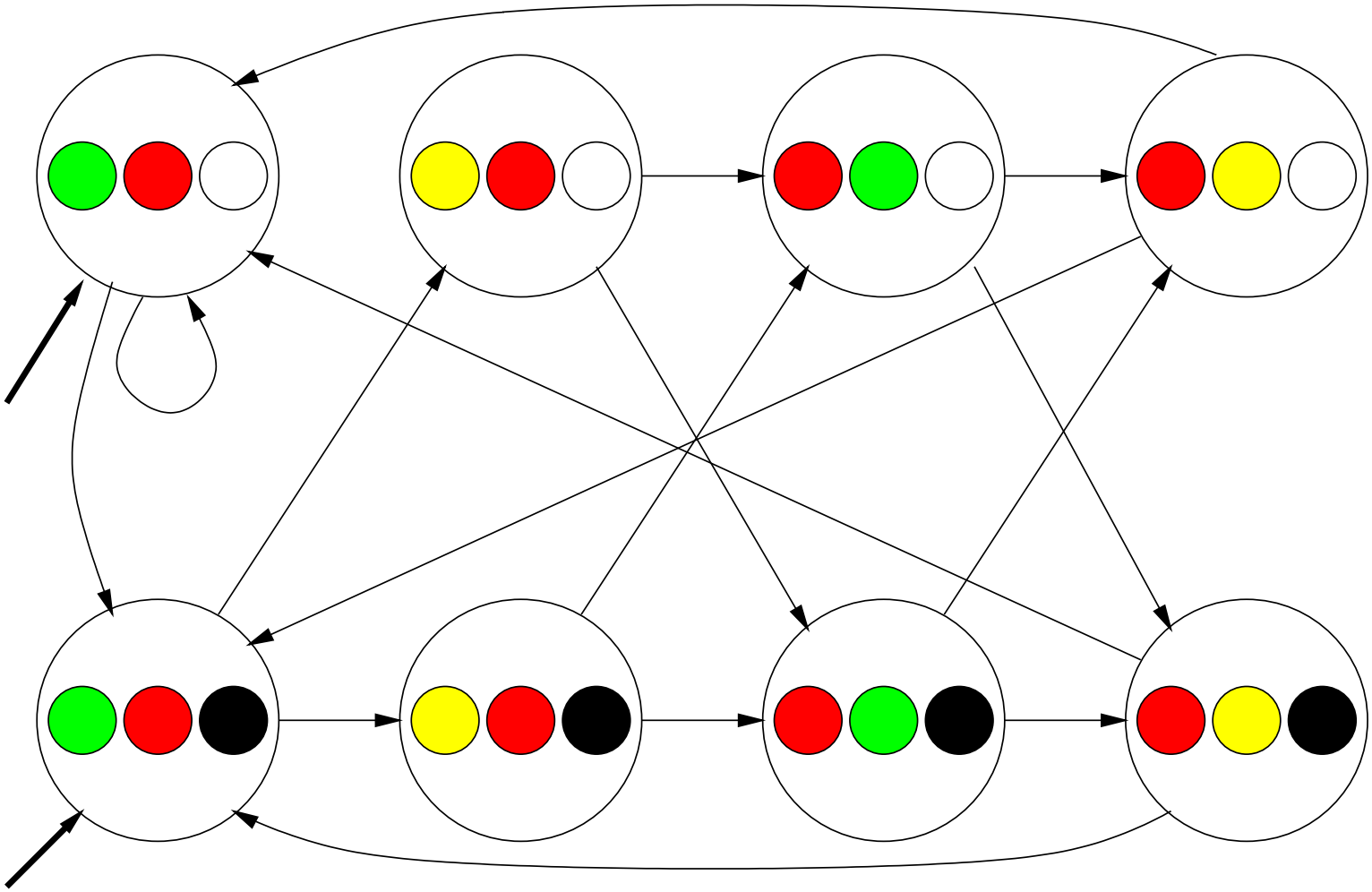
The Kripke structure



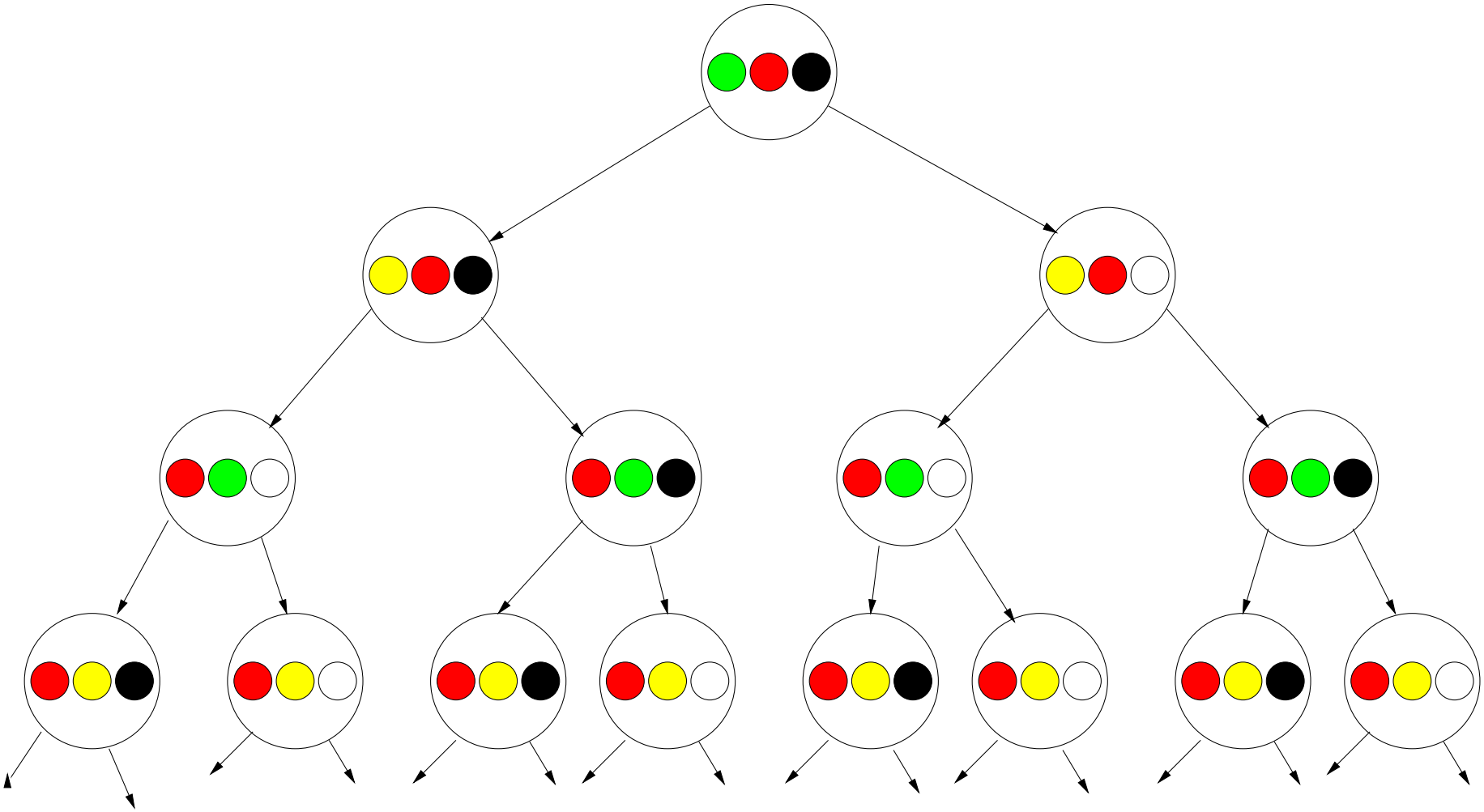
The Kripke structure



The Kripke structure



The computation tree of the system



Temporal logic

- Formulas in first order logic can specify properties of a single state.
- We want to express properties of the whole system.

A path $\rho = s_0, s_1, \ldots$ is an infinite sequence, such that for every i, $(s_i, s_{i+1}) \in R$.
In temporal logic we can specify in addition to state properties, path properties.

The CTL* temporal logic - syntax

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

The CTL* temporal logic - syntax

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

CTL* - semantics

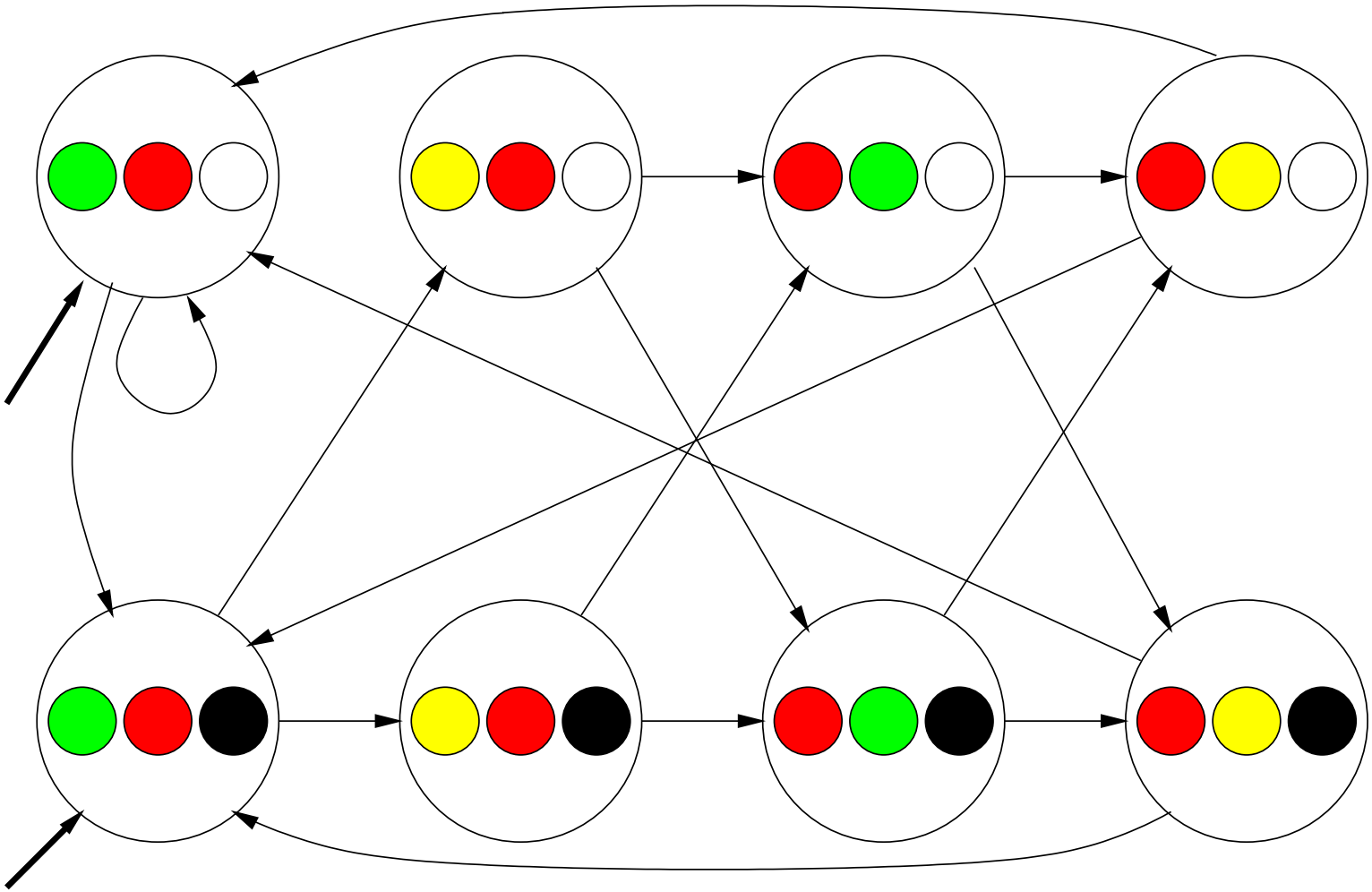
Let M be a Kripke structure, s a state in S and ρ a path in M. We use ρ^i to denote the suffix of ρ from it's ith state s_i.
Let f, f_1, f_2 be state formulas, and let g, g_1, g_2 be path formulas.

- $M, s \models p \Leftrightarrow p \in L(s)$.
- $M, s \models \neg f \Leftrightarrow M, s \not\models f$.
- $M, s \models f_1 \wedge f_2 \Leftrightarrow M, s \models f_1$ and $M, s \models f_2$.
- $M, s \models f_1 \vee f_2 \Leftrightarrow M, s \models f_1$ or $M, s \models f_2$.
- $M, s \models \mathbf{E}\, g \Leftrightarrow$ there exists a path ρ starting at s such that $M, \rho \models g$.
- $M, s \models \mathbf{A}\, g \Leftrightarrow$ every path ρ that starts at s satisfies $M, \rho \models g$.
- $M, \rho \models f \Leftrightarrow$ the first state s of ρ satisfies $M, s \models f$.

- $M, \rho \models \neg g \Leftrightarrow M, \rho \not\models g$.
- $M, \rho \models g_1 \wedge g_2 \Leftrightarrow M, \rho \models g_1$ and $M, \rho \models g_2$.
- $M, \rho \models g_1 \vee g_2 \Leftrightarrow M, \rho \models g_1$ or $M, \rho \models g_2$.
- $M, \rho \models \mathbf{X}\, g \Leftrightarrow M, \rho^1 \models g$.
- $M, \rho \models \mathbf{F}\, g \Leftrightarrow \exists k \geq 0$ such that $M, \rho^k \models g$.
- $M, \rho \models \mathbf{G}\, g \Leftrightarrow \forall k \geq 0$ it holds that $M, \rho^k \models g$.
- $M, \rho \models g_1\ \mathbf{U}\ g_2 \Leftrightarrow \exists k \geq 0$ such that $M, \rho^k \models g_2$ and $\forall 0 \leq j < k$ it holds that $\rho^j \models g_1$.
- $M, \rho \models g_1\ \mathbf{R}\, g_2 \Leftrightarrow \forall j \geq 0,$ if $\forall i \leq j, M, \rho^i \not\models g_1,$ then $M, \rho^j \models g_2$.

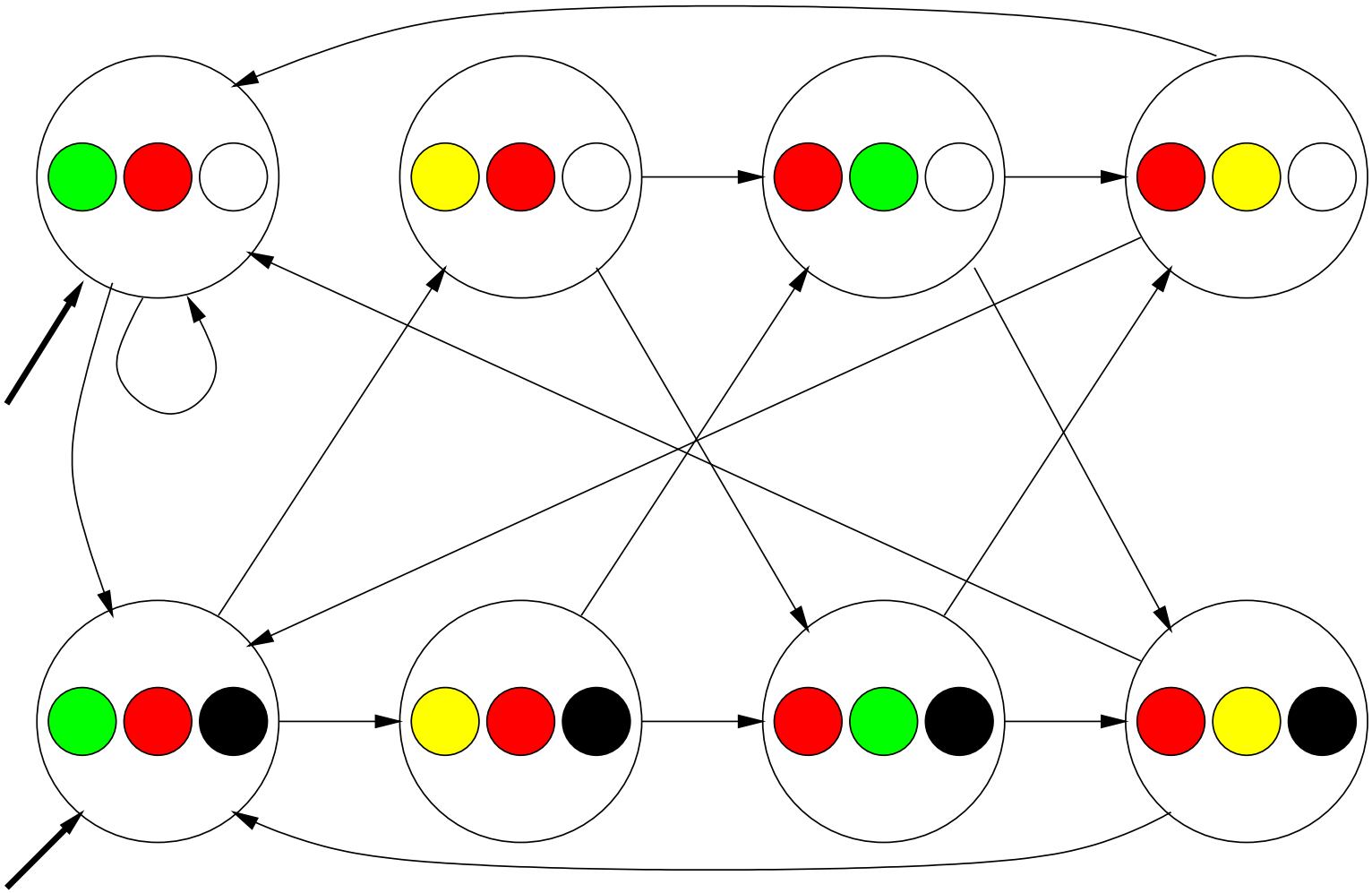
Example

We don't want cars to go in both directions



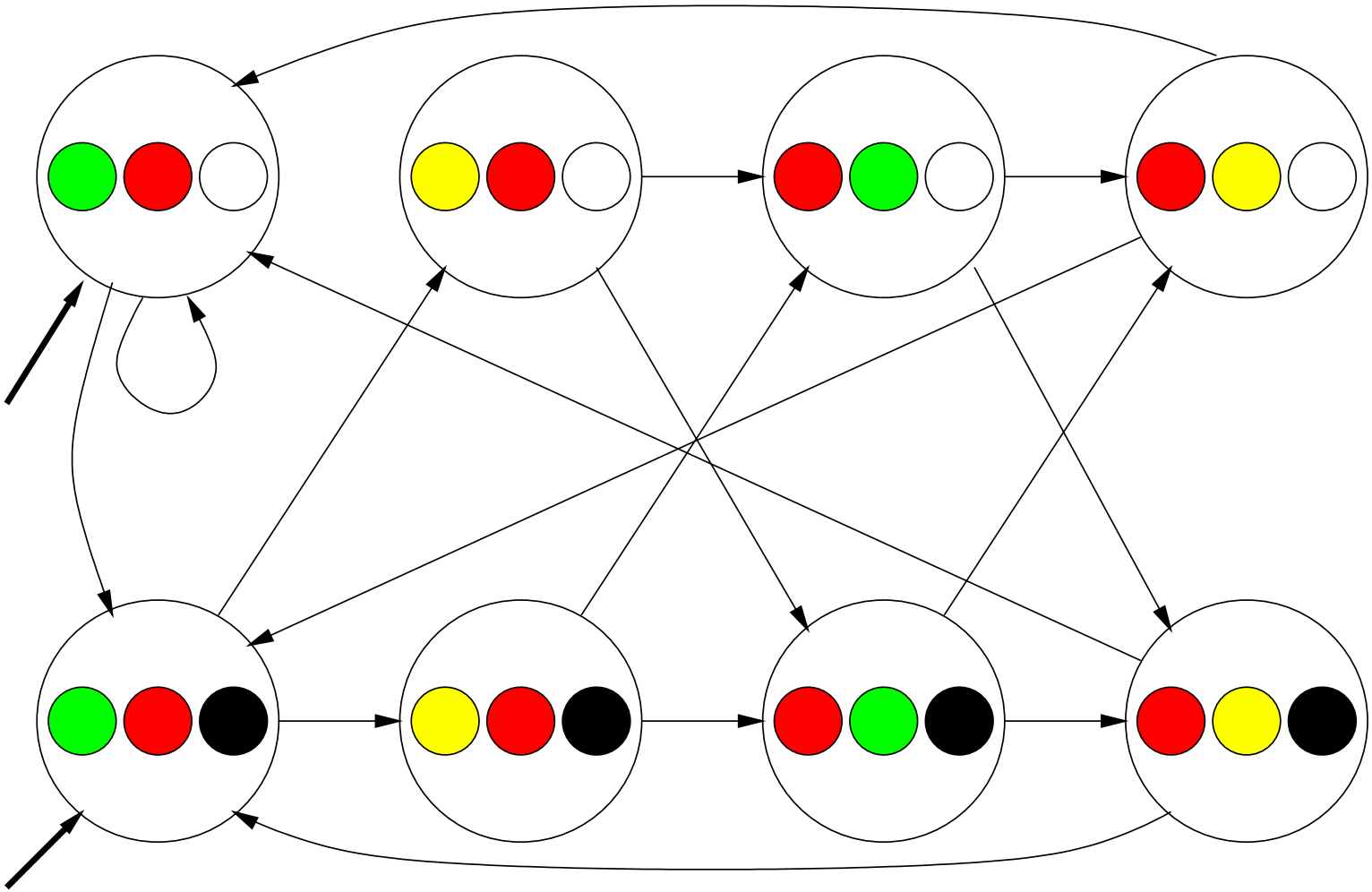
Example

$\mathbf{A}\,\mathbf{G}(t_1 = R \vee t_2 = R)$



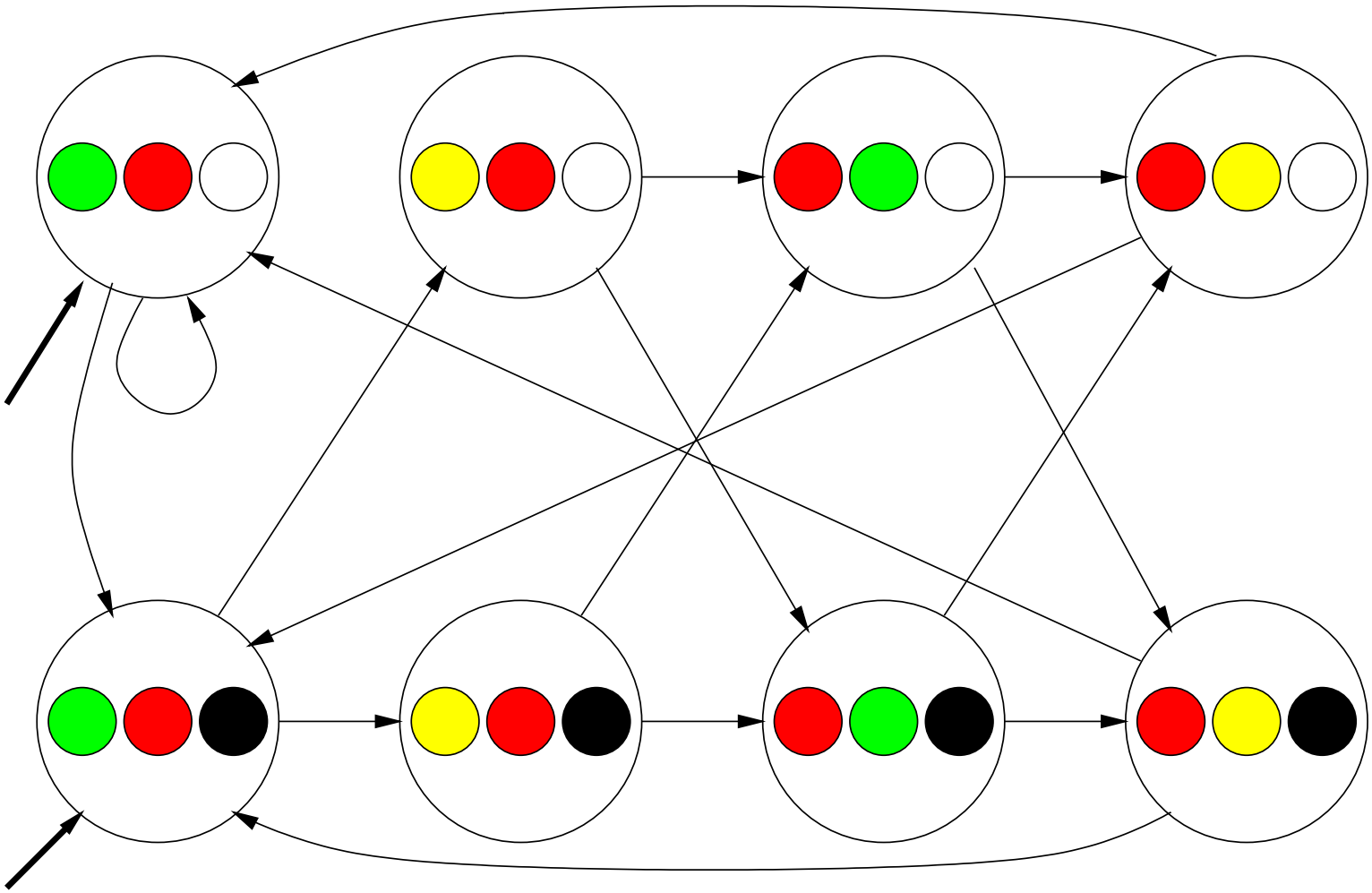
Example

Whenever a car waits for t_2 it eventually become green



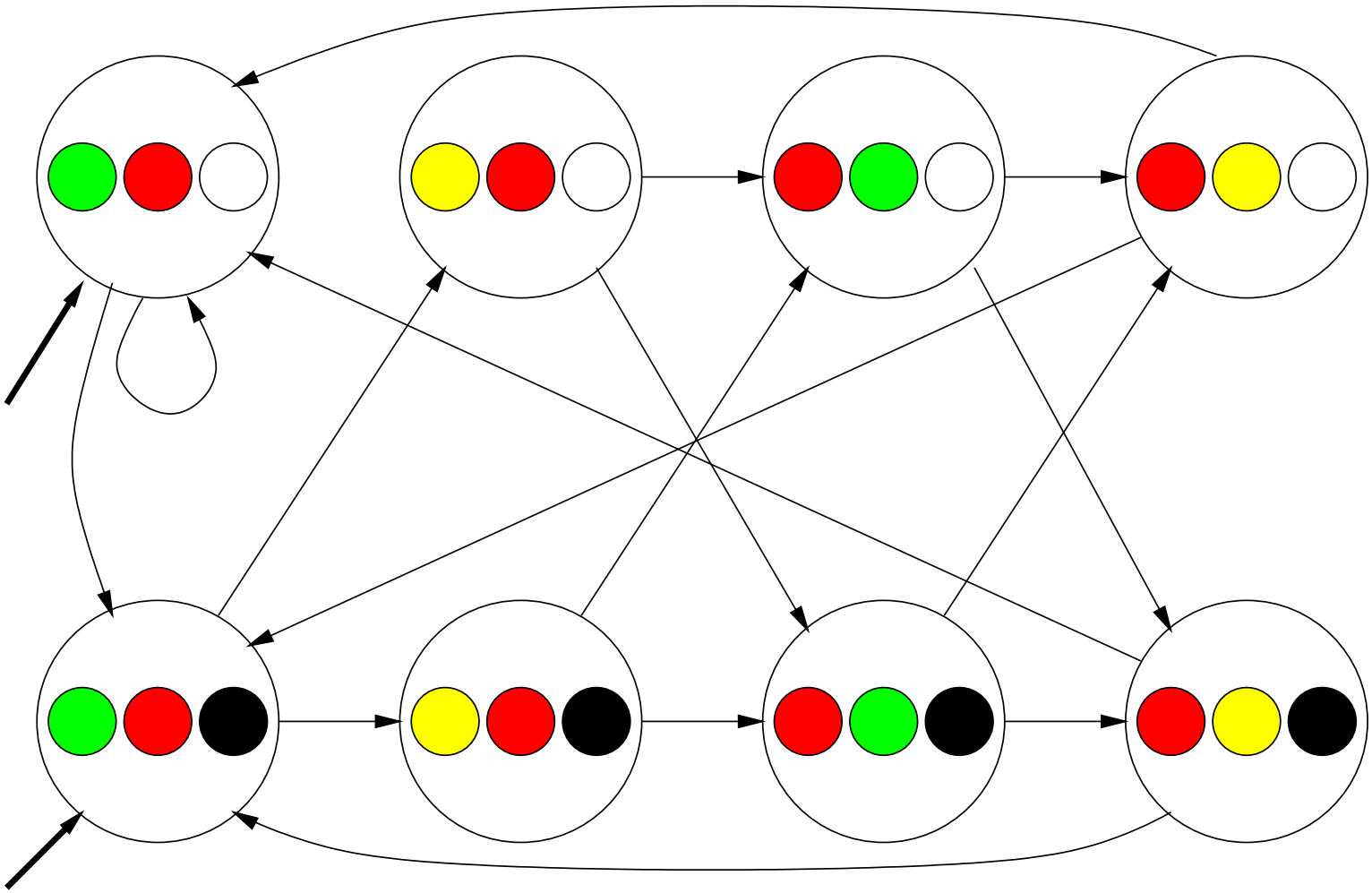
Example

$\mathbf{A}\,\mathbf{G}(sens = T \rightarrow \mathbf{A}\,\mathbf{F}\,t_2 = G)$



Example

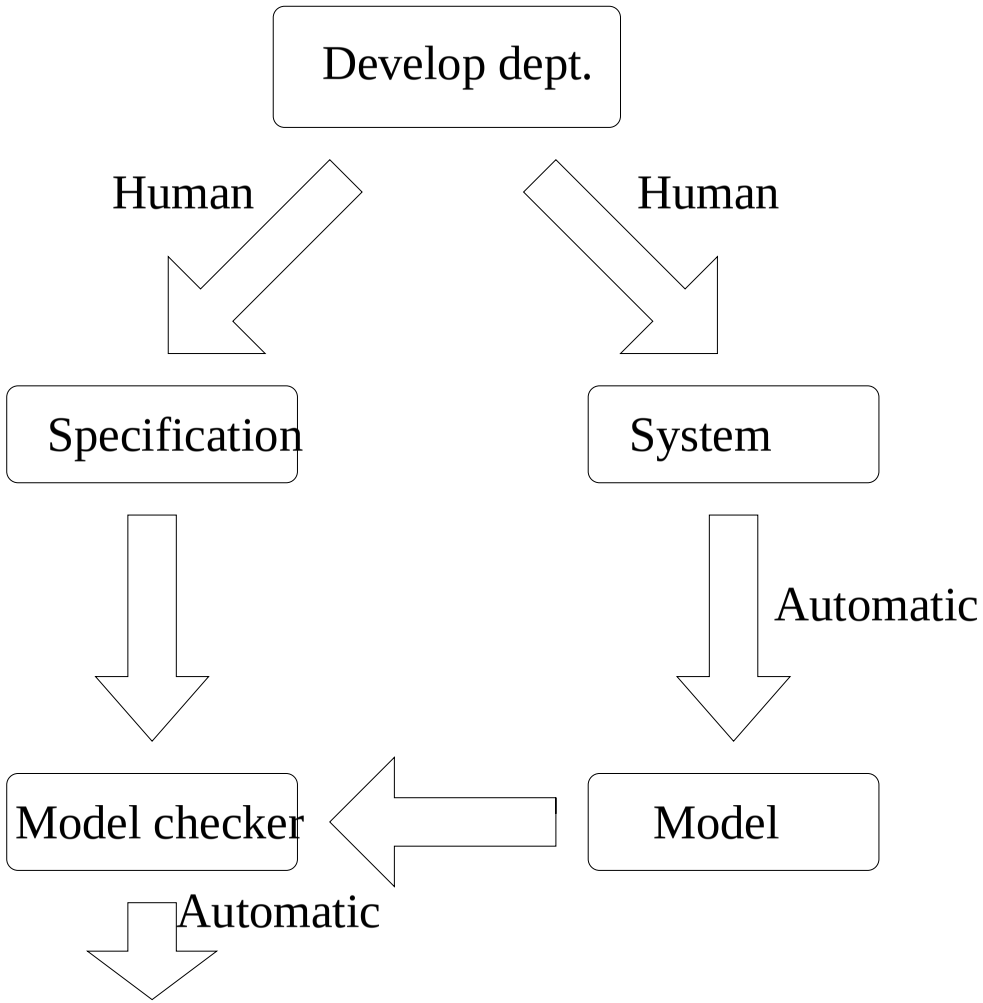
$\neg \mathbf{E}\,\mathbf{F}\,\mathbf{G}(sens = T \wedge \neg(t_2 = G))$



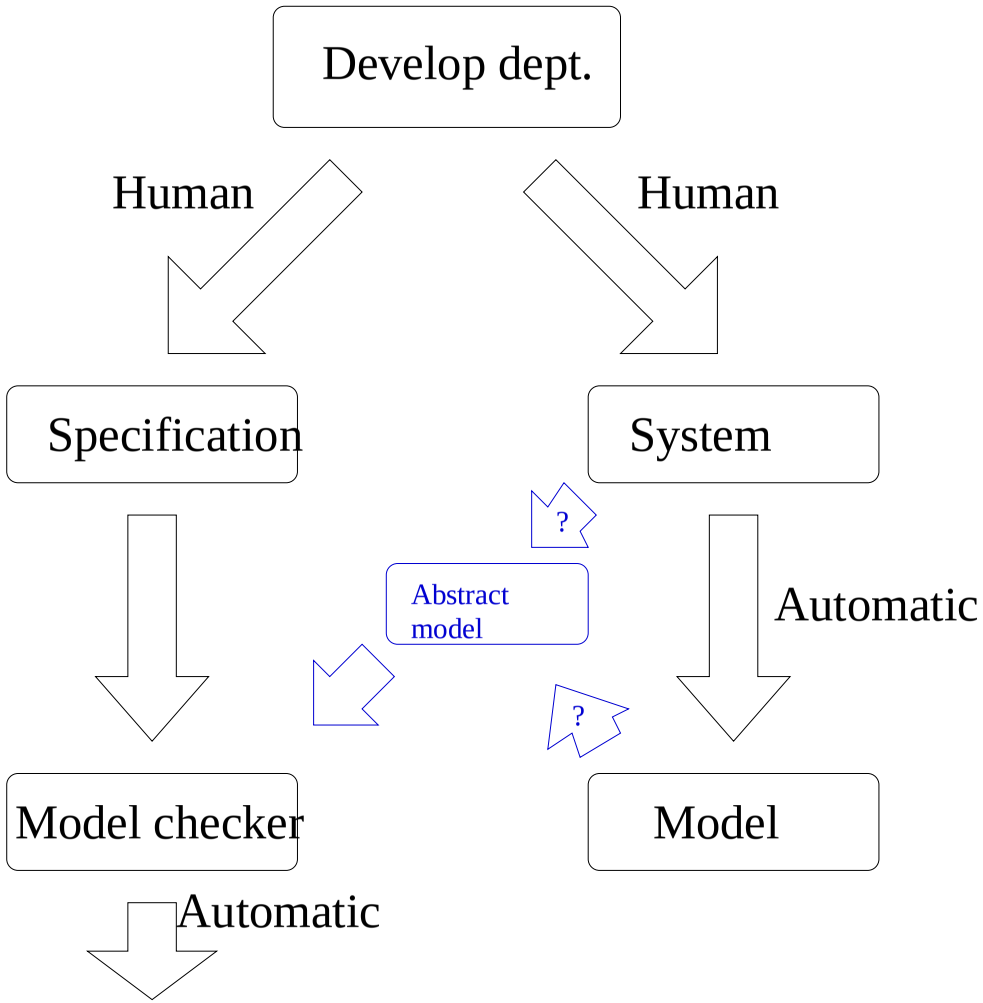
ACTL*

The temporal logic ACTL*is a restriction of CTL*where the **E** quantifier is not allowed, and the $\neg$ operator is used only on atom formulas.

General flow



Abstraction



Abstraction

Assume that we have a system K, a concrete model M of K, and an abstract model A of K. We want A to satisfy the following:

1. A should be smaller/simpler than M.
2. We should be able to verify K by running algorithms on A.
3. Constructing A should be cheaper than using M for verification.

Abstraction without approximation

- We want A to have to have the same properties as M (K).
- Can A have the same properties as M and be simpler than M at the same time?
- Yes, if equality is required only for a restricted set of the properties that we want to verify. In this case we say that M and A are equivalent with respect to this set of properties.

For example A and M are equivalent with respect to all the properties which can be specify by CTL*formulas, meaning $\forall\psi \in$CTL*, $M \models \psi \Leftrightarrow A \models \psi$.

Abstraction with over-approximation

Consider the temporal logic ACTL*. The formulas of ACTL*describe properties that all the paths of the system should satisfy.

Let A be an abstract model that over approximate M and let $\psi \in$ACTL*. Assume that $A \models \psi$, then all paths of A satisfy the property described by ψ. Since the set of paths of A contains the set of paths of M, all paths of M satisfy the property described by ψ, thus $M \models \psi$.
Hence, if A is an over approximation of M then $\forall \psi \in$ACTL*, $A \models \psi \rightarrow$ $M \models \psi$.

Abstraction with over-approximation

For the same reasons, if A is an under approximation of M then $\forall \psi \in$ACTL*, $M \models \psi \rightarrow A \models \psi$.

Bisimulation

Let $M =< S_1, I_1, R_1, L_1 >$ and $M_2 =< S_2, I_2, R_2, L_2 >$ be Kripke structures over the same set of atomic formulas A_t, a relation $B \subseteq S_1 \times S_2$ is *bisimulation relation* over M_1 and M_2 if and only if for every $(s_1, s_2) \in B$ the following conditions hold:

- $L_1(s_1) = L_2(s_2)$
- For every state t_1 such that $(s_1, t_1) \in R_1$, there exists a state t_2 such that $(s_2, t_2) \in R_2$ and $(t_1, t_2) \in B$.
- For every state t_2 such that $(s_2, t_2) \in R_2$, there exists a state t_1 such that $(s_1, t_1) \in R_1$ and $(t_1, t_2) \in B$.

Bisimulation

The structures M_1 and M_2 are bisimulation equivalent (denoted $M_1 \equiv M_2$) if there exists a bisimulation relation B over M_1 and M_2 which satisfy: for every $s_{01} \in I_1$ there exists $s_{02} \in I_2$ such that $(s_{01}, s_{02}) \in B$ and for every $s_{02} \in I_2$ there exists $s_{01} \in I_1$ such that $(s_{01}, s_{02}) \in B$.

Bisimulation

Theorem: $M_1 \equiv M_2$ if and only if for every formula ψ in CTL*, $M_1 \models \psi \Leftrightarrow M_2 \models \psi$.

Theorem: $\equiv$ is an equivalence relation.

Simulation

Let $M =< S_1, I_1, R_1, L_1 >$ and $M_2 =< S_2, I_2, R_2, L_2 >$ be Kripke structures with $A_{t2} \subseteq A_{t1}$, a relation $H \subseteq S_1 \times S_2$ is *Simulation relation* over M_1 and M_2 if and only if for every $(s_1, s_2) \in H$ the following conditions hold:

- $L_1(s_1) \cap A_{t2} = L_2(s_2)$
- For every state t_1 such that $(s_1, t_1) \in R_1$, there exists a state t_2 such that $(s_2, t_2) \in R_2$ and $(t_1, t_2) \in B$.

Simulation

M_2 simulates M_1 (denoted $M_1 \preceq M_2$) if there exists a Simulation relation H over M_1 and M_2 which satisfy: for every $s_{01} \in I_1$ there exists $s_{02} \in I_2$ such that $(s_{01}, s_{02}) \in H$.

Simulation

Theorem: $M_1 \preceq M_2$ if and only if for every formula ψ in ACTL*, $M_2 \models \psi \Rightarrow M_1 \models \psi$.

Theorem: $\preceq$ is a preorder over Kripke structures.

existential abstraction

Given a concrete structure $M =< S, I, R, L >$ we determine a subset $\hat{A}_t \subseteq A_t$ of "interesting" atomic formulas. Then, we partition S into sets of states, (we denote $[s]$ the set of s) such that every two states s_1 and s_2 of the same subset satisfy $L(s_1) \cap \hat{A}_t = L(s_2) \cap \hat{A}_t$ (notice that there could be many such partitions).
Based on the partition we define an abstract structure $A =< \hat{S}, \hat{I}, \hat{R}, \hat{L} >$:

- $\hat{S}$ is the set of sets of states.
- $\hat{s} \in \hat{I}$ if and only if there exist $s \in \hat{s}$ such that $s \in I$.
- $(\hat{s_1}, \hat{s_2}) \in \hat{R}$ if and only if there are $s_1 \in \hat{s_1}$ and $s_2 \in \hat{s_2}$ such that $(s_1, s_2) \in R$.
- $\hat{L}(\hat{s}) = \bigcup_{s \in \hat{s}} L(s) \cap \hat{A}_t$.

existential abstraction

Theorem: Given a partition, let A be the abstract structure that depend on the partition, then $M \preceq A$.

existential abstraction

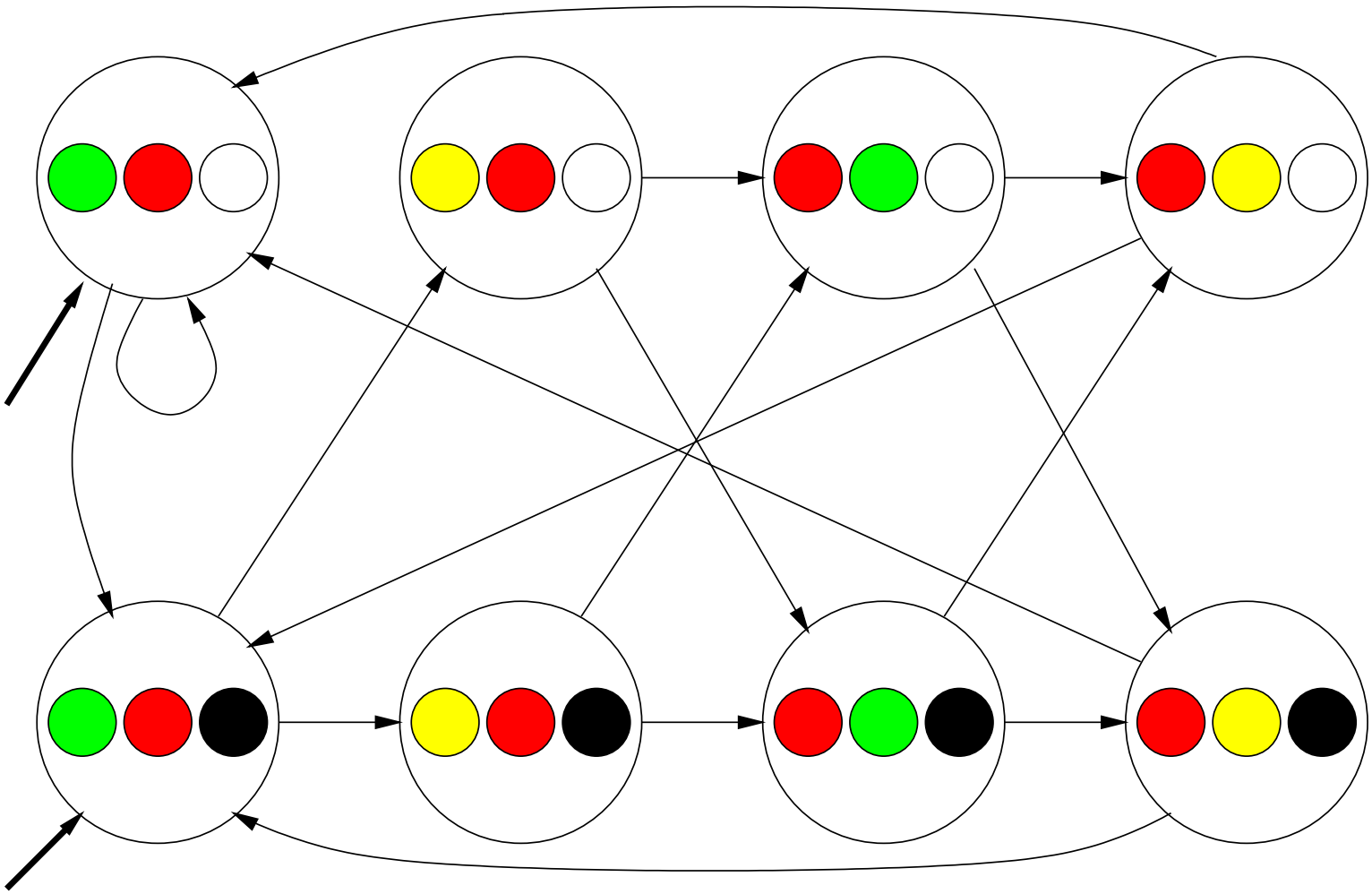
Proof: First we will show that the relation $H = \{(s, \hat{s}) | \hat{s} = [s]\}$ over M and A is a simulation relation.
Let $(s, [s])$ be an element of H then:

- $L(s) \cap \hat{A}_t = \hat{L}([s])$.
- Let t be a state such that $(s, t) \in R$. By the definition of $\hat{R}$, $([s], [t]) \in \hat{R}$ and by the definition of H, $(t, [t]) \in H$.

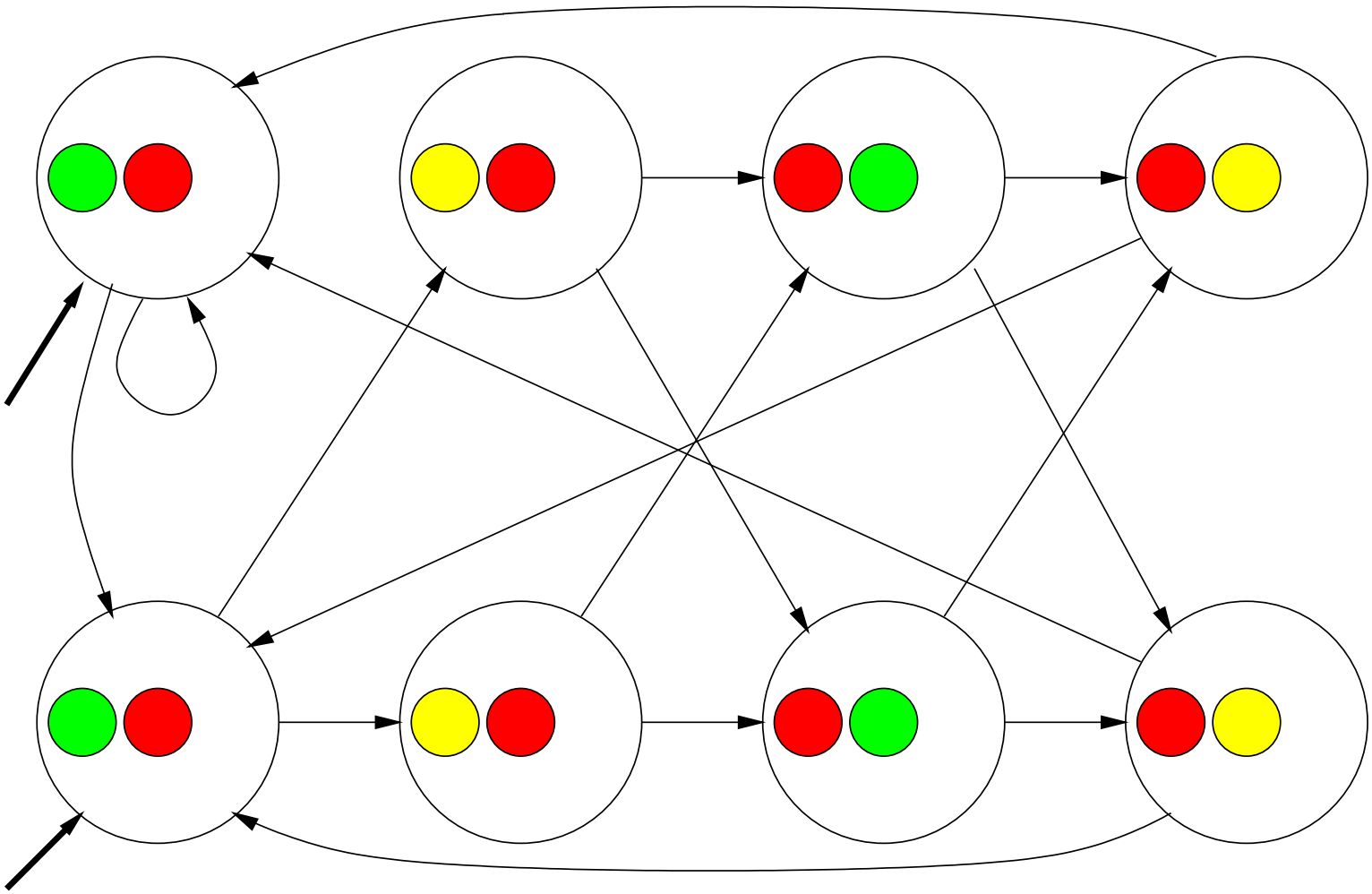
existential abstraction

Next, we show that for every initial state of M there exists a matching initial state in A.
Let s_0 be an initial state of M. By the definition of $\hat{I}$, $[s_0] \in \hat{I}$ and by the definition of H, $(s_0, [s_0]) \in H$.

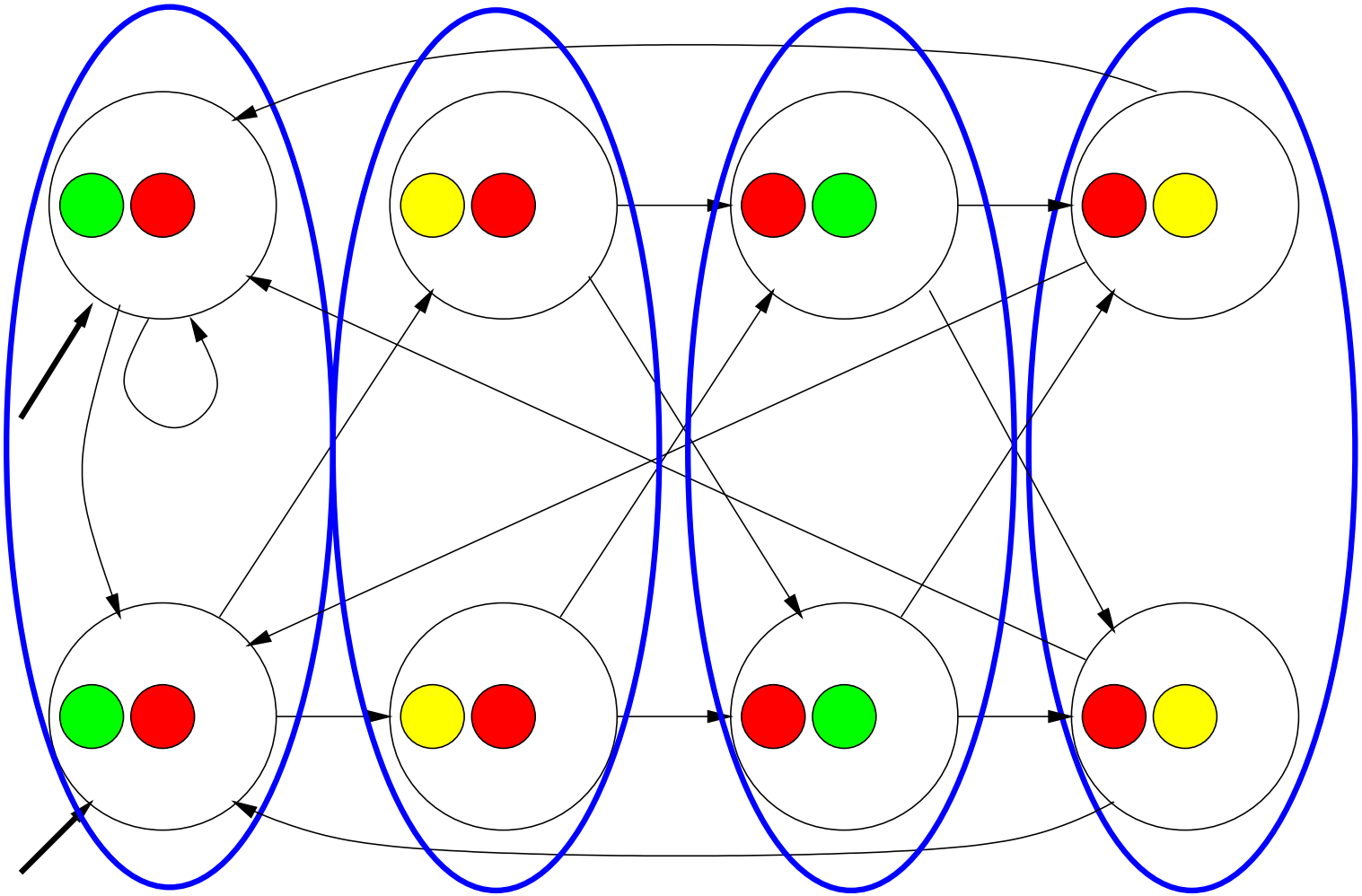
Example



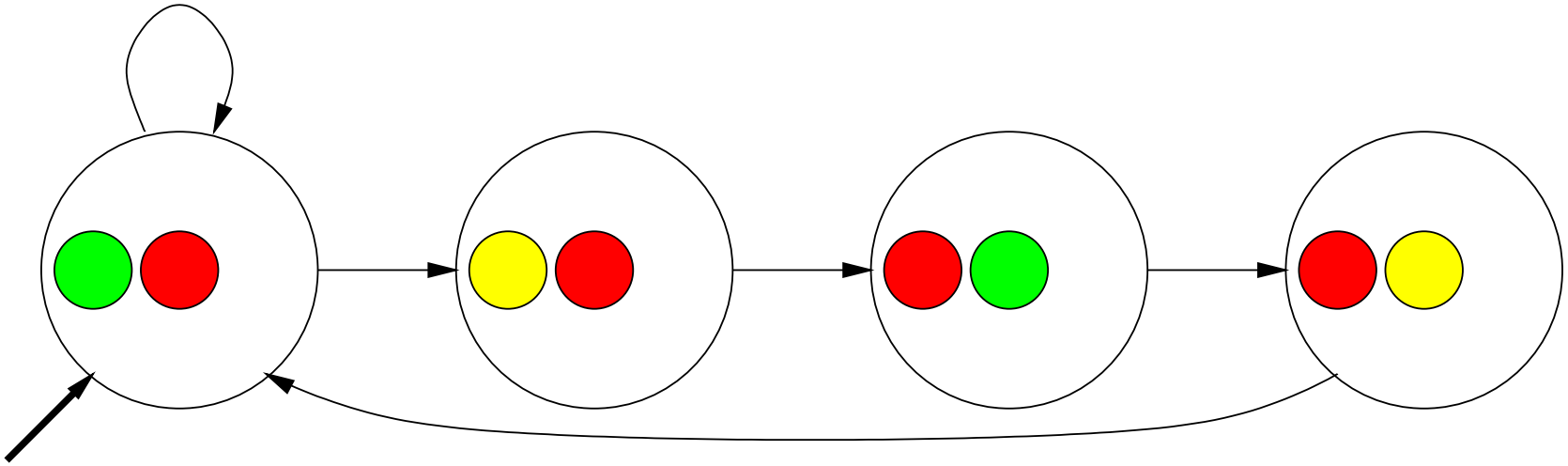
Example



Example

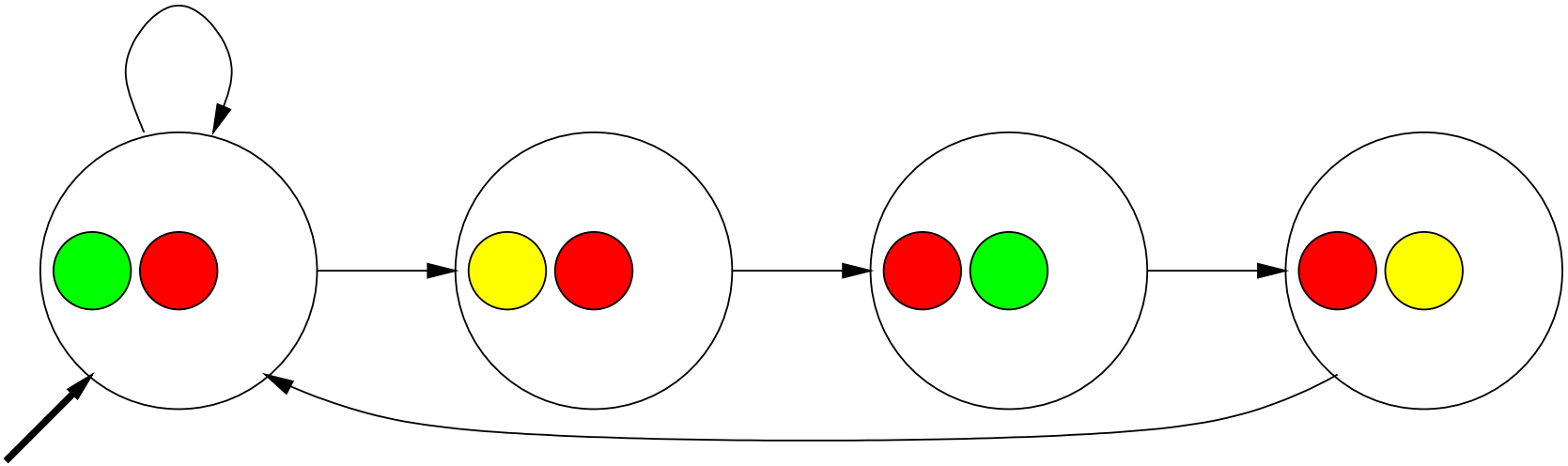


Example



Example

$$\mathbf{A}\,\mathbf{G}((\mathbf{A}\,\mathbf{X}\,t_1 = G) \vee (\mathbf{A}\,\mathbf{X}\,t_1 = Y) \vee (\mathbf{A}\,\mathbf{X}\,t_1 = R))$$



Being consistent with the thesis

- A partition can be represented by a surjection $h : S \to \hat{S}$ where two states s_1 and s_2 are in the same set if and only if $h(s_1) = h(s_2)$.
- The atomic sub-formulas A_ψ of an ACTL*formula ψ respect a partition if for every two state s_1, s_2 of the same set, $L(s_1) \cap A_\psi = L(s_2) \cap A_\psi$.
- Using H instead oh h result in the same partition and hence in the same abstract structure. The only thing that is different is the names of the new state.

Boolean functions

Given a domain D, a boolean function f over D ($f : D \to \{0, 1\}$) maps every element in D either to 0 or to 1.
Operations over boolean functions are defined as follows:

- Let $f = f_1 \cdot f_2$ then for every $d \in D$, $f(d) = f_1(d) \cdot f_2(d)$.
- Let $f = f_1 + f_2$ then for every $d \in D$, $f(d) = f_1(d) + f_2(d)$.
- Let $f = \overline{f_1}$ then for every $d \in D$, $f(d) = 1 - f_1(d)$.

Boolean functions

A boolean function $f : D : \{0, 1\}$ characterizes a subset $D_f \subseteq D$ as follow: for every $d \in D$, $d \in D_f$ if and only if $f(d) = 1$ (note that every subset characterizes a boolean formula) .
Let f_1 and f_2 be boolean functions over D then the following holds:

- $D_{f_1 \cdot f_2} = D_{f_1} \cap D_{f_2}$
- $D_{f_1 + f_2} = D_{f_1} \cup D_{f_2}$
- $D_{\overline{f_1}} = D \setminus D_{f_1}$

Boolean functions

In a similar way, boolean functions can characterize relations. Given a relation $R \subseteq D_1 \times D_2$ its characteristic boolean function is
$f_R : D_1 \times D_2 \to \{0, 1\}$ that defined as follow:
$f((d_1, d_2)) = 1 \Leftrightarrow (d_1, d_2) \in R$.
Let f_R be a boolean function over $D_1 \times D_2$, we define the following operations:

- Let $f = \exists d_1(f_R)$ then $f : D_2 \to \{0, 1\}$ is defined as follows:
  $f(d_2) = 1 \Leftrightarrow \exists d_1, f_R((d_1, d_2) = 1)$.

- Let $f = \forall d_1(f_R)$ then $f : D_2 \to \{0, 1\}$ is defined as follows:
  $f(d_2) = 1 \Leftrightarrow \forall d_1, f_R((d_1, d_2) = 1)$.

Using boolean functions to represent a Kripke structure

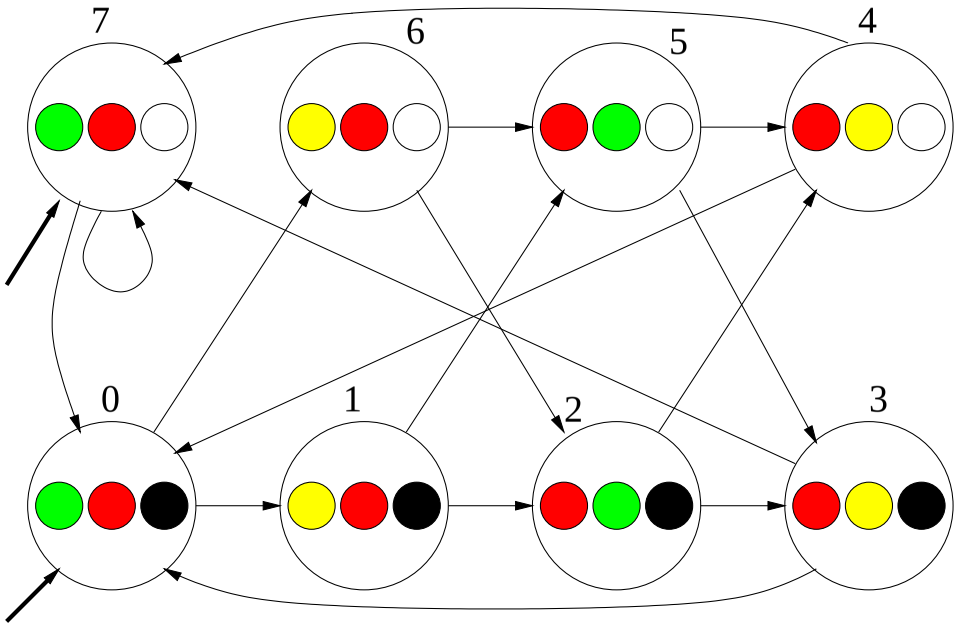
Given a Kripke structure $M =< S, I, R, L >$, we represent M as boolean functions over S.

- f_I is simply the function that characterizes I.
- $f_R : S \times S \rightarrow$ is the function that represents R.
- For each atomic formula ϕ we define a boolean formula f_ϕ such that $f_\phi(s) = 1 \Leftrightarrow s \models \phi$.

Example

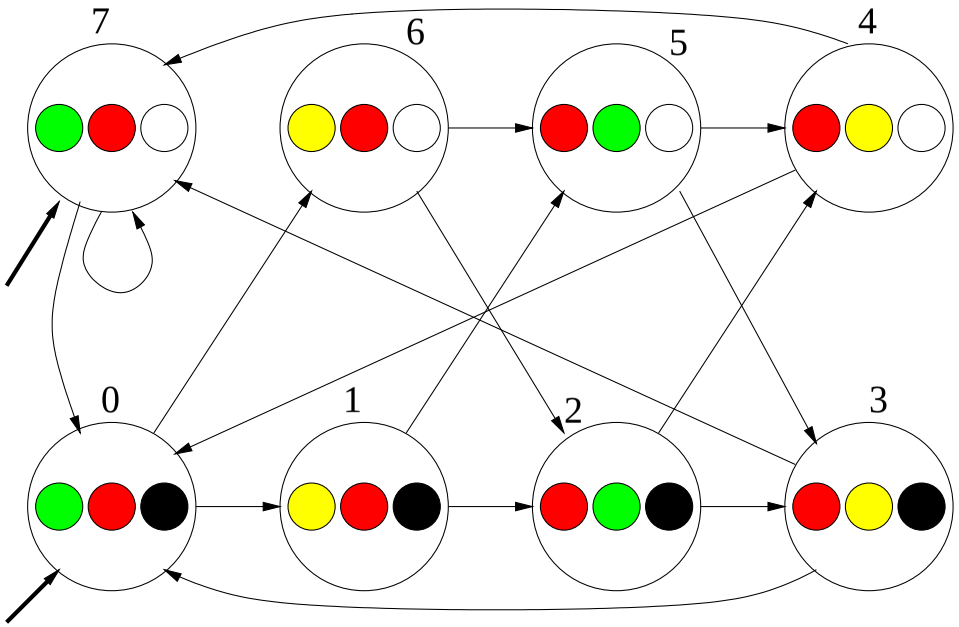
The domain of the functions is $\{0-7\}$

- $f_I(s) = 1 \Leftrightarrow s \in \{0, 7\}$.
- $f_R((s_1, s_2)) = 1 \Leftrightarrow (s_1, s_2) \in \{(0,1), (0,6), (1,2), (1,5), (2,3), (2,4), (3,0), (3,7), (4,0), (4,7), (5,4), (5,3), (6,5), (6,2), (7,7), (7,0)\}$



Example

Let us denote G_1 for $t_1 = G$, R_1 for $t_1 = R$, Y_2 for $t_2 = Y$ etc. Then L' is represented by 7 functions:
$f_{G_1} = \{0, 7\}$. $f_{Y_1} = \{1, 6\}$. $f_{R_1} = \{2, 3, 4, 5\}$. $f_{G_2} = \{2, 5\}$. $f_{Y_2} = \{3, 4\}$.
$f_{R_2} = \{0, 1, 6, 7\}$. $f_{sens} = \{0, 1, 2, 3\}$.



Symbolic model checking

Assume that we want to verify that $M \models \mathbf{A}\,\mathbf{G}((\mathbf{A}\,\mathbf{X}\,G_1) \vee (\mathbf{A}\,\mathbf{X}\,Y_1) \vee (\mathbf{A}\,\mathbf{X}\,R_1))$. We suggest the following algorithm :

1. Construct the set $S_{G_1} = \{s | s \models G_1\}$.
2. Construct the set $S_{\mathbf{A}\,\mathbf{X}\,G_1}$ that contains all states for which all their successors are in S_{G_1}.
3. Similarly construct the sets $S_{\mathbf{A}\,\mathbf{X}\,Y_1}$ and $S_{\mathbf{A}\,\mathbf{X}\,R_1}$.
4. Construct the set $S' = S_{\mathbf{A}\,\mathbf{X}\,G_1} \cup S_{\mathbf{A}\,\mathbf{X}\,Y_1} \cup S_{\mathbf{A}\,\mathbf{X}\,R_1}$.
5. Construct the set S'' of all the states that are reachable from an initial state. Check whether $S' = S''$.

Symbolic model checking

The same algorithm using boolean functions:

1. S_{G_1} is characterized by f_{G_1}.
2. $S_{\mathbf{A}\,\mathbf{X}\,G_1}$ is characterized by $f_{\mathbf{A}\,\mathbf{X}\,G_1} = \overline{\exists s_2(f_R \cdot \overline{f_{G_1}})}$.
3. $f_{\mathbf{A}\,\mathbf{X}\,Y_1}$ and $f_{\mathbf{A}\,\mathbf{X}\,R_1}$ can be calculated similarly.
4. S' is characterized by $f' = f_{\mathbf{A}\,\mathbf{X}\,G_1} + f_{\mathbf{A}\,\mathbf{X}\,Y_1} + f_{\mathbf{A}\,\mathbf{X}\,R_1}$
5. S'' is constructed inductively by constructing a sequence $S_0, S_1, \ldots$ where $S_0 = I$ and $S_{i+1} = S_i \cup \{$ successors of $S_i\}$. It easy to see that S_i contains all states which can be reached from an initial state within i steps. The inductive construction stop when $S_i = S_{i+1}$ after at most $|S|$ steps.
   Given a function f_i that characterizes S_i, $f_{i+1} = \exists s_1(f_R \cdot f_i)$ is the function that characterizes the set S_{i+1}.

Symbolic model checking

The advantage of symbolic model checking is that it operates on sets of states and not on each state separately, thus it save space and time.

Is it really?

That depends on the size of the functions representation and the complexity of the boolean operation.

Bad news: In general, for every type of representation there are sets (most of them) for which we cannot perform efficient boolean operations.

Why am I westing your time?

Symbolic model checking

We are looking for a representation of boolean functions that enables efficient operations in practical verification.

BDDs

Given a domain D we encode the elements of D by $n = log_2(|D|)$ boolean variables $v_0, v_1, \ldots, v_{n-1}$. Thus a boolean function $f : D \to \{0,1\}$ is seen as $f : \{0,1\}^n \to \{0,1\}$.
We define the Order Binary Decision Tree (OBDT) of f as follows: The BDT is a full binary tree of hight $n+1$. The nodes of the bottom level are *Terminal nodes* and the other nodes are *Internal nodes*. The internal nodes are labeled by boolean variables, each level has a unique variable v_i that labels its nodes. Every internal node x has two outgoing edges: A 0-edge and a 1-edge, each edge represents an assignment to the variable that labels x.

Thus each path of the tree represents an assignment to the variables $v_0, v_1, ..., v_{n-1}$. In other words each path represents an element $d \in D$. We assign the value $f(d)$ to the terminal node at the end of the path that represents d.

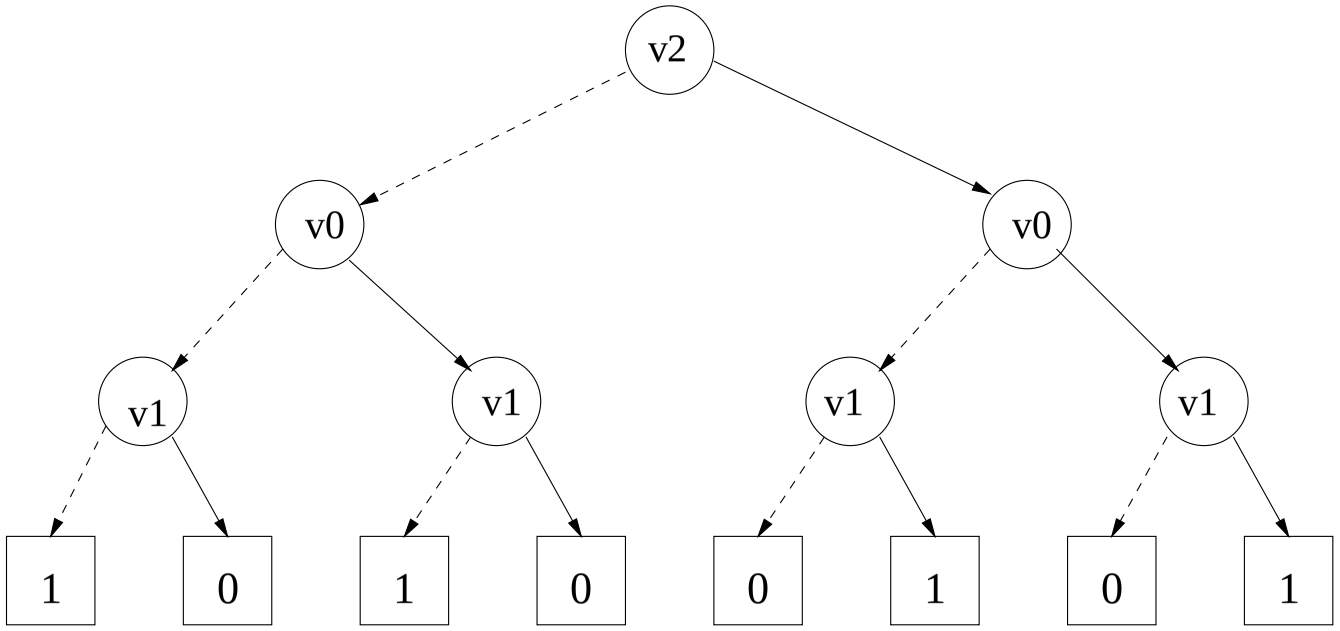
Example for $f_{R_2} = \{0, 1, 6, 7\}$

The domain of f_{R_2} is $\{0 - 7\}$ thus we use v_0, v_1, v_2 to encode it.

The truth table of f_{R_2} is:

v_2	v_1	v_0	f_{R_2}
0	0	0	1
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	1

A OBDT of f_{R_2} is

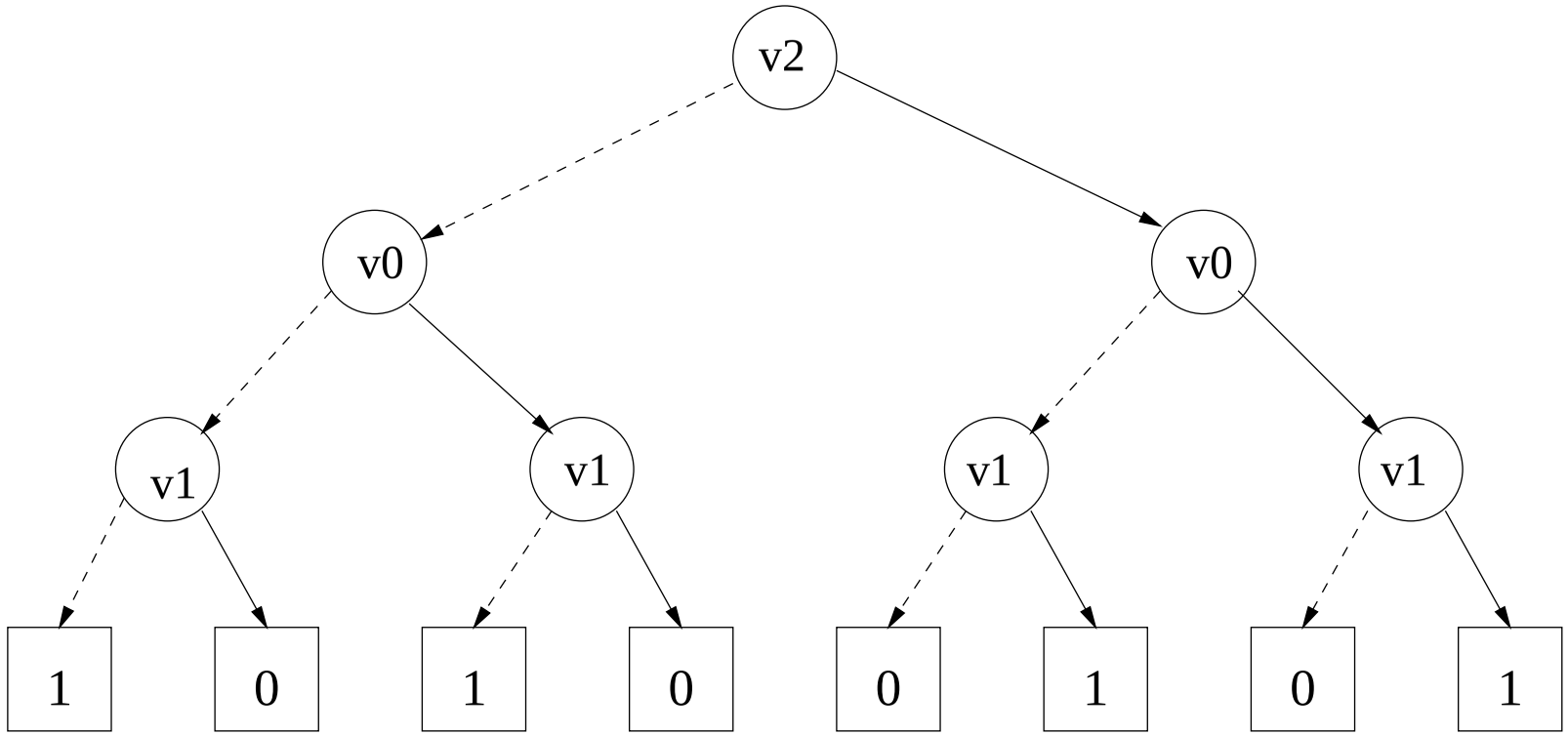


An OBDD

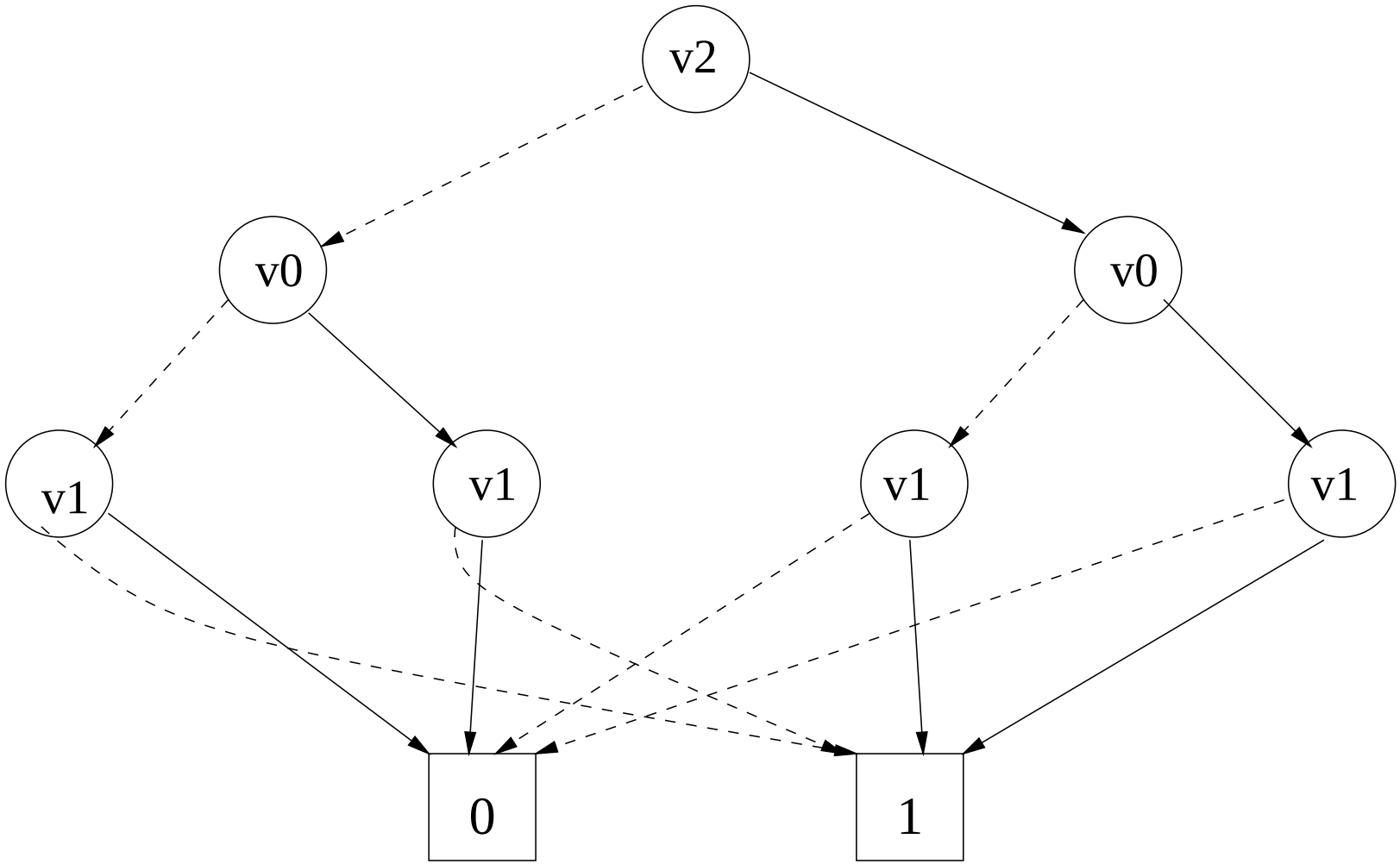
Given a boolean function f, an OBDD of f is the result of performing the following reductions on an OBDT of f:

- Merge all 0−terminal nodes into one 0−terminal node, merge all 1−terminal nodes into one 1−terminal node.
- If two nodes are labeled by the same variable and their subgraphs are isomorphic, merge them into one node.
- If for some node x_1 both edges goes to the same node x_2, then delete x_1 and redirect the edges that had previously pointed to node x_1 to x_2.

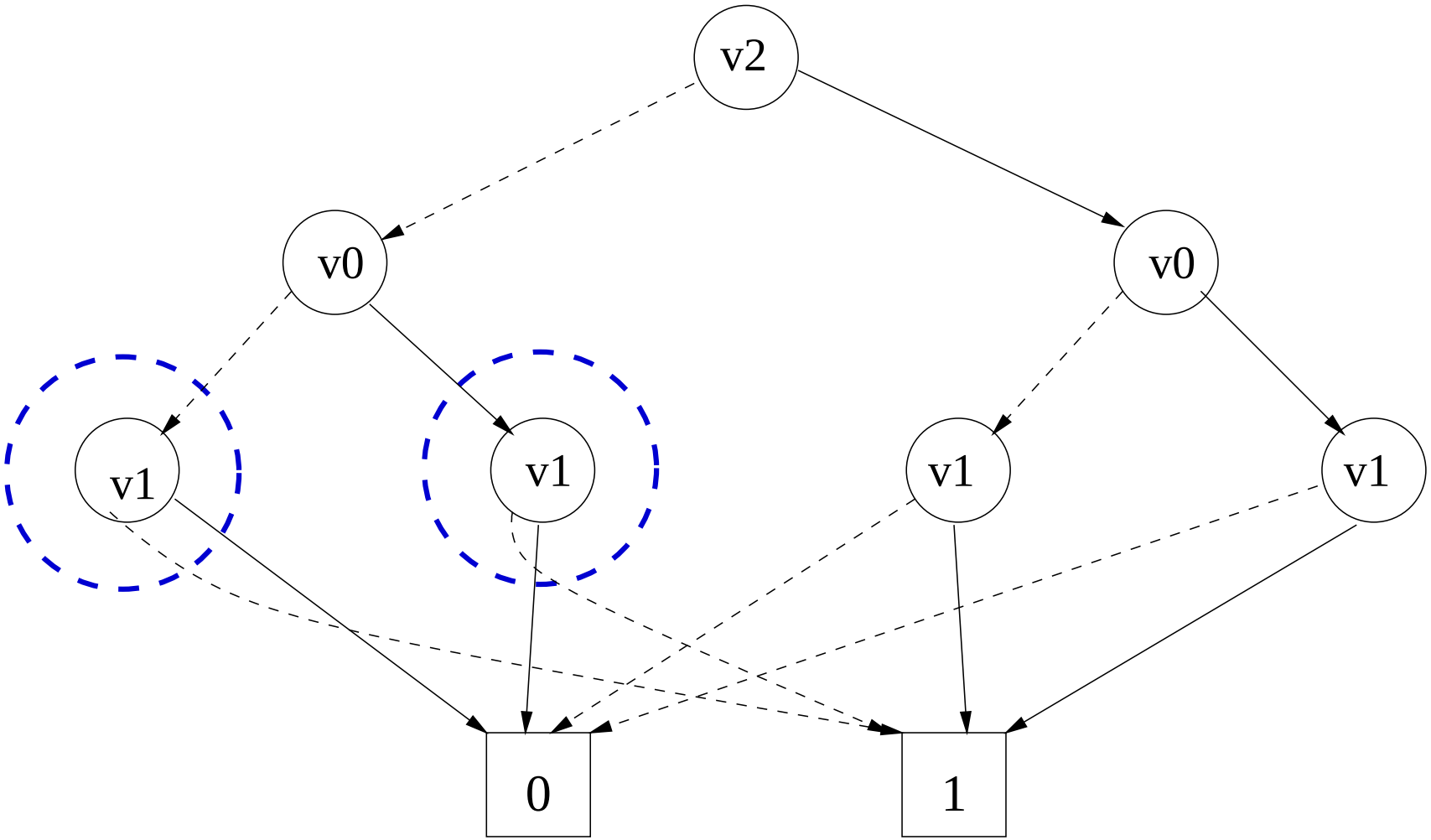
Example



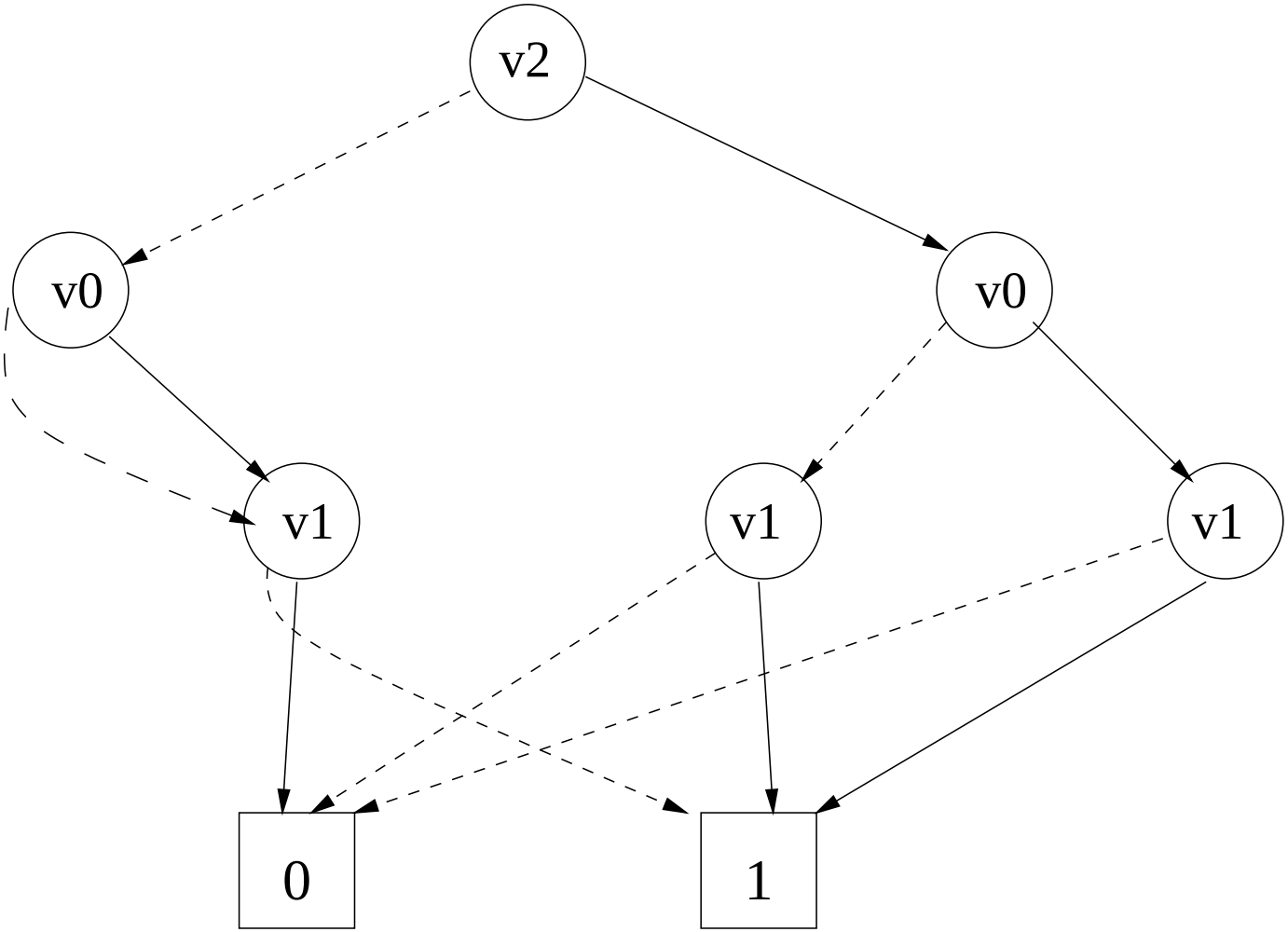
Example



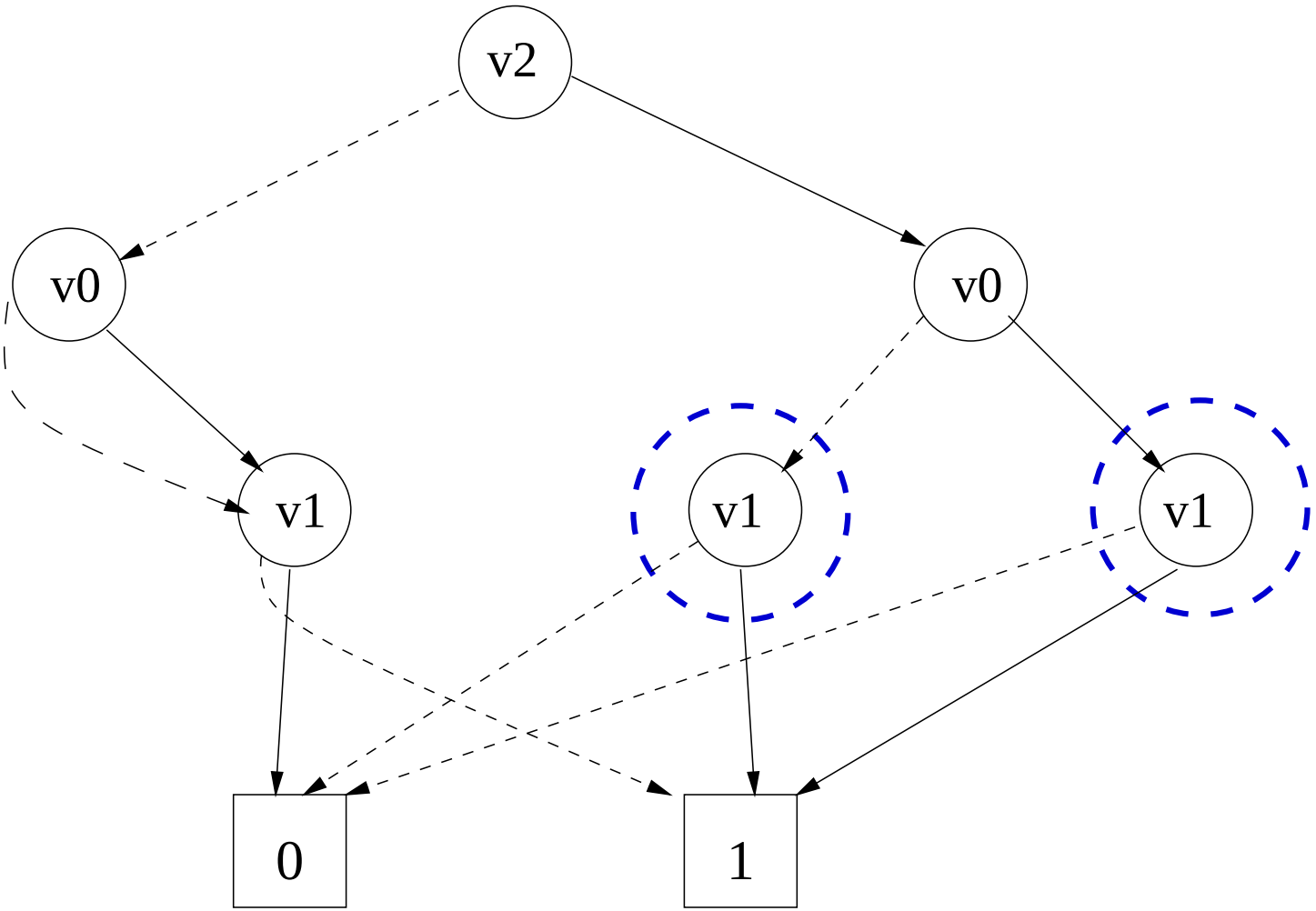
Example



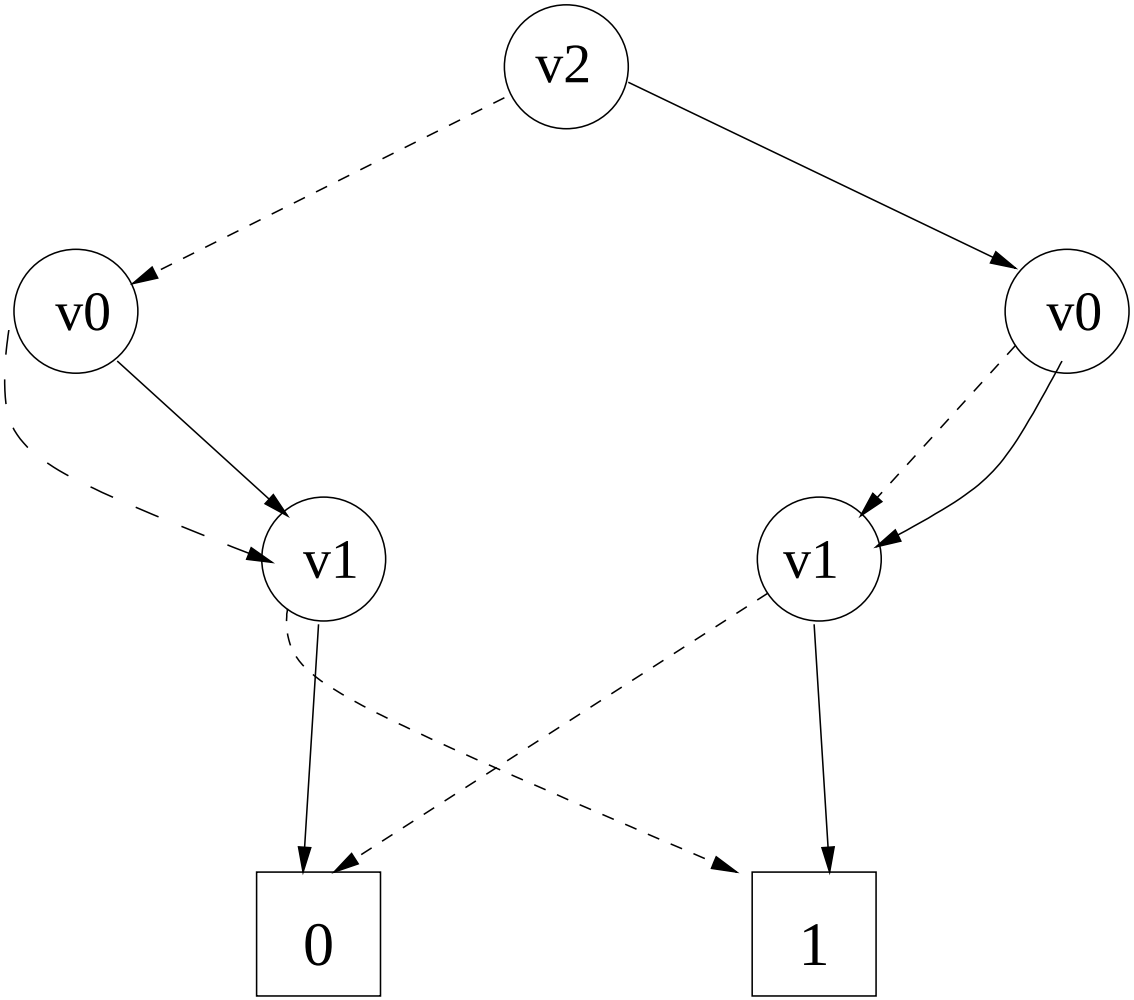
Example



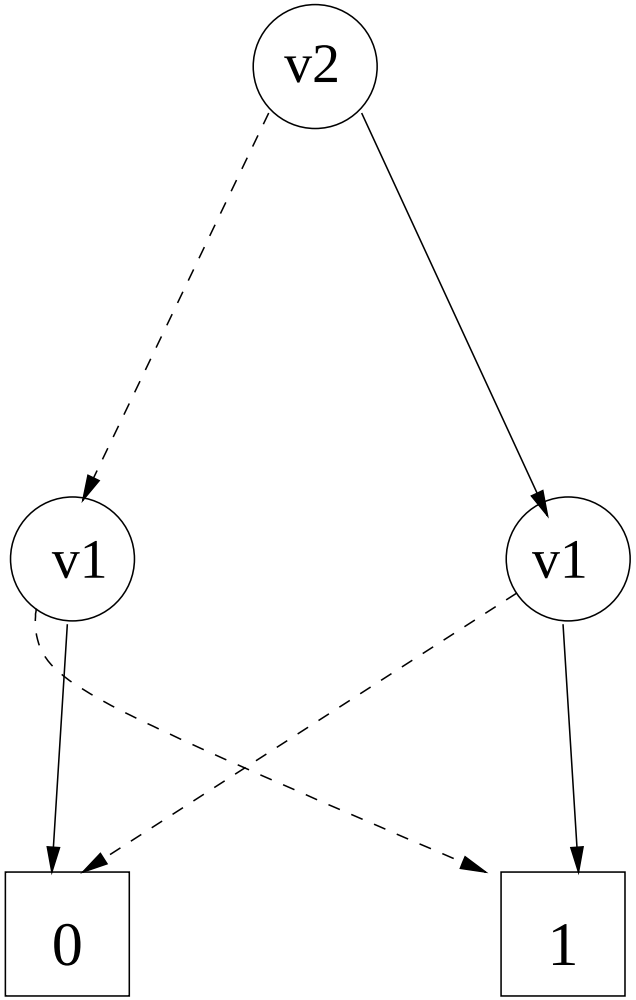
Example



Example



Example



Properties of BDDS

Given two BDDs B_1 and B_2 over the same order, the complexity of boolean operations over the BDDs is:

Operation	time complexity
$B_1 \wedge B_2$	$O(\|B_1\| \cdot \|B_2\|)$
$B_1 \vee B_1$	$O(\|B_1\| \cdot \|B_2\|)$
$\neg B_1$	$O(1)$
$\exists x_1 \ldots \exists x_n B_1$	$O(\|B_1\|^n)$

Properties of BDDS

For a fix order over $v_0, v_1, \ldots, v_{n+1}$, there exists a unique BDD for every boolean formula. Thus we can check whether functions $f_1 = f_2$ by checking whether B_1 is isomorphic to B_2.

Properties of BDDS

The size of the BDD is very sensitive to the order of the variables. For some boolean functions different orders result in exponential difference in the sizes of the BDDs.
Given a boolean formula $f : \{0,1\}^n \rightarrow \{0,1\}$, finding an order that minimizes the size of it's BDD is an NP-hard problem.