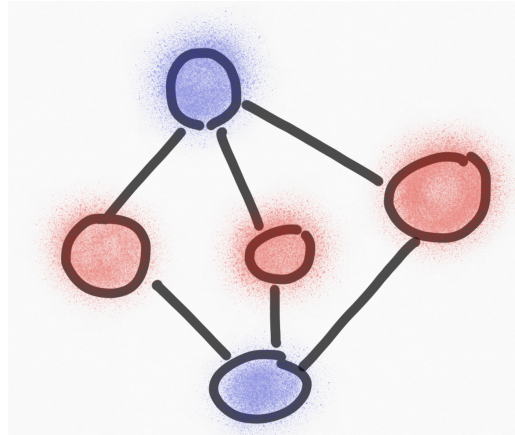
An introduction to descriptive complexity

Étienne Lozes
Dpt Info – UNICE

february, 19th 2018

P and NP

P is the family of decision problems for which a solution can be computed in polynomial time.
Example: two colouring of a graph



NP is the family of decision problems for which a solution can be checked in polynomial time.
Example: three colouring, sudoku

	2		6		8			
3	7					6		8
8		6		7				
4	9	1		5				6
		7				1		
2				4		3	7	5
				9		4		1
1		2					6	3
			5		3		2	

P $\subseteq$ **NP**, but we don't know if this inclusion is strict or not.

Descriptive complexity

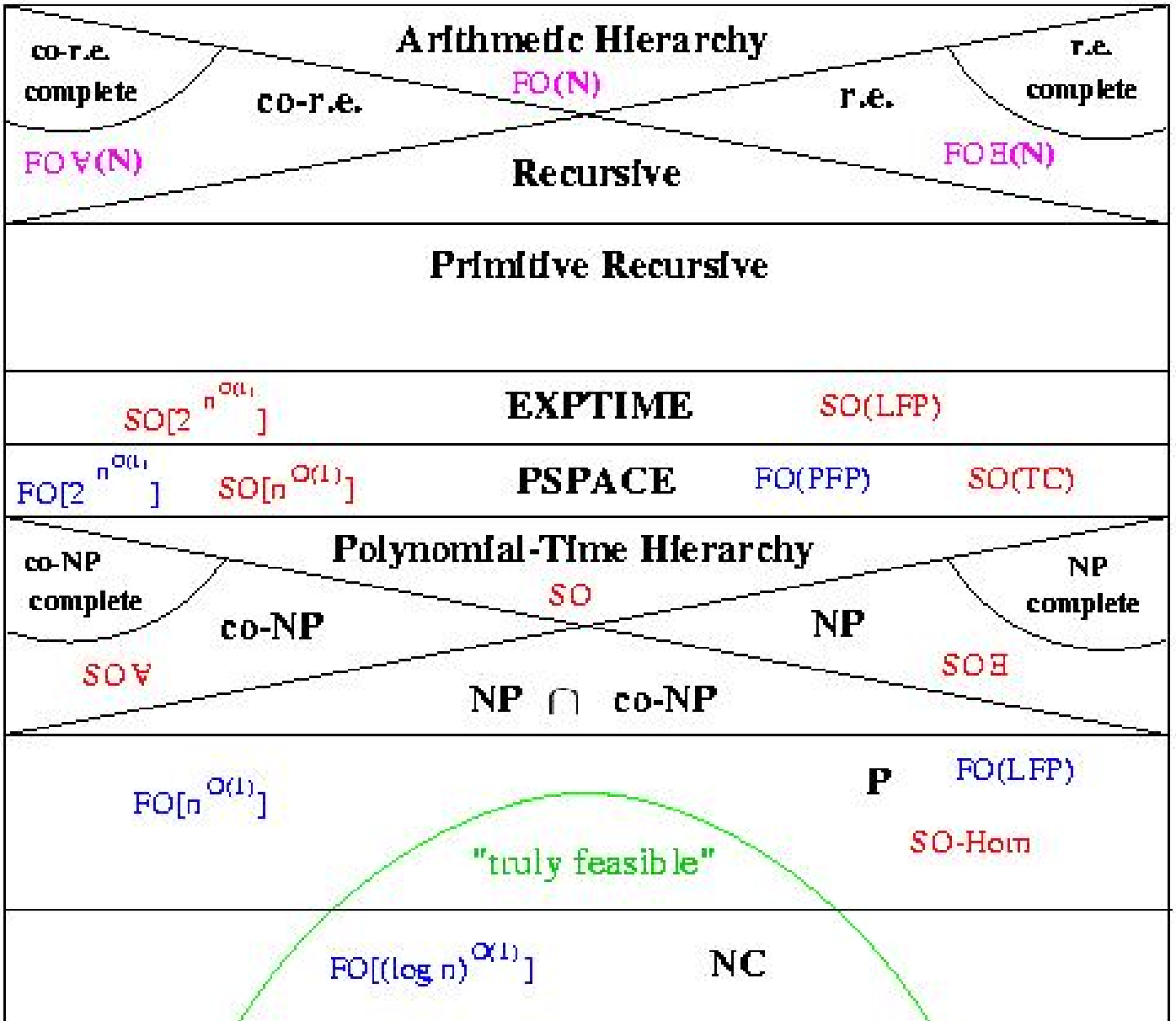
NP is the family of properties that can be expressed in *existential second-order logic* (**ESO**)

On totally ordered structures, **P** is the family of properties that can be expressed in *first order logic* with *least fixed point*.
(**FO(<)+LFP**)

Open problems in descriptive complexity

1. we don't know if the two logics **ESO** and **FO(<)+LFP** are equivalent on ordered structures
2. we don't know a logic that captures P on all structures

More than P and NP



Outline

1. Turing machines and complexity classes
2. First order logic, second order logic, and finite structures
3. Fagin's theorem
4. Logics with fixed points

Turing machine : definition

A Turing machine consists of

- a finite set of states K
- a finite set of symbols Σ, including \$ and $\square$
- an initial state $s_0 \in K$
- disjoint subsets Acc, Rej of K (accepting and rejecting states)
- a transition function δ that specifies for each state and symbol a next state, a symbol to overwrite the current symbol, and a direction for the tape head to move (**L**eft,**R**ight,**S**tay)

$$\delta : (K \times \Sigma) \to K \times \Sigma \times \{L, R, S\}$$

Words, configurations, runs

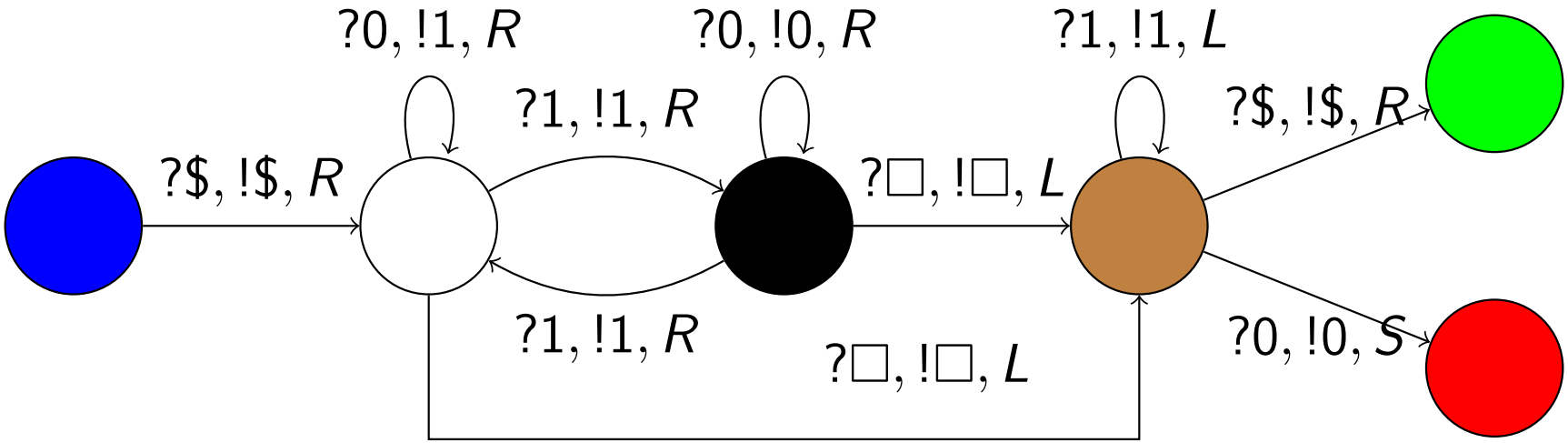
A Σ-word w is a finite sequence of symbols. Σ^* denotes the set of words. $w[i]$ is the symbol at index i of w. Indexes start from 0. $|w|$ denotes the length of w.

A configuration of a Turing machine $\mathcal{M} = (K, \Sigma, s_0, Acc, Rej, \delta)$ is a tuple $\gamma = (s, i, w) \in K \times \mathbb{N} \times \Sigma^*$

A run ρ of $\mathcal{M}$ is a finite or infinite sequence of configurations. ρ is accepting if it is finite and the last configuration is (s, i, w) with $s \in Acc$.

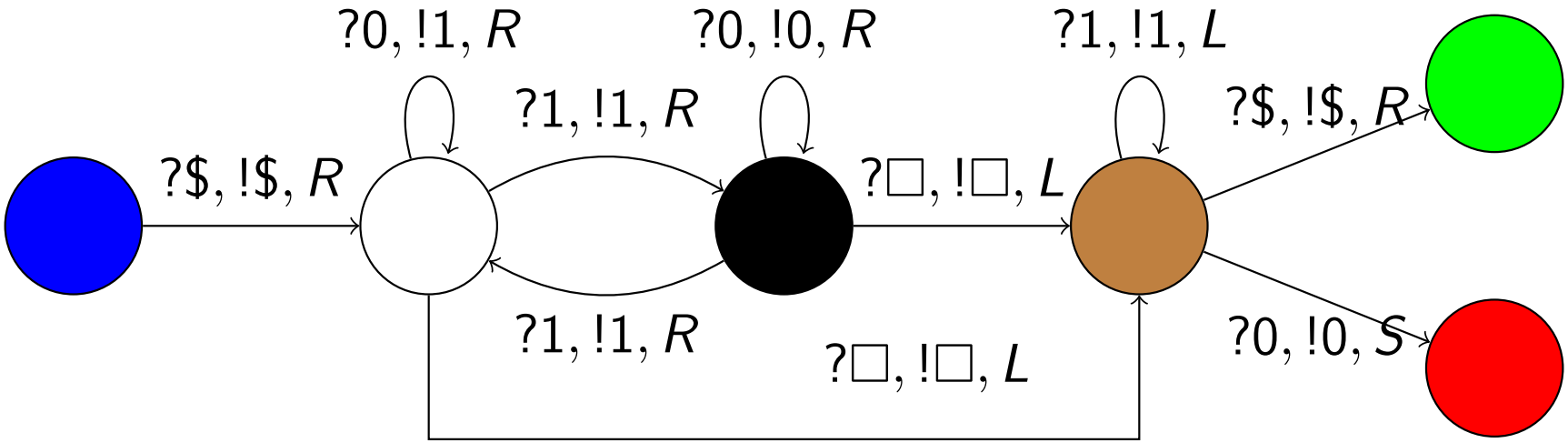
A rejecting run is defined similarly

Turing machine : example



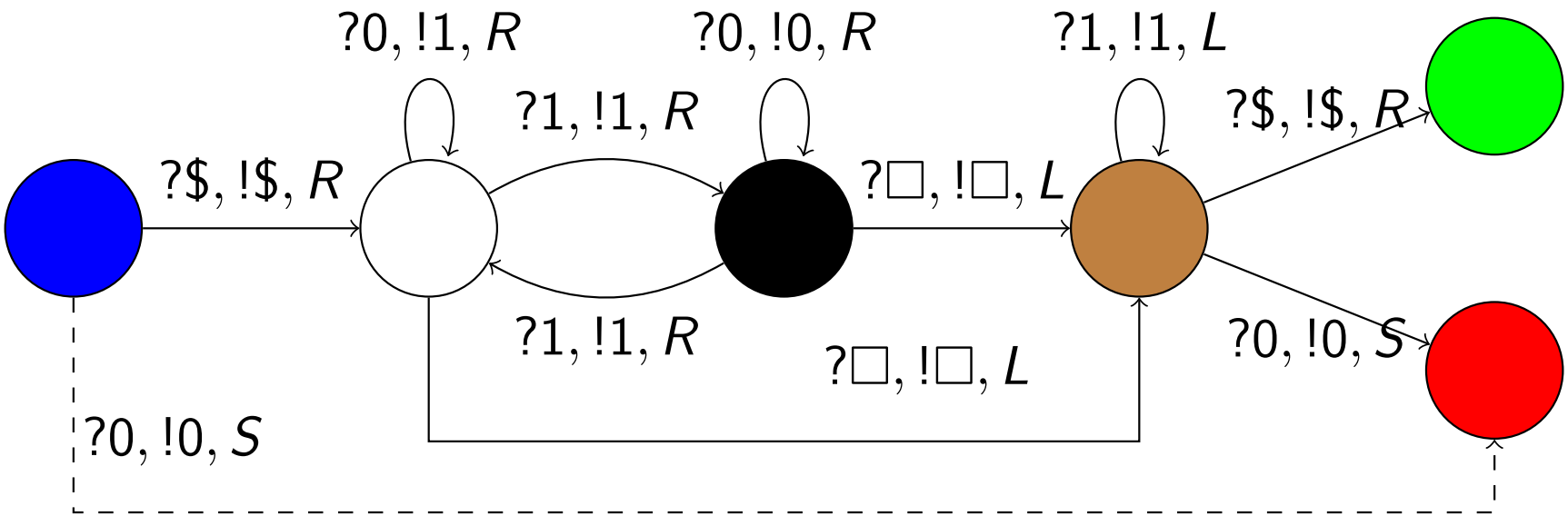
$$
\begin{array}{lcl}
K & = & \{●, ○, ●, ●, ●, ●\} \\
\Sigma & = & \{0, 1, \$, \square\} \\
s_0 & = & ● \\
\text{Acc} & = & \{●\} \\
\text{Rej} & = & \{●\}
\end{array}
$$

Turing machine : example



$$\begin{aligned} \delta(\bullet, \$) &= (\$, \circ, R) \\ \delta(\circ, 0) &= (1, \circ, R) \\ \ldots & \end{aligned}$$

Turing machine : example



$\delta(\bullet, \$) = (\$, \circ, R)$

$\delta(\circ, 0) = (1, \circ, R)$

$\ldots$

Transitions leading to ● are not drawn.

Example : $\delta(\bullet, 0) = (0, \bullet, S)$

One step relation

Small step semantics

$(s, i, w) \to_{\mathcal{M}} (s', i', w')$ if and only if

- $s \in K \setminus (Acc \cup Rej)$
- $\delta(s, w[i]) = (a, s', MV)$
- $i' = \begin{cases} i & \text{if } MV = S \\ i+1 & \text{if } MV = R \\ i-1 & \text{if } MV = L \end{cases}$
- either $0 \leq i' < |w|$ and w' is w where $w[i]$ is replaced with a
- or $i' = |w| + 1$ and w' is $w[0] \cdots w[i-1] \cdot a \cdot \square$

Determinism

There is at most one successor (none if $i' < 0$).

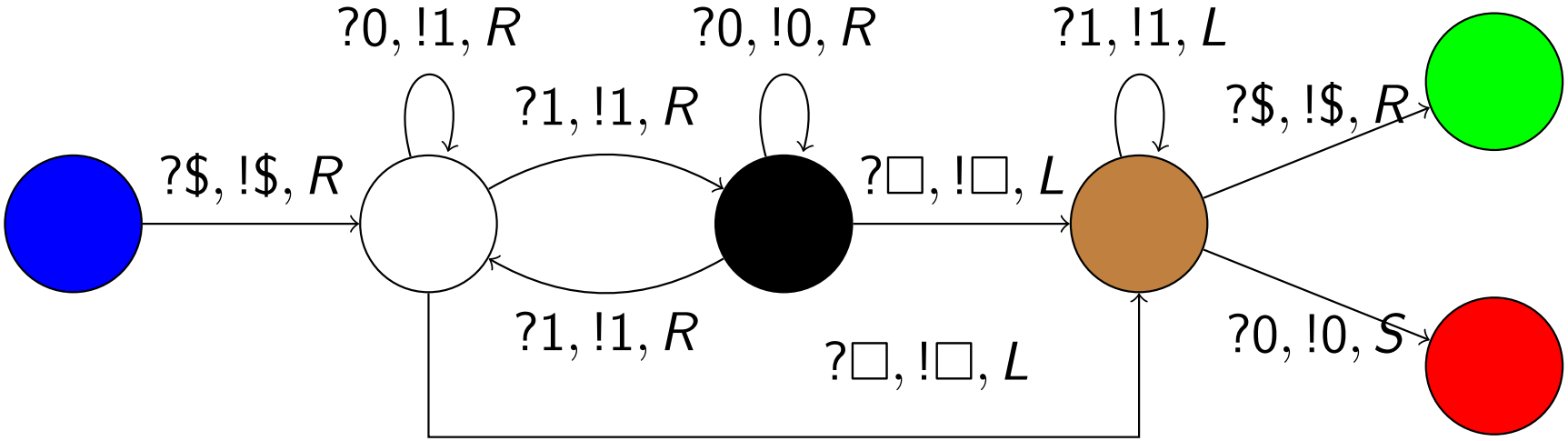
Run

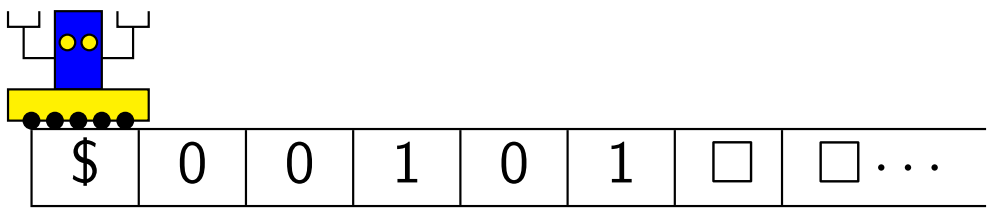
The run of $\mathcal{M} = (K, \Sigma, s_0, Acc, Rej, \delta)$ over the input tape w_0 is the maximal sequence

$$\text{Run}(\mathcal{M}, w_0) = \gamma_0 \to \gamma_1 \to \gamma_2 \to \dots$$

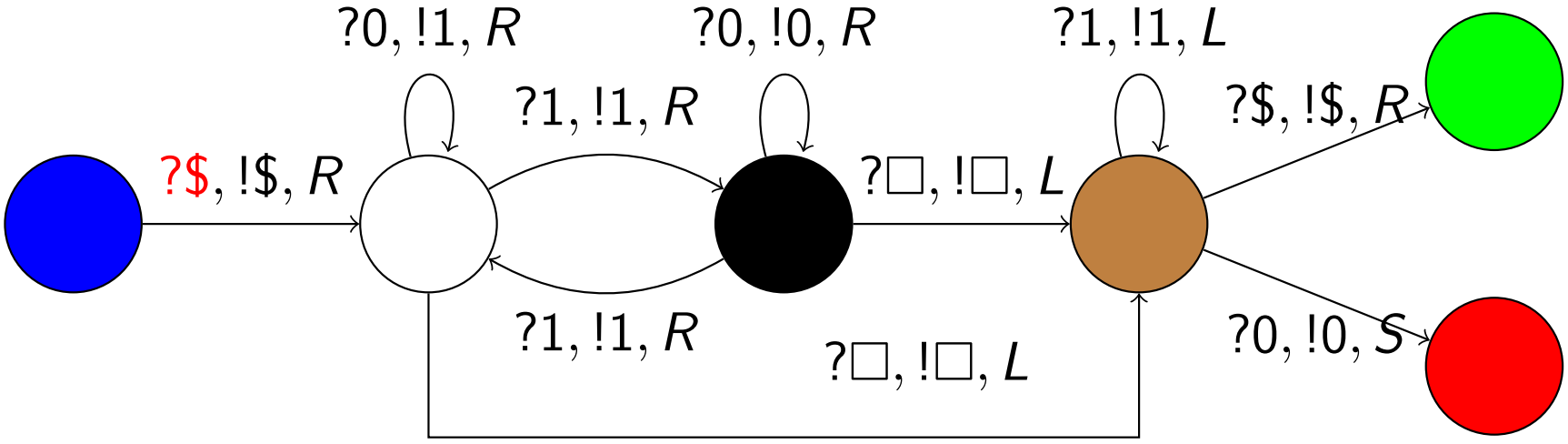
with $\gamma_0 = (s_0, 0, w_0)$.

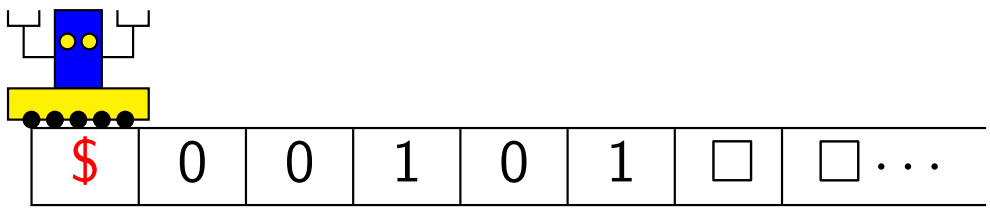
Run of a TM: example



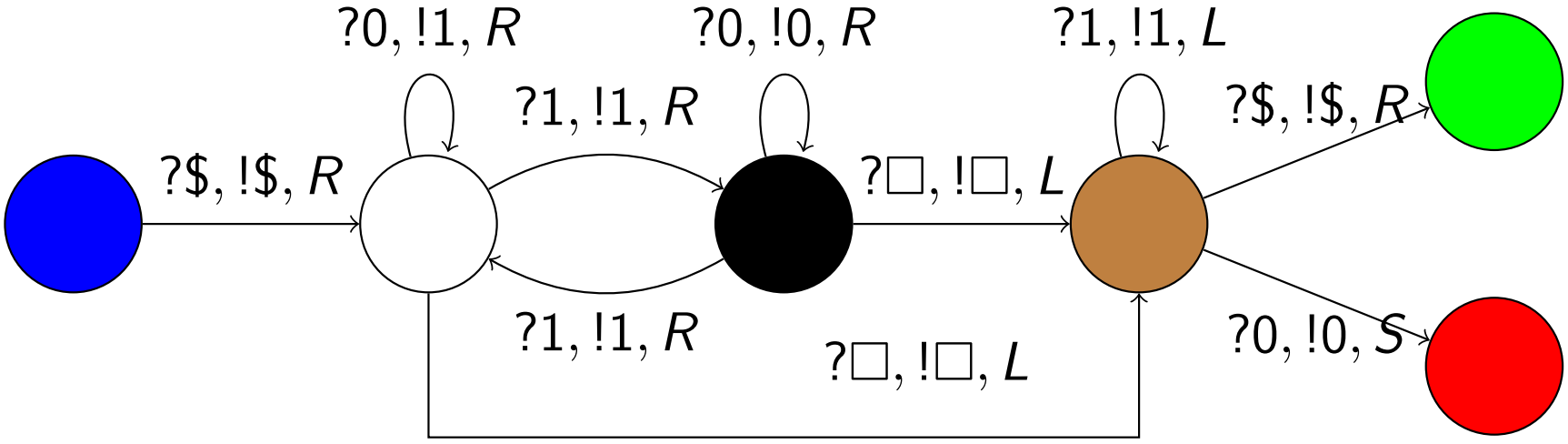


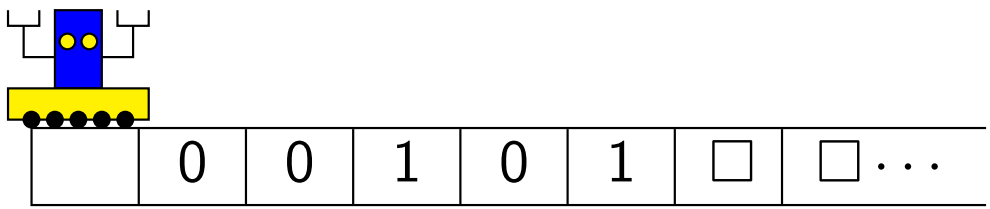
Run of a TM: example



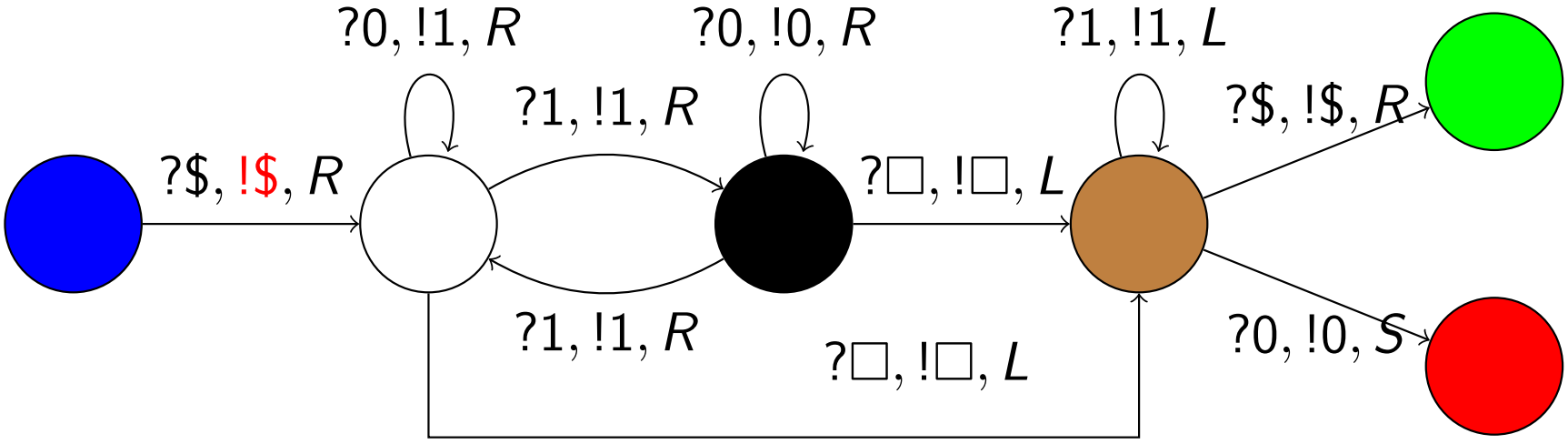


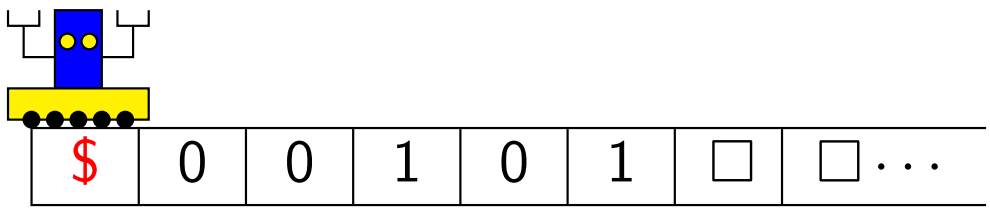
Run of a TM: example



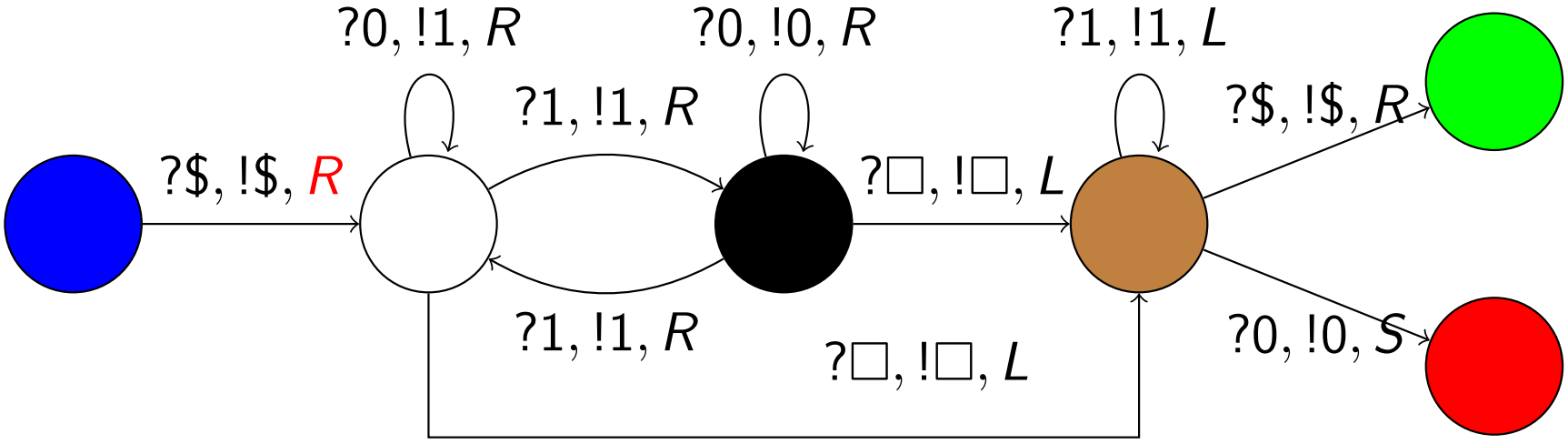


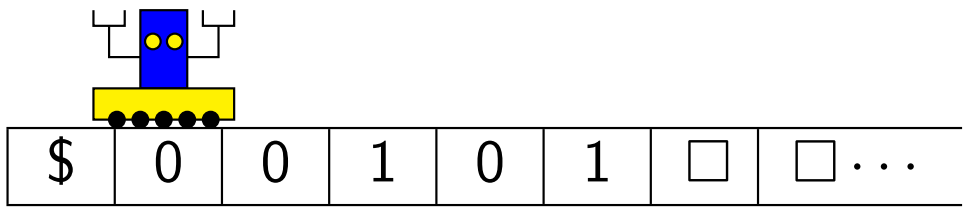
Run of a TM: example



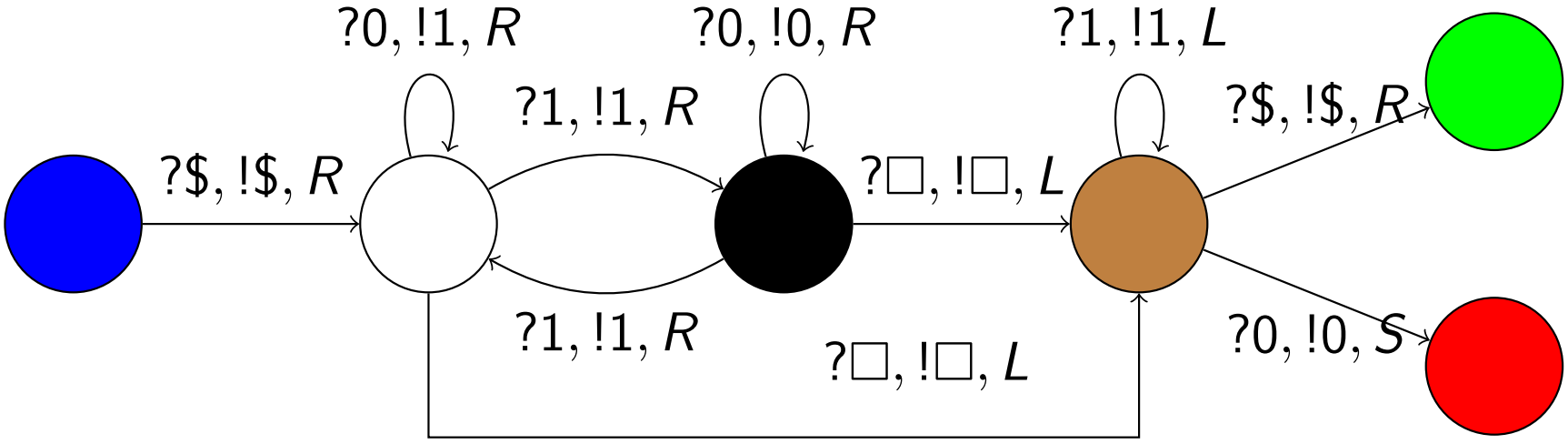


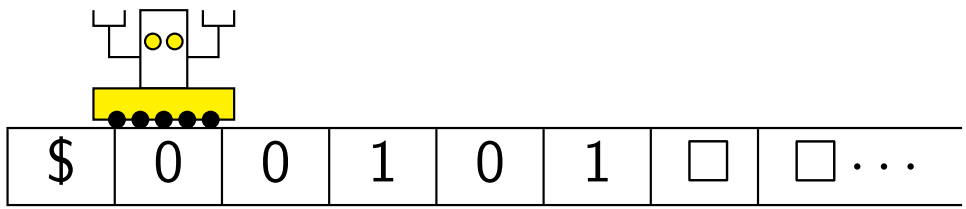
Run of a TM: example



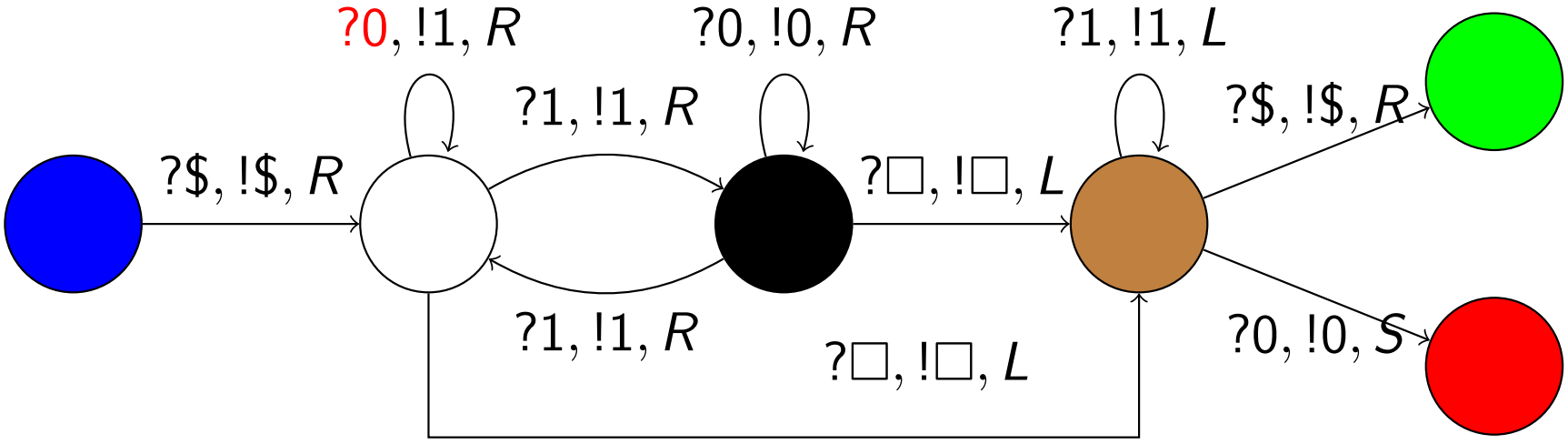


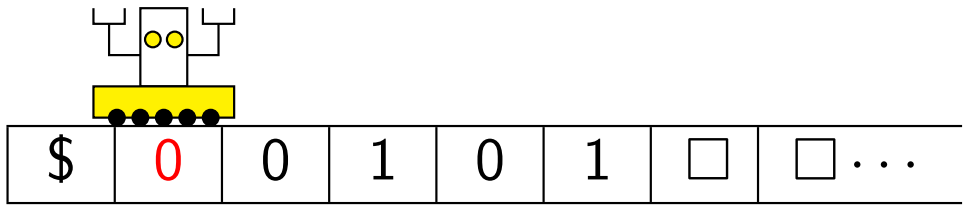
Run of a TM: example



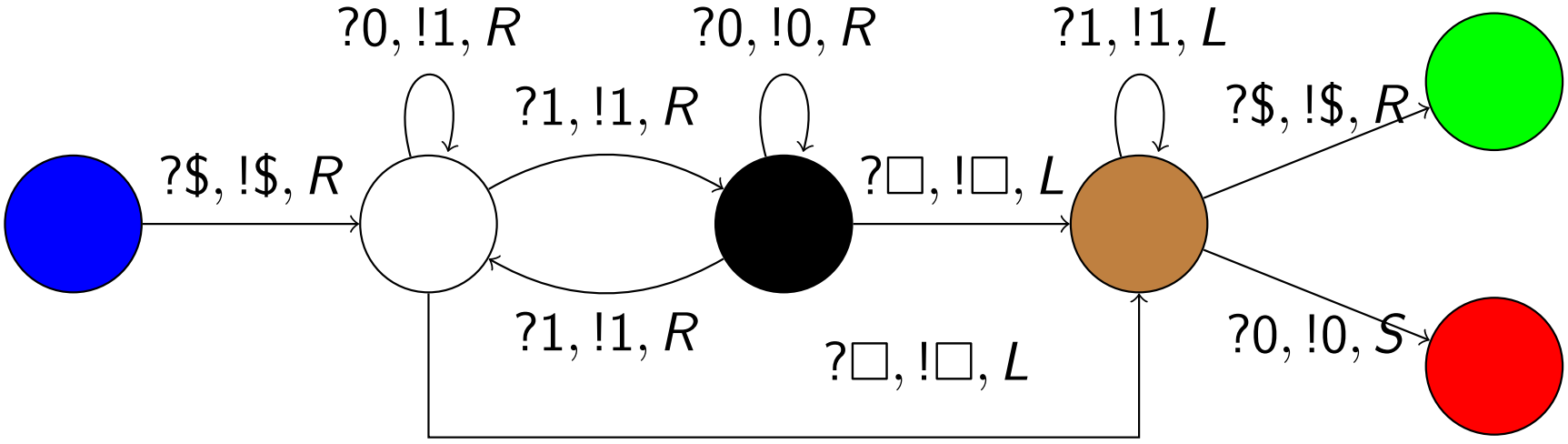


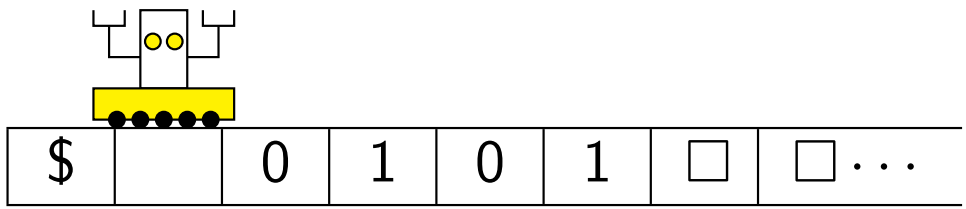
Run of a TM: example



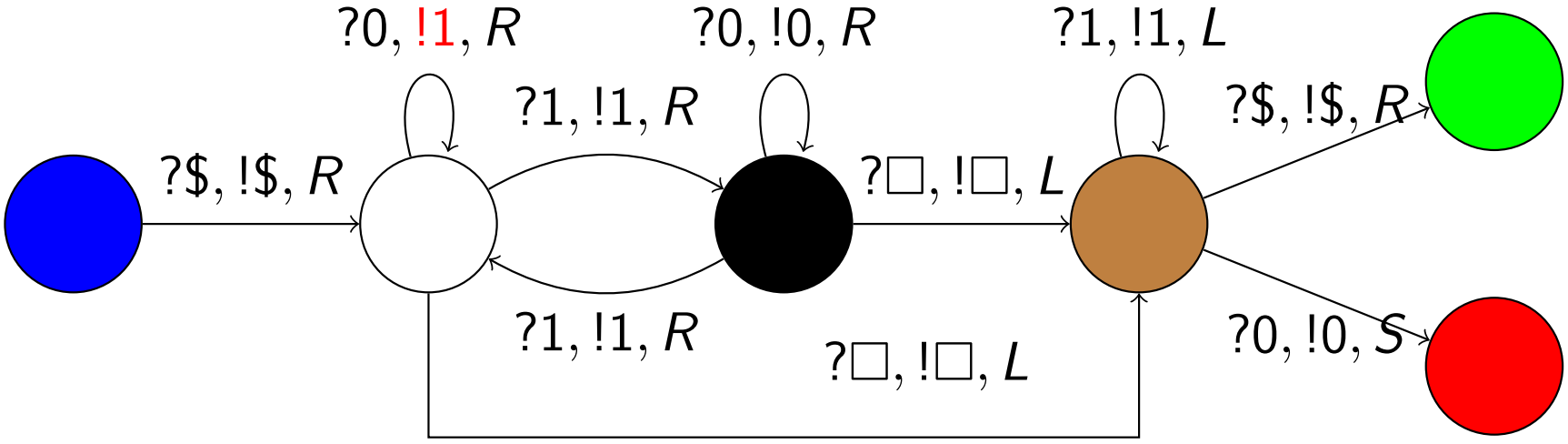


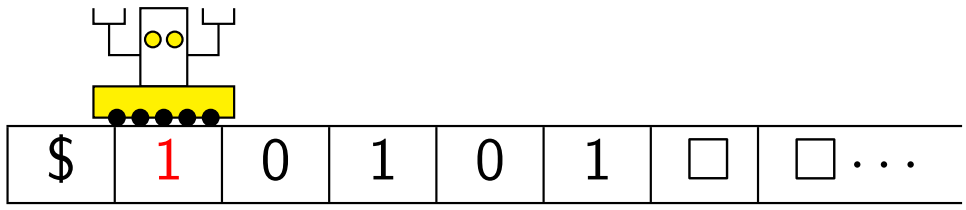
Run of a TM: example



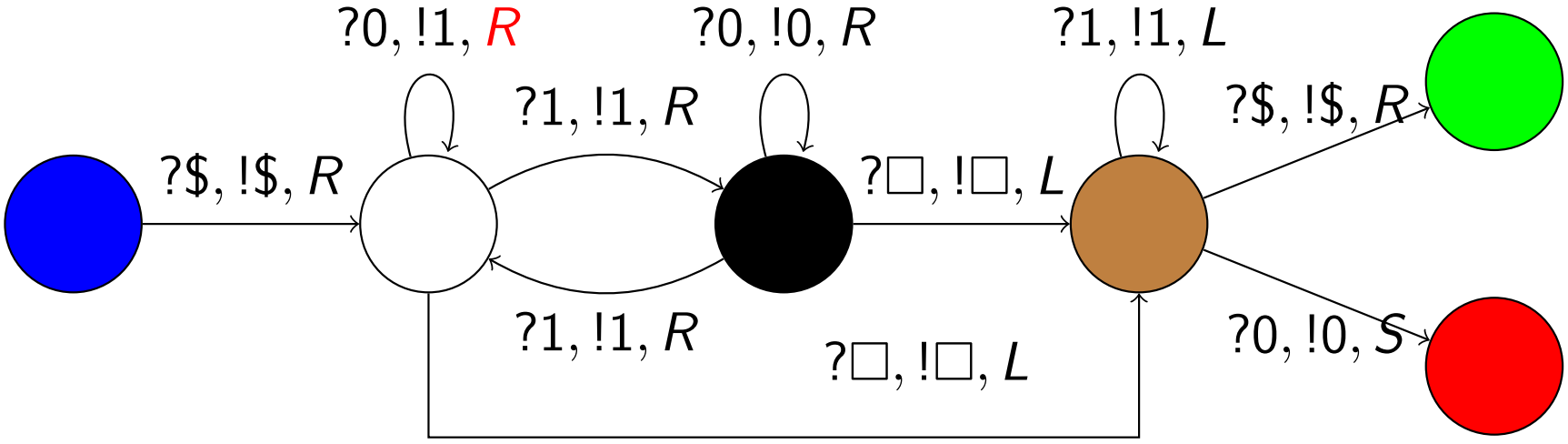


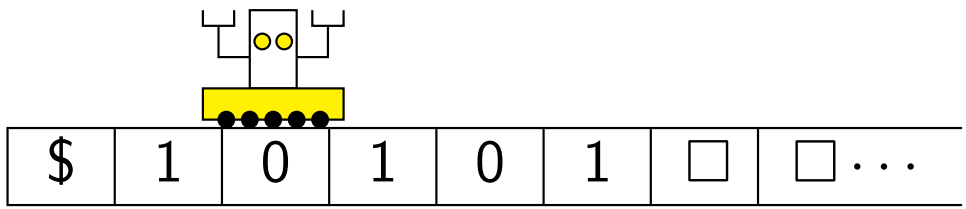
Run of a TM: example



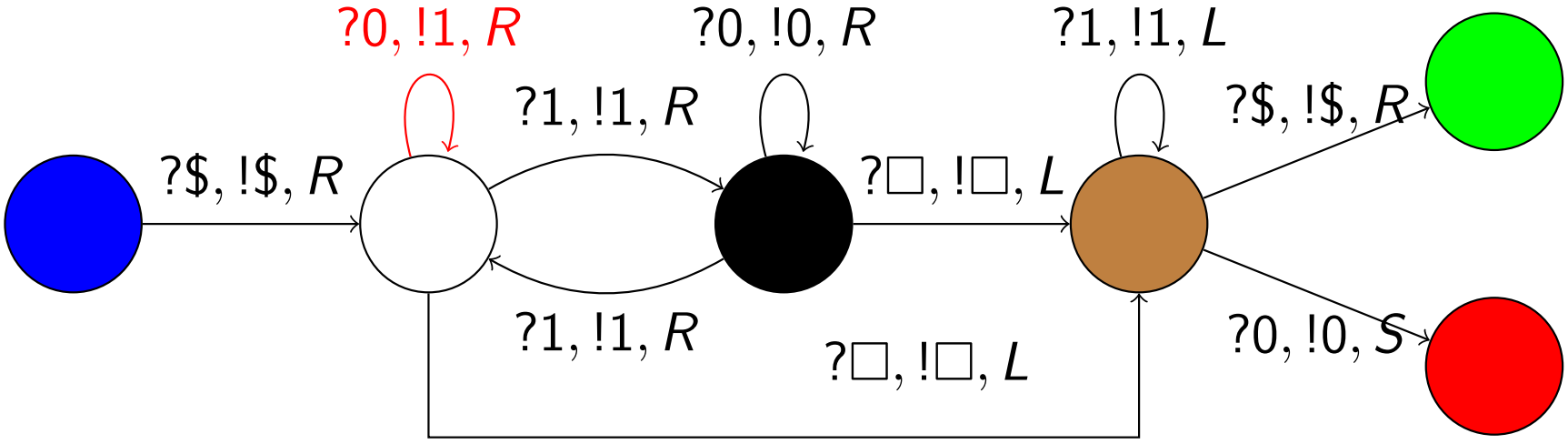


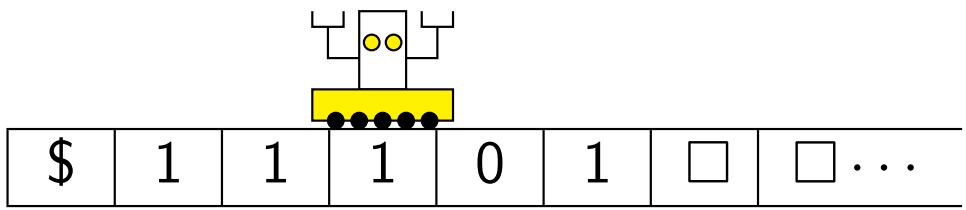
Run of a TM: example



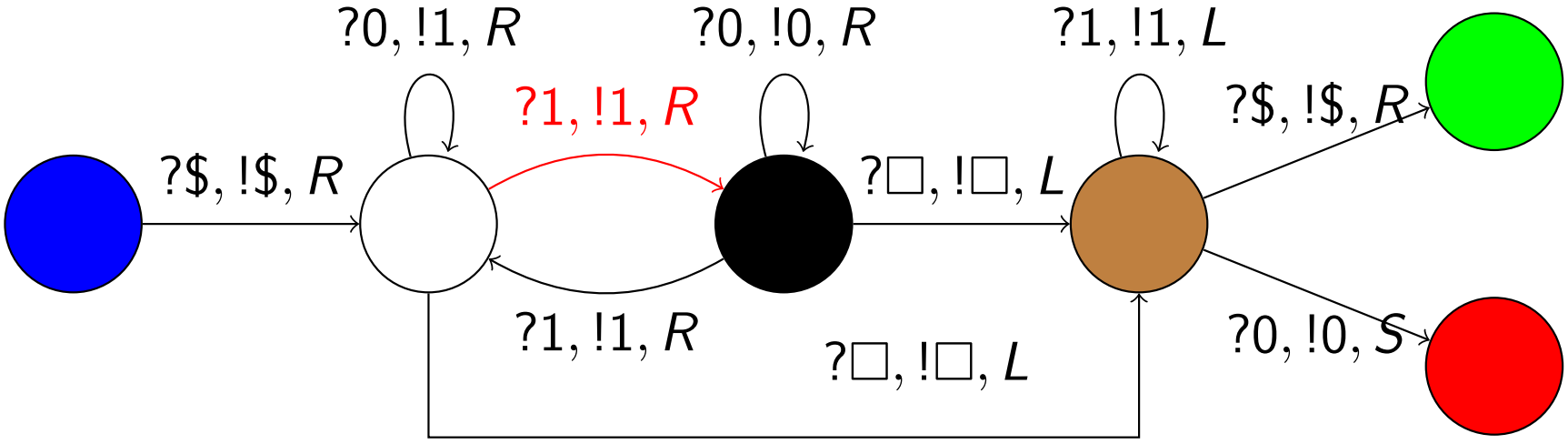


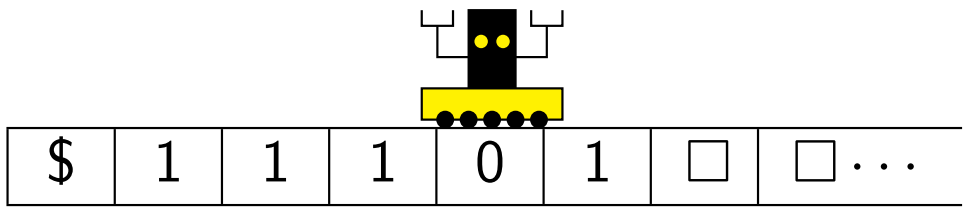
Run of a TM: example



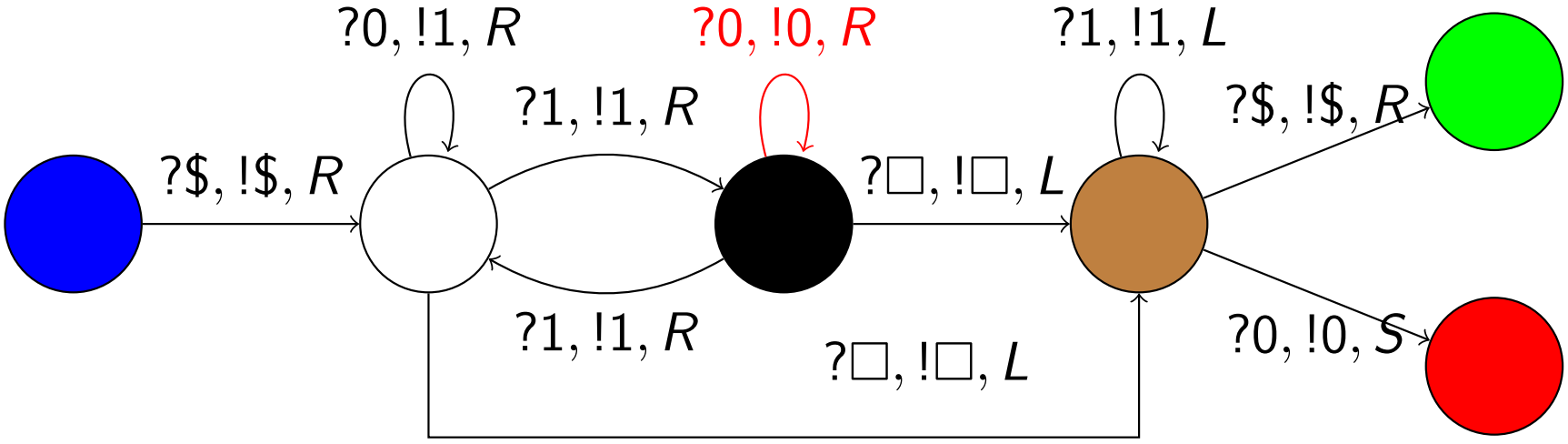


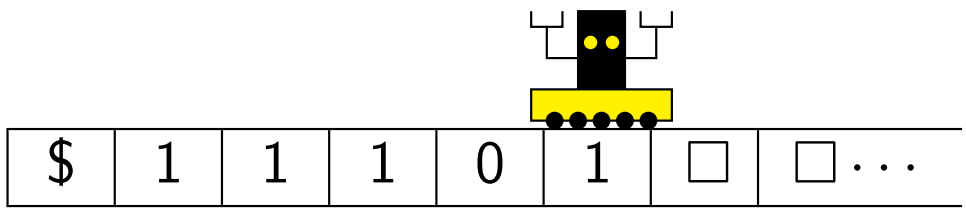
Run of a TM: example



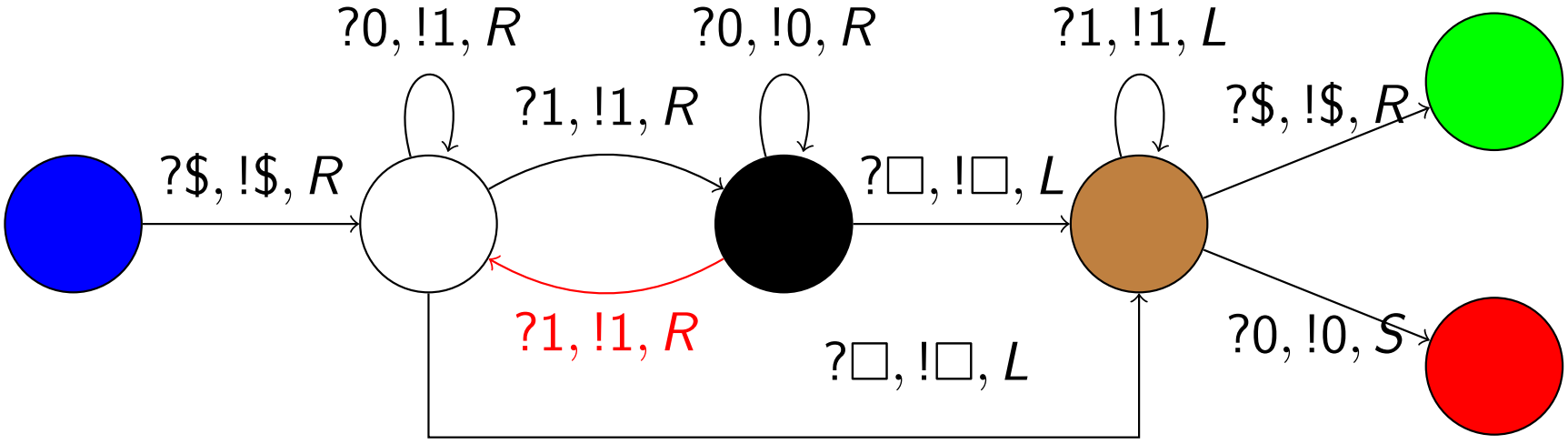


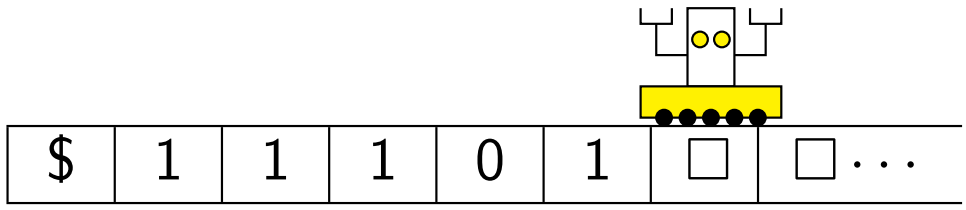
Run of a TM: example



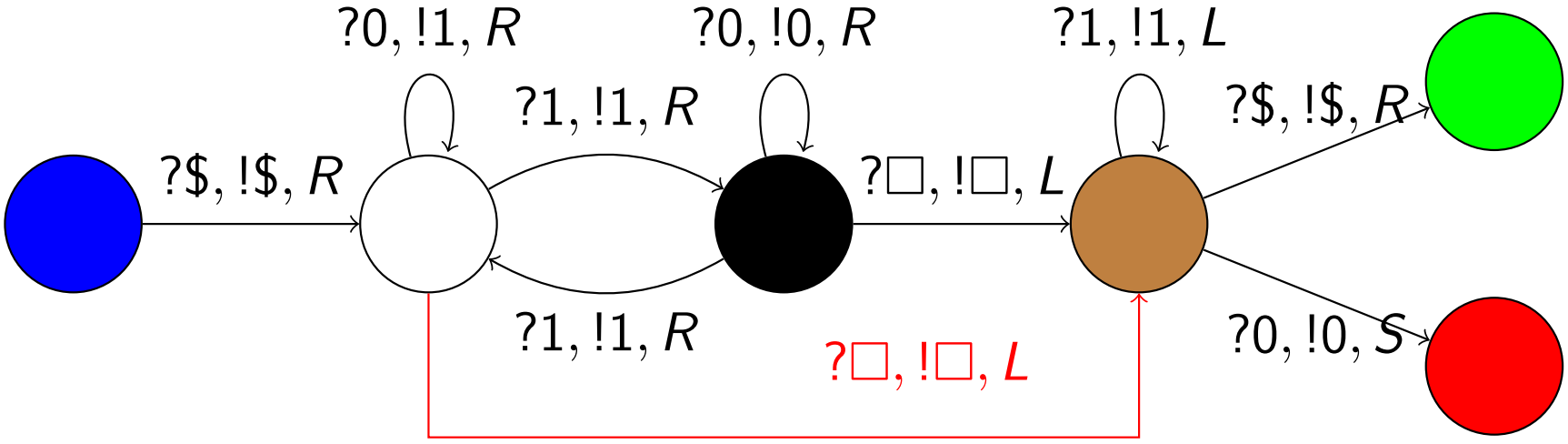


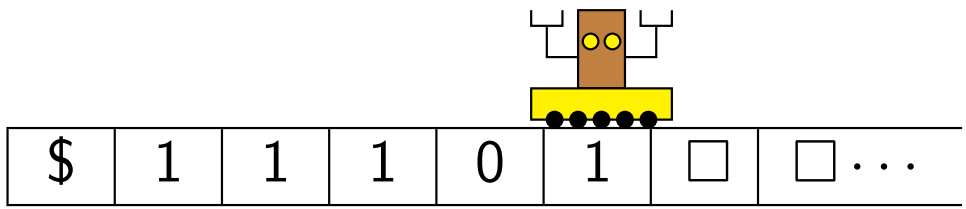
Run of a TM: example



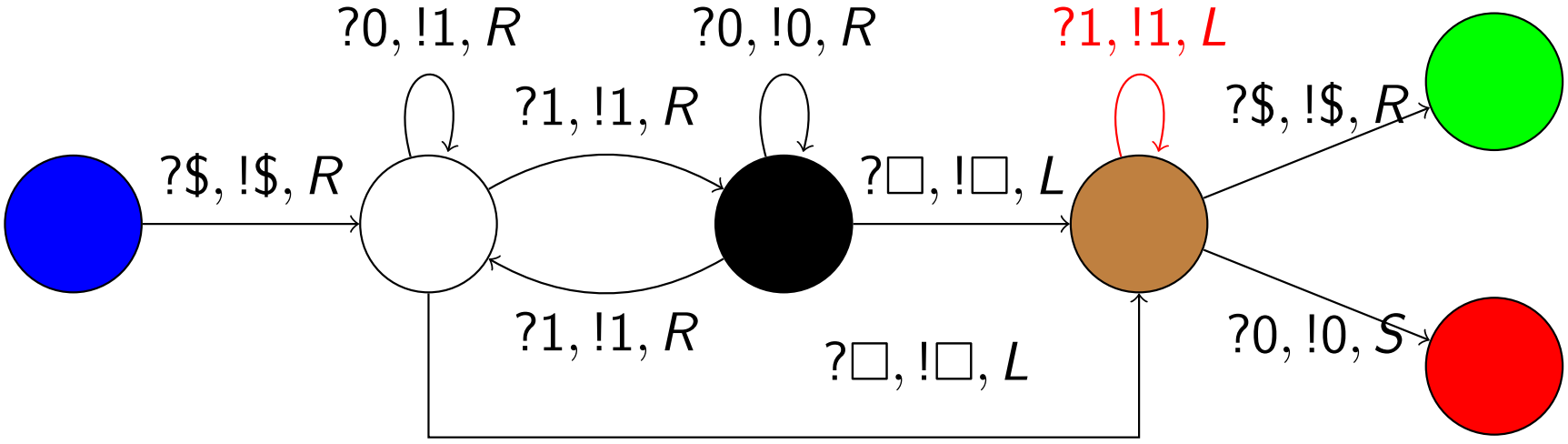


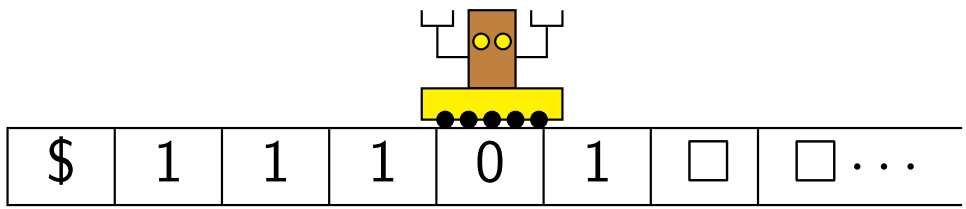
Run of a TM: example



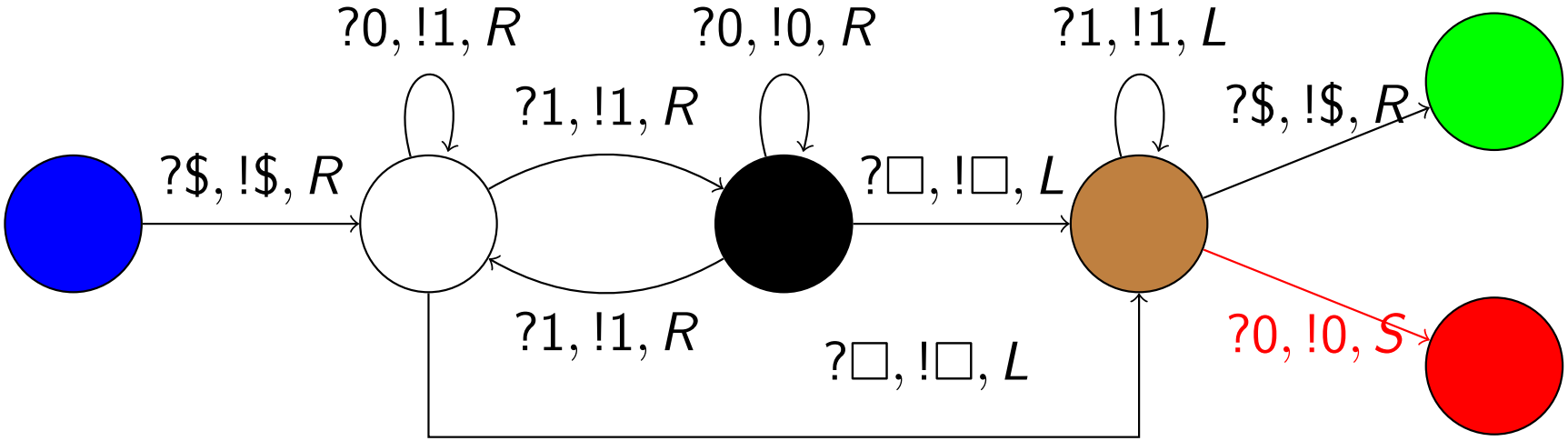


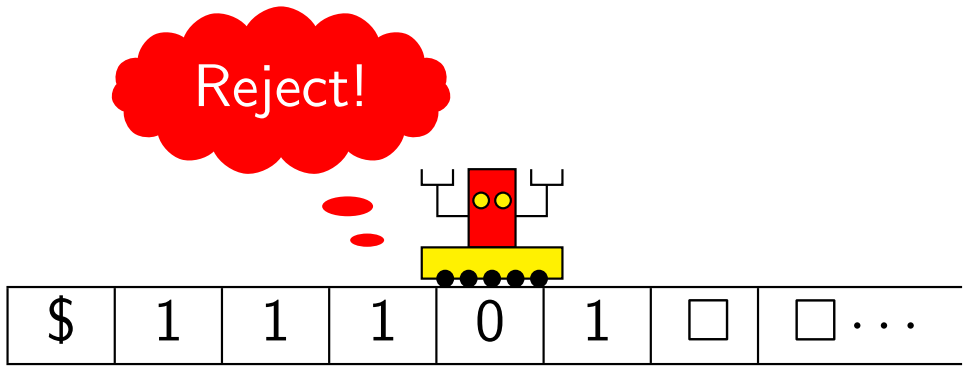
Run of a TM: example





Run of a TM: example



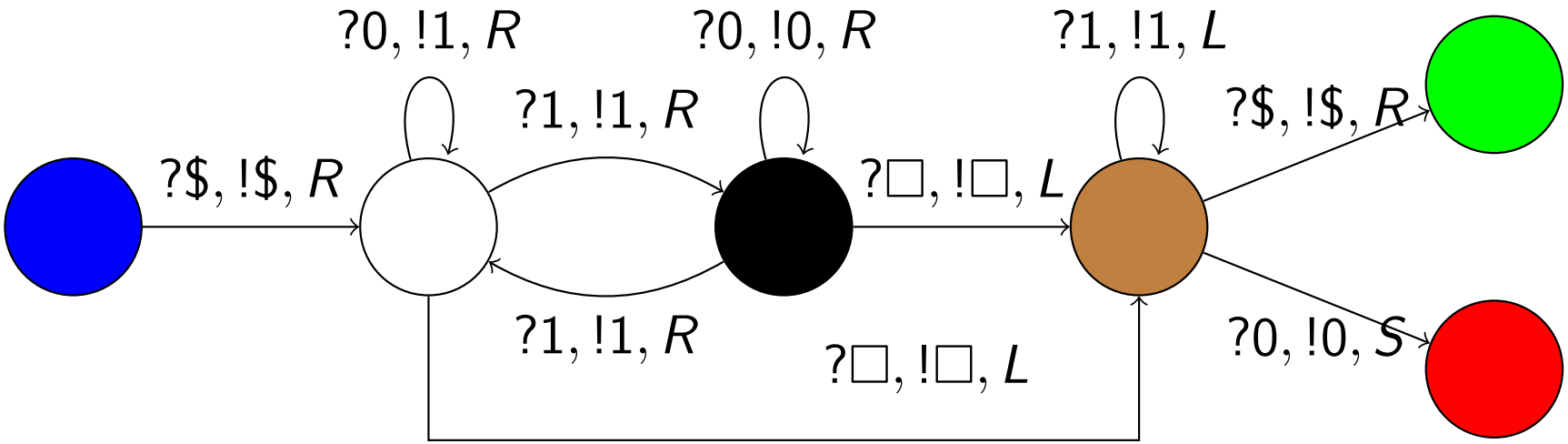


Language accepted by a Turing machine

$$L(\mathcal{M}) = \{w \in \Sigma^* \mid \text{Run}(\mathcal{M}, w) \text{ is accepting}\}$$

Example: $\$00101 \notin L(\mathcal{M}_0)$, where $\mathcal{M}_0$ is as before.

Question: what is $L(\mathcal{M}_0)$?



How the machine works

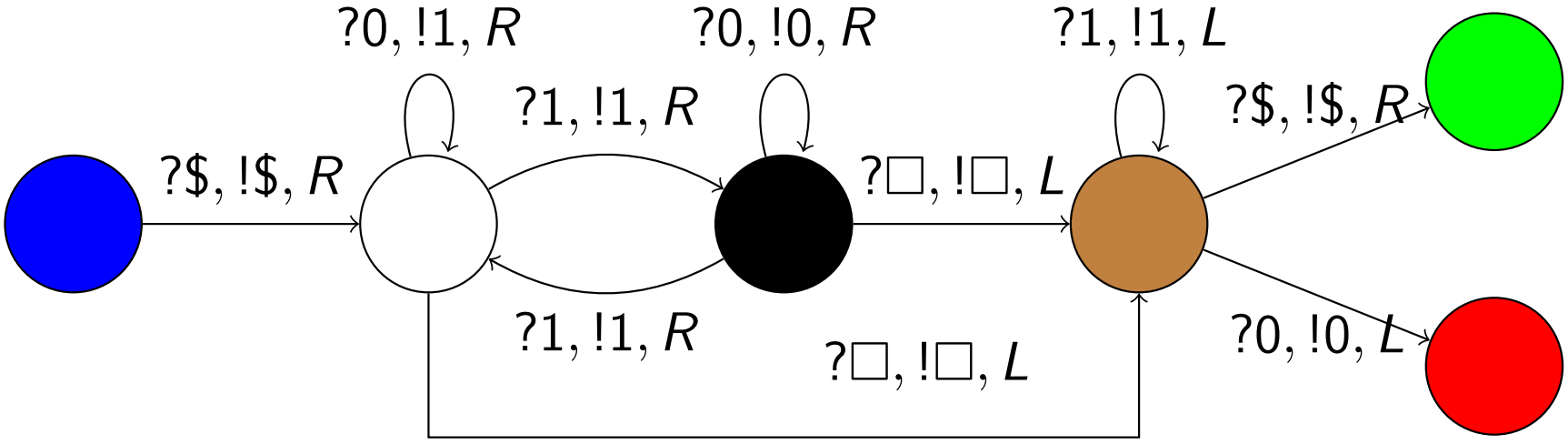
First phase

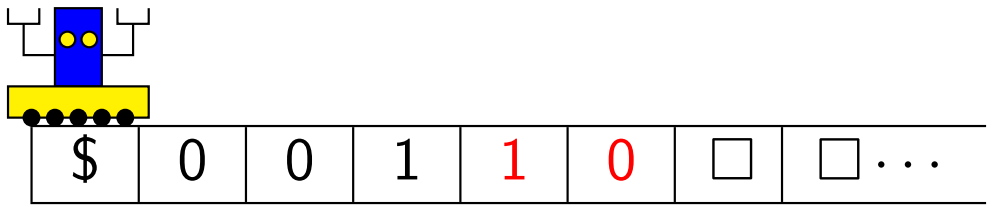
- it moves to the right until it reaches $\square$
- it swaps between $\circ$ and $\bullet$ when it reads a 1
- if it reads a 0 in $\circ$, it replaces it with a 1

Second phase

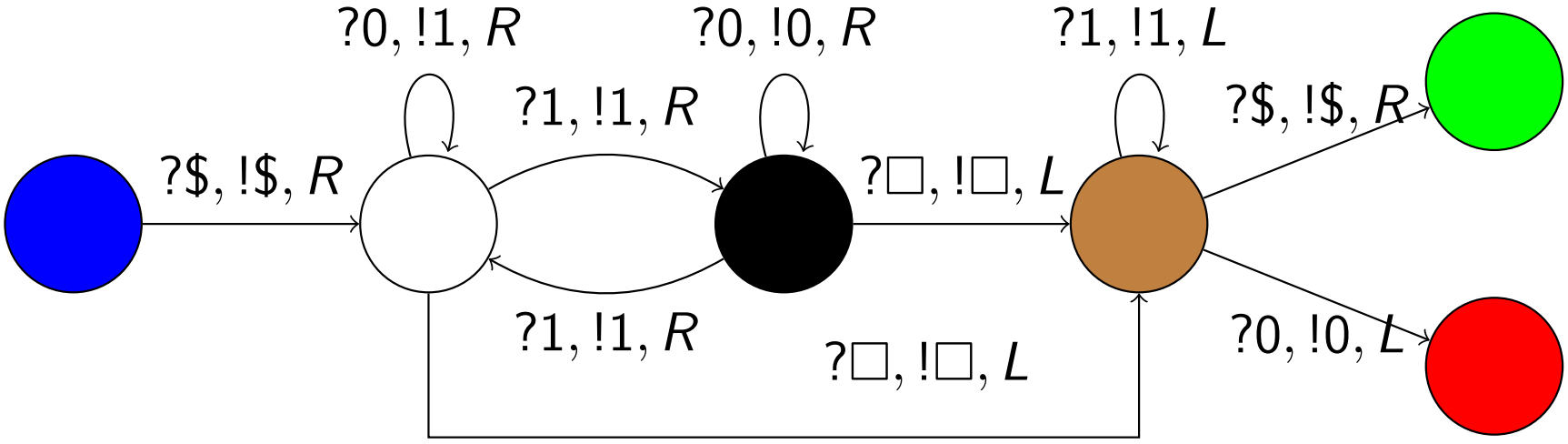
- it moves to the left until it reaches either \$ or 0
- it accepts if it reaches \$

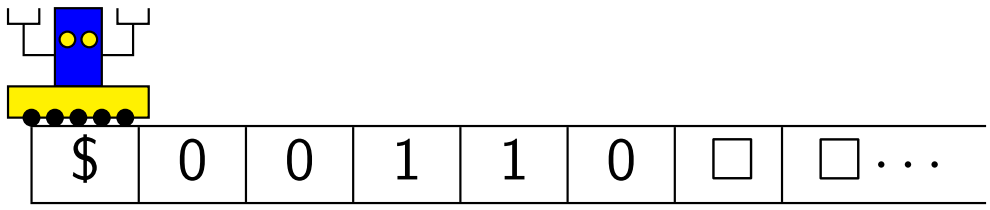
Run($\mathcal{M}_0$, \$00110)



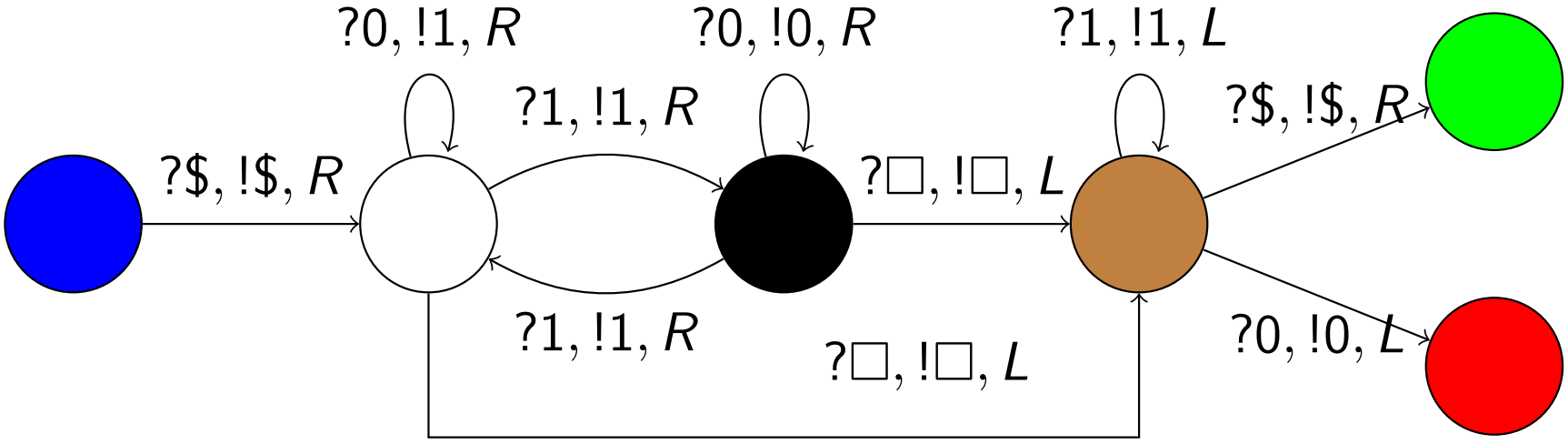


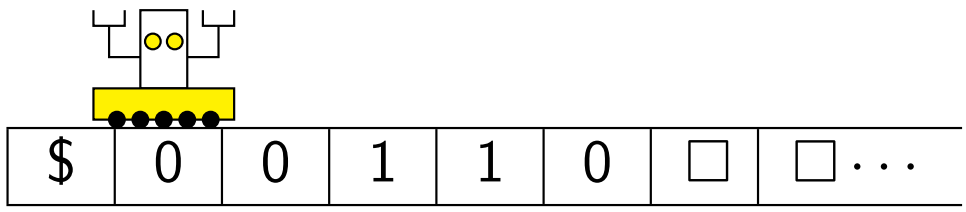
Run($\mathcal{M}_0$, \$00110)



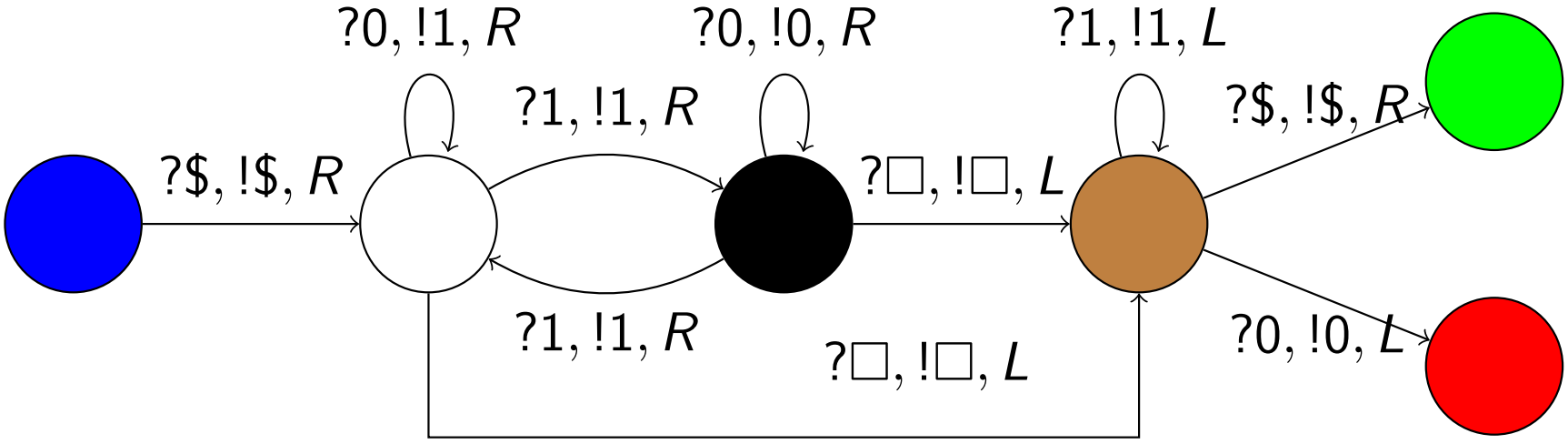


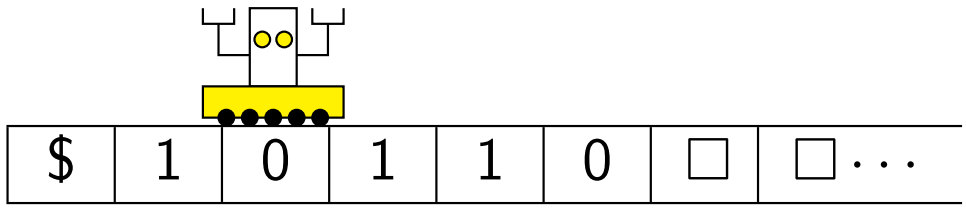
Run($\mathcal{M}_0$, \$00110)



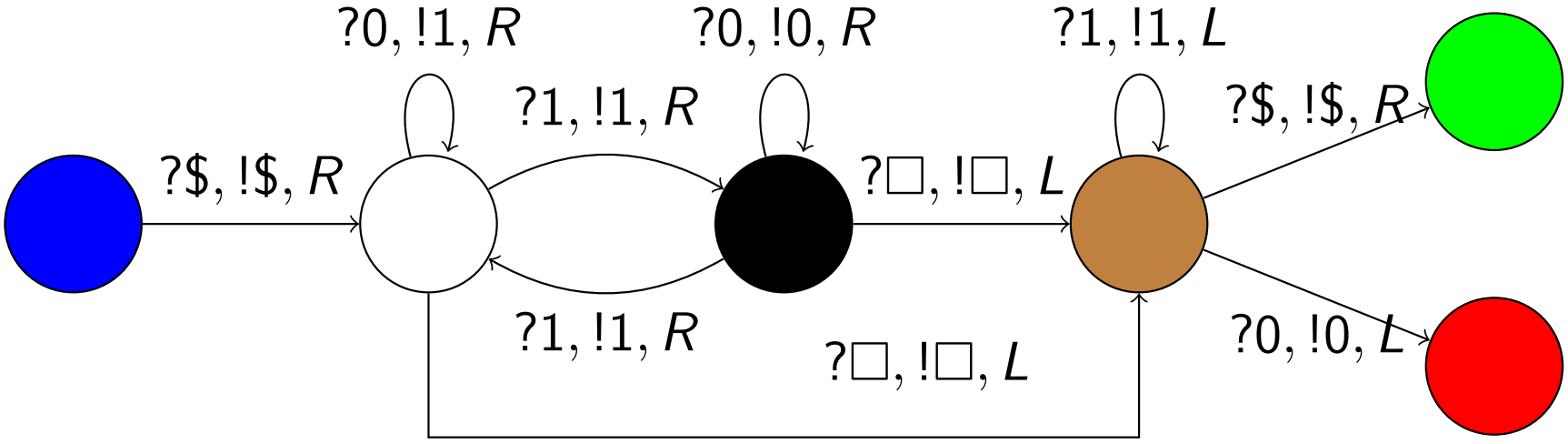


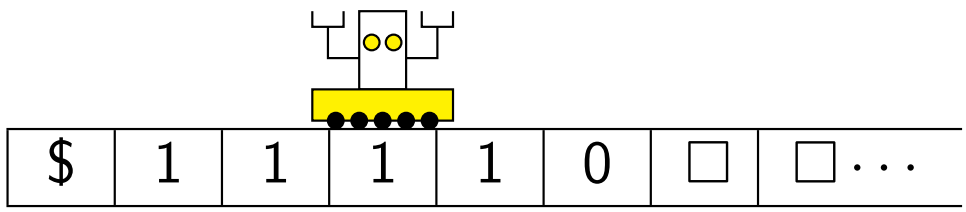
Run($\mathcal{M}_0$, \$00110)



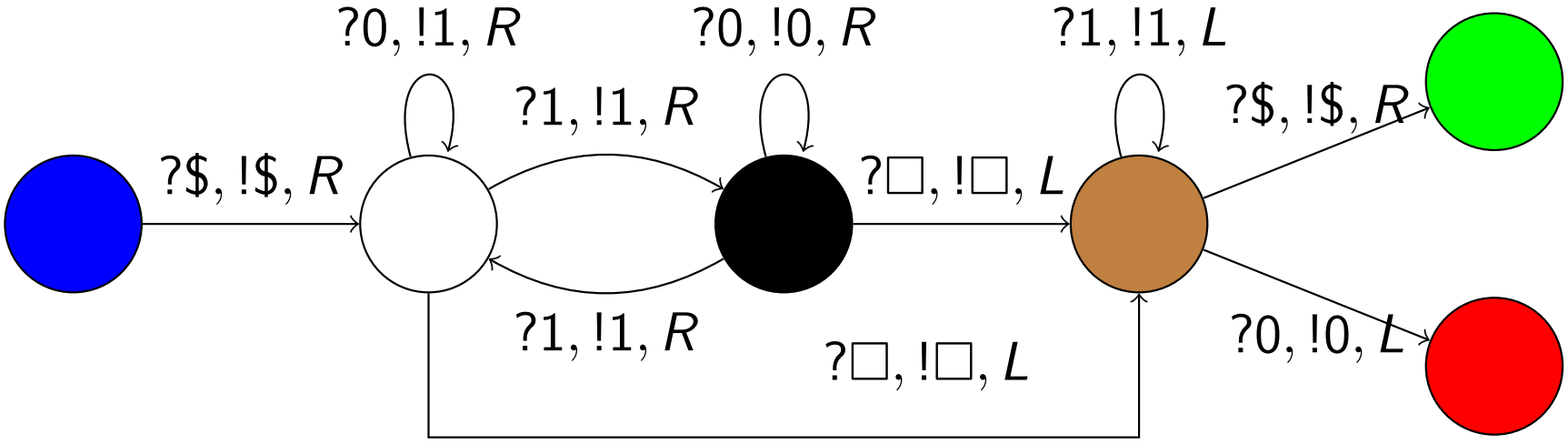


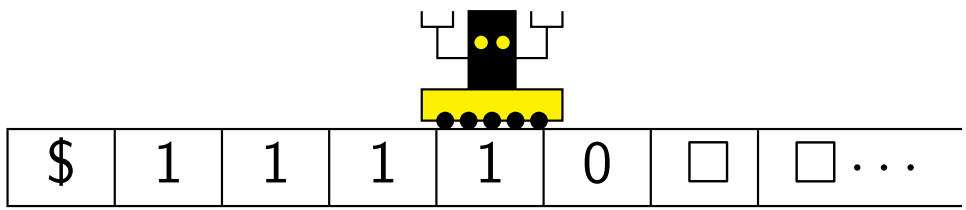
Run($\mathcal{M}_0$, \$00110)



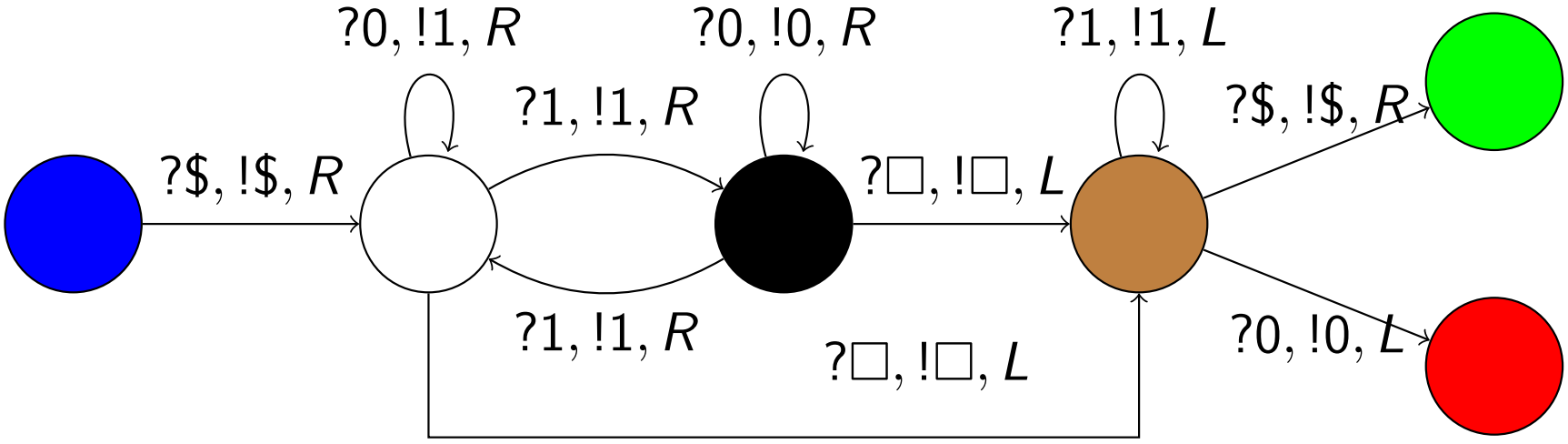


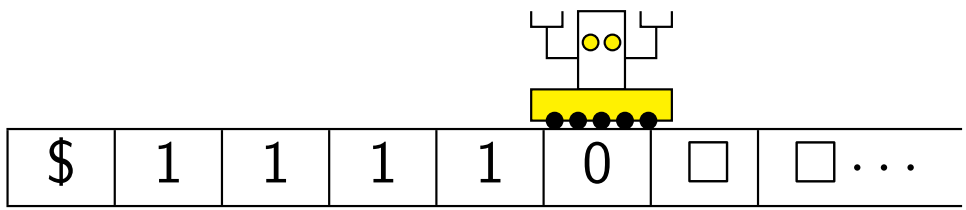
Run($\mathcal{M}_0$, \$00110)



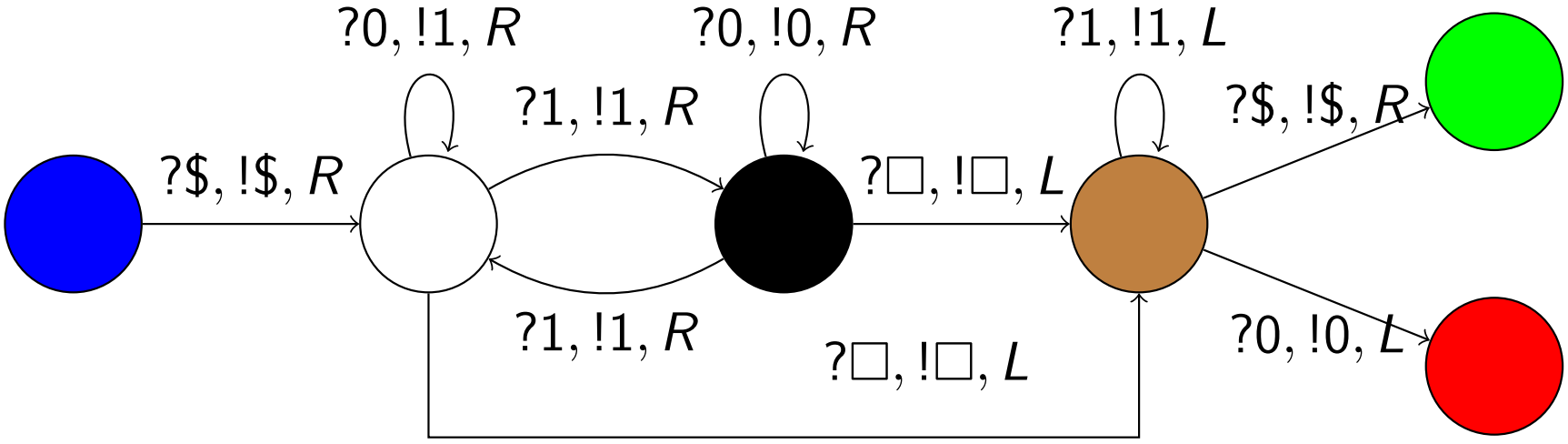


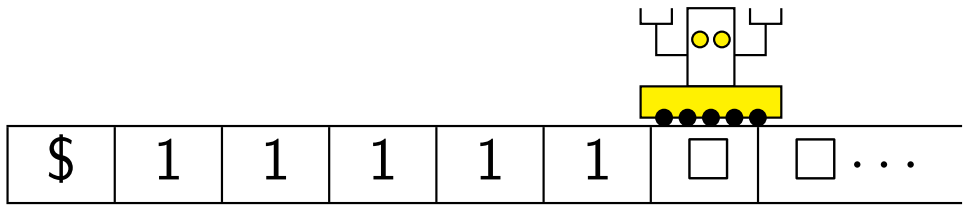
Run($\mathcal{M}_0$, \$00110)



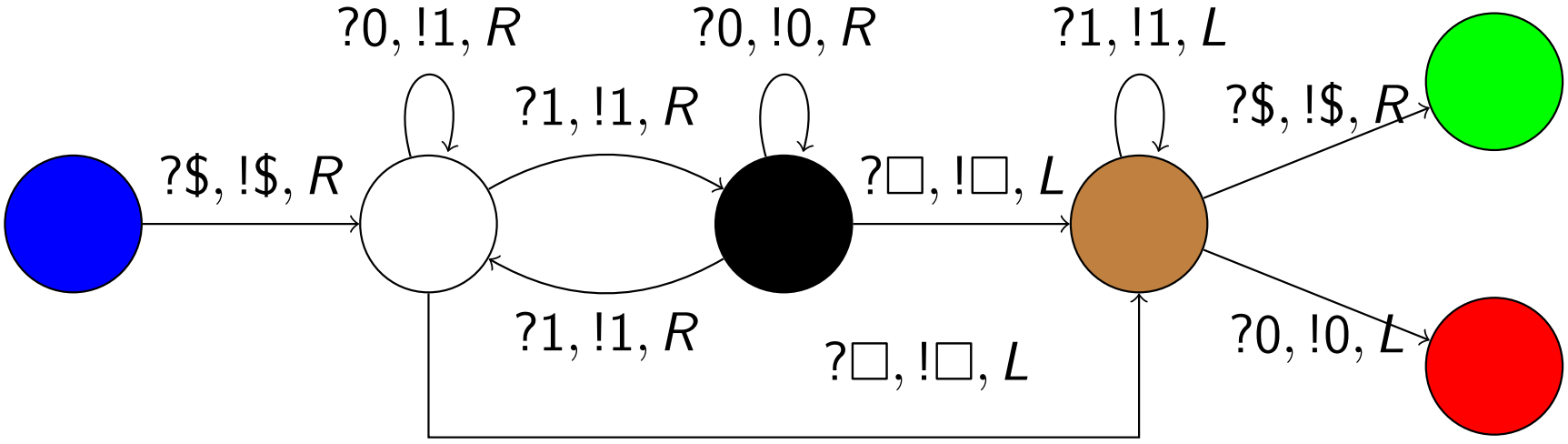


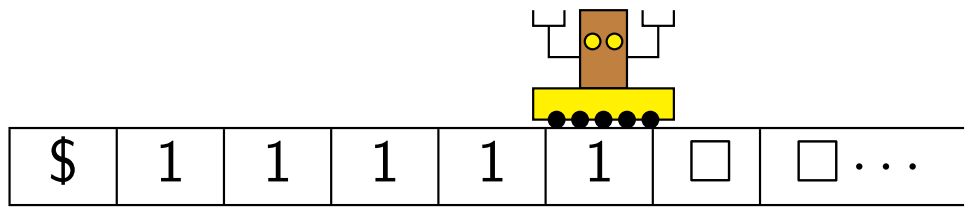
Run($\mathcal{M}_0$, \$00110)



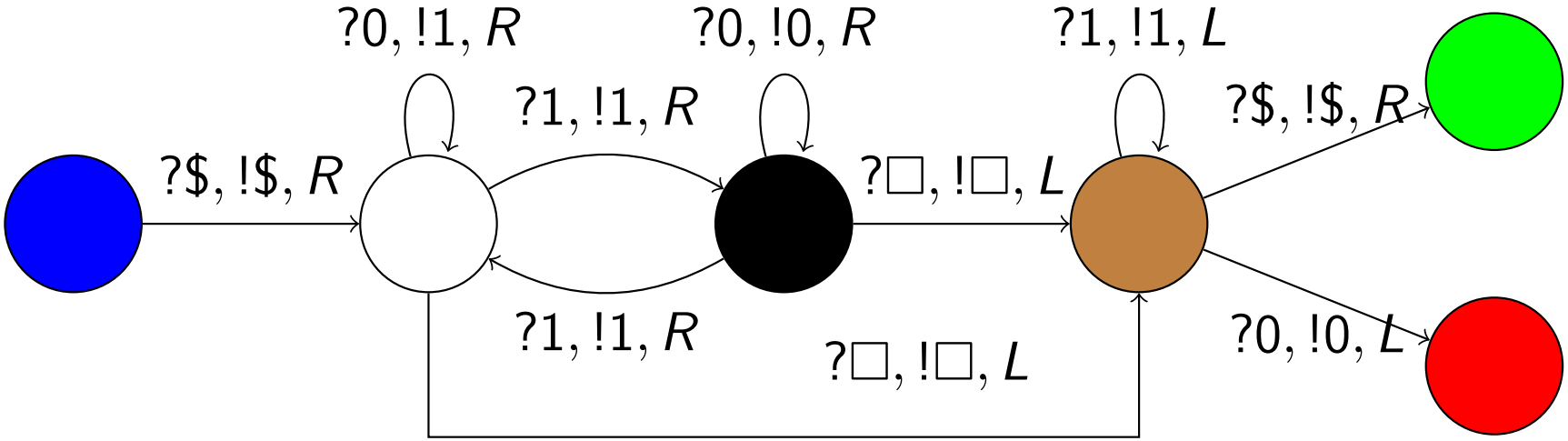


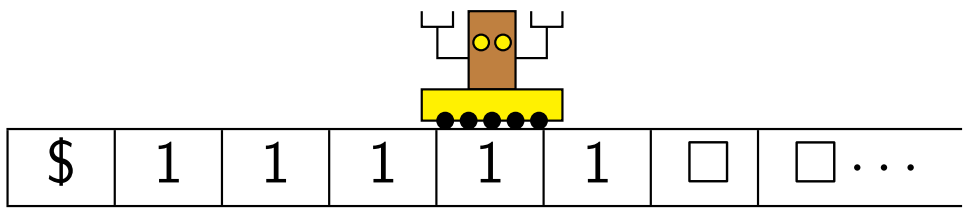
Run($\mathcal{M}_0$, \$00110)



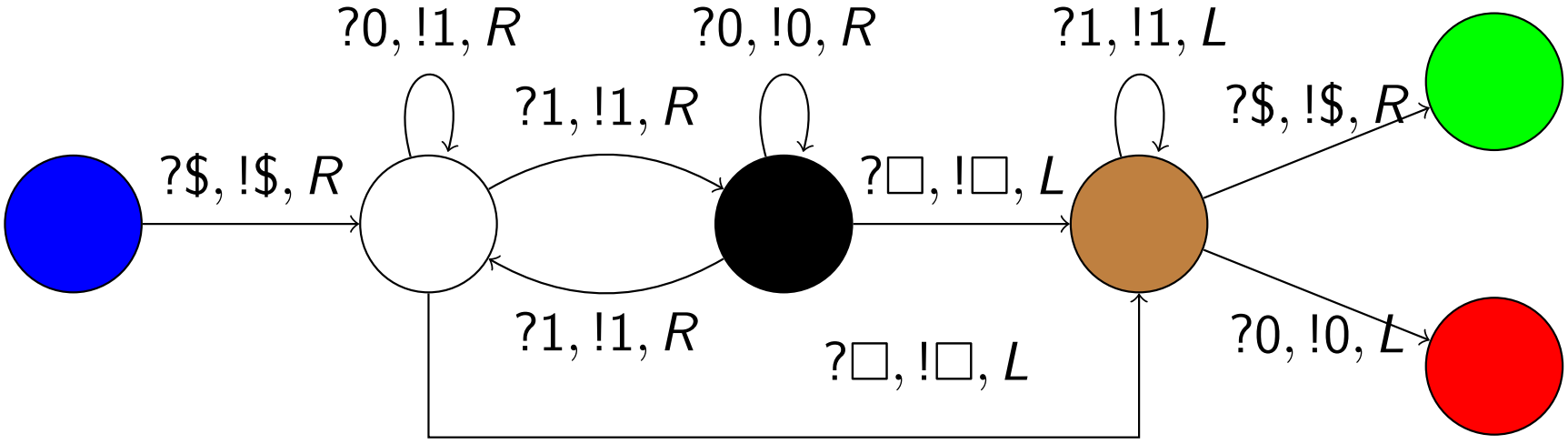


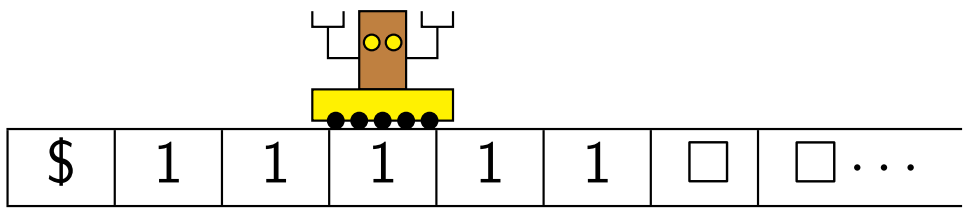
Run($\mathcal{M}_0$, \$00110)



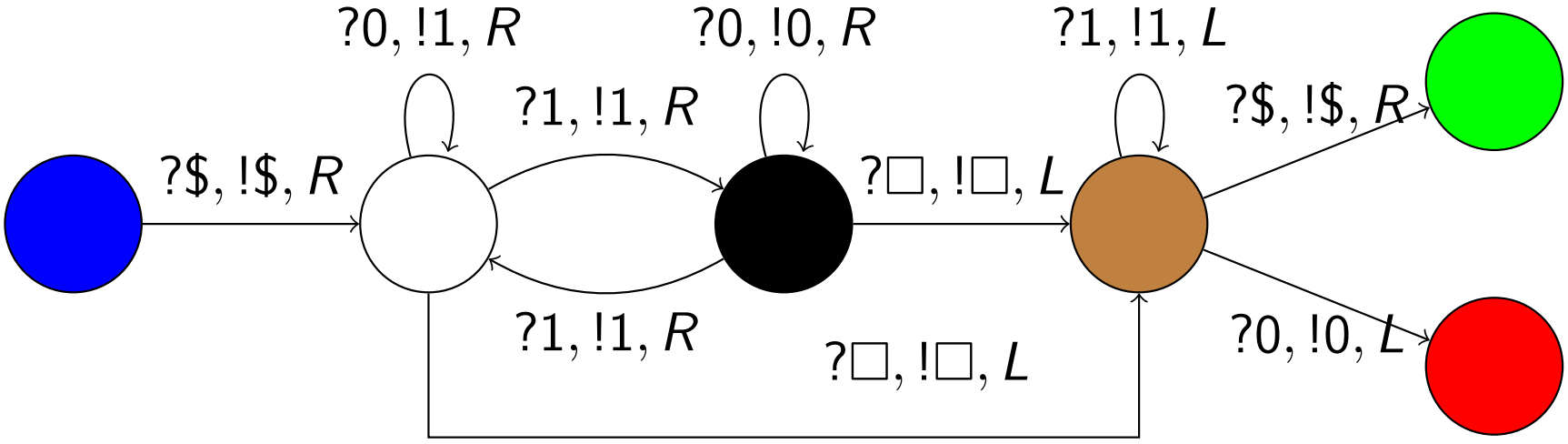


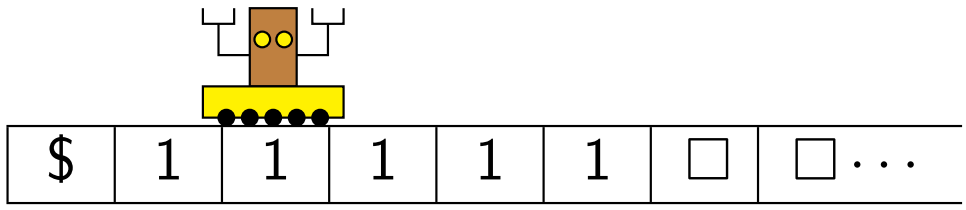
Run($\mathcal{M}_0$, \$00110)



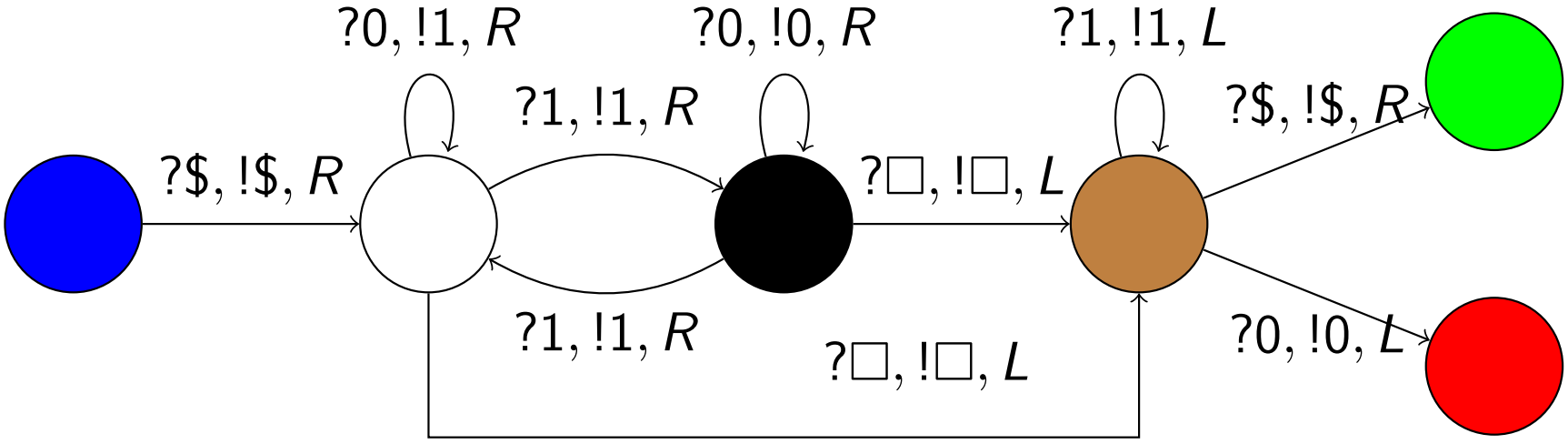


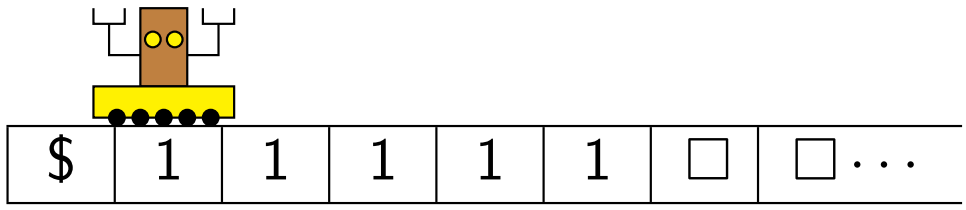
Run($\mathcal{M}_0$, \$00110)



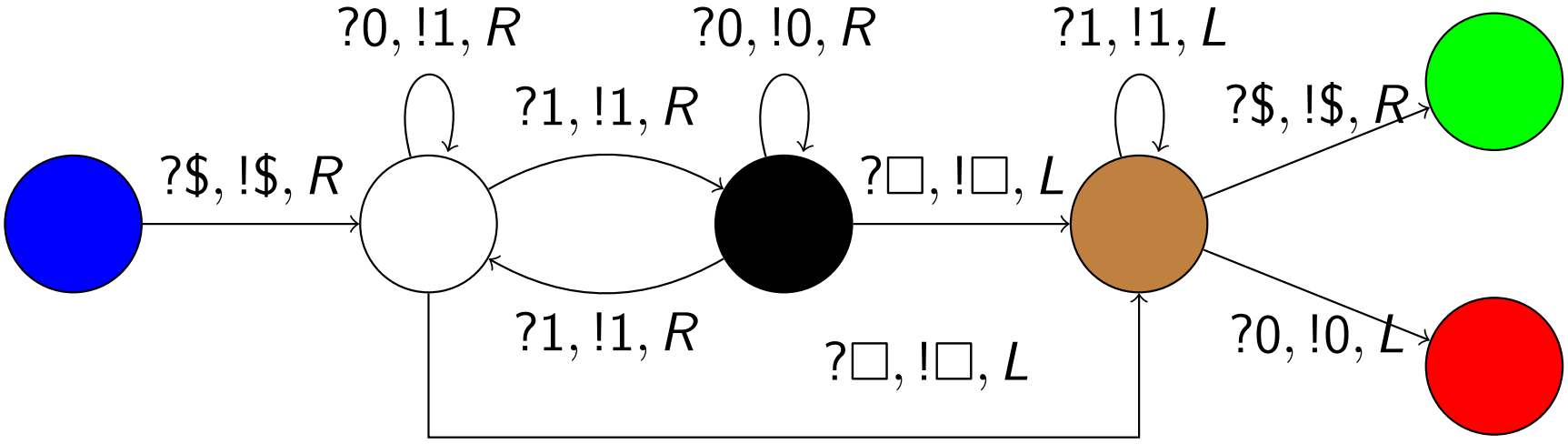


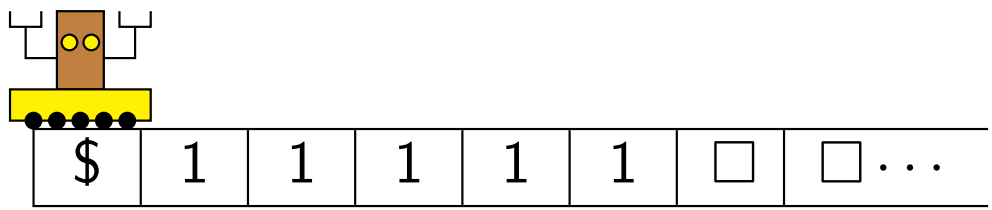
Run($\mathcal{M}_0$, \$00110)



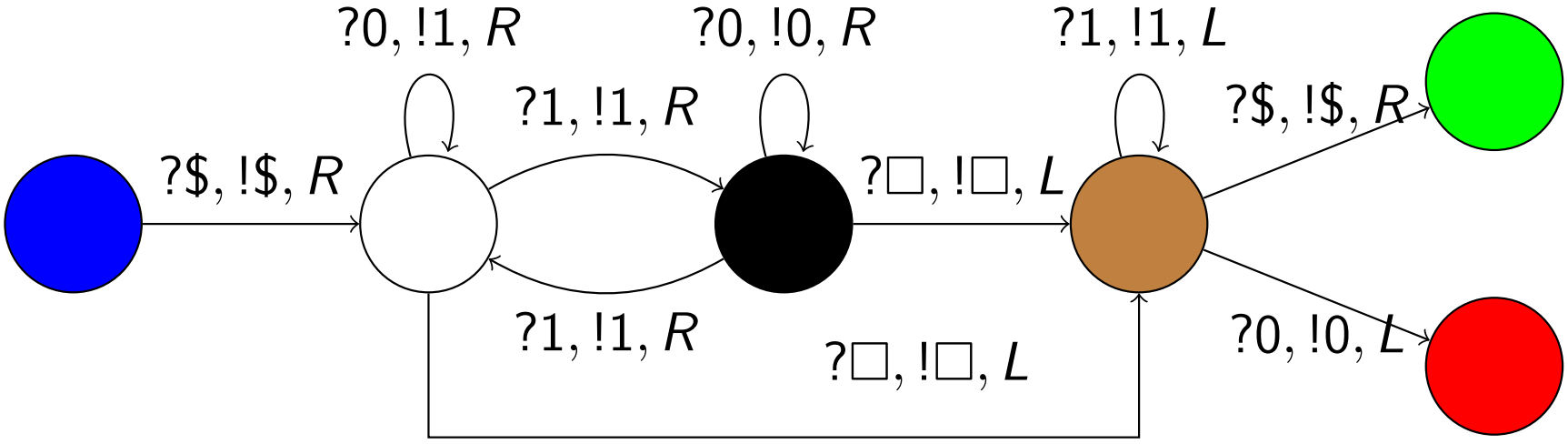


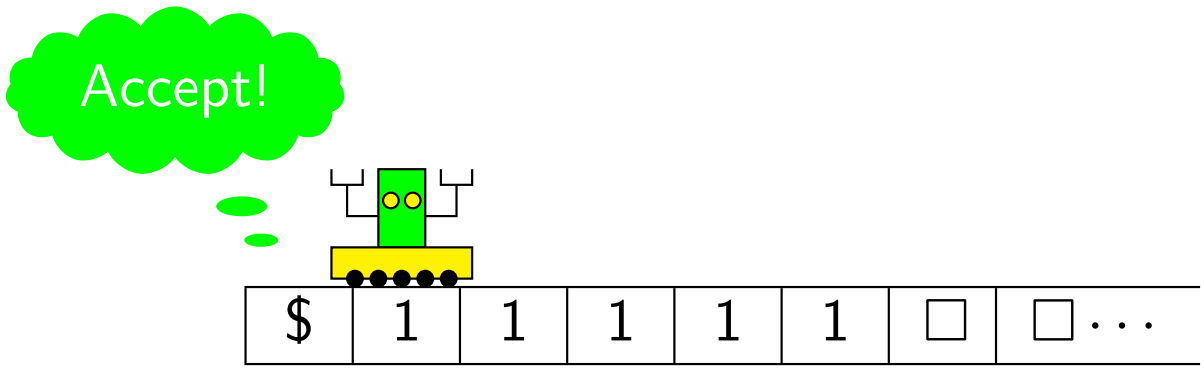
Run($\mathcal{M}_0$, \$00110)





Run($\mathcal{M}_0$, \$00110)





$L(\mathcal{M}_0)$

First phase

- it moves to the right until it reaches $\square$
- it swaps between $\circ$ and $\bullet$ when it reads a 1
- if it reads a 0 in $\circ$, it replaces it with a 1

Second phase

- it moves to the left until it reaches either \$ or 0
- it accepts if it reaches \$

$$\begin{aligned} L(\mathcal{M}_0) &= \{w \mid \text{all blocks of 1s are of even length}\} \\ &= \$(0+11)^*\square\Sigma^* \end{aligned}$$

Complexity

The time complexity $\mathrm{TIME}(\mathcal{M}, w)$ of the run of $\mathcal{M}$ on w is the number n of steps.
In other words, $(s_0, 0, w_0) \to_{\mathcal{M}}^{n} (s, i, w) \not\to_{\mathcal{M}}$

Example

For $\mathcal{M}_0$ as before, $\mathrm{TIME}(\mathcal{M}_0, w) \leq 2 \cdot |w|$

The space complexity $\mathrm{SPACE}(\mathcal{M}, w)$ of the run of $\mathcal{M}$ on w is the number m of distinct visited cells.
In other words, for all $(s, i, w) \in \mathrm{Run}(\mathcal{M}, w)$, $i \leq n$.

Example

for our TM, $\mathrm{SPACE}(\mathcal{M}_0, w) = |w| + 1$

Complexity classes

A complexity class is a collection of languages determined by three things:

- A model of computation
  (such as Turing machines, random access machines, circuits, etc)
- A resource
  (such as time, space or number of processors).
- A set of bounds

Time and space for Turing machines

Deterministic time

For any function $f : \mathbb{N} \to \mathbb{N}$ we say that a language L is in $\mathbf{DTIME}(f(n))$ if there is a machine $\mathcal{M}$ and a constant c such that

1. $L = L(\mathcal{M})$, and
2. for every $w \in L$, $\mathrm{TIME}(\mathcal{M}, |w|) \leq c \cdot f(|w|)$.

Deterministic space

Similarly, we define $\mathbf{DSPACE}(f(n))$ to be the class of languages accepted by a machine which uses $\mathcal{O}(f(n))$ tape cells[1] on inputs of length n.

---

[1] In defining space complexity, we assume a machine M, which has a read-only input tape, and a separate work tape, and we only count cells on the work tape.

Polynomial time computation

$$\mathbf{P} \stackrel{\text{def}}{=} \bigcup_{k=1}^{\infty} \mathbf{DTIME}(n^k)$$

The class of language decidable in polynomial time.

Why polynomial bounds?

By making the bounds broad enough, we can make our definitions fairly independent of the model of computation.
The collection of languages recognised in polynomial time is the same whether we consider Turing machines, random access machines, or any other deterministic model of computation.
The collection of languages recognised in linear time, on the other hand, is different on a one-tape and a two-tape Turing machine.

Closure properties

Union and intersection

if L_1, L_2 are in **P**, then so do $L_1 \cap L_2$ and $L_1 \cup L_2$

Proof: simulate two runs in one run

Complementation

if L is in **P**, then $\Sigma^* \setminus L$ is in **P**

Proof: ensure first that the machine halts on all inputs, then swap accepting and rejecting states

Erasure of first symbol

if L is in **P**, then $L' = \{w \mid aw \in L \text{ for some } a \in \Sigma\}$ is in **P**

Proof: try all possible erased symbol and simulate a run for each of them

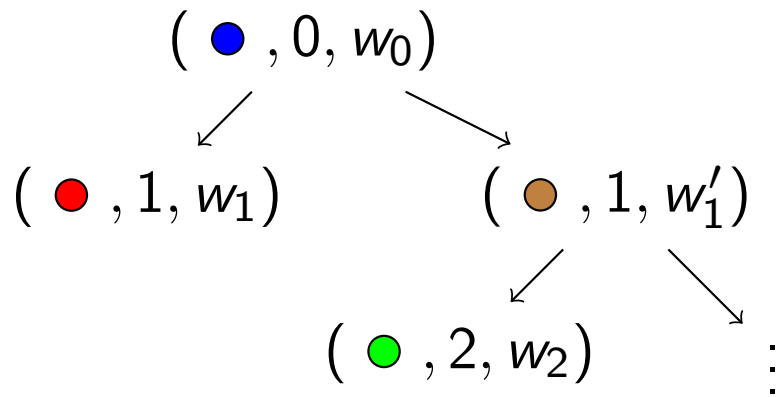
erasure: generalize to any fixed number of symbols erased at any position

Non-deterministic Turing machine

If, in the definition of a Turing machine, we relax the condition on δ being a function and instead allow an arbitrary relation, we obtain a nondeterministic Turing machine.

$$\delta \subseteq (K \times \Sigma) \times (K \times \Sigma \times \{L, R, S\}$$

The small step semantics $\rightarrow_{\mathcal{M}}$ is also no longer functional, and all runs form a computation tree.



Acceptance condition

$\mathcal{M}$ accepts w in time t and space s if at least one run does.

Nondeterministic Complexity

Nondeterministic time

For any function $f : \mathbb{N} \to \mathbb{N}$, we say that a language L is in $\mathbf{NTIME}(f(n))$ if there is a nondeterministic machine $\mathcal{M}$ such that $L = L(M)$ and for every $w \in L$ there is an accepting run of $\mathcal{M}$ on w in time $f(|w|)$.

Non-deterministic space

Similarly, we define $\mathbf{NSPACE}(f(n))$ to be the languages accepted by a nondeterministic machine which uses $\mathcal{O}(f(n))$ tape cells on inputs of length n. As before, we only count work space.

Nondeterministic polynomial time

$$\mathbf{NP} \stackrel{\text{def}}{=} \bigcup_{k=1}^{\infty} \mathbf{NTIME}(n^k)$$

That is, **NP** is the class of languages accepted by a nondetermistic machine running in polynomial time

$$\mathbf{P} \subseteq \mathbf{NP}$$

since a deterministic machine is just a nondeterministic machine in which δ is functional.

Closure properties

Union and intersection

if L_1, L_2 are in **NP**, then so do $L_1 \cap L_2$ and $L_1 \cup L_2$

Complementation?

we don't know if **NP** is closed under complementation.

Why? if **NP** is not closed under complementation, then **P** $\neq$ **NP**.

Erasure of the first half of symbols

if L is in **NP**, then

$L' = \{w \mid w' \cdot w \in L \text{ for some } w' \in \Sigma^* \text{ with } |w'| = |w|\}$ is in **NP**

Proof The machine "guesses" all erased symbols and then simulates the run. This strongly relies on non-determinism!

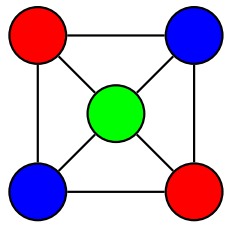
What about the closure of **P** under this erasure? We don't know!

Famous problems in NP

We identify a class of graphs with a language (see 2 next slides).

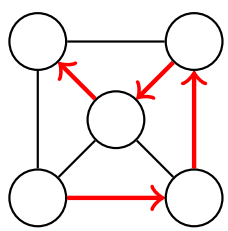
3 colorability

- Input : a finite graph $G = (V, E)$
- Question : is there $c : V \to \{1, 2, 3\}$ such that for all $(v_1, v_2) \in E$ we have $c(v_1) \neq c(v_2)$?



Hamiltonian path

- Input : a finite graph $G = (V, E)$
- Question : are there $v_1, v_2, \ldots, v_n \in V$ such that
  1. $(v_i, v_{i+1}) \in E$ for all i, and
  2. all v_i are distinct, and
  3. $V = \{v_1, \ldots, v_n\}$.



Signature and structure

A signature σ is a finite sequence of relation symbols

$$\sigma = (R_1, \ldots, R_m)$$

where every R_i has a fixed arity $k_i \geq 0$.

A σ-structure is a tuple

$$\mathfrak{A} = (A, R_1^{\mathfrak{A}}, \ldots, R_m^{\mathfrak{A}})$$

where A is a set and $R_i^{\mathfrak{A}} \subseteq A^{k_i}$.

Remarks

- We will always implicitly consider finite structures only.
- An oriented graph is a structure over a signature with a unique relation symbol of arity 2.
- In general a signature also allows function symbols and a structure has to provide an interpretation for them.

Coding a structure as a word

Let us fix $\Sigma = \{0, 1, \square\}$

If $\sigma = \{R_1, \ldots, R_p\}$, and $\mathfrak{A} = (A, R_1^{\mathfrak{A}}, \ldots, R_p^{\mathfrak{A}})$ is a finite σ-structure with $|A| = n$, we define its encoding as the concatenation of the encodings of relations.

$$\mathrm{enc}(\mathfrak{A}) = 0^n 1 \cdot \mathrm{enc}(R_1^{\mathfrak{A}}) \cdot \mathrm{enc}(R_2^{\mathfrak{A}}) \cdots \mathrm{enc}(R_p^{\mathfrak{A}})$$

In order to code the relations $R_i^{\mathfrak{A}}$, we need to fix an enumeration $a_0, \ldots, a_{n-1}$ of A. Once the enumeration is fixed, the k-tuple $t = (a_{i_1}, \ldots, a_{i_k})$ is identified by the number $\mathrm{enc}(t) = i_1 + n \cdot i_2 + n^2 \cdot i_3 + \cdots + n^{k-1} \cdot i_k$. The encoding of $R_i^{\mathfrak{A}}$ of arity k_i is the sequence of bits $b_0 b_1 \ldots b_{n^{k_i}-1}$ such that $b_i = 1$ iff $\mathrm{enc}^{-1}(i) \in R_i^{\mathfrak{A}}$.

Querying structures

Another way of thinking of a structure : a database

Example : a database with two tables

ETU

1029021	Camille	Tozzi
1072902	Moez	Zanad
⋮	⋮	⋮

MARK

1029021	17
1202131	13
⋮	⋮

$A = \{$ all students IDs, all names and surnames, all final exam marks $\}$

$\sigma = \{\text{ETU}, \text{MARK}\}$ with resp. arities 3 and 2

A request defines a new table from the previous ones

Example

the table TAKEN($name, surname$) that contains all names and surnames of the students that took the final exam

$(\text{Camille}, \text{Tozzi}) \in \text{TAKEN}^{\mathfrak{A}}$

First order logic

Let $X = \{x, y, \dots\}$ be a fixed set of variables.
Formulas of first order logic (FO) are defined by induction

- atomic formulas
  if $R \in \sigma$ is a relation symbol of arity n,
  then $R(x_1, x_2, \dots, x_n)$ is a formula
- Boolean combinations
  if $\varphi, \varphi_1, \varphi_2$ are formulas, then $\neg\varphi$, $\varphi_1 \vee \varphi_2$, $\varphi_1 \wedge \varphi_2$ and $\varphi_1 \Rightarrow \varphi_2$ are formulas
- quantification over elements of the structure if φ is a formula, then $\forall x\varphi$ and $\exists x\varphi$ are formulas

Example

A formula that defines the query $\mathrm{TAKEN}(x, y)$ is

$$\exists z(\mathrm{ETU}(z, x, y)) \wedge (\exists t \mathrm{GRADE}(z, t))$$

Semantics of FO

A valuation is a function $\upsilon : X \to A$.
$\mathfrak{A}, \upsilon \models \varphi$ is defined by induction

$\mathfrak{A}, \upsilon \models R(x_1, \ldots, x_n)$	if $\big(\upsilon(x_1), \ldots, \upsilon(x_n)\big) \in R^{\mathfrak{A}}$
$\mathfrak{A}, \upsilon \models \varphi_1 \vee \varphi_2$	if $\mathfrak{A}, \upsilon \models \varphi_1$ or $\mathfrak{A}, \upsilon \models \varphi_2$
$\mathfrak{A}, \upsilon \models \varphi_1 \wedge \varphi_2$	if $\mathfrak{A}, \upsilon \models \varphi_1$ and $\mathfrak{A}, \upsilon \models \varphi_2$
$\mathfrak{A}, \upsilon \models \neg\varphi$	if $\mathfrak{A}, \upsilon \not\models \varphi$
$\mathfrak{A}, \upsilon \models \exists x\varphi$	if there is $a \in A$ s.t. $\mathfrak{A}, \upsilon[x \mapsto a] \models \varphi$
$\mathfrak{A}, \upsilon \models \forall x\varphi$	if for all $a \in A\mathfrak{A}, \upsilon[x \mapsto a] \models \varphi$

Closed formulas, Boolean queries

A formula is closed if every occurence of a variable x is underneath a $\exists x$.

Examples

- $\exists z(\mathrm{ETU}(z, x, y)) \wedge (\exists t \mathrm{GRADE}(z, t))$ is not closed
  x and y are not quantified
- $\forall x(P(x) \vee \exists y Q(x, y))$ is closed
- $\exists y\big(Q(x, y) \vee \forall x P(x)\big)$ is not closed

We write $\mathfrak{A} \models \varphi$ if $\mathfrak{A}, v \models \varphi$ for all v.

Data complexity

Let $\mathrm{Mod}(\varphi) = \{\mathrm{enc}(\mathfrak{A}) \mid \mathfrak{A} \models \varphi\}$

Claim

Let φ be a fixed FO formula. Then $\mathrm{Mod}(\varphi) \in \mathbf{P}$

The straightforward algorithm proceeds recursively on the structure of φ. Each quantifier corresponds to a "for loop" enumerating A.

If $|A| = n$ and φ has at most m nested quantifiers, then the running time of the algorithm is in $\mathcal{O}(n^m)$.

Space complexity

We need to remember a counter between 0 and $n - 1$ for each quantifier, plus a call stack whose depth is bounded by the depth of the formula (which is a constant), so the space complexity is in $\mathcal{O}(m \log n)$.

Inexpressivity results

We just saw that $\{\mathrm{Mod}(\varphi) \mid \varphi \in \mathrm{FO}\} \subseteq \mathbf{P}$.

But is the inclusion strict?

The answer is YES:

- connectivity of a graph is in **P**, but cannot be expressed in **FO**
- evenness of $\mathfrak{A}$ is in **P**, but cannot be expressed in **FO**.

Second order logic

We extend first-order logic by a set of relational variables

For each $m \in \mathbb{N}$ there is an infinite collection of variables $\mathcal{V}^m = \{V_1^m, V_2^m \dots\}$ of arity m.

Second-order logic extends first-order logic by allowing second-order quatifiers

$$\exists X \varphi \qquad \text{for } X \in \mathcal{V}^m$$

A structure $\mathfrak{A}$ satisfies $\exists X \varphi$ if there is an m-ary relation R on the universe of $\mathfrak{A}$ such that $(\mathfrak{A}, X \to R)$ satisfies φ.

Existential second-order logic

Existential second-order logic (**ESO**) consists of those formulas of second-order logic of the form

$$\exists X_1 \ldots \exists X_k \varphi$$

Examples

Evenness

This formula is true in a structure if, and only if, the size of the domain is even.

$$
\begin{array}{ll}
\exists B \exists S & \forall x \exists y B(x,y) \wedge \forall x \forall y \forall z B(x,y) \wedge B(x,z) \rightarrow y = z \quad (1) \\
& \forall x \forall y \forall z B(x,z) \wedge B(y,z) \rightarrow x = y \quad (2) \\
& \forall x \forall y S(x) \wedge B(x,y) \rightarrow \neg S(y) \quad (3) \\
& \forall x \forall y \neg S(x) \wedge B(x,y) \rightarrow S(y)
\end{array}
$$

1. B is a functional relation
2. it is injective (therefore a permutation)
3. it maps S to its complement and vice versa

Examples

Transitive closure

This formula is true of a pair $(a, b) \in A$ if, and only if, there is an E-path from a to b is even.

$$\begin{aligned}
\exists P \quad & \forall x \forall y P(x, y) \rightarrow E(x, y) \\
& \exists x P(a, x) \wedge \exists x P(x, b) \wedge \neg \exists x P(x, a) \wedge \neg \exists x P(b, x) \\
& \forall x \forall y P(x, y) \rightarrow \big(\forall z P(x, z) \rightarrow y = z\big) \\
& \forall x \forall y P(x, y) \rightarrow \big(\forall z P(z, y) \rightarrow x = z\big) \\
& \forall x (x \neq a \wedge \exists y P(x, y)) \rightarrow \exists z P(z, x) \\
& \forall x (x \neq b \wedge \exists y P(y, x)) \rightarrow \qquad (5)
\end{aligned}$$

P is a partial function that associated with a path π from a to b. It maps a node of the path to its successor.

It only works for finite structures!

Examples

3-colourability

This formula is true in a graph $G = (V, E)$ if, and only if, it is 3-colourable

$$\begin{aligned}
\exists R \exists G \exists B \quad & \forall x R(x) \vee B(x) \vee G(x) \\
& \forall x \neg(R(x) \wedge G(x)) \wedge \neg(R(x) \wedge B(x)) \wedge \neg(G(x) \wedge B(x)) \\
& \forall x \forall y E(x, y) \rightarrow \big(\quad \neg(R(x) \wedge R(y)) \wedge \\
& \qquad\qquad\qquad\qquad \neg(G(x) \wedge G(y)) \wedge \\
& \qquad\qquad\qquad\qquad \neg(B(x) \wedge B(y))\big)
\end{aligned}$$

Fagin's theorem

Theorem (Fagin)

Let $\mathcal{C}$ be class of finite structures. The following two are equivalent

1. $\mathcal{C} = \mathrm{Mod}(\varphi)$ for some $\varphi \in \mathbf{ESO}$
2. $\{\mathrm{enc}(\mathfrak{A}) \mid \mathfrak{A} \in \mathcal{C}\} \in \mathbf{NP}$

In other words,

$$\text{“}\mathbf{ESO} = \mathbf{NP}\text{”}$$

One direction is easy: given $\mathfrak{A}$ and $\exists P_1 \ldots \exists P_m \varphi$, a nondeterministic Turing machine can guess an interpretation for $P_1, \ldots, P_m$ and then verify φ.

Fagin's theorem

Fix a nondeterministic machine $\mathcal{M}$ that accepts $\mathrm{enc}(\mathcal{C})$ in time $\mathcal{O}(n^k)$

We construct a first-order formula $\varphi_{\mathcal{M},k}$ such that

$$(\mathfrak{A}, <, \mathbf{X}) \models \varphi_{M,k} \quad \Leftrightarrow \quad \begin{array}{l}\mathbf{X} \text{ codes an accepting run of } \mathcal{M} \\ \text{of length at most } n^k \text{ on input } \mathrm{enc}(\mathfrak{A}, <)\end{array}$$

So, $\mathfrak{A} \models \exists < \exists \mathbf{X}\ \mathrm{order}(<) \wedge \varphi_{\mathcal{M},k}$ if, and only if, there is some total order $<$ on $\mathfrak{A}$ so that $\mathcal{M}$ accepts $\mathrm{enc}(\mathfrak{A}, <)$ in time n^k.

Constructing the formula

order($<$) is the **FO** formula

$$\begin{array}{ll} \forall x \forall y & (x \neq y) \leftrightarrow (x < y \vee y < x) \quad \wedge \\ \forall x \forall y \forall z & (x < y \wedge y < z) \rightarrow x < z \end{array}$$

the lexicographical order on k-tuples is expressed by the formula

$$\bigvee_{i<k} \Big((\bigvee_{j<i} x_j = y_j) \wedge x_i < y_i \Big)$$

a k-tuple $\mathbf{x}$ codes a number in $\{0, \ldots, n^k - 1\}$ and we can express some simple arithmetic on individuals and on k-tuples

$$\begin{array}{ll} x = 0 & \text{stands for } \forall y (x \leq y) \\ x = y + 1 & \text{stands for } \forall z (z \leq x) \rightarrow (z < y) \\ \mathbf{x} < n^a & \text{stands for } \bigwedge_{i \leq k-a} x_i = 0 \\ \ldots & \end{array}$$

Constructing the formula

Let $\mathcal{M} = (K, \Sigma, s, Acc, Rej, \delta)$.

The second order variables $\mathbf{X}$ appearing in $\varphi_{\mathcal{M},k}$ include S_s, T_a, and H. The formula $\varphi_{\mathcal{M},k}$ will enforce that they have the following meaning:

- $S_s(\mathbf{x})$
  "the state of the machine at time $\mathbf{x}$ is s"
- $T_a(\mathbf{x}, \mathbf{y})$
  "at time $\mathbf{x}$, the symbol at position $\mathbf{y}$ of the tape is a"
- $H(\mathbf{x}, \mathbf{y})$
  "at time $\mathbf{x}$, the tape head is pointing at tape cell $\mathbf{y}$"

Constructing the formula

1. initial state is s_0 and the head is initially at the beginning of the tape

$$\left(S_{s_0}(\mathbf{0}) \wedge H(\mathbf{0},\mathbf{0})\right)$$

2. the machine is never in two states at once

$$\forall \mathbf{x} \bigwedge_{s \in K} \left(S_s(\mathbf{x}) \rightarrow \bigwedge_{s' \neq s} \neg S_{s'}(\mathbf{x})\right)$$

3. the head is never in two places at once

$$\forall \mathbf{x} \forall y \left(H(\mathbf{x},\mathbf{y}) \rightarrow (\forall \mathbf{z}(\mathbf{y} \neq \mathbf{z}) \rightarrow \neg H(\mathbf{x},\mathbf{z}))\right)$$

4. each tape contains only one symbol

$$\forall \mathbf{x} \forall \mathbf{y} \bigwedge_{a \in \Sigma} T_a(\mathbf{x},\mathbf{y}) \rightarrow \bigwedge_{b \neq a} \neg T_b(\mathbf{x},\mathbf{y})$$

Constructing the formula

5. the tape does not change except under the head

$$\forall \mathbf{x} \forall \mathbf{y} \forall \mathbf{z} (\mathbf{y} \neq \mathbf{z} \rightarrow (\bigwedge_{a \in \Sigma} H(\mathbf{x}, \mathbf{y}) \wedge T_a(\mathbf{x}, \mathbf{z}) \rightarrow T_a(\mathbf{x}+1, \mathbf{z})))$$

6. each tape is according to δ

$$\begin{aligned} \forall \mathbf{x} \forall \mathbf{y} \quad & \Big(\textstyle\bigwedge_{a \in \Sigma} \bigwedge_{s \in K} H(\mathbf{x}, \mathbf{y}) \wedge S_s(\mathbf{x}) \wedge T_a(\mathbf{x}, \mathbf{y}) \Big) \\ & \rightarrow \textstyle\bigvee_{(s', b, D) \in \delta} \Big(H(\mathbf{x}+1, \mathbf{y}_D) \wedge S_b(\mathbf{x}+1) \wedge T_b(\mathbf{x}+1, \mathbf{y}) \Big) \end{aligned}$$

7. some accepting state is reached

$$\exists \mathbf{x} \bigvee_{s \in Acc} S_s(\mathbf{x})$$

8. the initial content of the tape is $\mathrm{enc}(\mathfrak{A}, <)$

Initial tape content

Remember that $\mathrm{enc}(\mathfrak{A}) = 0^n 1 \cdot \mathrm{enc}(R_1) \cdots \mathrm{enc}(R_m)$.
So we can express the property with

$$\begin{aligned} \forall \mathbf{x} \quad & \mathbf{x} < n \to T_0(\mathbf{1}, \mathbf{x}) \wedge T_1(\mathbf{1}, \mathbf{n}) \\ & \mathbf{x} \leq n^{k_1} \to \Big(T_1(\mathbf{1}, \mathbf{x} + n + 1) \leftrightarrow R_1(\mathbf{x}_{|k_1}) \Big) \\ & \dots \end{aligned}$$

where $\mathbf{x} = \mathbf{y} + n$ stands for

$$x_0 = y_0 \wedge \bigvee_{0<i<k-1} x_i = y_i + 1 \wedge \bigwedge_{0<j<i} x_j = 0 \wedge y_j = n - 1 \wedge \bigwedge_{i<j} x_j = y_j$$

The polynomial hierarchy

We can define further classes by allowing other second-order quantifier prefixes

- $\Sigma_1^1 = \mathbf{ESO}$ $(\exists^*)$ corresponds to **NP**
- $\Pi_1^1 = \mathbf{USO}$ $(\forall^*)$ corresponds to $co - \mathbf{NP}$, the class of problems that can be accepted by a demoniac nondeterministic machine
- Σ_{n+1}^1 is the collection of properties definable by a formula of the form $\exists^* \mathbf{X} \varphi$ with $\varphi \in \Pi_n^1$
- Σ_{n+1}^1 is the collection of properties definable by a formula of the form $\exists^* \mathbf{X} \varphi$ with $\varphi \in \Pi_n^1$
- $\mathbf{PH} = \bigcup_{i \geq 1} \Sigma_i^1 = \bigcup_{i \geq 1} \Pi_i^1$ is the polynomial hierarchy

Remarks

$\mathbf{NP} \subseteq \mathbf{PH} \subseteq \mathbf{PSPACE}$

$\mathbf{P} = \mathbf{NP}$ if, and only if, $\mathbf{P} = \mathbf{PH}$

Alternating Turing machines

An alternating Turing machine is a machine with both angelic and demoniac non-determinism.

$$\mathcal{M} = (K = K_\forall \uplus K_\exists, \Sigma, \delta, s_0, \text{Acc}, \text{Rej})$$

The run $\text{Run}(\mathcal{M}, w)$ is a two-player game between Ang$\exists$l and D$\forall$emon; w is accepted by $\mathcal{M}$ if Ang$\exists$l has a winning strategy

Standard theorems in computational complexity

Theorem (Chandra, Stockmeyer, Kozen)

for all $f(n) \geq \log(n)$.

$$\mathbf{ATIME}(f(n)) \subseteq \mathbf{DSPACE}(f(n))$$
$$\mathbf{ASPACE}(f(n)) = \bigcup_{c>0} \mathbf{DTIME}(2^{c \cdot f(n)})$$
$$\mathbf{NSPACE}(f(n)) \subseteq \bigcup_{c>0} \mathbf{ATIME}(c \cdot f(n)^2)$$

Theorem (Savitch)

$$\mathbf{NSPACE}(f(n)) \subseteq \mathbf{DSPACE}(f(n)^2)$$

$$\mathbf{L} \subseteq \mathbf{NL} \subseteq \mathbf{AL} = \mathbf{P} \subseteq \mathbf{NP} \subseteq \mathbf{PH} \subseteq \mathbf{AP} = \mathbf{PSPACE} = \mathbf{NPSPACE}$$

FO + LFP

Let $\varphi(\mathbf{x}, R)$ be a formula that is monotone in R, i.e.

$$R \subseteq R' \qquad \text{implies} \qquad \varphi(\mathbf{x}, R) \to \varphi(\mathbf{x}, R')$$

then by Knaster-Tarski fixed point theorem there is a unique R_ω such that

$$\forall \mathbf{x} \quad R_\omega(\mathbf{x}) \leftrightarrow \varphi(\mathbf{x}, R_\omega)$$

Moreover, over a finite structure $\mathfrak{A}$ with $|\mathfrak{A}| = n$,

$$R_0 = \bot \qquad R_{i+1}(\mathbf{x}) = \varphi(\mathbf{x}, R_i)$$

stabilizes after n^k steps, where k is the arity of $\mathbf{x}$

We write $\mathbf{LFP}[\mathbf{x}, R]\varphi$ for R_ω.

FO + LFP is the logic that extends **FO** with **LFP**

Examples

Reachability

$$E^* = \mathbf{LFP}[x, y, R]\big(x = y \vee \exists z\ E(x, z) \wedge R(z, y)\big)$$

Reachability game

Let $\sigma = (\text{Angel}, \text{Demon}, E)$ be the signature of arenas (game graphs).

$$\begin{aligned}\mathbf{LFP}[x, y, R]\big(\quad & x = y \qquad \vee \\ & \text{Angel}(x) \wedge \exists z\ E(x, z) \wedge R(z, y) \qquad \vee \\ & \text{Demon}(x) \wedge \forall z\ R(x, z) \rightarrow R(z, y)\big)\end{aligned}$$

is true for (x, y) if Angel has a strategy for reaching y starting from x

Capturing **P** over ordered structures

Assume $\sigma = \{<, \dots\}$. A structure $\mathfrak{A}$ is ordered if $<^{\mathfrak{A}}$ is a total order on A.

Theorem (Immerman, Vardi)

FO$[<] + $**LFP** captures **P**: for all class $\mathcal{C}$ of ordered structures, the following two are equivalent

1. $\mathcal{C} = \mathrm{Mod}(\varphi)$ for some φ in **FO**$[<] + $**LFP**
2. $\{\mathrm{enc}(\mathfrak{A}) \mid \mathfrak{A} \in \mathcal{C}\}$ is in **P**

(1) $\rightarrow$ (2) is by finite fixed point iteration
(2) $\rightarrow$ (1) : since **P** = **ASPACE**$(\log(n))$, we need to encode the existence of a winning strategy for Angel in an alternating Turing machine with logarithmic space. Since the machine uses logarithmic space, a configuration can be coded as a tuple **x**, and therefore the game graph can be coded by a **FO**$[<]$ formula.

NL and FO + TC

the extension of **FO** with transitive closure is defined by

$$\mathbf{TC}[\mathbf{x},\mathbf{y},\varphi] = \mathbf{LFP}[\mathbf{x},\mathbf{y},R]\big(\mathbf{x}=\mathbf{y} \vee \exists \mathbf{z}\varphi(\mathbf{x},\mathbf{z}) \wedge R(\mathbf{z},\mathbf{y})\big)$$

Theorem (Immerman, Szelepcsényi)

FO$[<] + $ **TC** captures **NL**, and as a corollary, **NL** = **coNL**.

Similar encoding of a nondeterministic logspace Turing machine. Deciding reachability in logspace : the machine guesses a path and remembers how many nodes it visited.

The difficult point is negation: how to decide non-reachability in non-deterministic logspace? nice trick there!

Sources and references

On complexity theory

- **C.H. Papadimitriou** Computational Complexity. Addison-Wesley. 1994
- **S. Arora and B. Barak** Computational Complexity. CUP. 2009.
- **S. Perifel** Complexité algorithmique. Ellipses. 2014.

On descriptive complexity

- **L. Libkin** Elements of finite Model Theory. Springer) -2012 (online)
- **N. Immerman** Descriptive Complexity. Springer. 2009.
- **Anuj Dawar** Topics in Logic and Complexity (slides) Lecture in Cambridge
  this presentation owes a lot to his lecture