Formal Specification Techniques

References for study:

- D. Basin, *Formal Methods for Security*, CyBOK 1.1, 2021, Chapter 2

Formal Methods

- **Formal specification**
  - Build a formal (mathematical) model (abstraction) of a system
  - Can be used at different development stages
    - Requirements specification, structural and behavioral specifications at different abstraction levels (at different design stages)
- **Formal verification**
  - Check the self-consistency of formal models
  - Check that a formal behavioral model satisfies its formal requirements specification
  - Check the cross-consistency of two formal behavioral models (at different abstraction levels)

Formal Specifications

- Characteristics of a formal specification
  - **Unambiguous** (no multiple interpretations)
  - **Consistent** (no internal contradictions)
  - **Complete** (all **relevant** information represented)
- Examples
  - Combinational circuit -> boolean function
  - Sequential circuit -> Finite State Machine
  - Less immediate for software and systems

Formal Specification Styles (behavioral models)

- Operational (imperative)
  - description of the system actions (e.g. state machine)

Especially used for system design/impl. models

- Descriptive (declarative)
  - description of the system properties (e.g. square function with input x, output y: $y = x^2$)

Especially used for system requirements models

System Behavior Classification

- **Computational (or transformational) systems**
  - Their task is to receive some input X and produce a corresponding output Y. After having finished this task, they terminate.
  - Operationally, they can be described by an algorithm (compute Y from X)
  - Descriptively, they can be described by the mathematical function or relationship that binds Y to X

System Behavior Classification

- **Reactive systems**
  - Their task is to interact in a predefined way with their environment (respecting some temporal constraints). They may not terminate.
  - Their specification must describe how they interact with the environment (the possible sequences of interactions)
  - Operationally, they can be described by a state machine (state-transition model)
  - Descriptively, they can be described by (temporal) logic formulas (more on this later on)

Discussion

- This classification has nothing to do with the distinction between concurrent and sequential systems.
- Reactive systems are a superclass of computational systems

  = > Specification techniques for reactive systems are themselves a superclass of specification techniques for computational systems

State-Transition Models

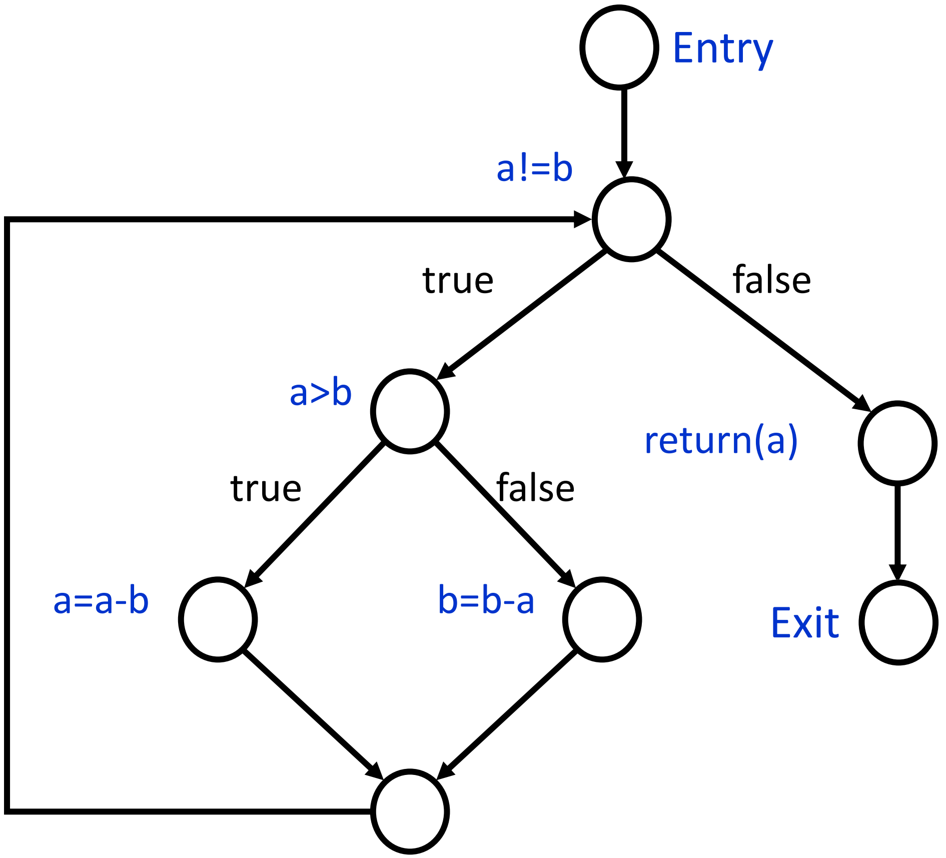
- **Transition System (TS):** (**S, init,** ρ)
  - **S**: set of states
  - **init**: initial state (init $\in$ S)
  - ρ: transition relation ($\rho \subseteq S \times S$)
- **Labelled Transition System** (**LTS): (S, init, L,** ρ)
  - **S**: set of states
  - **init**: initial state (init $\in$ S)
  - **L**: set of labels (events)
  - ρ: transition relation ($\rho \subseteq S \times L \times S$)

Example: State-Transition model of a sequential program execution

- The control flow of a sequential program can be described by a Control Flow Graph (CFG)
  - a directed graph including entry, exit, assignment and test vertices).

Example of Control Flow Graph

```
int f(int a, int b)
{
  while (a!=b) {
    if (a>b)
      a = a-b;
    else
      b = b-a;
  }
  return(a);
}
```



Only captures **control flow** of a **single sequential** program

Modeling Variables

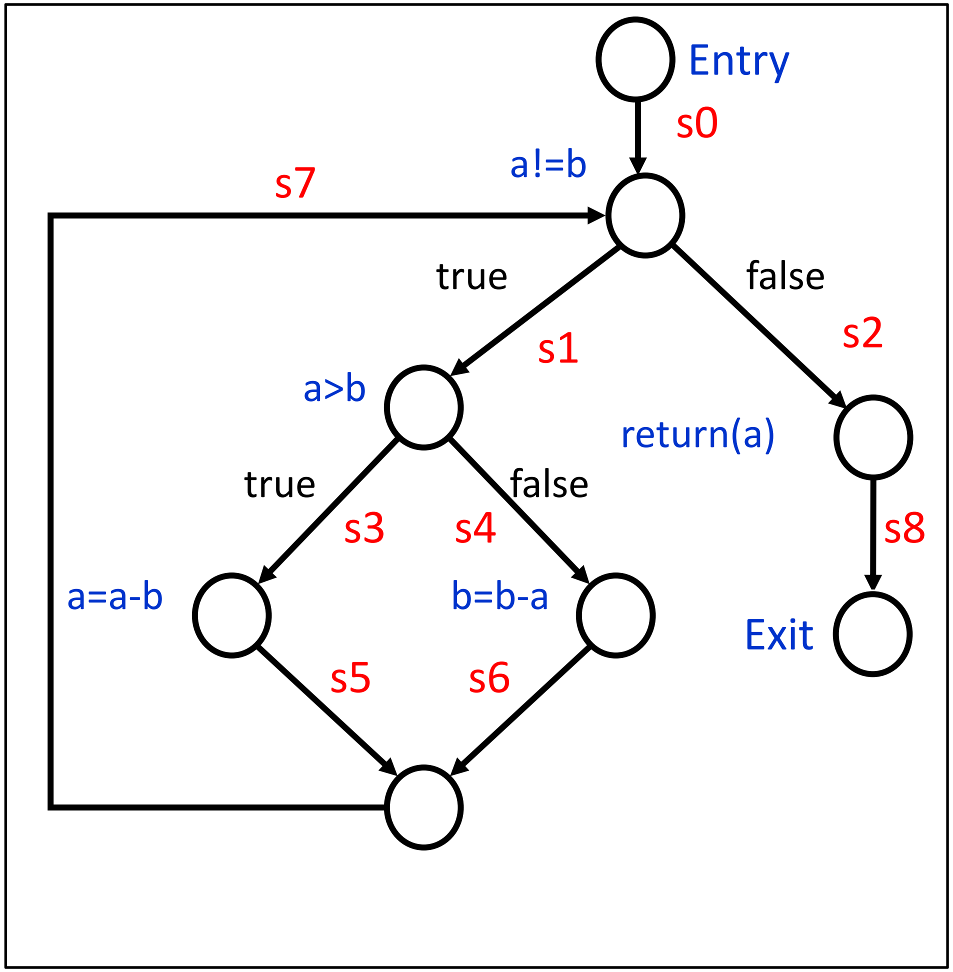
- The possible contents of variables can be modeled as elements of sets
- Examples:
  - a single int variable => modeled by the set of integers N
  - two int variables a and b => modeled by the cartesian product NxN
- In general, V: set of all possible contents of variables

Putting all together

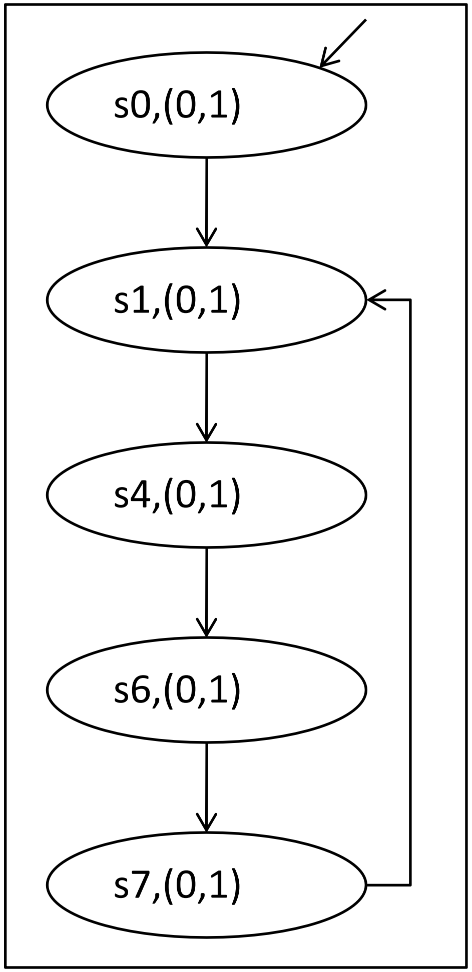
- The full model of the program is a TS: (S, init, ρ), where:
  - States (S): set of pairs <s,v> where
    - s: control state (an edge of the CFG)
    - v: contents of variables (an element of V)
  - initial state (init): <s0, v0i> where
    - s0: the edge outgoing from the entry vertex
    - v0i: an element of V (e.g. the element of V corresponding to “all variables not initialized”)
  - state transitions (ρ): determined by the semantics of program statements

Example with initial values 0,1

CFG



TS



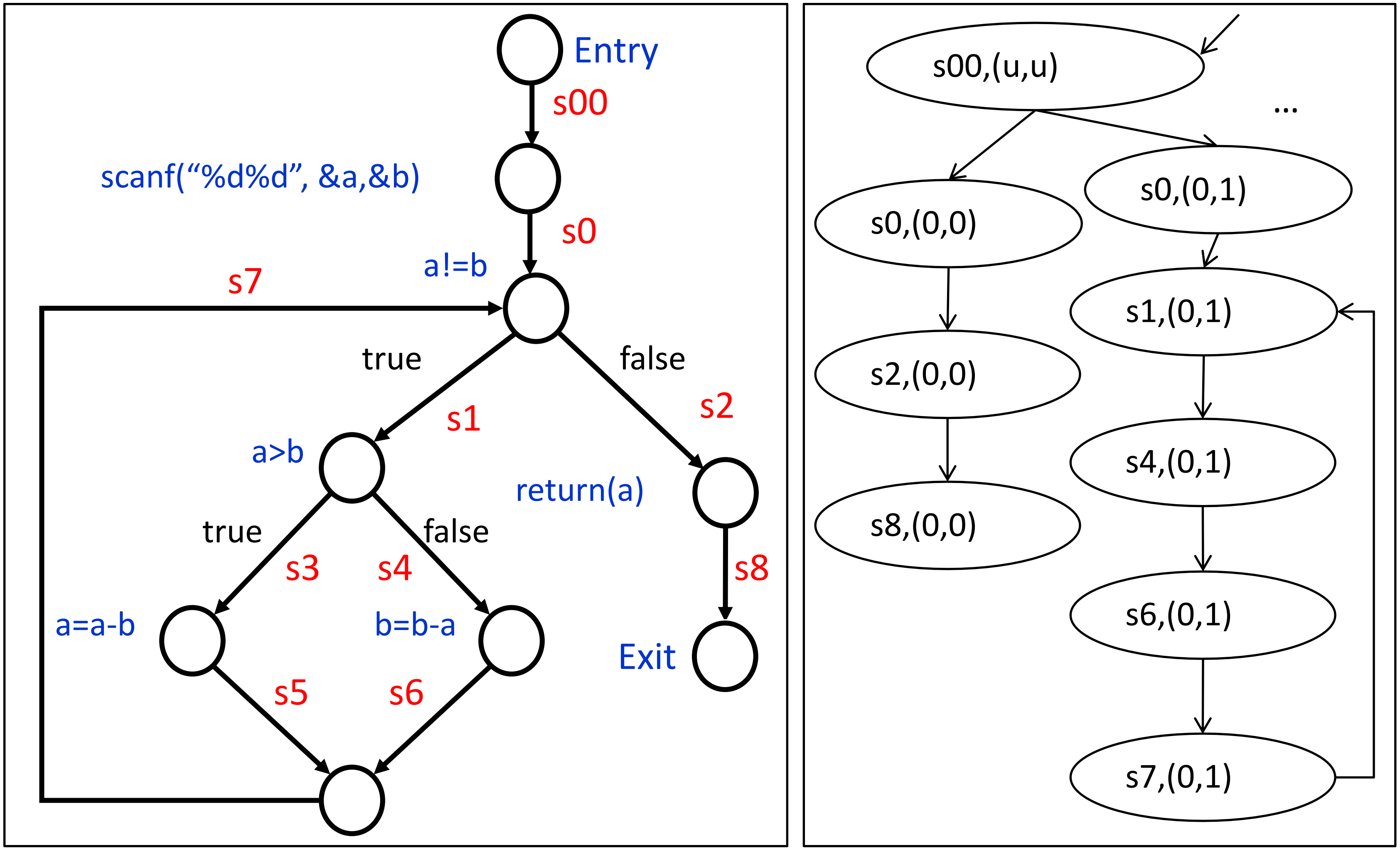
Non-determinism

- Abstract models use non-determinism for representing execution aspects that are not known a-priori
  - The inputs of a program
  - how concurrent processes are scheduled by the operating system
- Non-determinism is also used to represent different possible implementation choices

Example with nondeterminism

CFG

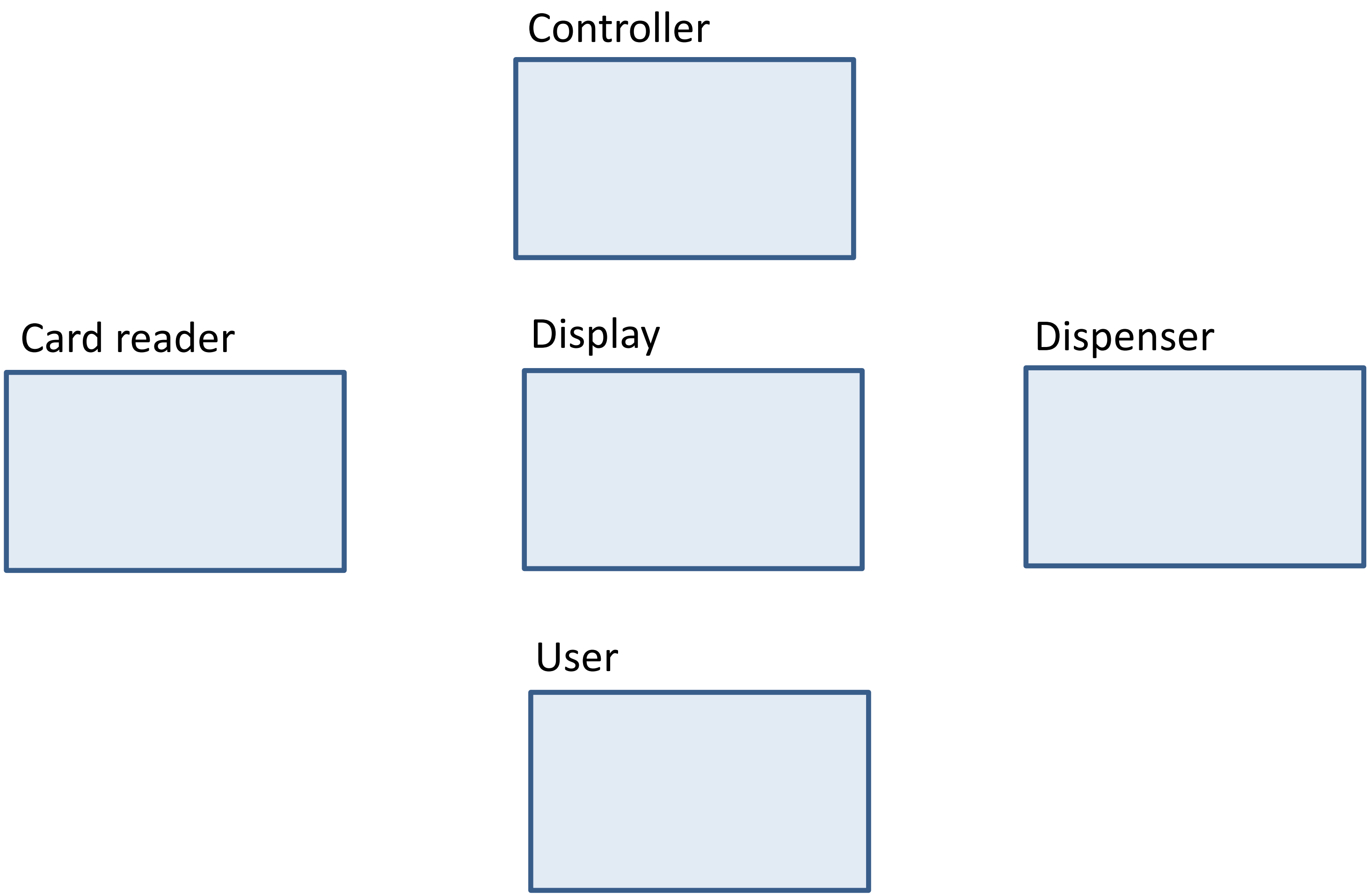
TS



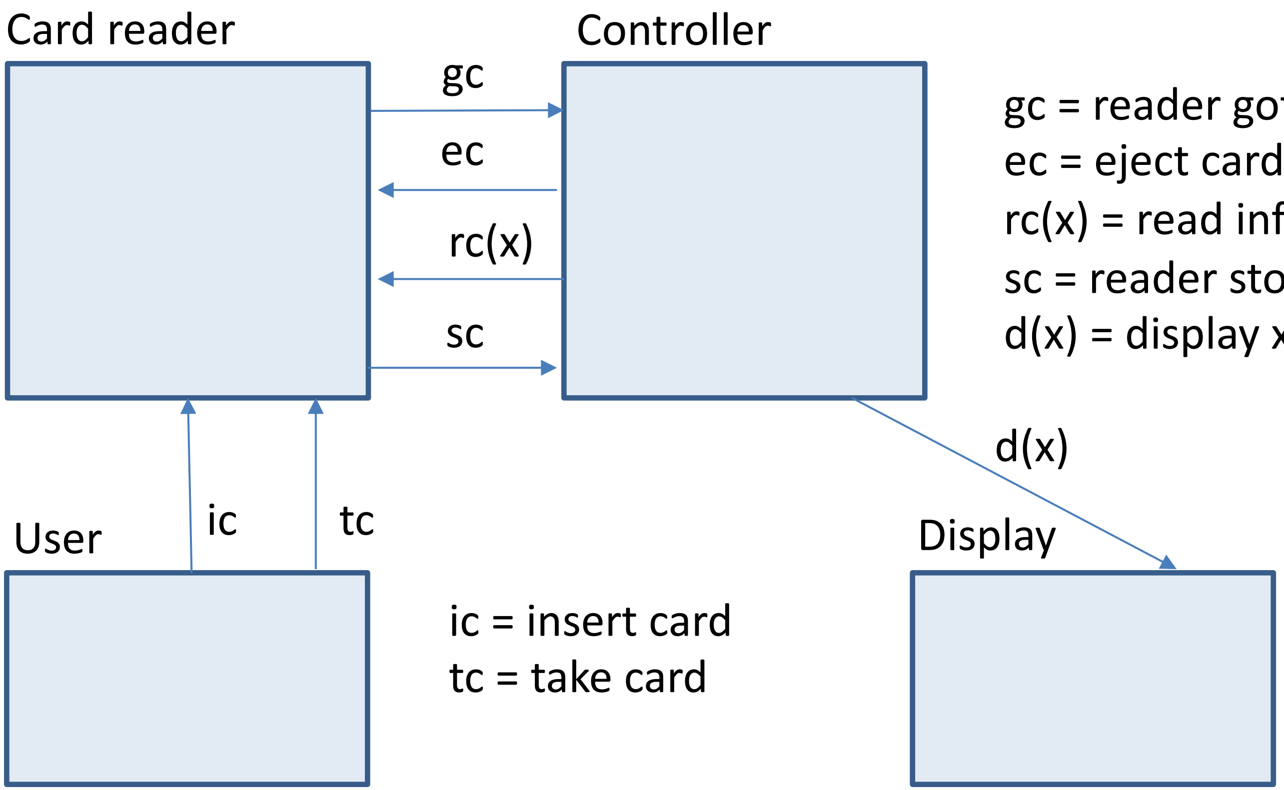
Example: State-transition model of a **concurrent** program execution

- Each sequential process that is part of a concurrent system can be represented by a TS
- The whole concurrent system can be represented by a product TS:
  - Set of states S=S1 x S2 X…
  - Initial state: init=<init1,init2,…>
  - Transition relation: ρ(<s1,s2,…>,<s1',s2',…>) iff ρi(si,si') for some i, and si=si' for the others.

Example:
Operational Model of an ATM as LTS

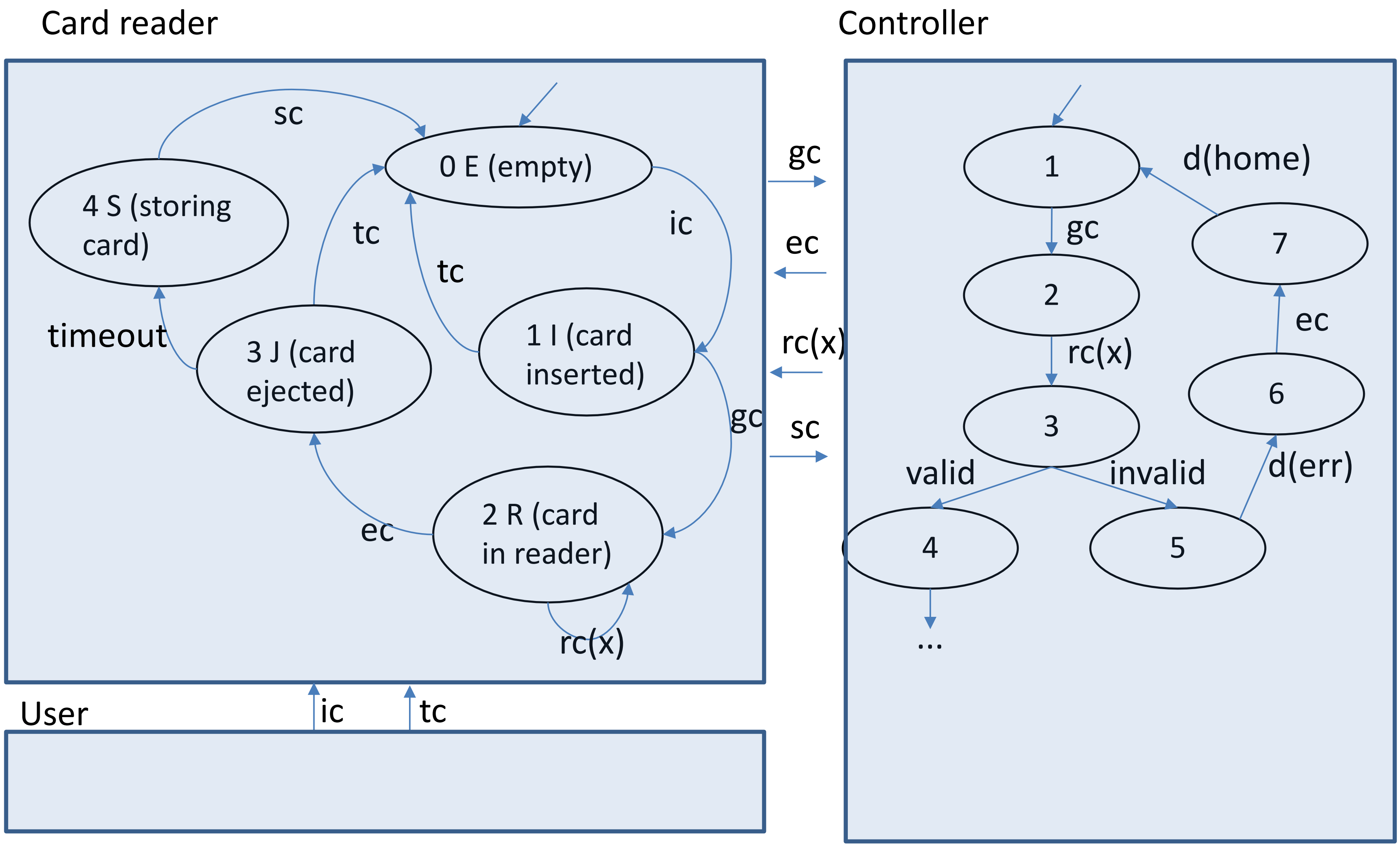


Events of Some Model Processes

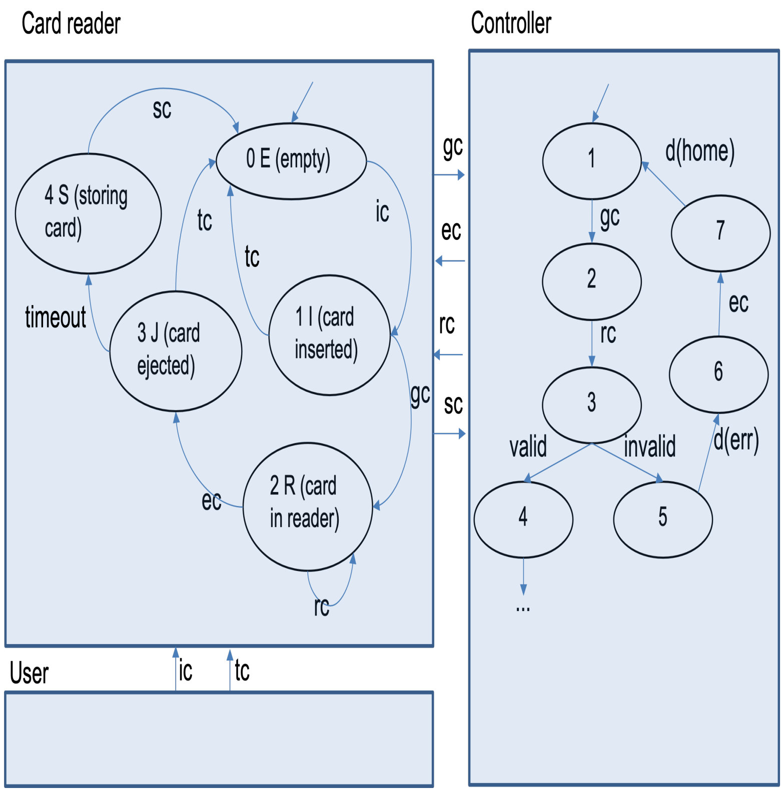


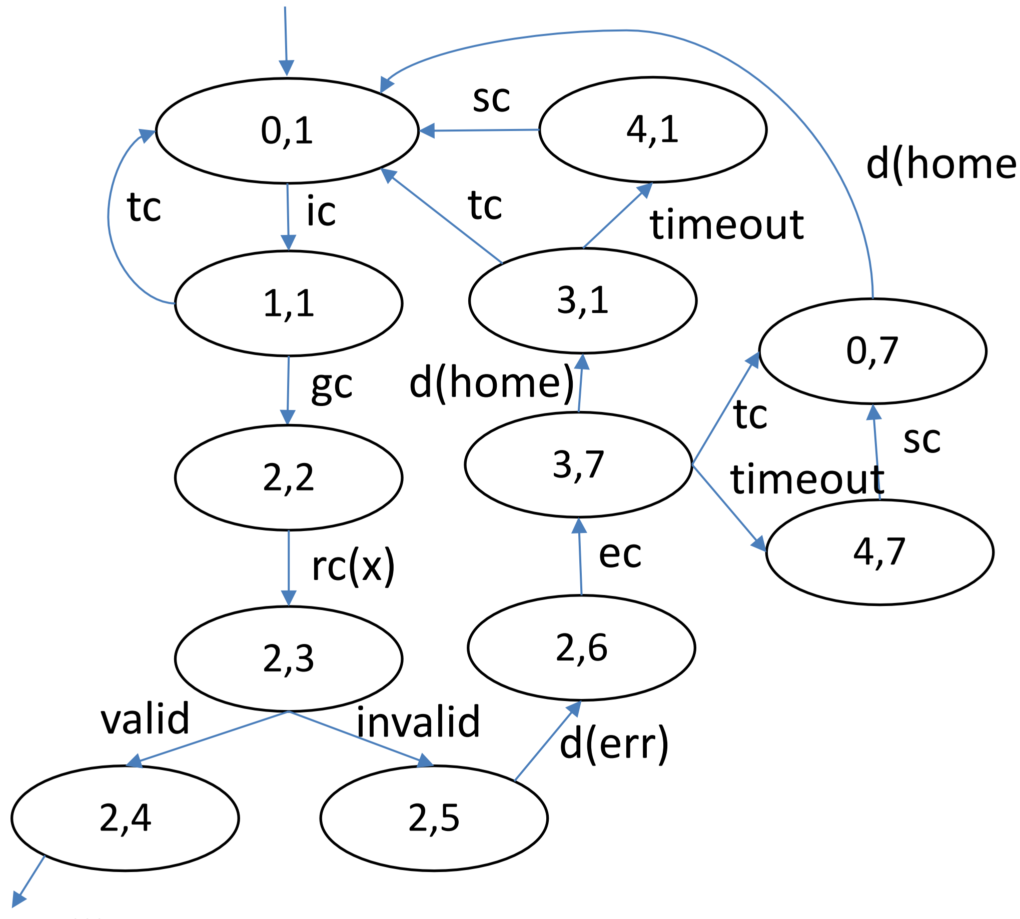
gc = reader got card
ec = eject card
rc(x) = read info x from card
sc = reader stored card
d(x) = display x

ic = insert card
tc = take card


Card reader
Controller
sc
0 E (empty)
4 S (storing card)
tc
tc
ic
1 I (card inserted)
timeout
3 J (card ejected)
gc
ec
2 R (card in reader)
rc(x)
gc
ec
rc(x)
sc
1
d(home)
gc
7
2
ec
rc(x)
6
3
valid
invalid
d(err)
4
5
...
User
ic
tc

Overall LTS (Product LTS)





State Explosion

- Concurrency tends to make the number of states/transitions explode

  => strategies are necessary to manage the issue
  (more on this later on)

Descriptive Formal Specifications

- A formal model of a system property can be expressed by a **logic formula**
- Different logics can be used for this purpose
  - Propositional logic
  - Predicate (I order) logic
  - Temporal logics
  - I order logics (specializations of predicate logic)

The basis for other more specialized logics

Propositional Logic

- A possible minimal definition:
  - Syntax:

Example:
$(P \wedge \neg Q) \Rightarrow \neg R$

$$
\begin{aligned}
\mathit{formula} ::= \quad & P \mid Q \mid R \mid \ldots \quad \text{(atomic propositions)} \\
& \mid \neg\, \mathit{formula} \\
& \mid \mathit{formula} \vee \mathit{formula} \\
& \mid (\, \mathit{formula}\,)
\end{aligned}
$$

$f1 \wedge f2 \equiv \neg\, ((\neg\, f1) \vee (\neg\, f2))$

$f1 \Rightarrow f2 \equiv (\neg\, f1) \vee f2$

$f1 \Leftrightarrow f2 \equiv (f1 \Rightarrow f2) \wedge (f2 \Rightarrow f1)$

...

Propositional Logic

– Semantics (interpretation):

- Interpretation of atomic propositions:

  can be formalized as a function I: AP $\rightarrow$ {F,T}

- Interpretation of operators

  can be formalized as boolean functions (truth tables)

f	$\neg$ f
F	T
T	F

f1	f2	f1 $\vee$ f2
F	F	F
F	T	T
T	F	T
T	T	T

We write I $\models$ f to mean f is true with interpretation I

Abstract Reasoning

The interpretation of operators lets us reason *independently* of the interpretation of AP:

- **Tautology**: formula that is always true (independently of how APs are interpreted)
  – Examples: $Q \Rightarrow (P \Rightarrow Q)$ $\quad$ $P \vee (\neg P)$
- **Contradiction**: formula that is always false (it is the negation of a tautology)
  – Examples: $P \wedge (\neg P)$

Satisfiability and Validity

- A formula is said **satisfiable** if it is true for at least one interpretation of APs
- A formula is said **valid** if it is true for all interpretations of APs (i.e., it is a tautology)
- Duality of validity and satisfiability:

$$f \text{ is valid} \quad \Leftrightarrow \quad \neg f \text{ is not satisfiable}$$

Predicate Logic (I order logic)

- An extension of propositional logic where:
  - Atomic propositions are replaced by *predicates*
  - the new concepts of **constant**, variable, function, relation and the $\forall$ and $\exists$ quantifiers are introduced
- Formula sample:

$$\forall\, k\ (\ (\mathbf{1} \leq k \leq n) \Rightarrow (v(k) < v(k+\mathbf{1}))\ )$$

Predicate Logic (I order logic)

- **Minimal syntax**

data

term ::=	**a**	(**a** constant)
	| x	(x variable)
	| f(*term*, … ,*term*)	(f function)
	| (*term*)	

assertions

atomic formula ::=	A(*term*, …, *term*)	(A predicate)
formula ::=	*atomic formula*	
	| $\neg$ *formula*	
	| *formula* $\vee$ *formula*	
	| ($\forall$ x) *formula*	(x variaile)
	| (*formula*)	

n is free

Example:

$\forall$ k ((**1**$\leq$ k $\leq$ n) $\Rightarrow$ (v(k) < v(k+**1**)))

k is bound

Predicate Logic (I order logic)

- **Derived formulas**

$$f1 \wedge f2 \equiv \neg ((\neg f1) \vee (\neg f2))$$

$$f1 \Rightarrow f2 \equiv (\neg f1) \vee f2$$

$$f1 \Leftrightarrow f2 \equiv (f1 \Rightarrow f2) \wedge (f2 \Rightarrow f1)$$

…

$$(\exists x) f \equiv \neg ((\forall x) (\neg f))$$

Predicate Logic (I order logic)

- **Semantics (Interpretation)**

  Domain (set D of the possible values of terms)

  Interpretation of constants (function C $\rightarrow$ D)

  Interpretation of functions (function F $\rightarrow$ fun(D))

  Interpretation of predicates (function P $\rightarrow$ rel(D))

  Interpretation of logical connectives

  - same as in propositional logic

  Interpretation of ($\forall$ x) f

  - true iff f is true for any substitution of x in f with any term

Predicate Logic (I order logic)

- For closed formulas (without free variables)
  - Interpretation maps each formula onto an element of {F,T}
- For open formulas (with n free variables)
  - Interpretation maps each formula onto a relation on D^n

Another possible formalization of a logic: A Formal System

- A **Formal System (Theory)** is defined by:
  - A **formal language**
    - An alphabet of symbols
    - A set of well formed formulas (sequences of symbols belonging to the language)
  - A **deductive apparatus** (or deductive system)
    - A set of **axioms** (formulas to which the true value is assigned axiomatically)
    - A set of **inference rules** (each one expressing that a certain formula is a *direct consequence* of certain other formulas)

What formulas can I write?

How do I give a truth value to a formula?

Example: A possible formal system for propositional logic (Lukasiewicz)

- Formal language: propositional logic syntax, with the only two primitive operators $\Rightarrow$ $\neg$
- Deductive apparatus: Axioms

  A1) $f1 \Rightarrow (f2 \Rightarrow f1)$

  A2) $(f1 \Rightarrow (f2 \Rightarrow f3)) \Rightarrow ((f1 \Rightarrow f2) \Rightarrow (f1 \Rightarrow f3))$

  A3) $(\neg f2 \Rightarrow \neg f1) \Rightarrow (f1 \Rightarrow f2)$
- Deductive apparatus: Inference rules

  I1) $\dfrac{f1,\ f1 \Rightarrow f2}{f2}$ (modus ponens)

Theorems and Proofs

- **Proof**

  a sequence of wff f1,…,fn such that, for each i, fi is an axiom or it is a direct consequence of some of the preceding formulas, according to an inference rule.

- **Theorem**

  a wff f such that there exists a proof that terminates with f

We write $\vdash f$ to mean f is a consequence of the Formal System axioms and rules (a theorem)

Example: Theorem and Proof

proof

1. $P \Rightarrow ((Q \Rightarrow P) \Rightarrow P)$
2. $(P \Rightarrow ((Q \Rightarrow P) \Rightarrow P) \Rightarrow ((P \Rightarrow (Q \Rightarrow P)) \Rightarrow (P \Rightarrow P))$
3. $(P \Rightarrow (Q \Rightarrow P)) \Rightarrow (P \Rightarrow P)$
4. $P \Rightarrow (Q \Rightarrow P)$
5. $\boxed{P \Rightarrow P}$

A1 with f1=P and f2=Q⇒P

A2 with f1=P, f2=Q⇒P, and f3=P

from 1. and 2. by modus ponens

A1 with f1=P and f2=Q

from 4. and 3. by modus ponens

theorem

$$\vdash (P \Rightarrow P)$$

Temporal properties

- Proposition and predicate logics describe **static facts** (immutable in time)
- Instead, the facts related to a program execution or to a dynamic system are typically **time-varying**
- If we refer to a particular state (e.g. the final state of a program run) static properties are adequate, otherwise temporal properties are necessary.

Temporal Properties

- Examples:
  - Variable x takes positive value during the whole program execution
  - It is not possible that, during any session of the ATM (i.e. between the time when the card is inserted and the time when the home page is displayed), a user gets money without having inserted the right pin code
  - The time between the start of a purchase operation and the end of the same operation must be less than 30 seconds

Possible solutions

- Use predicate logic with a variable t interpreted as (continuous or discrete) time

  Example:

  $\forall t\ (x(0)>0 \Rightarrow x(t)>0)$

- Use a specialized logic (temporal logic)

Temporal Logics

- Extensions of classical logics that also let the temporal evolution of facts to be described
- Can be defined in various ways
  - Propositional vs I order **logic**
  - Discrete vs Continuous, Implicit vs Real, Linear vs Branching **time**
  - Event vs State, Instant vs Interval, Past vs Future **modalities**

LTL (Linear Temporal Logic)

- The main temporal operators of LTL are:

  ○ (X) Next

  ○ f : f is true in the next state

  [] (G) Always in the future (globally)

  [] f : f is true in all future states

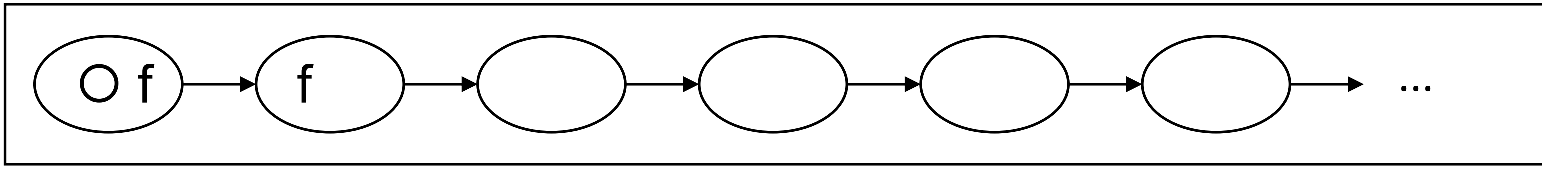
  ◊ (F) Eventually in the future

  ◊ f : f is true at least in one future state

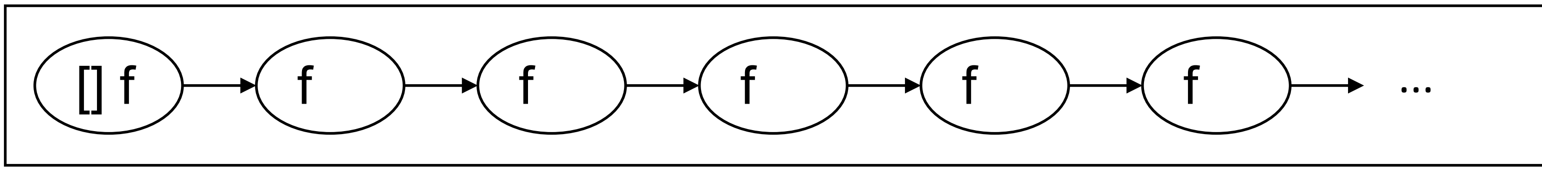
  U Until

  f1 U f2 : f1 keeps true until f2 becomes true

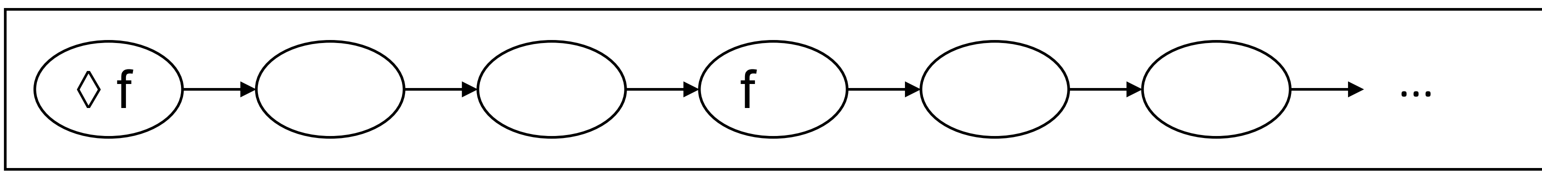
$\bigcirc f$: f is true in next state



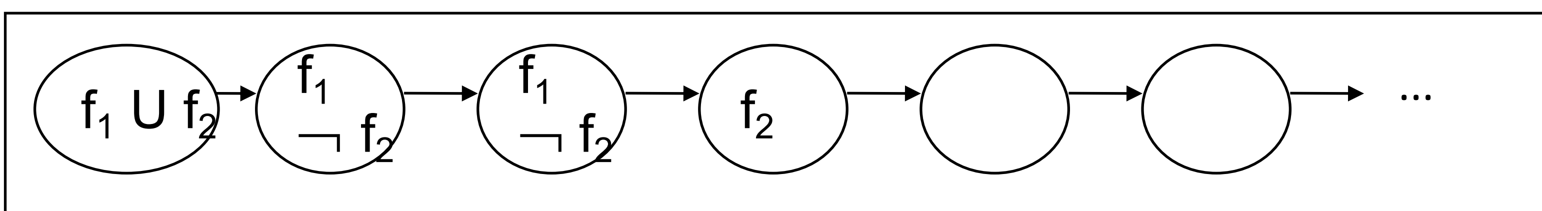
[] f : f is true in all future states



$\Diamond f$: f is true at least in one future state



$f_1 \; U \; f_2$: f1 keeps true until f2 becomes true



LTL (Linear Temporal Logic)

- Minimal syntax

formula ::= P | Q | R | ... (atomic propositions)
| $\neg$ *formula*
| *formula* $\vee$ *formula*
| $\bigcirc$ *formula*
| U *formula*
| (*formula*)

$f1 \wedge f2 \equiv \neg ((\neg f1) \vee (\neg f2))$

...

$\Diamond f \equiv T \; U \; f$

$[] f \equiv \neg \Diamond \neg f$

LTL (Linear Temporal Logic)

- Semantics (Interpretation):
  - Kripke Structure $K=(\underbrace{S,init,\rho}_{TS},I)$

    Interpretation of APs
    $I: S \times AP \rightarrow \{T,F\}$

  - K defines paths: linear sequences of states bound by the transition relation ρ
  - Formula f is true for interpretation K iff f is true for each path π of K

    $(K \models f) \Leftrightarrow (\pi \models f$ for each path π of $K)$

LTL (Linear Temporal Logic)

– For each path π of K=(S,init,ρ,I) :

$\pi \models P$	iff	P is true in **first** state of π according to I
$\pi \models \bigcirc f$	iff	f is true in sub-path of π starting at **second** state of π
$\pi \models f_1 \, U \, f_2$	iff	f_1 is true for all sub-paths of π starting at **first k** states of π and f_2 is true for all sub-paths of π starting at states of π **after the first k**

– Boolean operators are interpreted by the usual truth tables

Examples

- Variabile x has positive value during the whole program execution

  [] x_positive

- After a card has been inserted, if the user does not remove the card, the card is stored by the card reader

  $[]\ (\ (ic \wedge \neg(\Diamond\ tc\ )\ ) \Rightarrow (\Diamond\ sc\ )\ )$