Tecniche di Specifica e di Verifica

Modeling with Transition Systems

An example

The Dining Philosophers

- Possible problems:
  - ***Deadlock***: system state where no action can be taken (no transition possible)
  - ***Livelock***: When system component is prevented to take any action, or a particular one (individual starvation)
  - ***Starvation***: obvious.

Fairness

The Dining Philosophers

- Possible solution to deadlock:
  - ***pick up right fork only if both are present***

  Assumptions:
  - ***weak fairness***: any trans. continuously enabled, will eventually fire (eating philosophers will finish)
  - ***strong fairness***: any trans. enabled infinitely often, will eventually occur (if 2 fork available infinitely often, phil. will eventually eat).

Livelock

The Dining Philosophers

- Possible solution:
  - ***pick up fork only if both are present***

  Assumptions:
  - ***strong fairness***: any transition enabled infinitely often, will eventually occur (if 2 fork available infinitely often, philosopher will eventually eat).

  strong fairness is not enough to prevent ***livelock***

  Why? Think of the case with 4 philosophers!

  Sol.(?): Try ***preventing consecutive eating***.

  Still suffers from ***livelock*** with 5 phils! ***Why***?

Outline

- The model – Transition systems
- Some features
  - Paths
  - Computations
  - Branching
- First order representation

Transition systems

- A transition system (***Kripke structure***) is a structure

$$\mathbf{TS = (S, S_0, R)}$$

where:

- **S** is a finite set of states.
- $\mathbf{S_0}$ Í **S** is the set of initial states.
- **R** Í **S** ´ **S** is a transition relation
  - **R** must be ***total***, that is
    - "**s** Î **S** $**s'** Î **S** . **(s, s')** Î **R** or, equivalently,
    - For every state **s** in **S**, there exists **s'** in **S** such that **(s, s')** is in **R**.

Notions and Notations

- **TS = (S, S_0, R)**
- **(s, s') Î R R(s, s') s ® s'**
- A (finite) ***path*** from **s** is a sequence

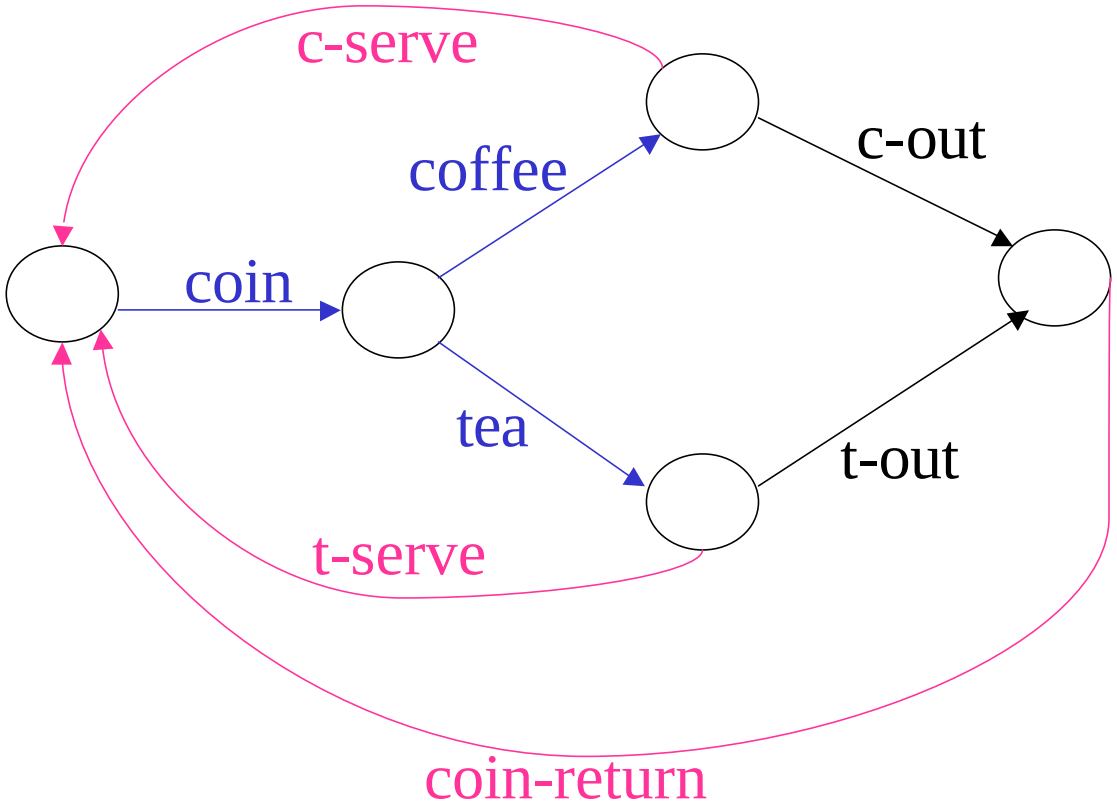
  $s_1, s_2, \ldots, s_n$

  such that
  - **$s = s_1$**
  - **s_i ® s_{i+1}** for $0 < i < n$.
- It is from **s** to **s'** if **$s_n = s'$**.
- An **infinite** path from **s** is an ***infinite sequence*** …..

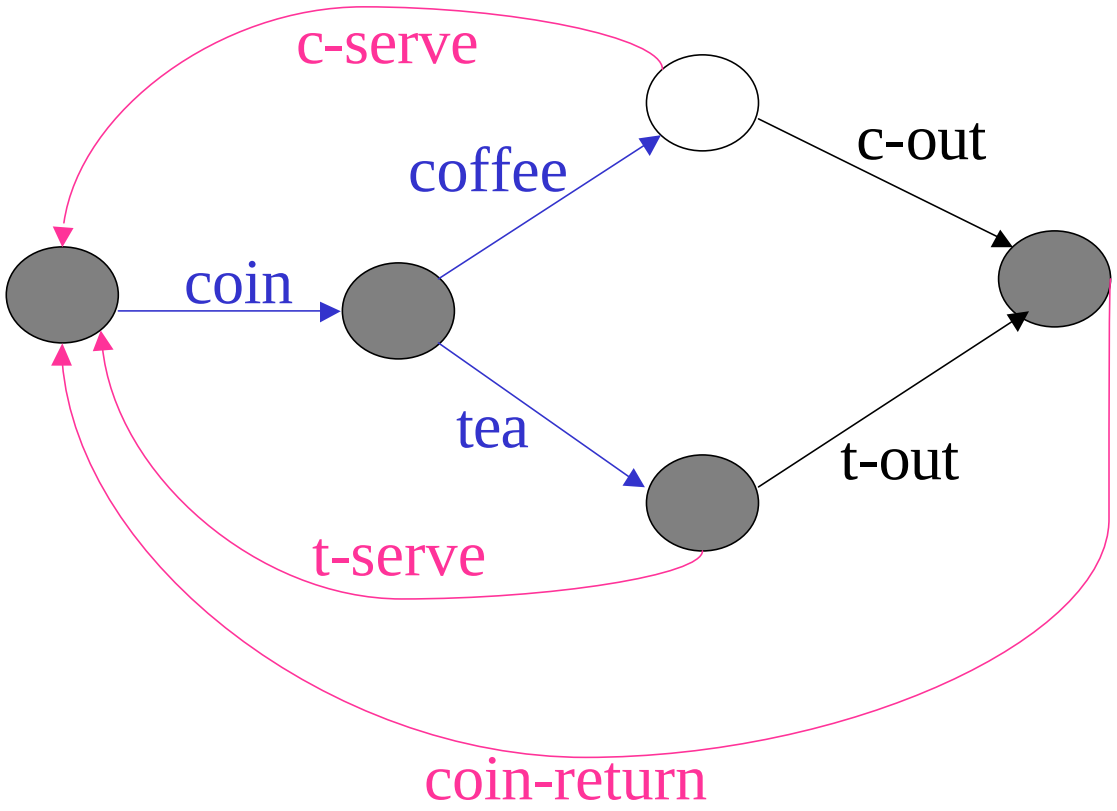
Labeled transition systems

- Sometimes we may use a finite set of actions:
  - **Act = {a, b, ..}**
- The actions will be used to label the transitions.
- **TS = (S, S_0, Act, R)**
  - **R Í S ´ Act ´ S**, labeled transitions.
  - **(s, a, s') Î R - R(s, a, s') - s $\xrightarrow{a}$ s'**

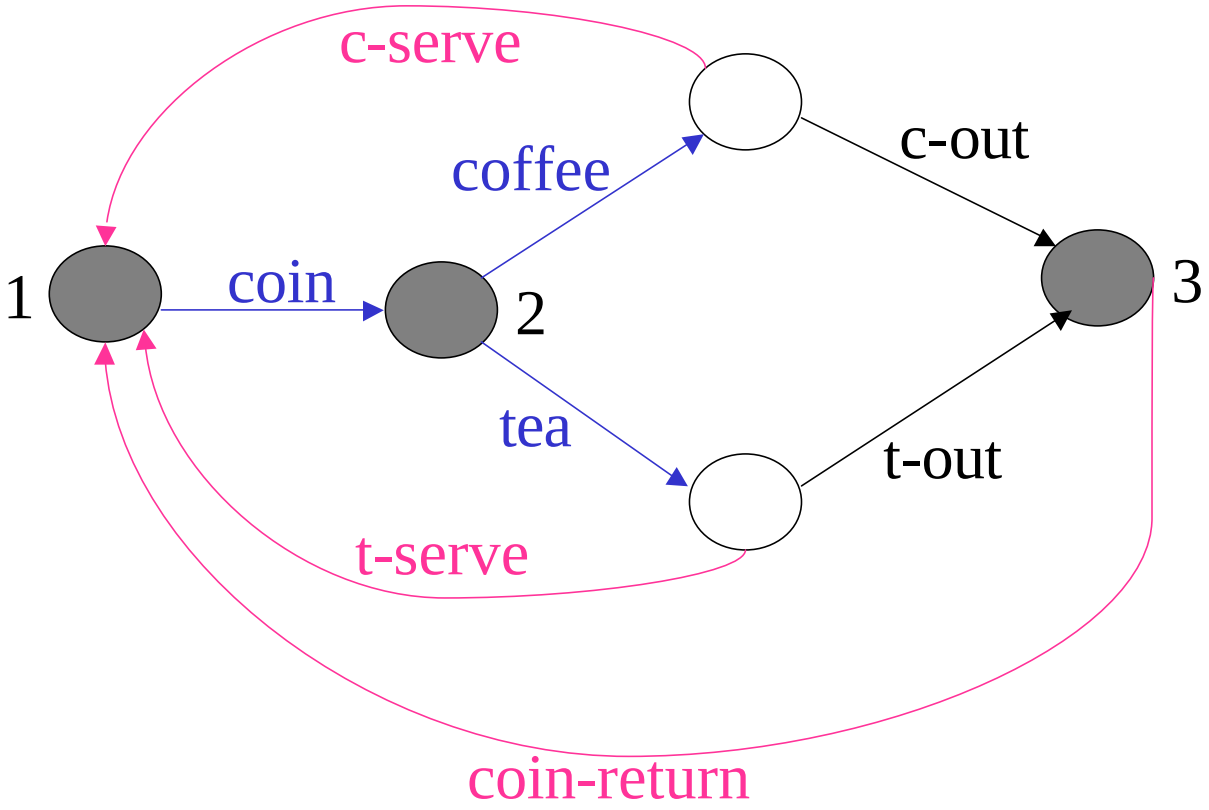
A vending machine



A path



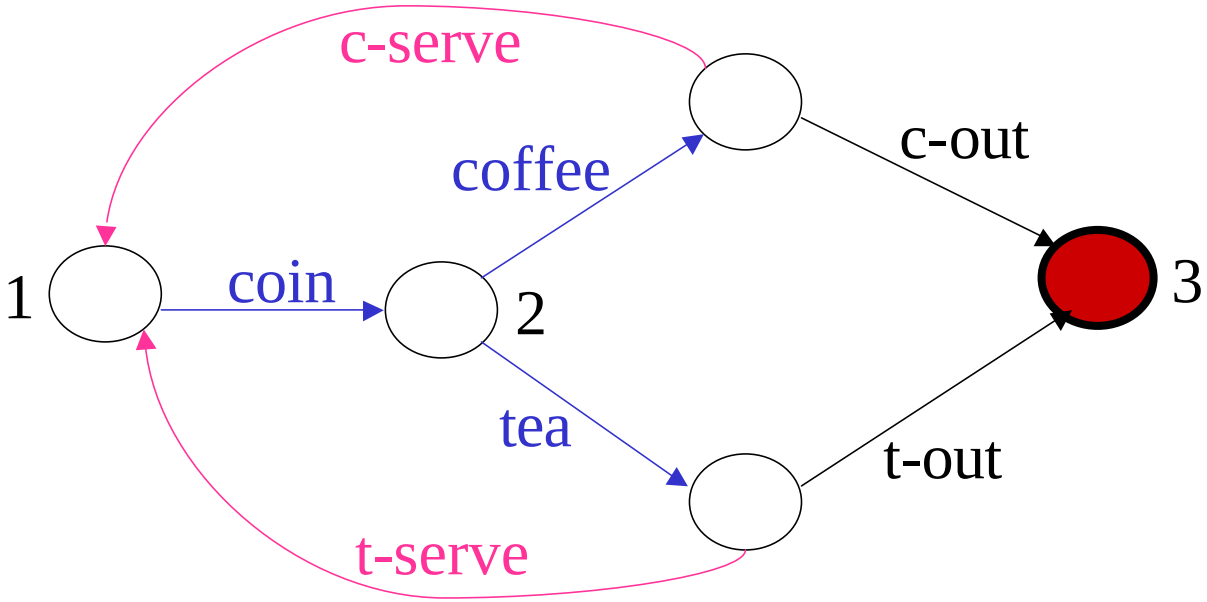
A non-path



1 2 3 No!

3 1 2 yes!

A non-total transition relation



State space

- The ***state space*** of a system (e.g. program) is the set of all its possible states.
- For example, if **V={a, b, c}** and the variables range over the naturals, then the ***state space*** includes:

  <a=0,b=0,c=0>, <a=1,b=0,c=0>, <a=1,b=1,c=0>, <a=932,b=5609,c=6658>…

Atomic transition

- Each ***atomic transition*** represents a small peace of code (or execution step), such that no smaller peace of code (or step) is observable.
- Is a:=a+1 atomic?
- In some systems, e.g., when a is a register and the transition is executed using an inc command.

(Non)Atomicity (race conditions)

- Execute the following when **x=0** in two concurrent processes:

  P1:a=a+1

  P2:a=a+1

- Result: **a=2**.
- Is this always the case?

- Consider the actual translation:

P1:load R1,a
inc R1
store R1,a

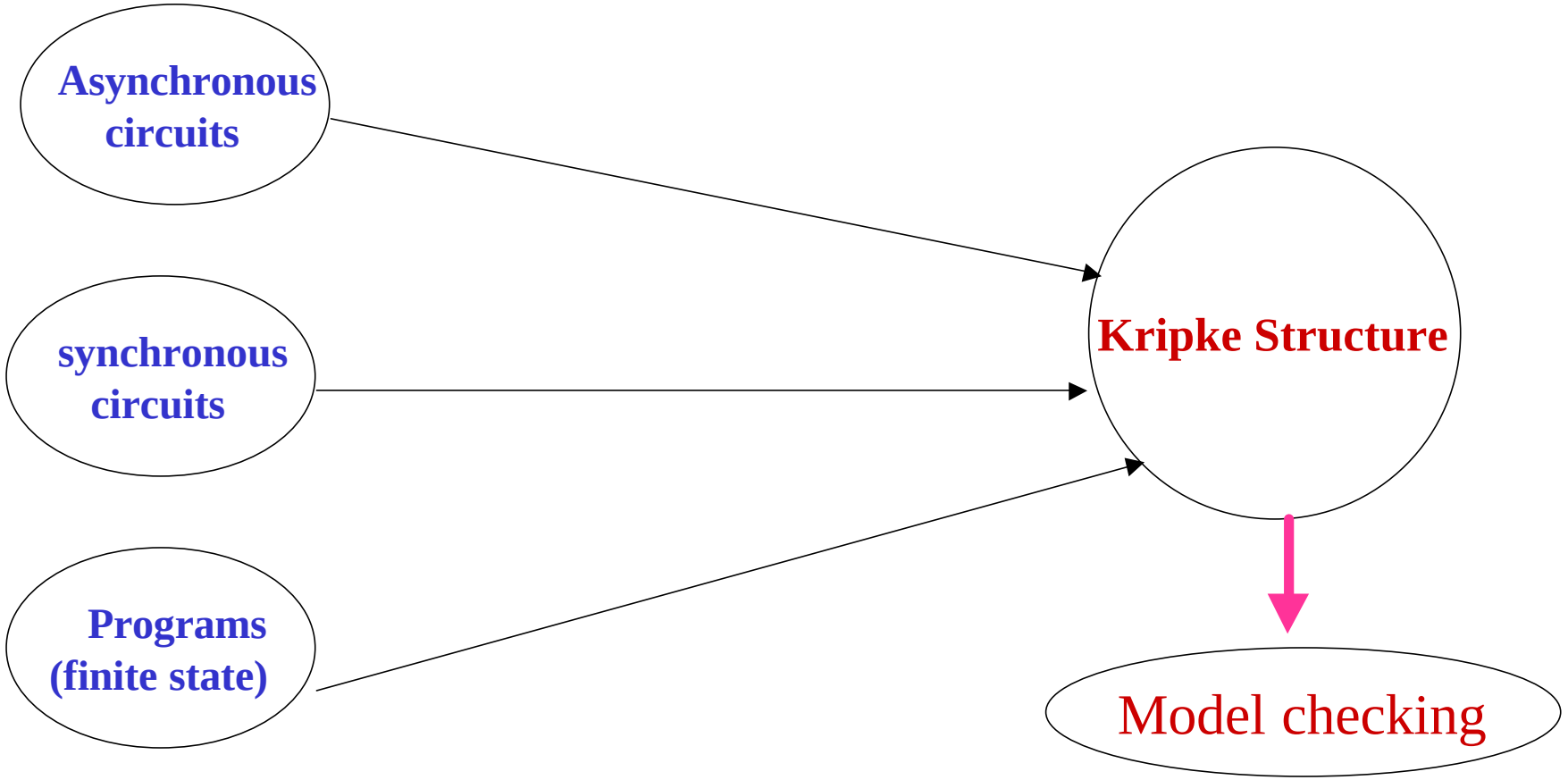
P2:load R2,a
inc R2
store R2,a

- **a may also be 1**

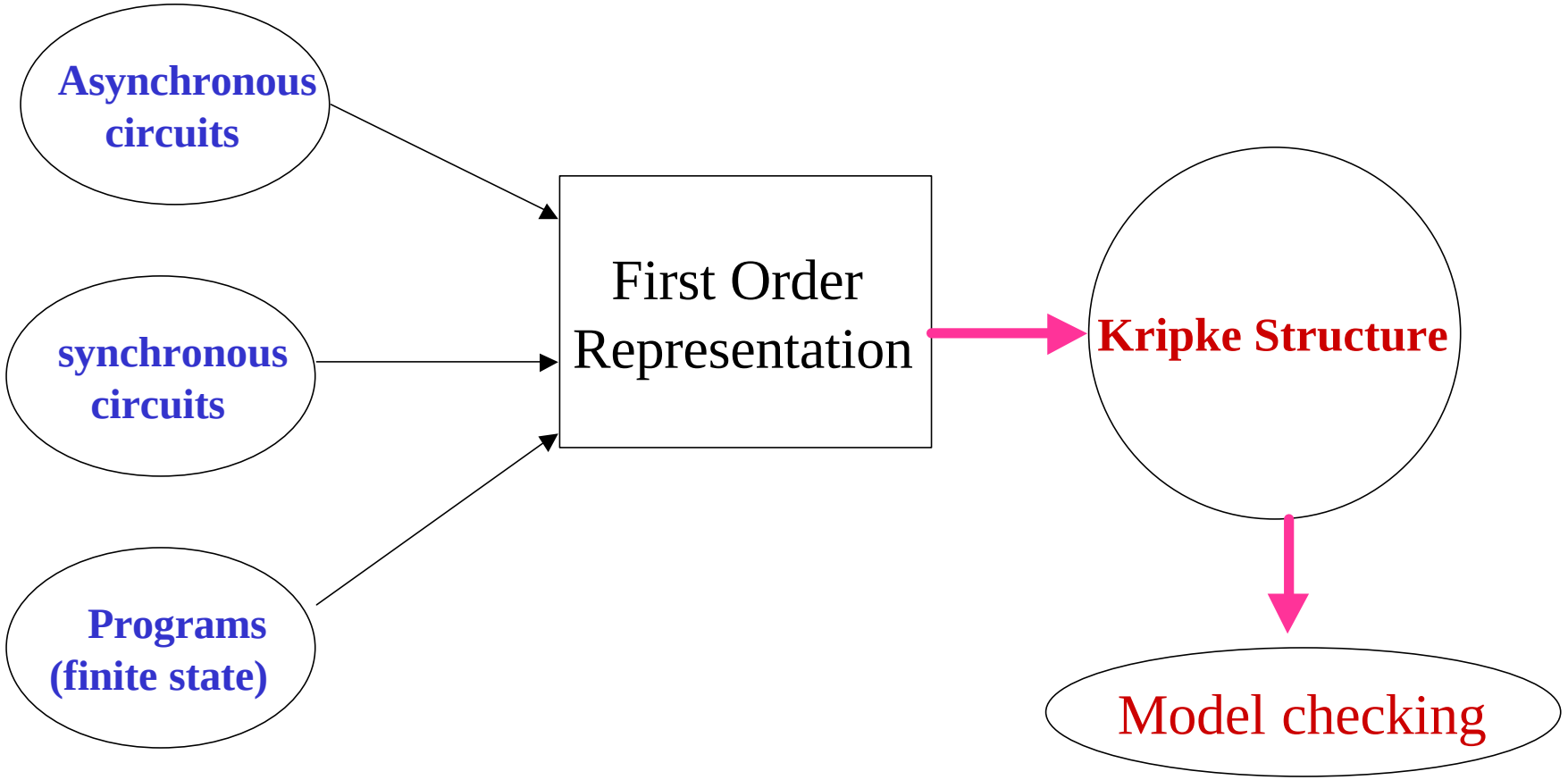
The common framework

- Many systems need to be modeled.
  - Digital circuits
    - **Synchronous**
    - **Asynchronous**
  - Programs
- Strategy : Capture the main features using a logical framework (nothing to do with temporal logics!) : ***First order representation***

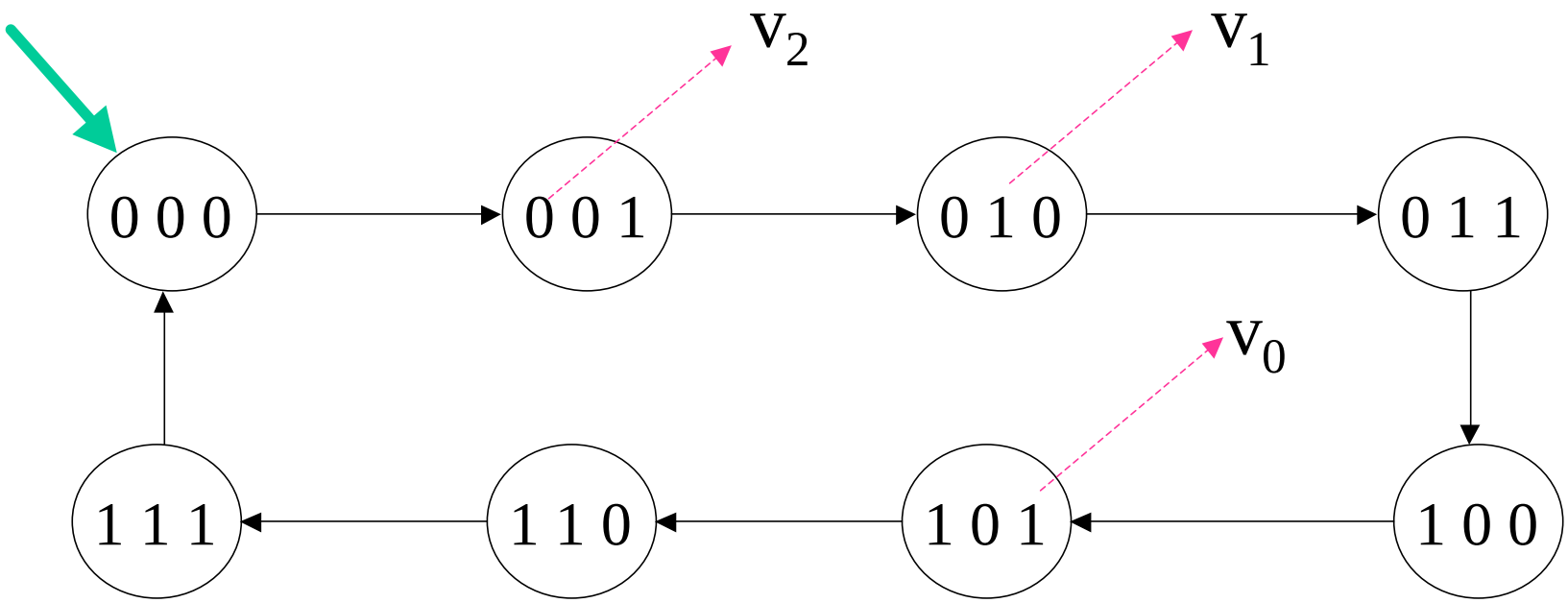
The inefficient way



The efficient way



A mod-8 counter

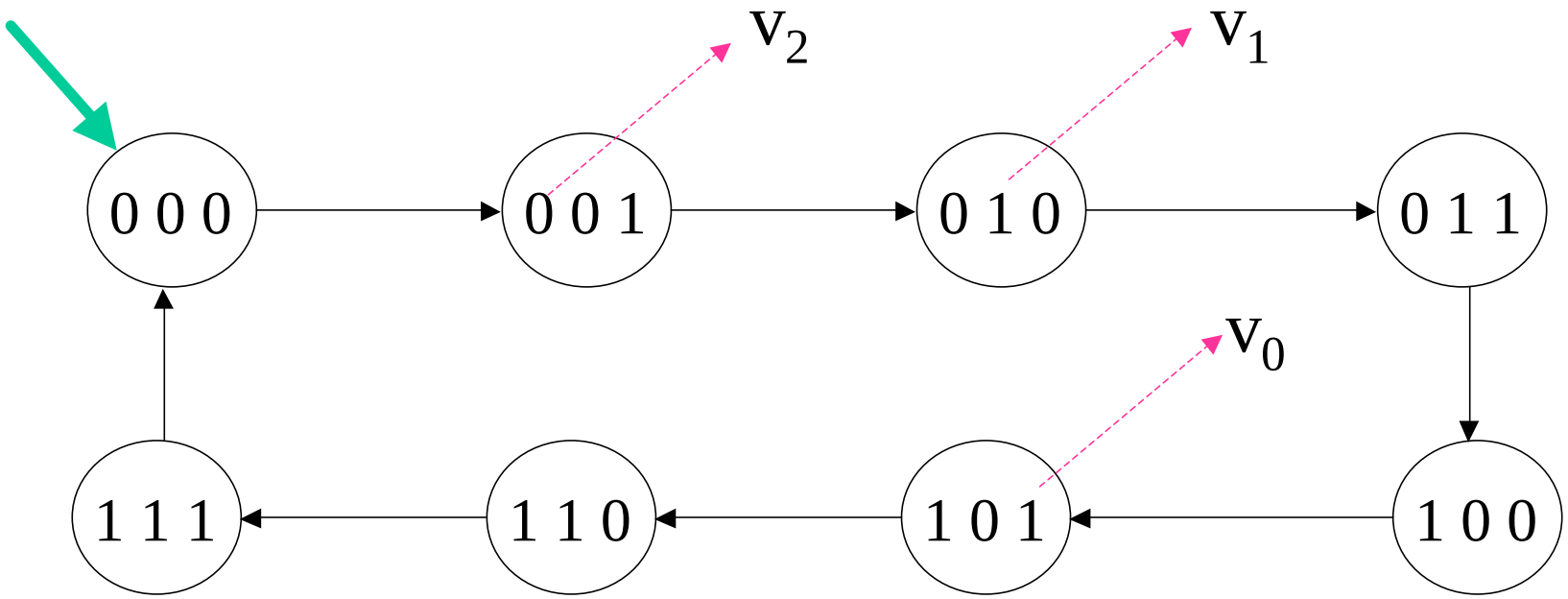


The mod-8 counter

- System variables : v_2 v_1 v_0
- Domain of v_2 = {0, 1}

  Same domain for v_1 and v_0
- Special case : These variables are boolean
- A state s can be seen as a function which assigns to each variable a value in its domain.
  - **$s(v_0) = 0$ $s(v_1) = 1$ $s(v_2) = 1$**
  - It is the state (1 1 0) !

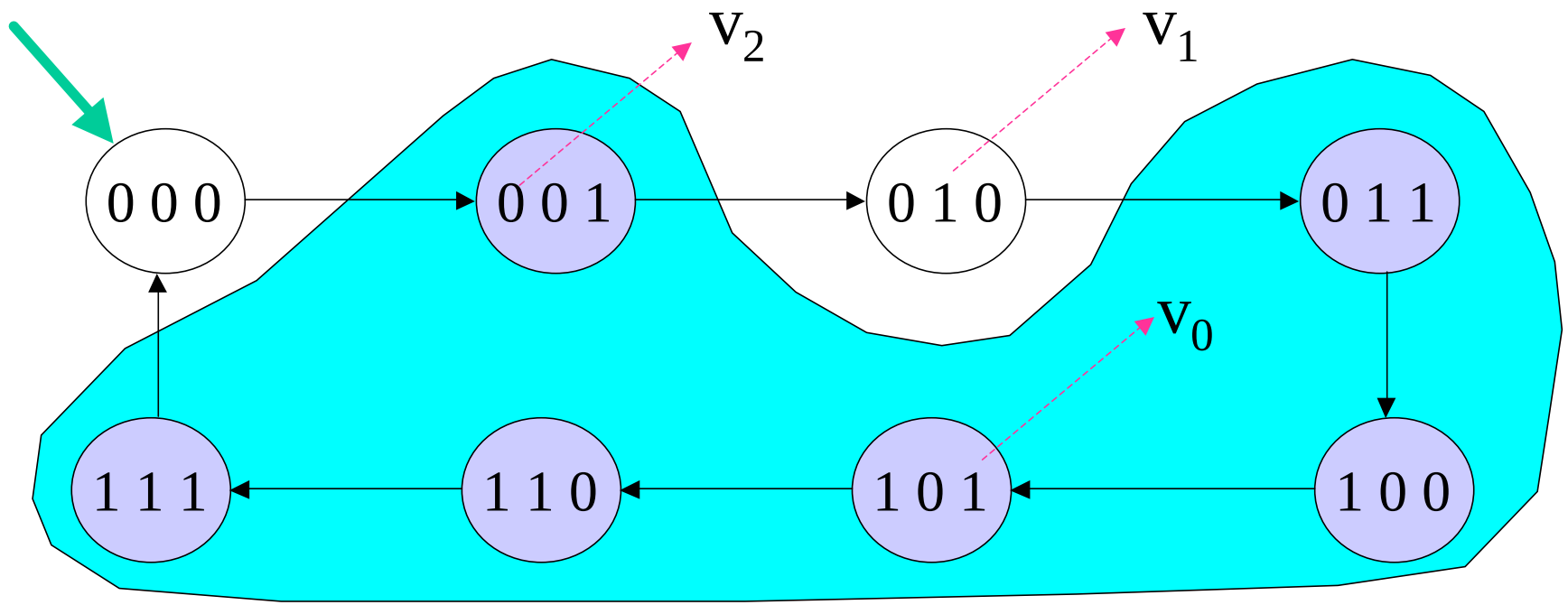
State Predicates



A set of states can be picked out by a formula;

X = v_2 Ú v_0 is the set {...}

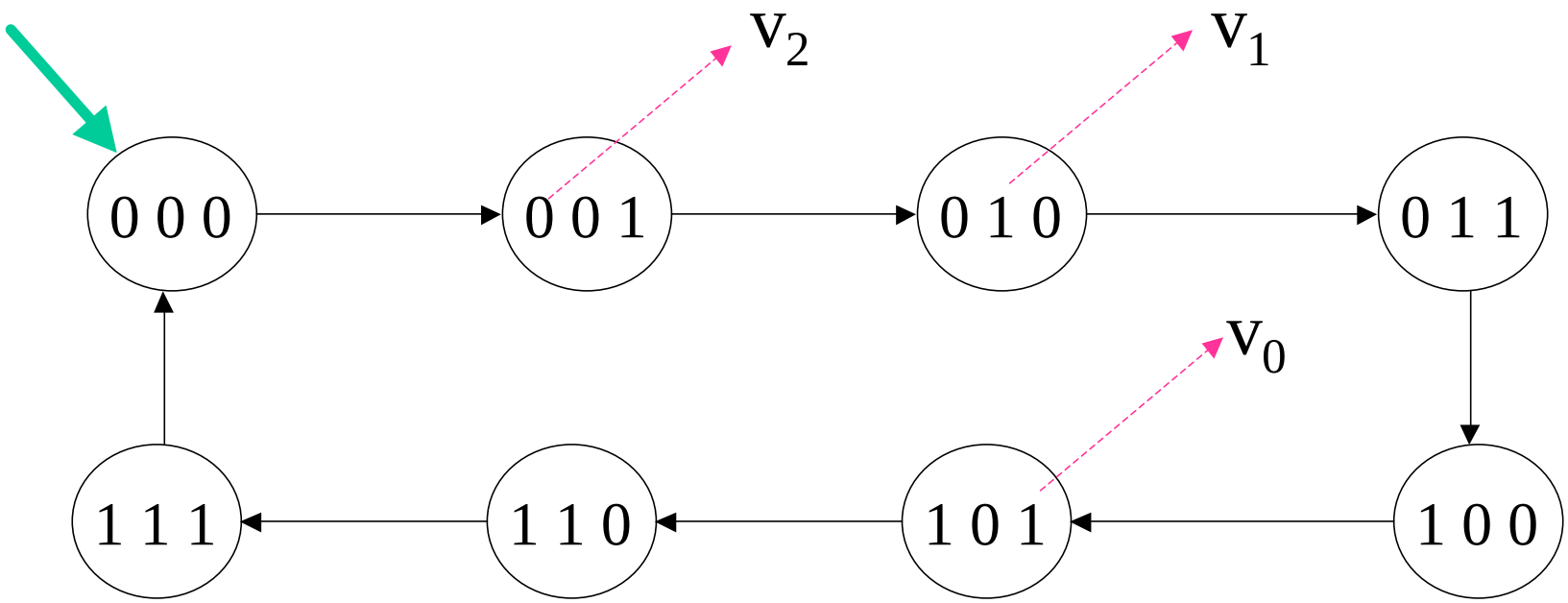
State Predicates



A set of states can be picked out by a formula;

X = v_2 Ú v_0 is the set {100, 101, 110, 111, 001, 011}

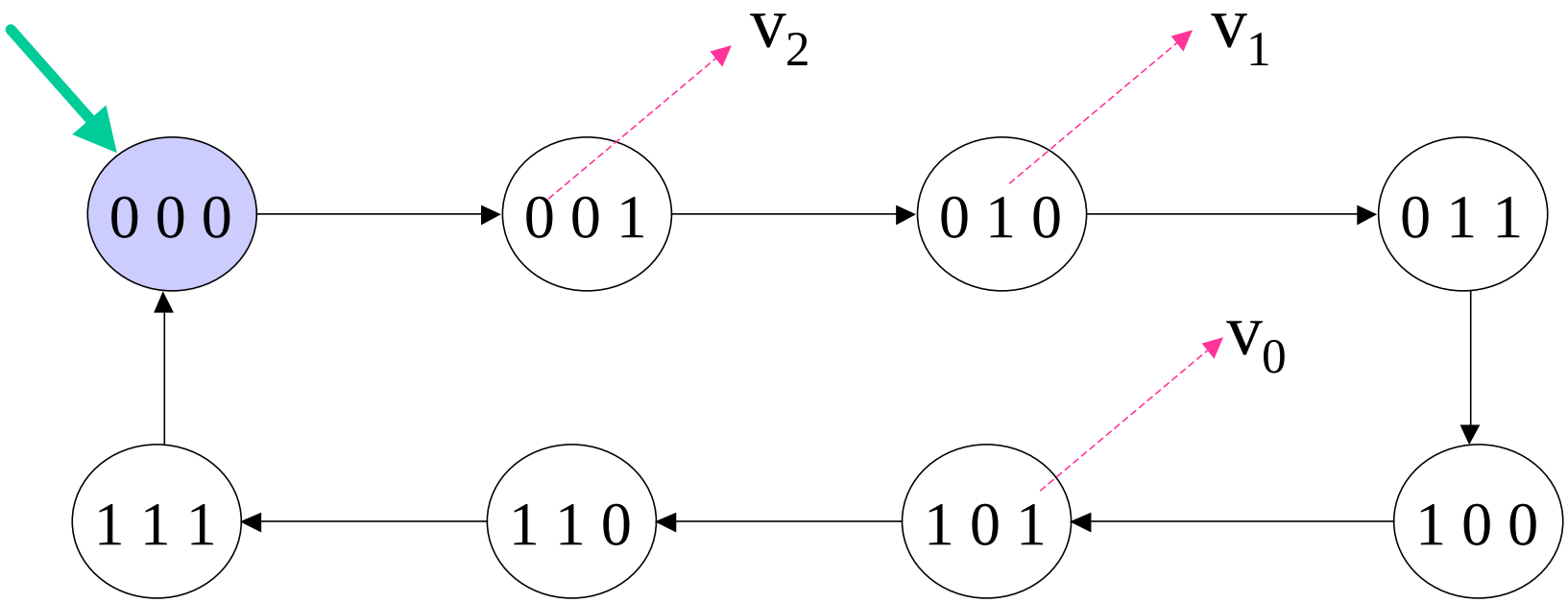
Initial States Predicate



A set of states can be picked out by a formula;

X’ = Ø$\mathbf{v_2}$ Ù Ø$\mathbf{v_1}$ Ù Ø$\mathbf{v_0}$

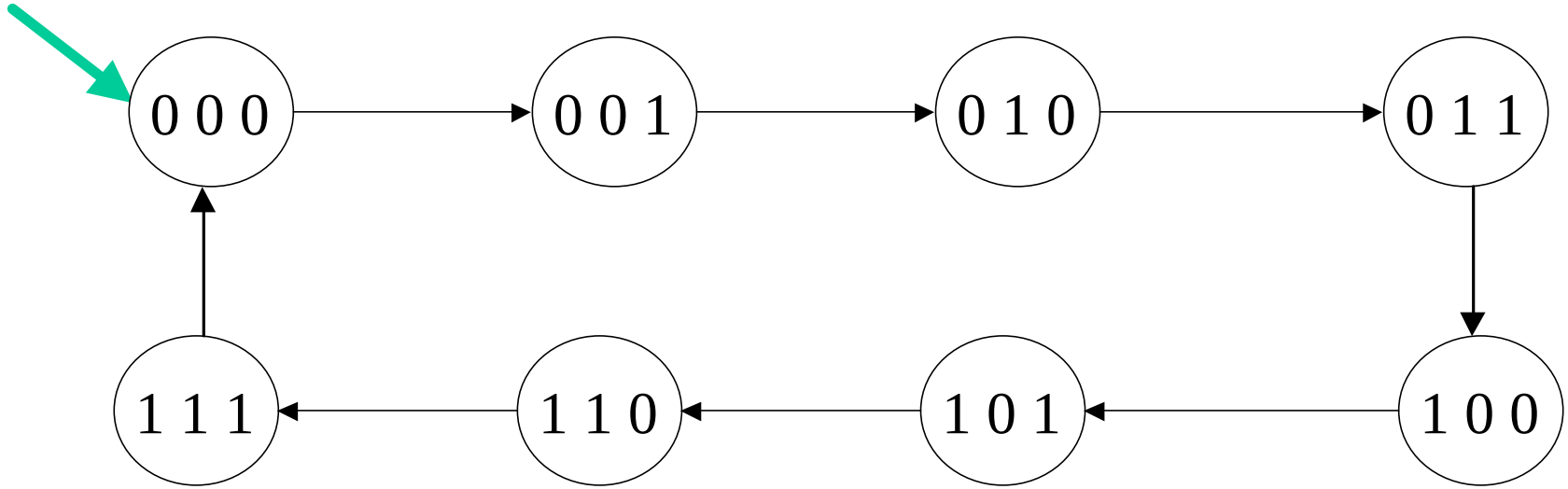
Initial States Predicate



A set of states can be picked out by a formula;

X' = Ø$\mathbf{v_2}$ Ù Ø$\mathbf{v_1}$ Ù Ø$\mathbf{v_0}$ **X'** = { $\mathbf{S_0}$ } = { 000 }

Transition relation predicate

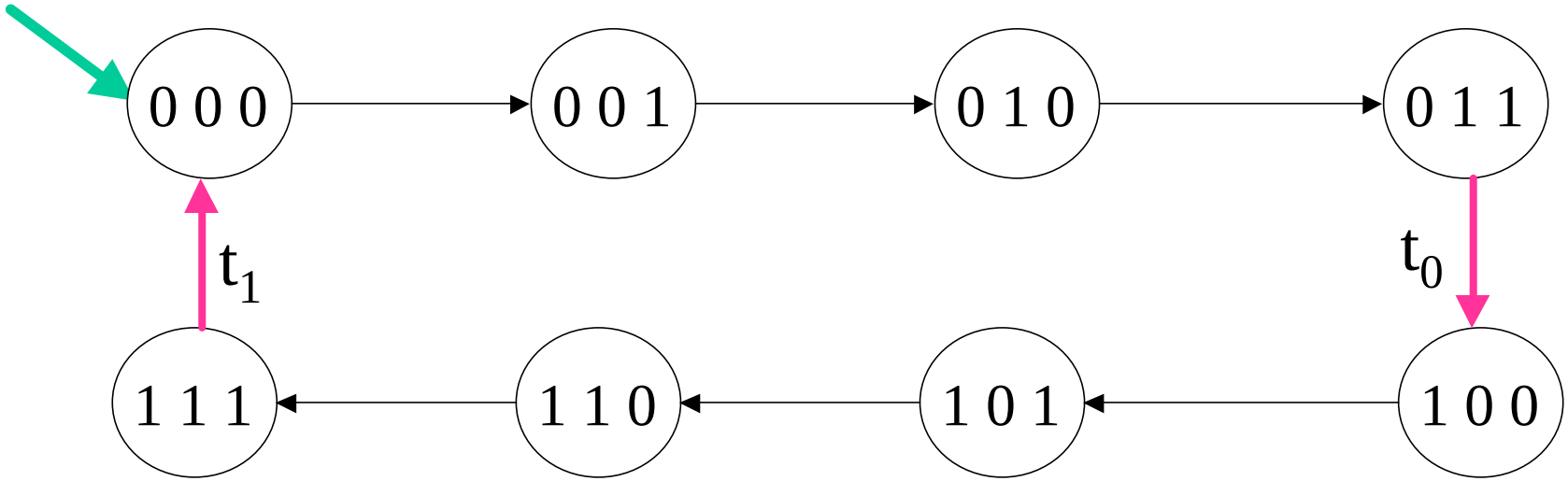


A set of transitions can also be picked out by a formula.

$\mathbf{R_2} = \mathbf{v_2'} \equiv (\mathbf{v_0} \wedge \mathbf{v_1}) \vee \mathbf{v_2}$

$\mathbf{v_2}$ – current value $\mathbf{v_2'}$ – next value

Transition relation predicate

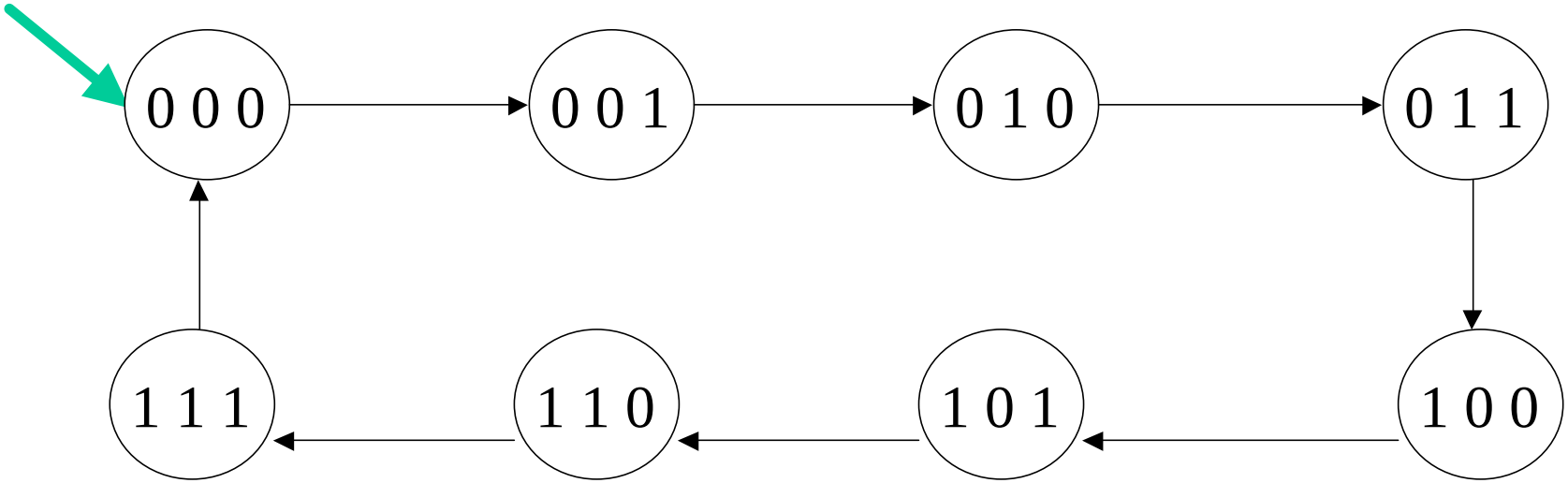


A set of transitions can also be picked out by a formula.

$\mathbf{R_2}$= $v_2' \equiv (v_0 \wedge v_1) \vee v_2$ v_2 – current value v_2' – next value

$\{t_0, t_1\} \subseteq \mathbf{R_2}$

Transition relation predicate

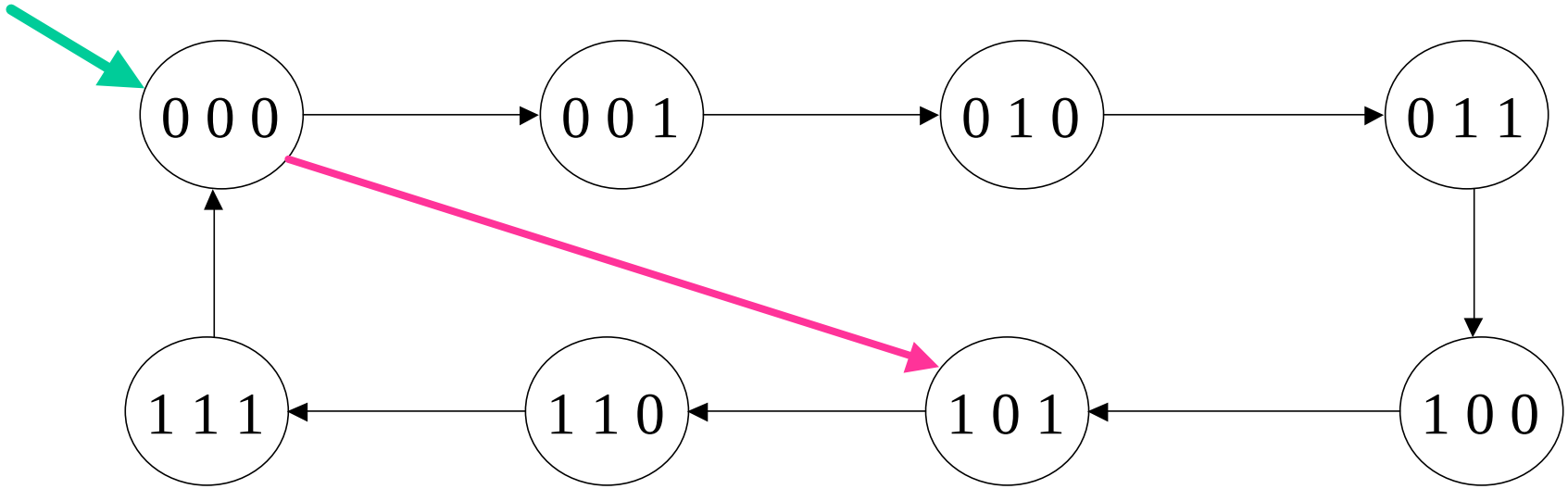


T = **Formula**(v_2, v_1, v_0, v_2', v_1', v_0')

Not all formulae will define subsets of transitions.

You must pick the right formula .

Transition relation predicate

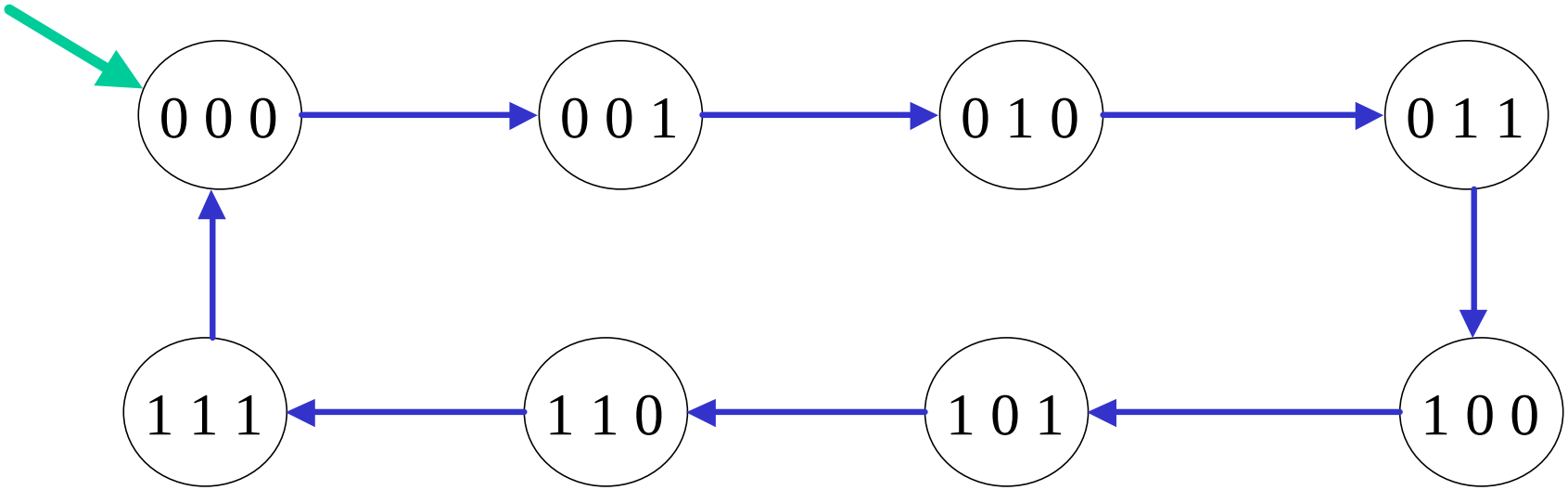


$\mathbf{T_0}$ = v_0' 1 v_0 v_0 – current value v_0' – next value

$\mathbf{T_0}$ = {(000) ⟶ (101) ,……..}

But this is not a transition!

Transition relation predicate



$T_0 = v_0' \leftrightarrow v_0$ $\quad v_i$ – current value $\quad v_i'$ – next value

$T_1 = v_1' = (v_0 \oplus v_1)$

$T_2 = v_1' = (v_0 \wedge v_1) \oplus v_2$

$T = T_0 \wedge T_1 \wedge T_2$

Summary of Predicates

- System variables $v_0, v_1, v_2, \ldots, v_n$.
- Each v_i has a domain of values
  - Boolean , {a,b,c,..}, {5,8,0,7}…
  - Each domain is required to be ***finite***.
- A state is a function **s** which assigns to each system variable a value in its domain.
- The set of states is ***finite*****.**

Summary

- Predicates can be used to pick out –succinctly- sets of states (useful for identifying initial states).
- **X** = **Formula**(v_0, v_1, v_2,...,v_n)
- But this works only when all domains are boolean.
- In general Formula can be a first order formula.

Summary

- A set of transitions can also be picked out using predicates.
- **T = Formula(v_0, v_1,…, v_n, v_0', v_1',…,v_n')**
- **T** is the set of all transitions

  (v_0, v_1,…,v_n) ⟶ (v_0', v_1',…,v_n')

  such that Formula (above!) is satisfied.
- Not all (state or transition) formulas will be legitimate.

Why use formulae?

- ***Formulae*** allow us to compactly describe a system and its dynamics
- It's easy to go from a "***logical***" description to ***Kripke structures***.
- Once we have a ***Kripke structure***, we are in business.
- We can use
  - ***Temporal Logics*** to specify properties
  - ***Model checking*** to verify these properties.

First Order Logic

- The general structure :
  - Syntax
    - Formulas
  - Semantics
    - When is a formula true ?
    - Models
      - Interpretations
      - Valuations

Syntax

- Terms
  - Variables
  - Functions symbols, constant symbols
- Atomic formulas
  - Relation symbols, equality, terms
- Formulas
  - Atomic formulas
  - Propositional connectives
  - ***Existential and universal quantifiers***

Syntax

- (individual) variables --- **x, y, v_3, v'**,…
  - System variables in our context
- Function symbols : $\mathbf{f}^{(n)}$
  - **n** is the arity of **f**.
  - Add $^{(2)}$
  - Next $^{(1)}$
- Function symbols will capture the functions used in the programs, circuits, …

Constant symbols

- Apart from variables, it will also be convenient to have constant symbols.
  - ***zero*** , ***five***, ….
- Variables can be assigned different values but a constant symbol is assigned a fixed value.

Terms

- Terms are used to point at values.
- A variable is a term.
  - **x , v , v''**
- A constant symbol is a term.
- Suppose *f* is a function symbol of arity n and $t_1, t_2, \ldots, t_n$ are terms then $f(t_1, t_2, \ldots, t_n)$ is also a term.

Terms

- Let Plus be a function symbol of arity 2.
- **v_1 , v_2, Plus(v_2, Plus(v_1, v_1))** are terms.
  - the semantics of the last term is intuitively
    $v_2 + 2v_1$
- Let weird_op be a function symbol of arity 3
- Then

  Plus(weird_op(**v**, Plus($\mathbf{v}_1$, $\mathbf{v}_2$), ***five***), Plus(**v**, **v”**))

  is a term.

Predicates

- Relation (predicate) symbols :
  - ***P*** which also has an arity
  - ***Greater-Than*** has arity 2
  - ***Prime*** has arity 1
  - ***Middle*** has arity 3 -- ***Middle*(t_1, x, t_2)**
    - intuitively, **x** lies between t_1 and t_2
- ***Equal*** has arity 2
  - will be denoted as **=**
  - It is a "**constant**" relation symbol.

Atomic formulas.

- If $\mathbf{t_1}$ and $\mathbf{t_2}$ are terms then $\mathbf{=(t_1, t_2)}$ is an atomic formula.
  - also written $\mathbf{t_1 = t_2}$
- Suppose $\boldsymbol{P}$ has arity **n** and $\mathbf{t_1, t_2, \ldots, t_n}$ are terms.
- Then $\boldsymbol{P}\mathbf{(t_1, t_2, \ldots, t_n)}$ is an atomic formula.

Atomic formulas

- ***Greater-Than*(*five*, *zero*)**
- ***Greater-Than*(*two*, *four*)**
- ***Prime*(Plus(v_1, v”))**
- **Plus(v,Zero) = weird_op(v,v,four)**
- **v = *Greater_Than*(v_1,v_2)** is ***not*** an atomic formula !

Terms and Predicates

- A term is meant to denote a value.
  - Makes no sense to talk about a term being true or false.
- An atomic formula may be true or false (depends on the interpretation).
  - Does not make sense to associate a value with an atomic formula.

Formulas

- Every atomic formula is a formula.
- If j is a formula then Ø j is a formula.
- If j and j' are formulas then j Ú j' is a formula.
- j Ù j' abbreviates: Ø(Ø j Ú Ø j')
- j É j' abbreviates : Ø j Ú j'
- j º j' abbreviates : (j É j') Ù (j' É j)

Formulas

- If **j** is a formula and **x** is a variable then **$x. j** is a formula.
- **"x. j** abbreviates : **Ø$x.Ø j**
- These are ***existential*** and ***universal*** quantifiers.
- The power of first order logic comes from these operators!

Semantics

- **Models** :
  - ***Domain of interpretation***
  - ***Interpretation***
    - For the function, constant and relation symbols.
      - ***Fixed for all formulas***.
    - For the individual variables, on a “per formula” basis.
      - ***Valuations***.

Semantics

- ***Domain***
  - Each variable will have its domain of values.
  - We pretend all these domains are the same.
  - Or rather, a big enough “universe” that will contain all these domains.
- Fix **D** the universe of values.

Semantics

Interpretation function I

- Assigns a concrete function to each function symbol (of the same arity!)
- Assigns a concrete member of **D** to each constant symbol.
- Assigns a concrete relation to each relation symbol (of the same arity!).

Semantics II

- Assign a concrete function to each function symbol (of the same arity!)
- Assign a concrete member of **D** to each constant symbol.
- Assign a concrete relation to each relation symbol (of the same arity!).

Semantics II

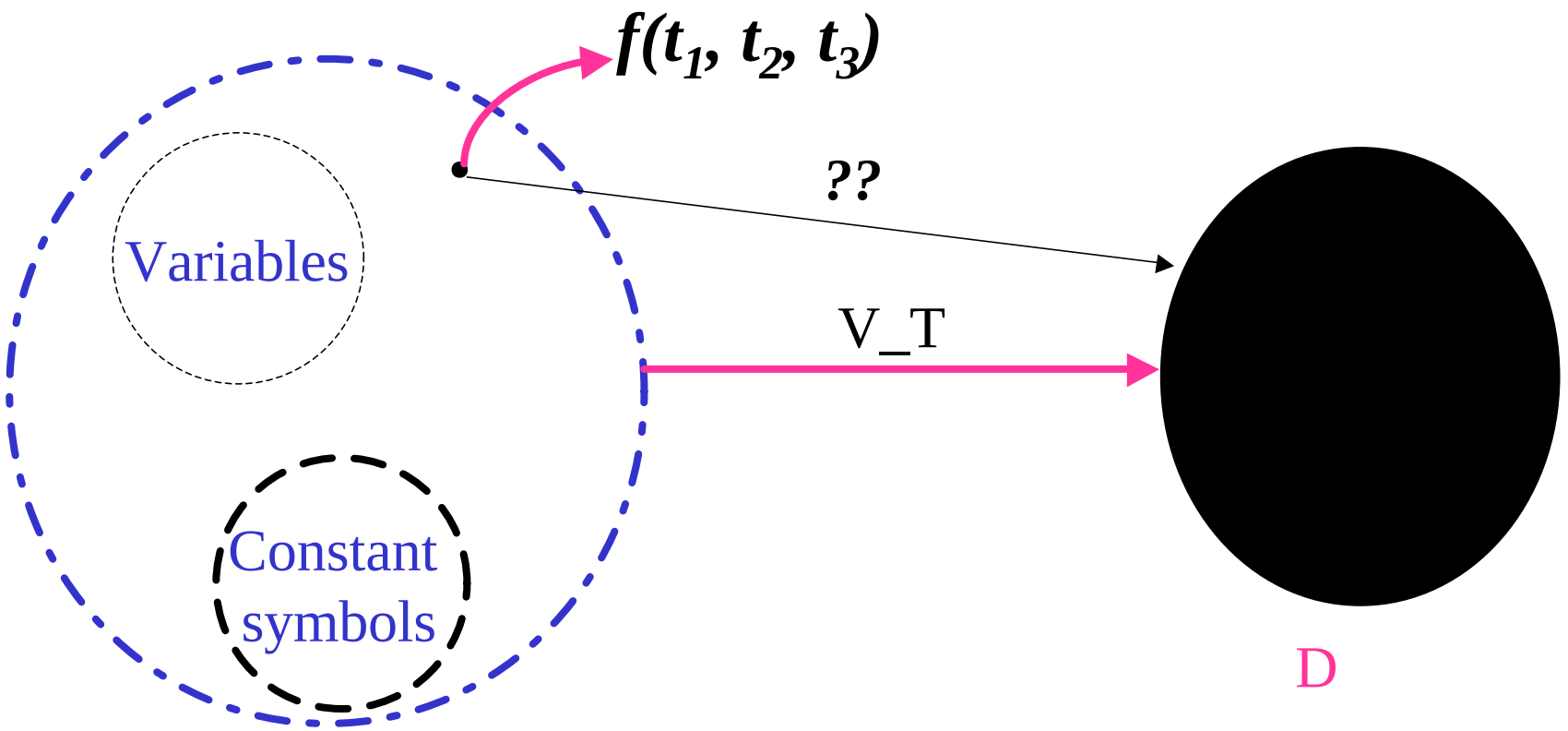
- Assume we have fixed an interpretation for all function symbols, constant symbols and relational symbols.
- Let j be a formula. Fix a **valuation** (assignment) **V** which assigns a member of **D** to each variable.
- **V** : Variables $\longrightarrow$ **D**

Lift V to All Terms

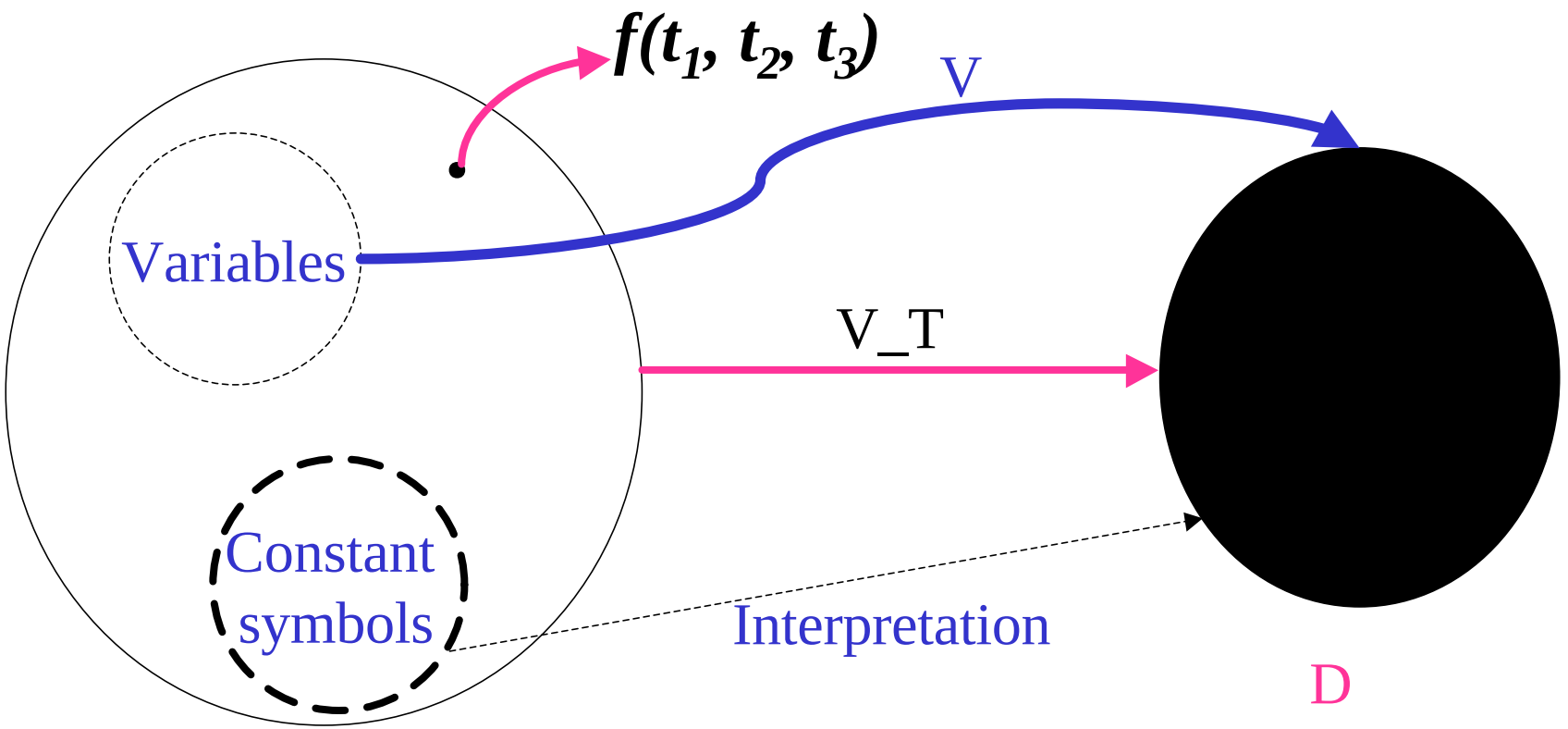
- We have :
  - An interpretation for the function symbols and constant symbols.
  - An assignment **V** : Variables $\longrightarrow$ **D**
- Using these, we can construct (uniquely!)

  V_T : Terms $\longrightarrow$ **D**

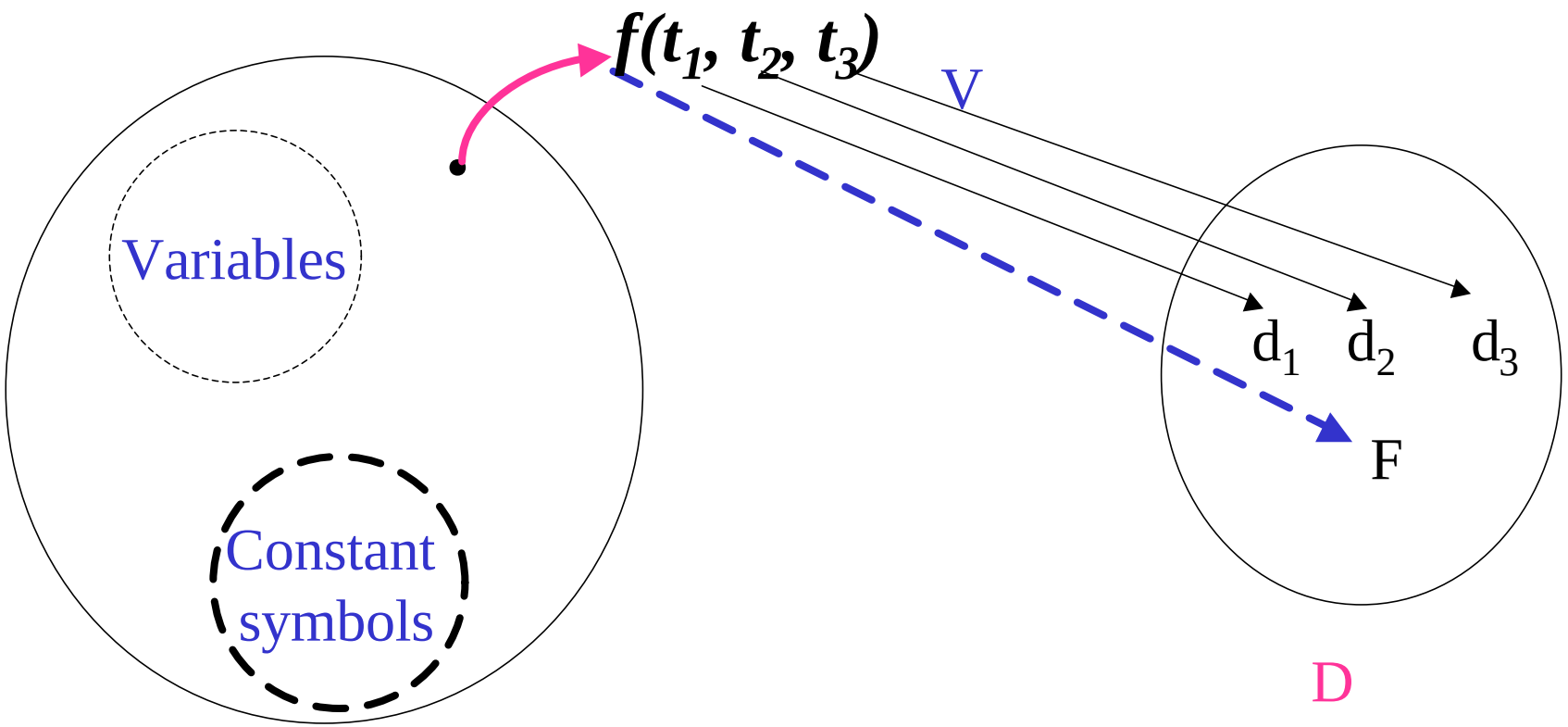
Constructing V_T



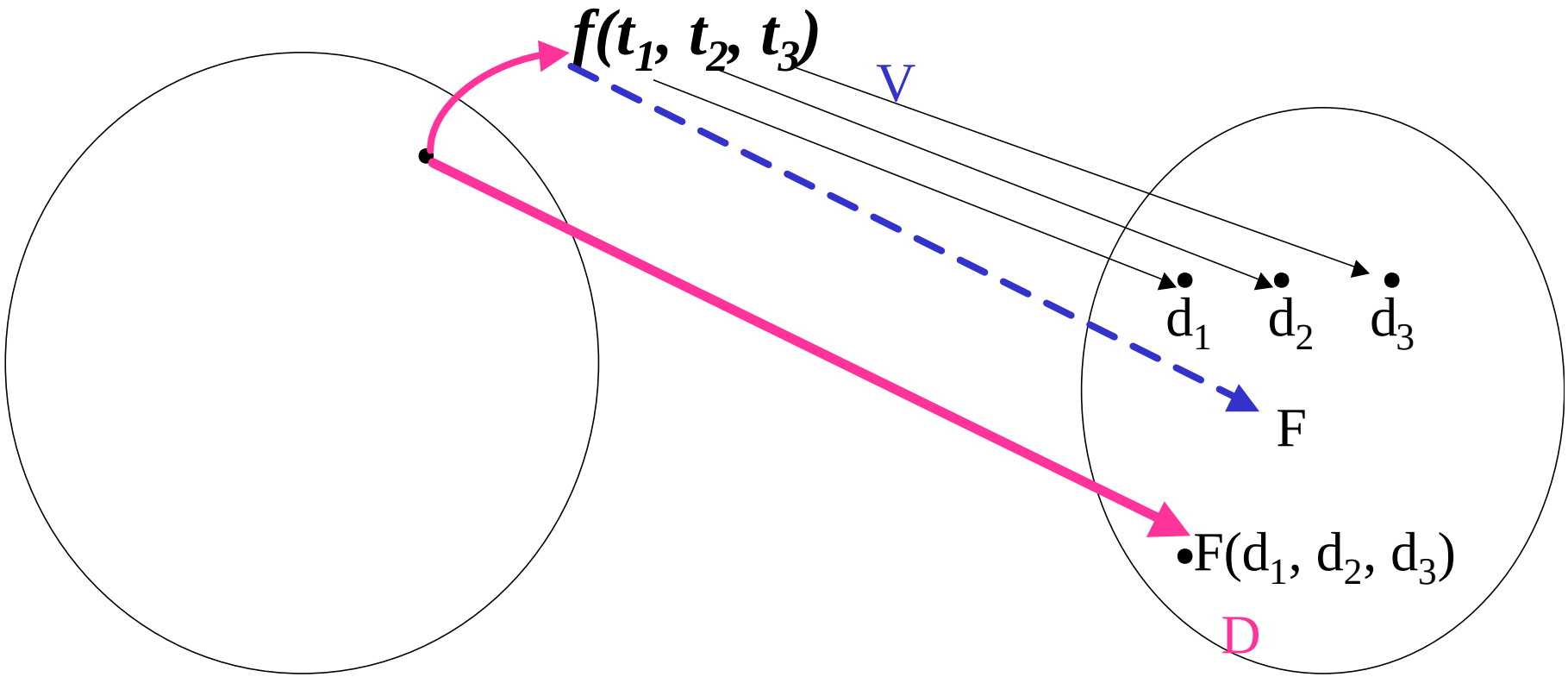
Constructing V_T



Constructing V_T



Constructing V_T



Semantics II

- Let j be a formula. Fix a valuation **V** which assigns a member of **D** to each variable.
- So we now have **V_T** that assigns a member of **D** each term.
- j is satisfied under **V** (and the interpretation we have fixed for all formulas) if :

Semantics II

- Suppose $\boldsymbol{P(t_1, t_2,.., t_n)}$ is an atomic formula and $\mathbf{V\_T}(\boldsymbol{t_1}) = d_1, \ldots \mathbf{V\_T}(\boldsymbol{t_n}) = d_n$ and PCON is the relation assigned to ***P*** by our interpretation.
- Then $\boldsymbol{P(t_1, t_2,.., t_n)}$ is satisfied under **V** iff $PCON(d_1, d_2,\ldots,d_n)$ holds in **D**.

  $(d_1, d_2, \ldots,d_n) \in PCON$ Í **D** ´ **D** ´…´ **D**

Semantics II

- Suppose φ is of the form $\neg \varphi'$.
- Then φ is satisfied under **V** iff φ' is not satisfied under **V**.
- Suppose φ is of the form $\varphi_1 \wedge \varphi_2$
- Then φ is satisfied under **V** iff φ_1 is satisfied under **V** or φ_2 is satisfied under **V**.

Semantics II

- ***Greater-Than*(Plus(*v*, *3*), Multi(*x*, *2*))**

  t_1 t_2

- V(v) = 2 V(x) = 1
- V_T(t_1) = 5 V_T(t_2) = 2
- (5, 2) $\in$ > $\subseteq$ Integers $\times$ Integers
- V'(v) = 1 V'(x) = 6
- Under V', the atomic formula is not true.

Semantics II

- The only case left is when ψ is of the form

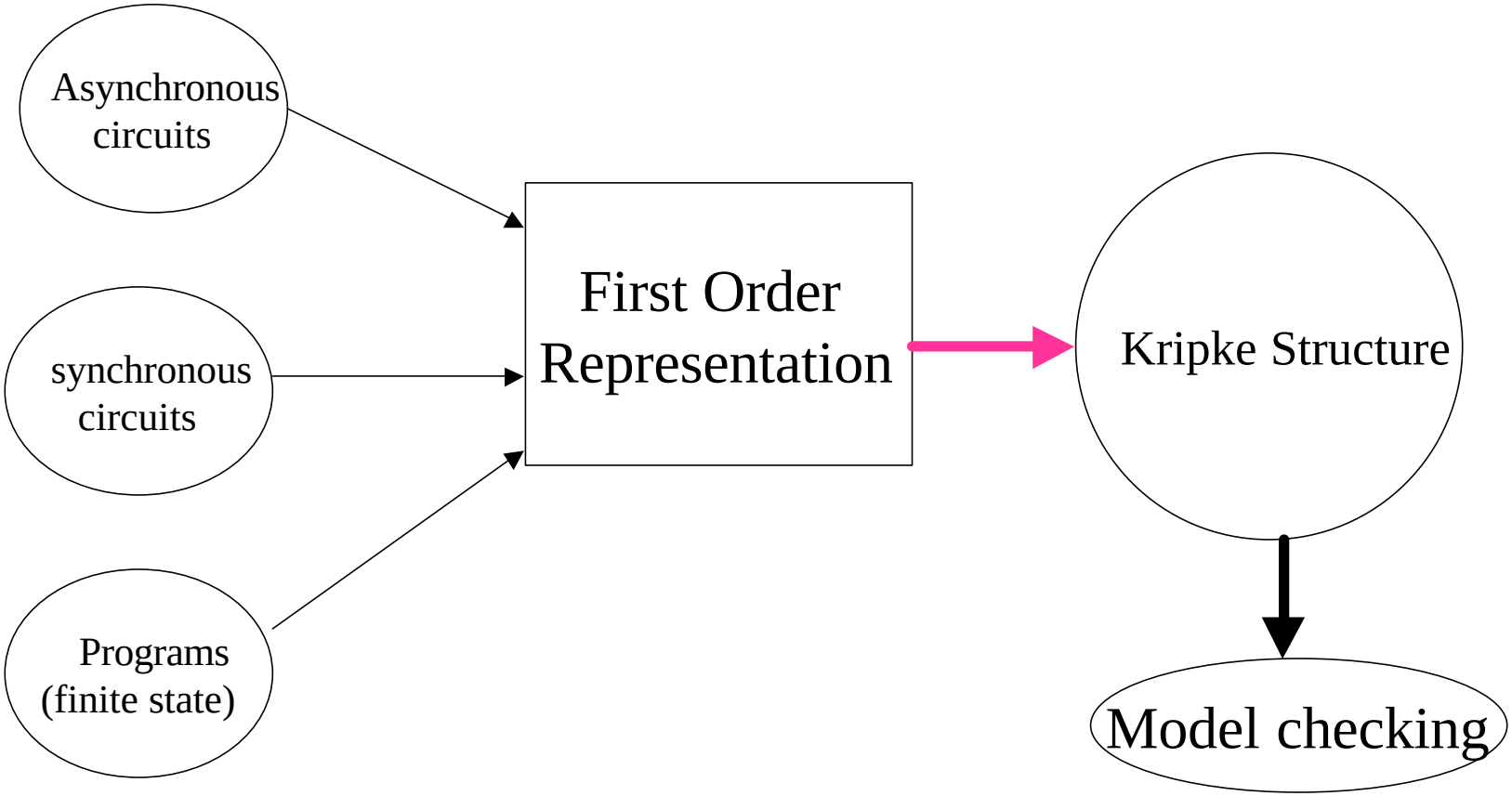
  $\$\mathbf{x}.\ \psi'$

- ψ is satisfied under **V** iff there is a valuation **V'** such that ψ' is satisfied under **V'** and **V'** is required to meet the condition:
  - **V'** is exactly **V** for all variables except **x**.
  - For **x** , **V'** can assign any value of its choosing.

Semantics II

- Whether $**x.** j is true or not under **V**
  - does not depend on what **V** does on **x** !
- **$x. 2x = y** is true under **V(y) = 4**, **V(x) = 1**!
- Because, we can find **V'** with **V'(y) = 4** but **V'(x) = 2**.
- One says **x** is bound in the formula and **y** is free.

The efficient way



First Order Representation to Transition Systems

- **$\{v_1, v_2, \ldots, v_n\}$---** System variables.
- **$D_1, D_2, \ldots, D_n$ ---** The corresponding domains.
- **$D = \dot{E}\ D_i$**
- **$s : \{v_1, v_2, \ldots, v_n\} \longrightarrow D$** such that
  $s(v_1)\ \hat{I}\ D_1$ …..
- **S ---** The set of states.

Initial States

- $S_0(v_1, v_2, \ldots, v_n)$ is a FO formula describing the set of initial states.
- Atomic formula
  - $v = d$ where v is is a system variable and d is a constant symbol interpreted as a member of the domain of v.

Example:

- "S_0 is the set of all states where the **pc = 0** and input is a power of 2"
- $\$n.\ (input = EXP(n))$ Ù $(pc = 0)$

Transition relation

- ***R*** **$(v_1, v_2, ..v_n, v_1', v_2', .., v_n')$** is a FO formula involving the variables $v_1, v_2, ..v_n$ (the system variables) and the new variables $(v_1', v_2', .., v_n')$.
- **$(d_1, d_2, .., d_n) \longrightarrow (d_1', d_2', .., d_n')$** iff

  R **$(v_1, v_2, ..v_n, v_1', v_2', .., v_n')$** is true under the valuation **$v_1 = d_1, \ldots, v_n = d_n, v_1' = d_1', .. v_n' = d_n'$**.

Transition Relation

- V = {x, y, z}
- Program : {x, y, z, pc}

  l_0 : begin

  l_1 : statement$_1$

  l_2 : statement$_2$

  ….

  l_5 : if even(x) then x = x/2 else x = x –1

  l_6 : ….

Transition Relation

- **V = {x, y, z}**
- Program : **{x, y, z, pc}**

  l_5 : if even(x) then x = x/2 else x = x –1

  l_6 : ….

- **j (x, y, z, pc, x', y', z', pc')**
- **pc = l_5 Ù pc' = l_6 Ù ($n. (x = 2n) É x' = x/2) Ù**
  (Ø $n. (x = 2n) É x' = x-1) Ù same(y, z)

which is equivalent to

- **pc = l_5 Ù pc' = l_6 Ù**
  (($n.(x=2n) Ù x'=x/2) Ú (Ø$n.(x=2n) Ù x'=x-1))) Ù
  same(y, z)
- **same(y, z) --- y' = y Ù z' = z**

Transition Relation

- In a similar fashion , we can construct transition relation formulas for :
  - Assignment statement
  - While statements
  - etc.etc.
  - See the text book!

Kripke Structures

- **AP** is a finite set of atomic propositions.
  - "**value of x is 5**"
  - "**x = 5**"
- **M = (S, S_0, R, L)**, a Kripke Structure.
  - **(S, S_0, R)** is a transition system.
  - **L : S $\longrightarrow$ 2^{AP}**
  - **2^{AP}** ---- The set of subsets of AP

Kripke Structures

- The atomic propositions and **L** together convert a transitions system into a model.
- We can start interpreting ***formulas*** over the ***Kripke structure***.
- The atomic propositions make basic (easy) assertions about system states.

Automata and Kripke Structures

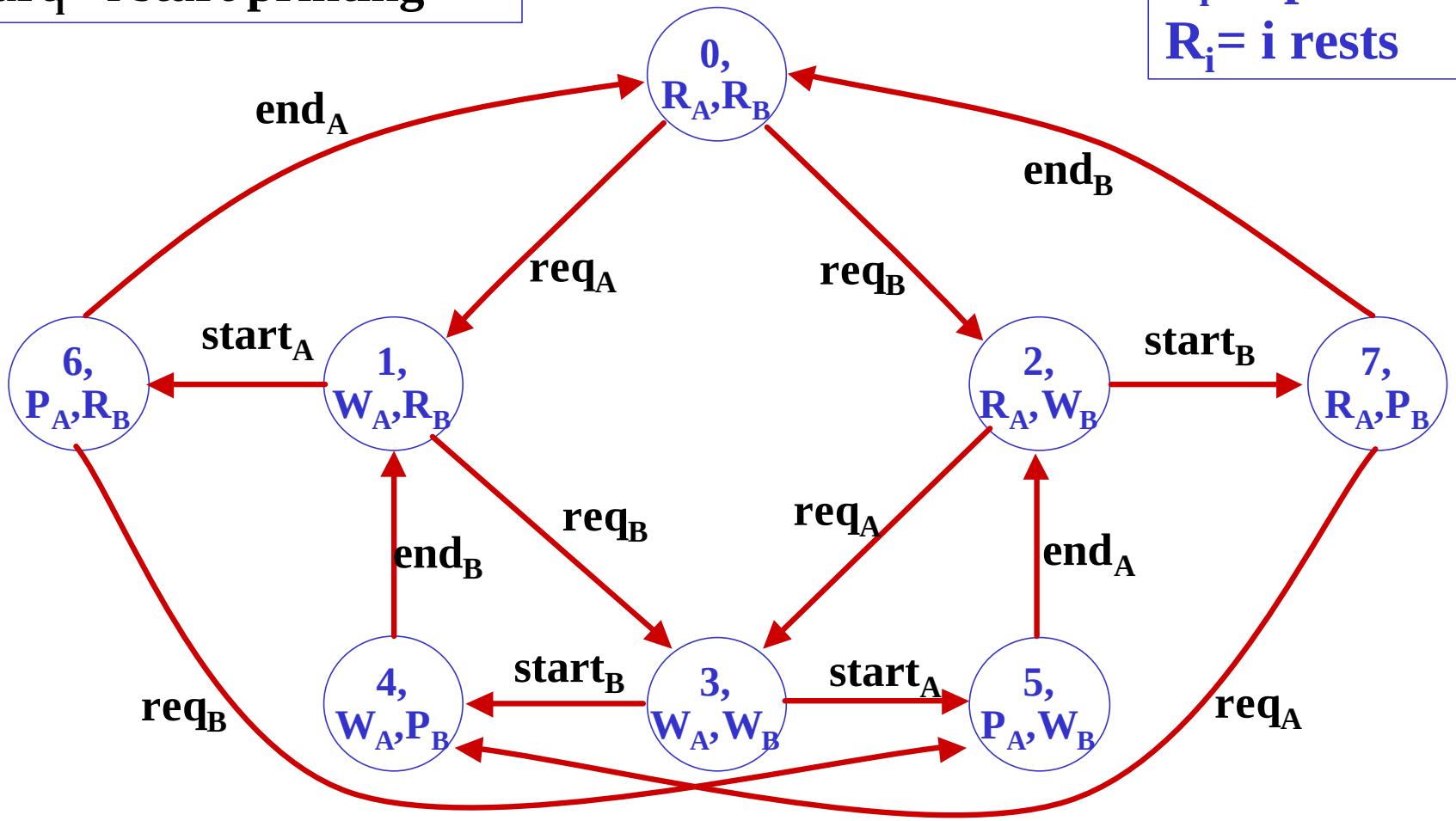
- **AP - set of elementary property**
- **$<S,A,R,s_0,L>$**
- **S - set of states**
- **A - set of transition labels**
- **$R \subseteq S \times A \times S$ - (labeled) transition relation**
- **L - interpretation mapping $L:S \longrightarrow 2^{AP}$**

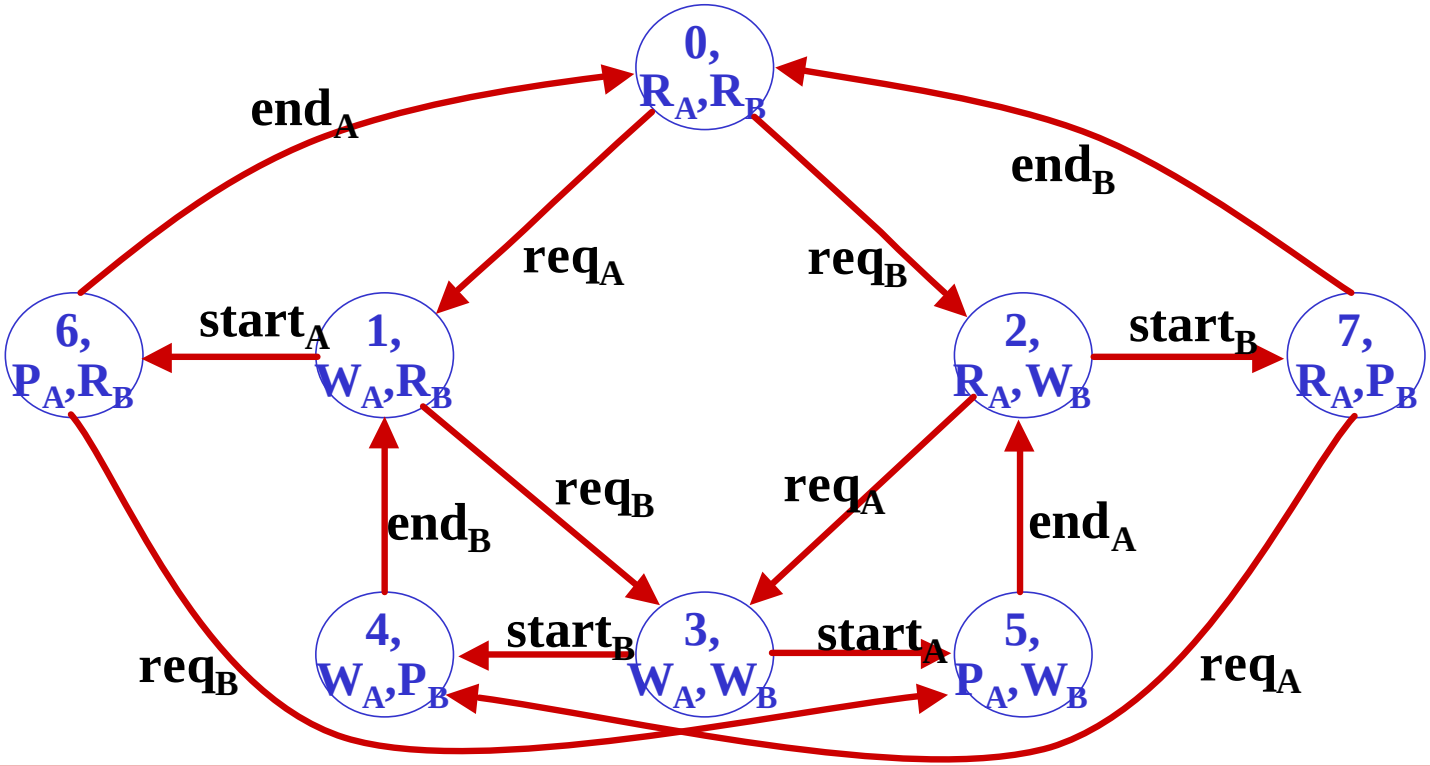
Example: a print manager

end_i= i ends printing
req_i= i requests printing
$start_i$= i start printing

AP

W_i= i waits
P_i= i prints
R_i= i rests





- S = {0,1,2,3,4,5,6,7}
- A = {end_A,end_B, req_A, req_B, $start_A$, $start_B$}
- R = {(0,req_A,1), (0,req_B,2), (1,req_B,3), (1,$start_A$,6), (2,req_A,3), (2,$start_B$,7), (3,$start_A$,5), (3,$start_B$,4), (4,end_B,1), (5,end_A,2), (6,end_A,0), (6,req_B,5), (7,end_B,0), (7,req_A,4),}
- L = {0® {R_A,R_B}, 1® {W_A,R_B}, 2® {R_A,W_B}, 3® {W_A,W_B}, 4® {W_A,P_B}, 5® {$P_A W_B$}, 6® {P_A,R_B}, 7® {$R_A P_B$} }

Properties of the printing systems

- Every state in which $\mathbf{P_A}$ holds, is preceded by a state in which $\mathbf{W_A}$ holds
- In any state in which $\mathbf{W_A}$ holds is followed (possibly not immediately) by a state in which $\mathbf{P_A}$ holds.
- The first can easily be checked to be true
- The second is false (e.g. 0134134134…) - in other words the system is ***not fair***.

Synchronization

- Usually complex systems are composed of a number of smaller ***subsystems*** (***modules***)
- It is natural to model the whole system starting from the models of the subsystems.
- And then define how they cooperate.
- There are many ways to define cooperation (***synchronization***).

Synchronization: no interaction

The system model is just the ***cartesian product*** of the simple modules.

Let $TS_1,\ldots,TS_n$ be n automata (or **TS**), where $TS_i=<S_i,A_i,R_i,s_{i0}>$

The system is then defined as $TS=<S,A,R,s_0>$ where

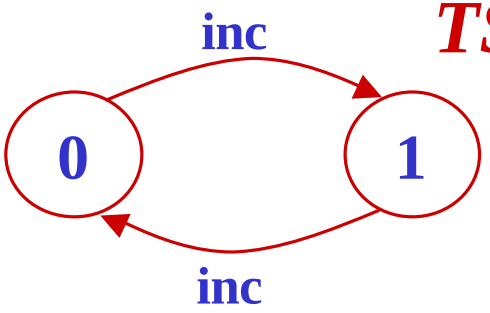
$S = S_1 \times S_2 \times \ldots \times S_n$

$A = A_1 \cup \{-\} \times A_2 \cup \{-\} \times \ldots \times A_n \cup \{-\}$

$R = \{((s_1,s_2,\ldots,s_n),(a_1,a_2,\ldots,a_n),(s'_1,s'_2,\ldots,s'_n)) \mid$ ***forall*** i

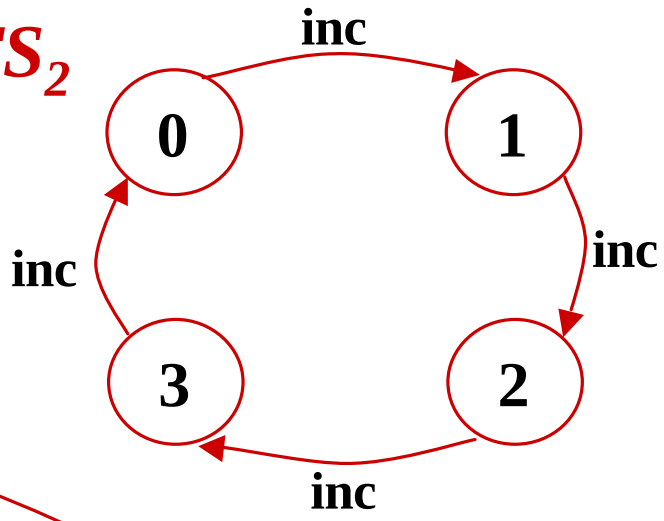
$a_i \neq -$ ***and*** $(s_i,a_i,s'_i) \in R_i$, ***or*** $a_1=-$ ***and*** $s'_i = s_i\}$

$s_0 = (s_{10},s_{20},\ldots,s_{n0})$



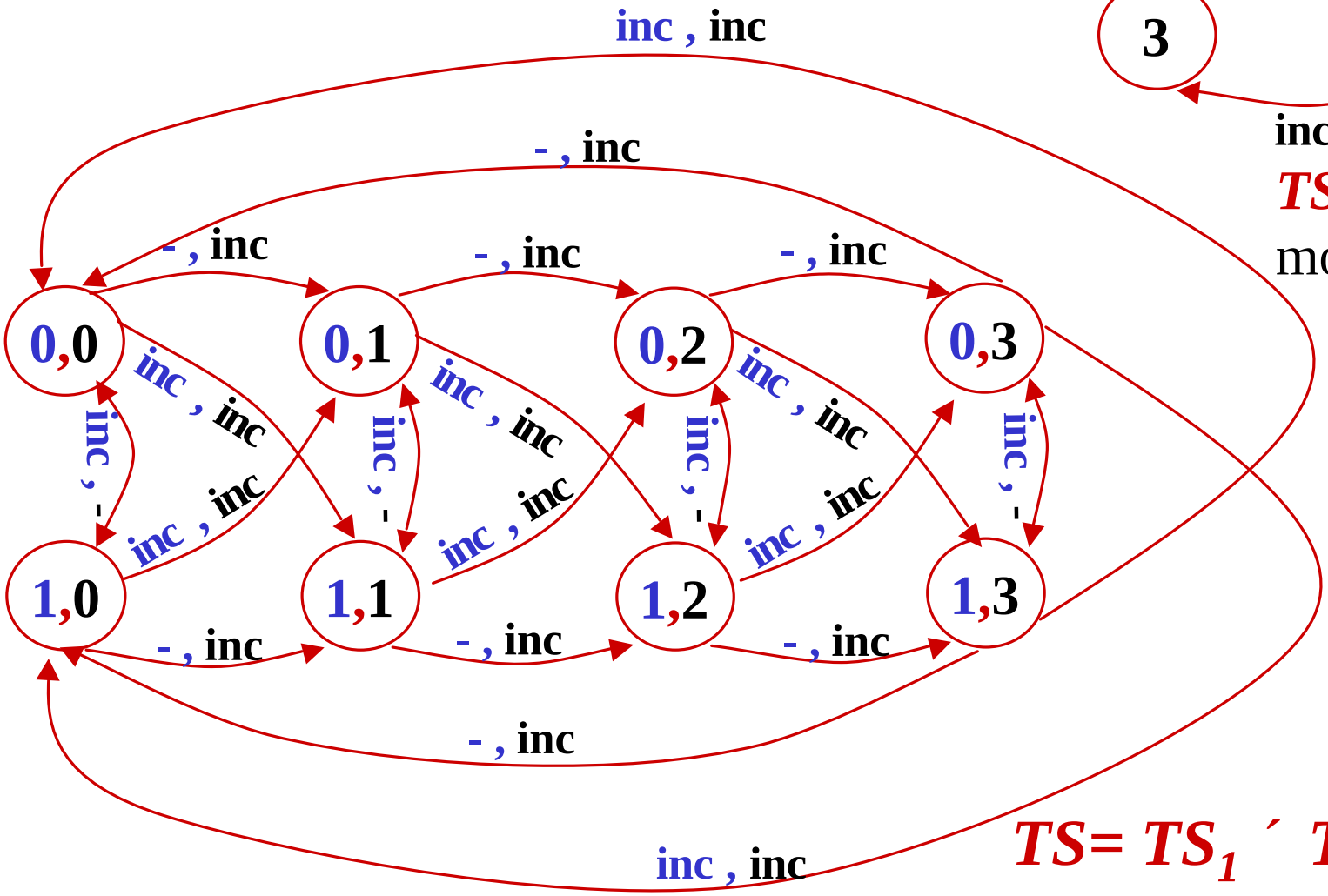
TS_1

TS_1 contatore modulo 2



TS_2

TS_2: contatore modulo 4



$TS = TS_1 \acute{} TS_2$

Synchronization: interaction

To allow for interaction, or synchronization on specific actions we can introduce a Synchro-nization Set (to inhibit undesired transitions) :

- ***Synchronization set is just a subset of the composite actions:***

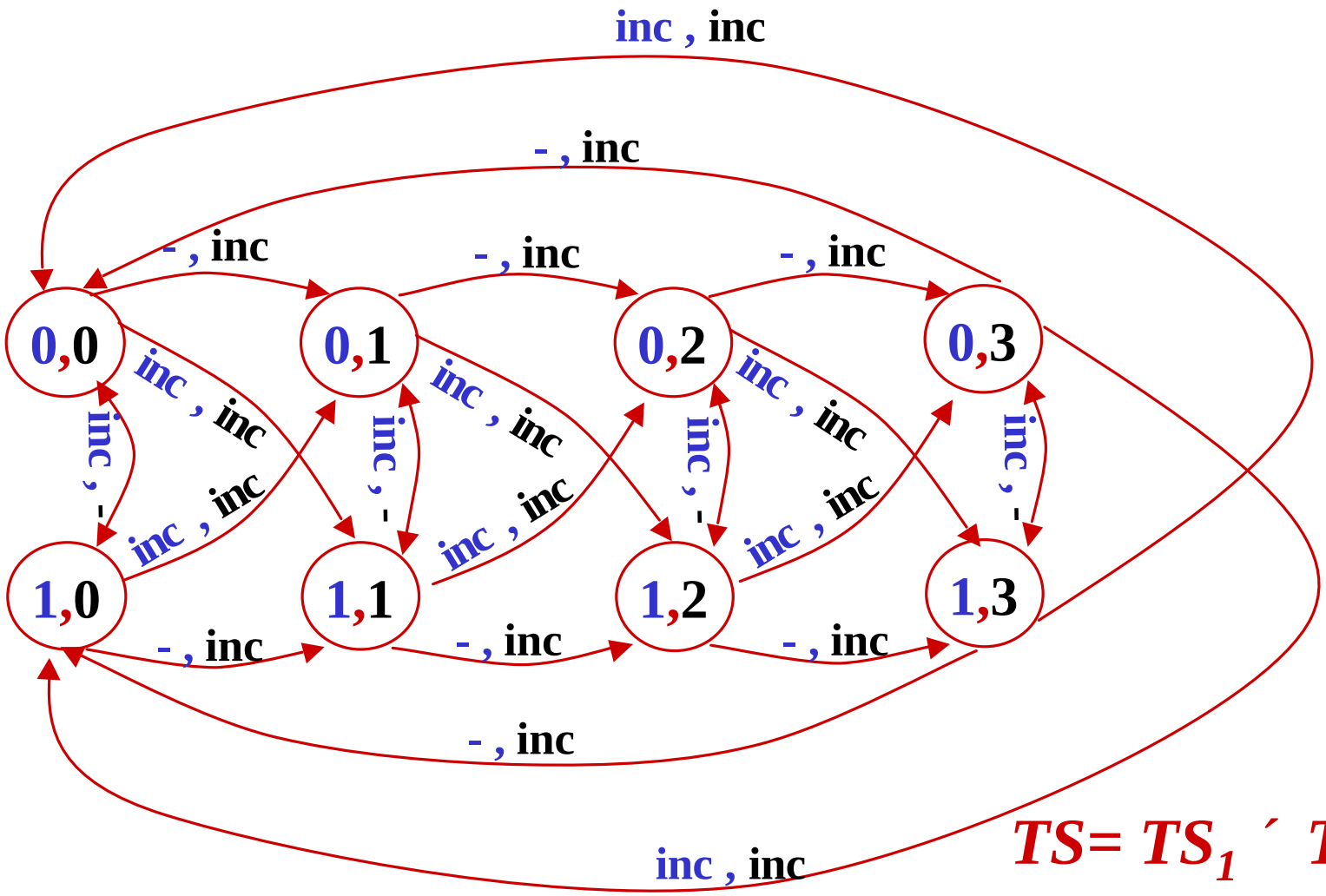
$$Sync \subseteq A_1 \cup \{-\} \times A_2 \cup \{-\} \times \ldots \times A_n \cup \{-\}$$

- ***Then we will have to define the possible transitions as:***

$$R=\{((s_1,s_2,\ldots,s_n),(a_1,a_2,\ldots,a_n),(s'_1,s'_2,\ldots,s'_n)) \mid (a_1,a_2,\ldots,a_n) \in Sync \text{ and for all } i, a_i^1 \text{ - and } (s_i,a_i,s'_i) \in R_i, \text{ or } a_1=- \text{ and } s'_i=s_i\}$$

Free synchronization (Asynchronous systems):

***Sync* = {*inc*,-} ´ {*inc*,-}**



TS= TS$_1$ ´ TS$_2$

Free synchronization

Asynchronous systems:

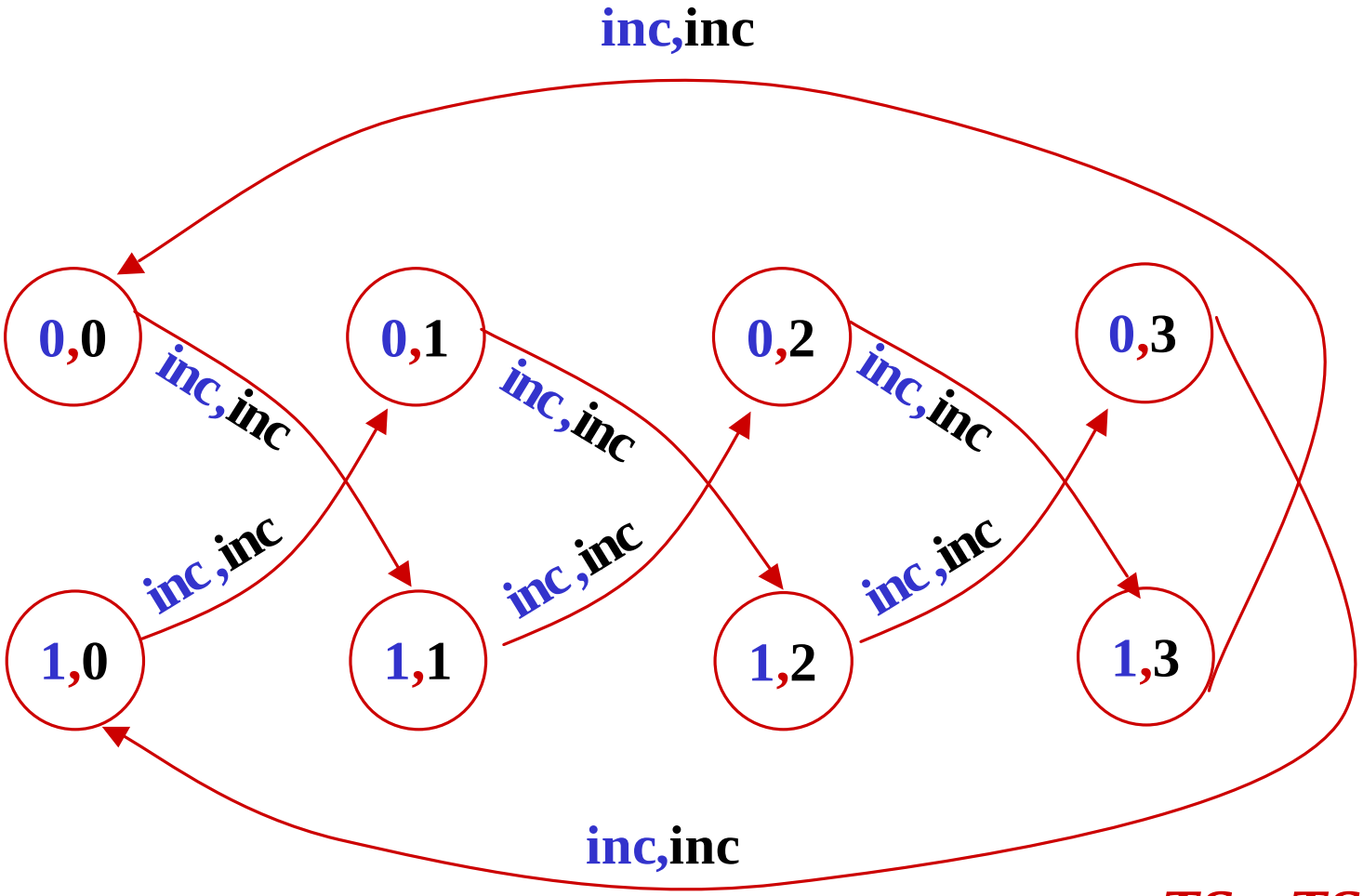
***Sync* = {*inc*,-} ´ {*inc*,-}**

- $R(V,V') = \bigwedge_{i \in I} (R_i(v_i,v_i') \vee \mathbf{same}(v_i)) \wedge \neg \bigwedge \mathbf{same}(v_i)$

if one wants to *discard* the situation where *no component acts*

Synchronization on all actions (Synchronous systems):

***Sync* = {(*inc*,*inc*)}**



$TS = TS_1 \acute{} TS_2$

Synchronous systems

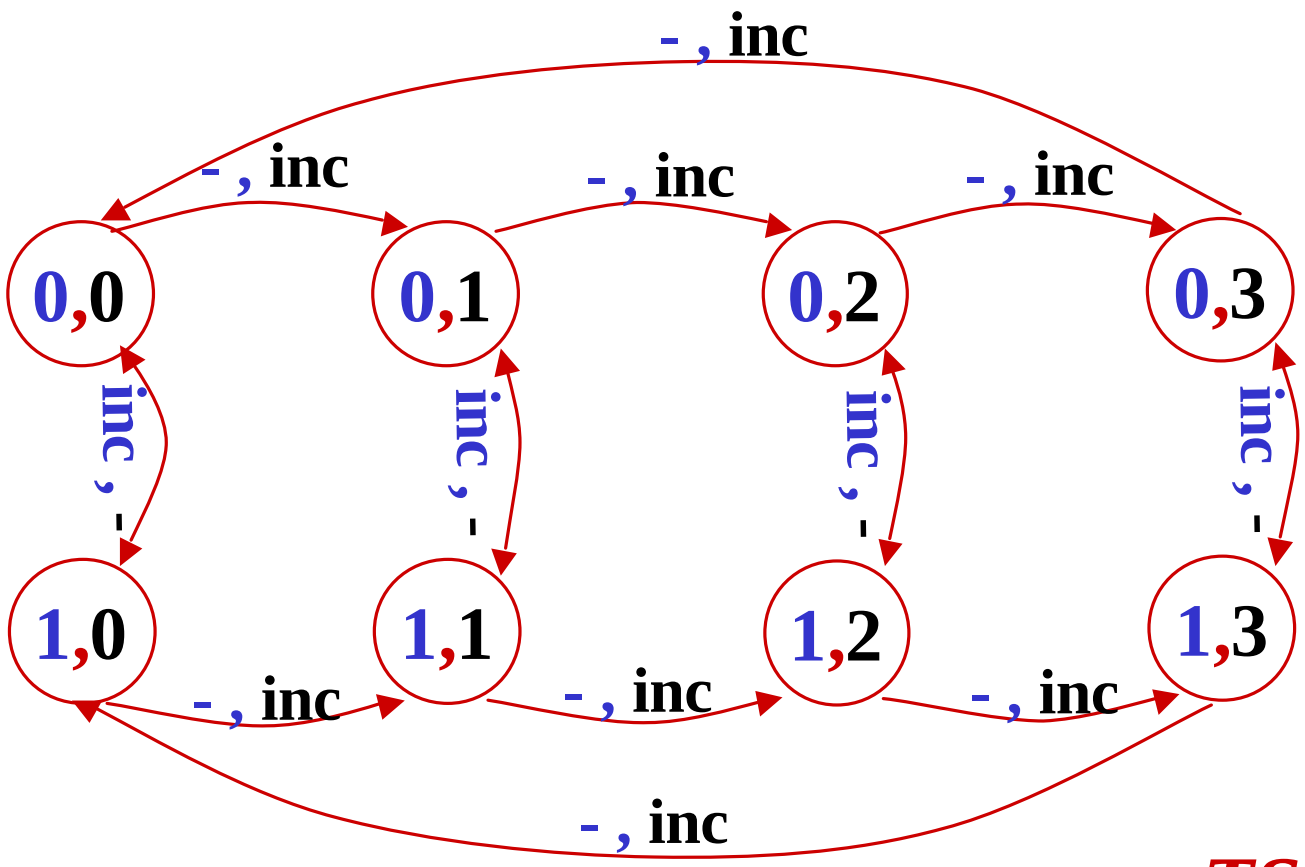
Synchronous systems:

***Sync* = {(*inc*,*inc*)}**

- $R(V,V') = \bigwedge_{i \in I} R_i(v_i,v_i')$

Asynchronous systems with interleaving (only one component acts at any time):

***Sync* = {(-,*inc*),(*inc*,-)}**



$TS = TS_1 \acute{} TS_2$

Asynchronous systems: Interleaving

Asynchronous systems:

***Sync* = {*inc*,-} ´ {*inc*,-}**

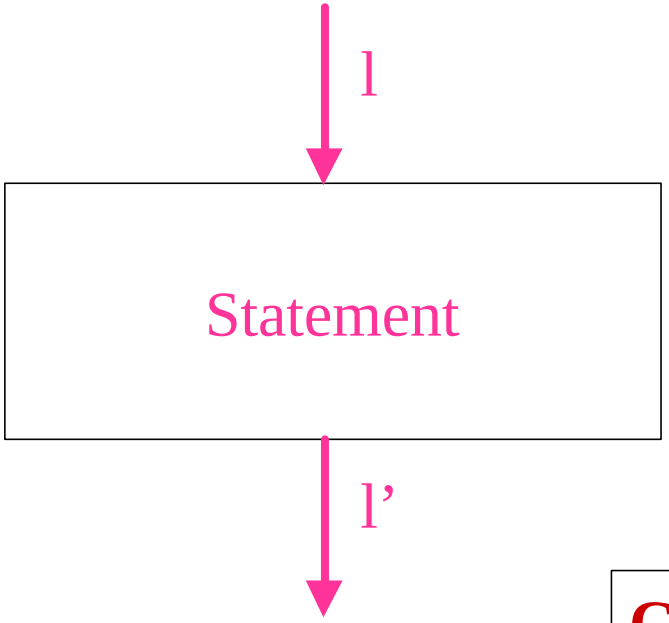
- $R(V,V') = \bigcup_{i \in I} (R_i(v_i,v_i') \wedge \bigwedge_{i \neq i} \mathbf{same}(v_j))$

Concurrent programs

- Many systems to be verified can be viewed as concurrent programs
  - operating system routines
  - cache protocols
  - communication protocols
- **P = cobegin (P_1 || P_2 || …|| P_n) coend**
- **P_1, P_2,..P_n ---** Sequential Programs.
- ***Usually interleaving semantics is assumed***

Sequential Programs

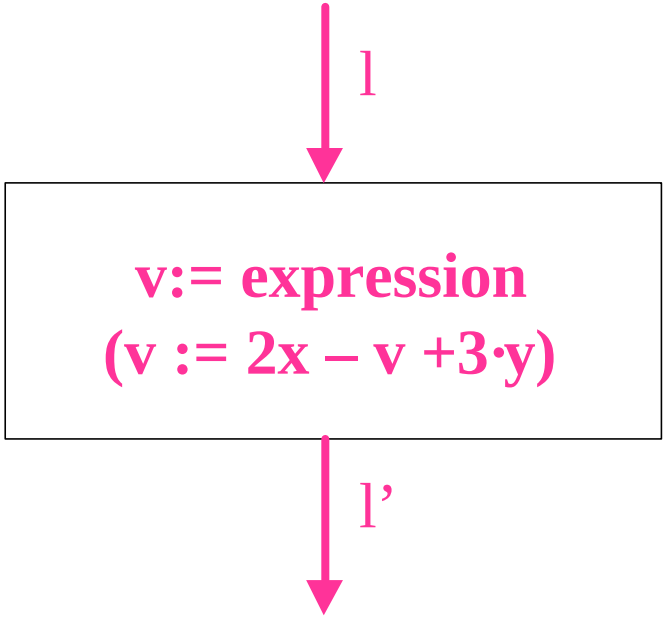
General Structure



C(l, statement, l')

C is a transition procedure which takes a *label*, a *program statement* and a *label* and gives the *FOL* formula specifying the transition relation.

Assignments

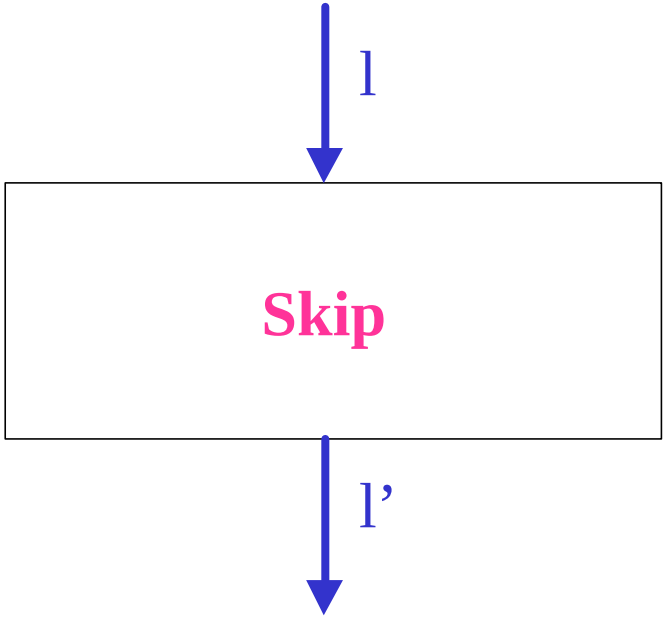


C(l, assignment, l’)

pc = l Ù pc’ = l’ Ù
v’ = expression Ù same (V – {v})

[Y = {y_1, y_2, ..y_m}
y_1’ = y_1 Ù y_2’ = y_2 Ù… Ù y_m]

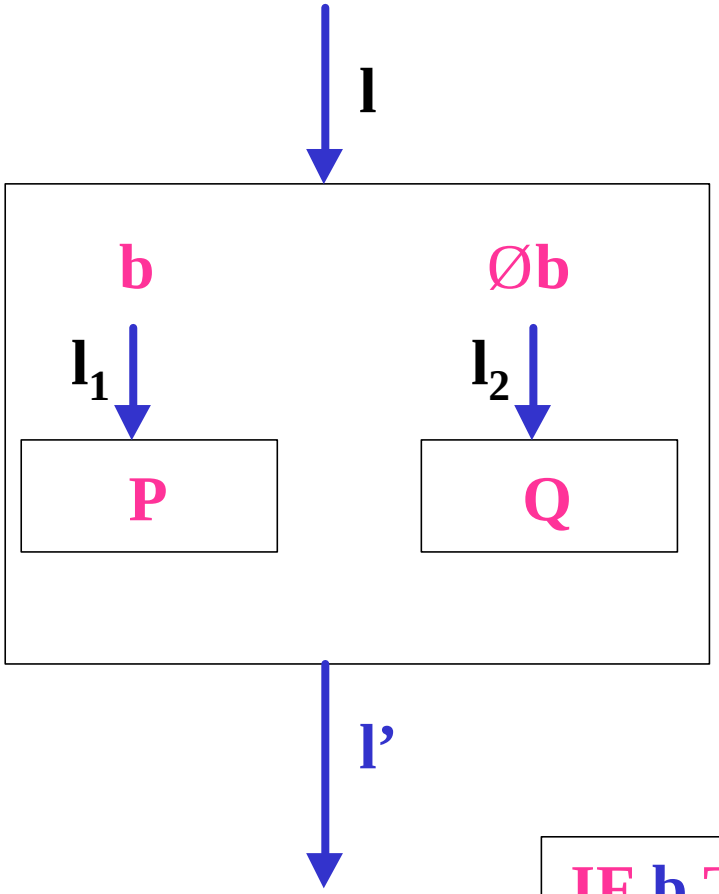
Skip



C(l, skip, l’)

pc = l Ù pc’ = l’ Ù same (V)

Conditional statement



C(l, b, IF-THEN-ELSE, l_1, l_2, l')

(pc = l Ù pc' = l_1 Ù b Ù same(V)) Ú

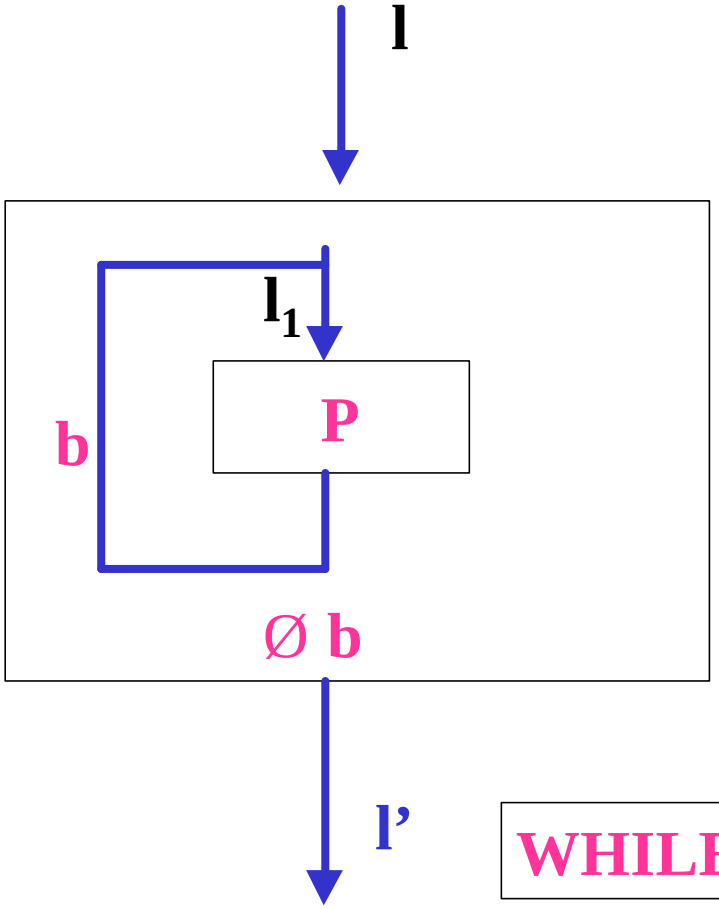
(pc = l Ù pc' = l_2 Ù Ø b Ù same(V)) Ú

C(l_1, P, l') Ú

C(l_2, Q, l')

IF b THEN P ELSE Q FI

While statement



C(l, b, l_1, WHILE, l')

(pc = l Ù pc' = l_1 Ù b Ù same(V)) Ú

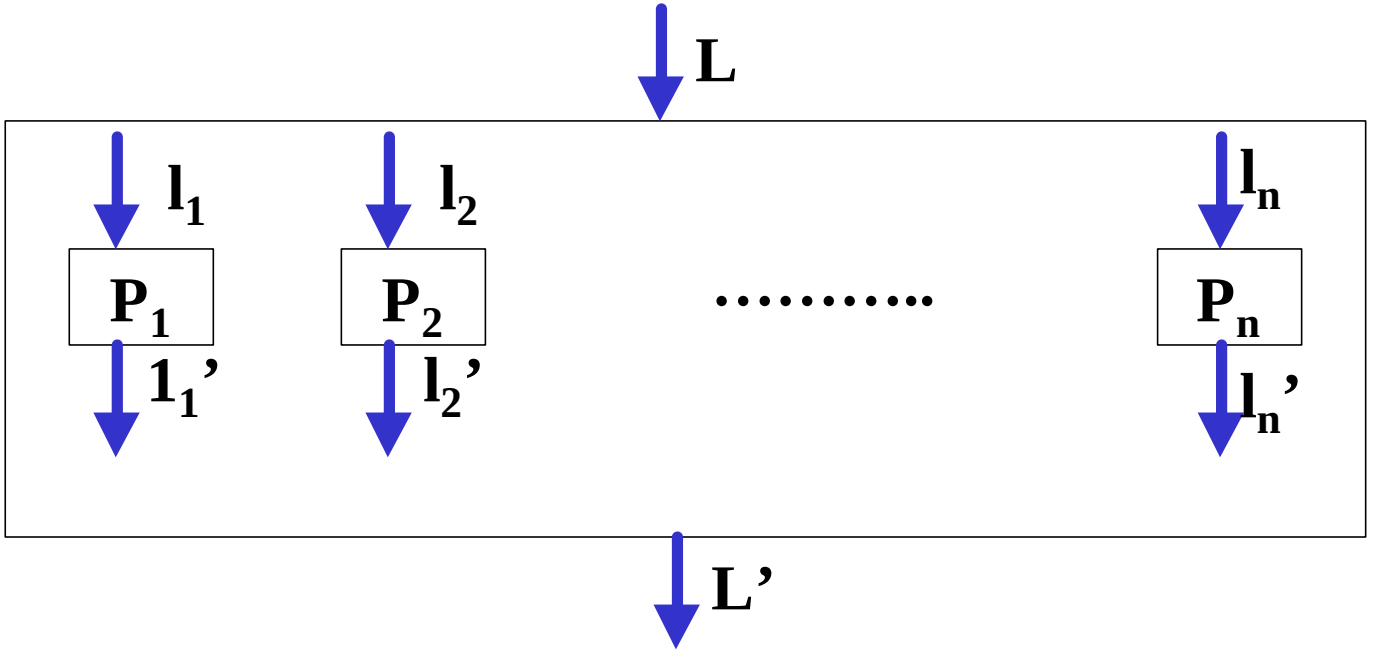
(pc = l Ù pc' = l' Ù Ø b Ù same(V)) Ú

C(l_1, P, l)

WHILE b DO P END_WHILE

Concurrent programs

- **P = cobegin ($P_1 \| P_2 \| \ldots \| P_n$) coend**
- **$P_1, P_2, .. P_n$** --- Sequential Programs.



Concurrent programs

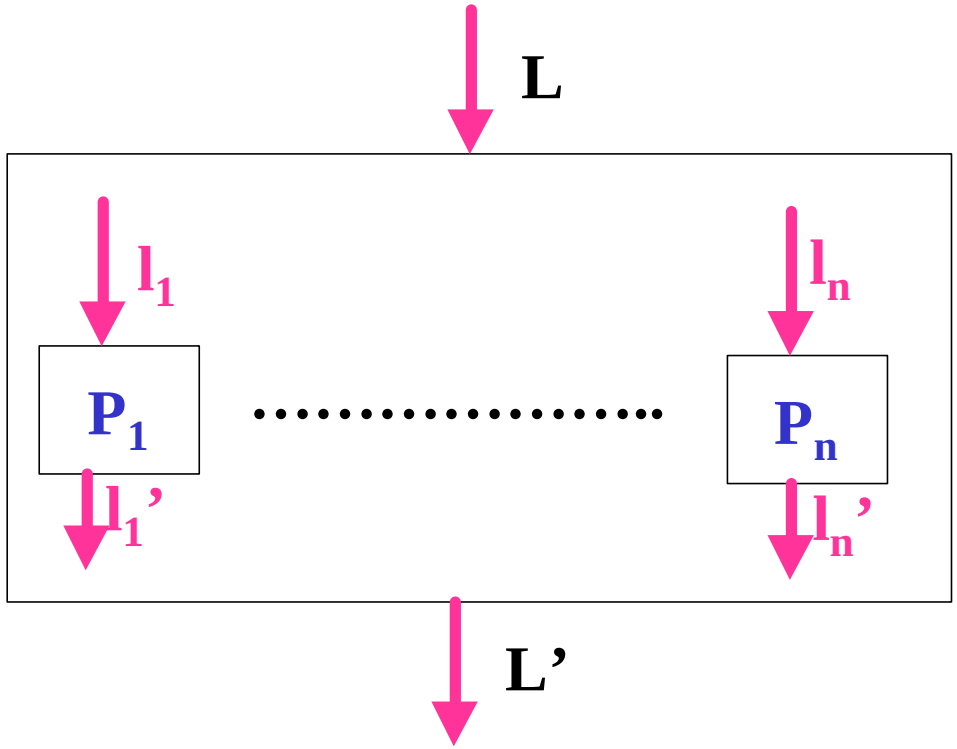
- **P = cobegin (P_1 || P_2 || …|| P_n) coend**
- **P_1, P_2,..P_n** --- ***Sequential Programs***.
- **C(l_1, P_1, l_1')** --- The transitions of **P_1** (defined inductively!).
- **V_i** ---- The set of variables of **P_i**.
- Programs may ***share*** variables !
- **pc_i** – The program counter of **P_i**.

Concurrent programs

- **pc** ---- the program counter of the concurrent program; it could be part of a larger program!
- ^ denotes the program counter value is ***undefined*** .
- $S_0(V, PC)$ = **pre(V)** Ù **pc = L** Ù

  $pc_1 = ^ \text{ Ù } \ldots\ldots \text{ Ù } pc_n = ^$

The Transition Predicate



$$(pc = L \wedge pc_1' = l_1 \wedge \ldots.$$
$$\wedge pc_n' = l_n \wedge pc' = \wedge) \vee$$
$$(pc = \wedge \wedge pc_1 = l_1' \wedge \ldots$$
$$pc_n = l_n' \wedge pc' = L' \wedge$$
$$pc_1' = \wedge \wedge \ldots pc_n' = \wedge) \vee$$
$$(C(l_1, P_1, l_1') \wedge Same\ (V - V_1)$$
$$\wedge Same(PC - \{pc_1\})) \vee \ldots$$
$$C(l_n, P_n, l_n') \wedge Same\ (V - V_n)$$
$$\wedge Same(PC - \{pc_n\}))$$

Summary

- System variables
- Domain of values
- States
- Initial state predicate
- Transition predicate
- pc values (for programs)