Introduction to Formal Method

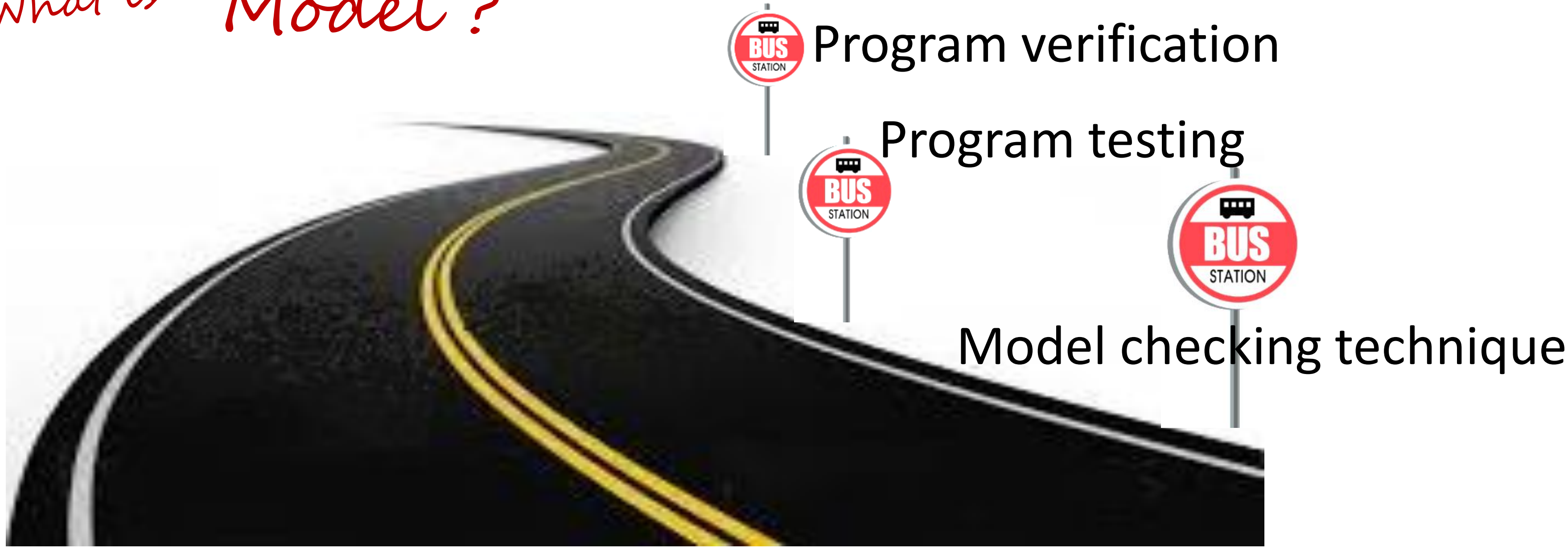
Part 2 : Principle of Model Checking

Fatemeh Ghassemi
Hossein Hojjat
University of Tehran

Outline

- Formal Methods
  - Model checking technique ?

What is Model ?

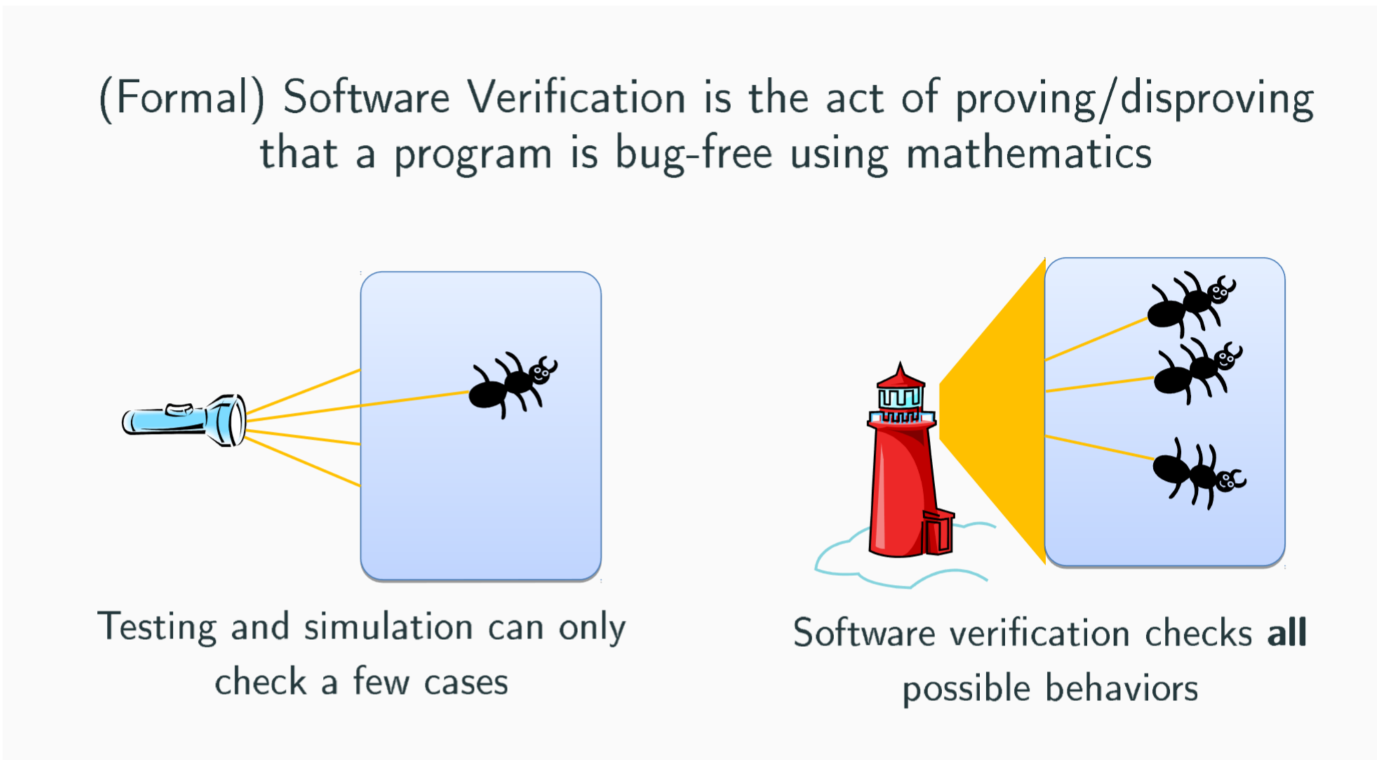


Let’s have a fun

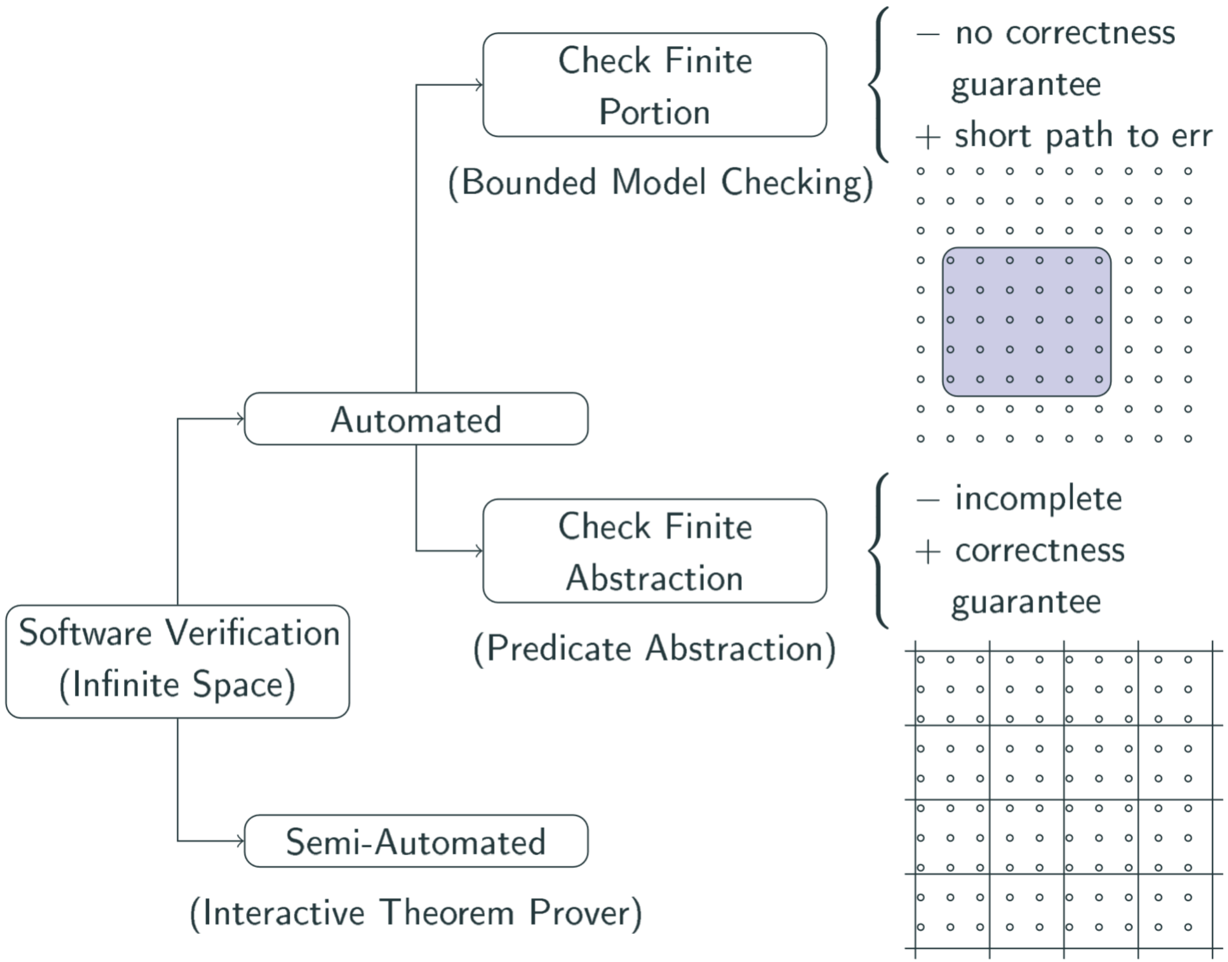
- How do you model the game to find the movements for taking the red car out ?
  - states : the status of cars
  - Transitions: movements

What we have learned so far

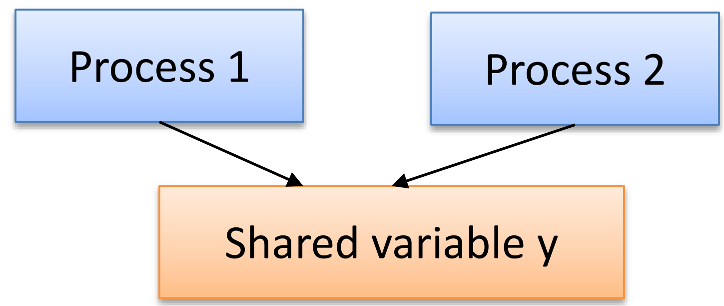
- Our focus was mainly on programs



Spectrum of approaches for program verfication



Example on mutual exclusion

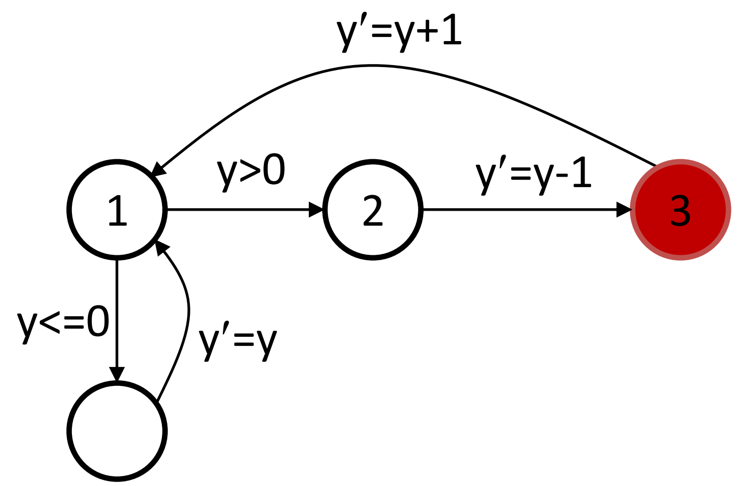



```
Loop forever
  l1: non-critical section
      while y<= 0 wait
  l2: y := y-1;
  l3: critical section
  y = y+1 ;
End loop
```

Do processes enter the critical section simultaneously ?

Example on mutual exclusion (Con.)

Loop forever
  l_1: non-critical section
    while y<= 0 wait
  l_2: y := y-1;
  l_3: critical section
  y = y+1 ;
End loop



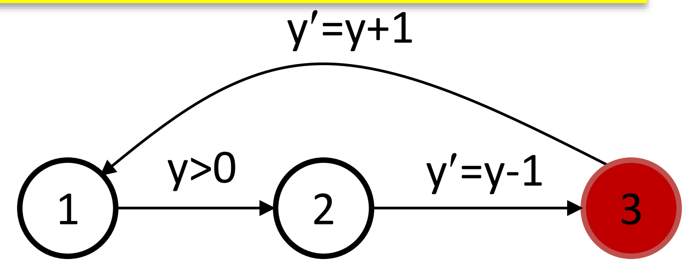
The parametrized relation for two processes:
$R(P_i, y, l_1, l_2)$

Error : R(1,y,3,3) and R(2,y,3,3)

Prove using your horn clauses

```
(set-logic HORN)
(declare-fun R (Int Int Int Int) Bool)
(assert (forall ((y Int) (id Int)) (R id y 1 1)))
; local for thread 1
(assert (forall ((y Int) (l2 Int))          (=> (and (> y 0) (R 1 y 1 l2))        (R 1 y  2 l2))))
(assert (forall ((y Int) (yp Int) (l2 Int)) (=> (and (= yp (- y 1)) (R 1 y 2 l2)) (R 1 yp 3 l2))))
(assert (forall ((y Int) (yp Int) (l2 Int)) (=> (and (= yp (+ y 1)) (R 1 y 3 l2)) (R 1 yp 1 l2))))
; local for thread 2
(assert (forall ((y Int) (l1 Int))          (=> (and (> y 0) (R 2 y l1 1))        (R 2 y  l1 2))))
(assert (forall ((y Int) (yp Int) (l1 Int)) (=> (and (= yp (- y 1)) (R 2 y l1 2)) (R 2 yp l1 3))))
(assert (forall ((y Int) (yp Int) (l1 Int)) (=> (and (= yp (+ y 1)) (R 2 y l1 3)) (R 2 yp l1 1))))
; owicki gries
(assert (forall ((y Int) (l2 Int))          (=> (and (R 1 y 1 l2) (R 2 y 1 l2) (> y 0))        (R 2 y  2 l2))))
(assert (forall ((y Int) (yp Int) (l2 Int)) (=> (and (R 1 y 2 l2) (R 2 y 2 l2) (= yp (- y 1))) (R 2 yp 3 l2))))
(assert (forall ((y Int) (yp Int) (l2 Int)) (=> (and (R 1 y 3 l2) (R 2 y 3 l2) (= yp (+ y 1))) (R 2 yp 1 l2))))

(assert (forall ((y Int) (l1 Int))          (=> (and (R 1 y l1 1) (R 2 y l1 1) (> y 0))        (R 1 y  l1 2))))
(assert (forall ((y Int) (yp Int) (l1 Int)) (=> (and (R 1 y l1 2) (R 2 y l1 2) (= yp (- y 1))) (R 1 yp l1 3))))
(assert (forall ((y Int) (yp Int) (l1 Int)) (=> (and (R 1 y l1 3) (R 2 y l1 3) (= yp (+ y 1))) (R 1 yp l1 1))))
; corrctness
(assert (forall ((y Int)) (=> (and (R 1 y 3 3) (R 2 y 3 3)) false)))
(check-sat)
(get-model)
```

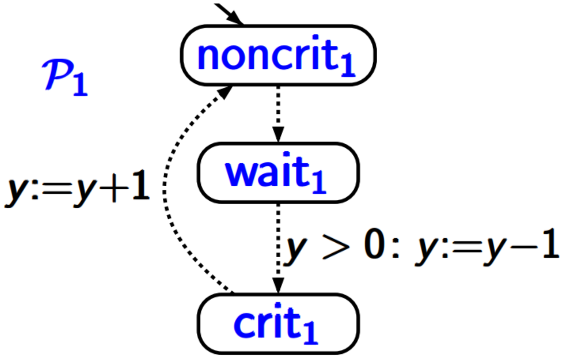


unsat 0: FALSE -> 5, 1
1: R(2, 0, 3, 3) -> 2
2: R(2, 1, 3, 2) -> 3
3: R(2, 1, 3, 1) -> 4, 8
4: R(1, 2, 2, 1) -> 12
5: R(1, 0, 3, 3) -> 6
6: R(1, 1, 2, 3) -> 9, 7
7: R(2, 2, 2, 2) -> 8
8: R(2, 2, 2, 1) -> 12, 11
9: R(1, 2, 2, 2) -> 10
10: R(1, 2, 1, 2) -> 12, 11
11: R(2, 2, 1, 1)
12: R(1, 2, 1, 1)

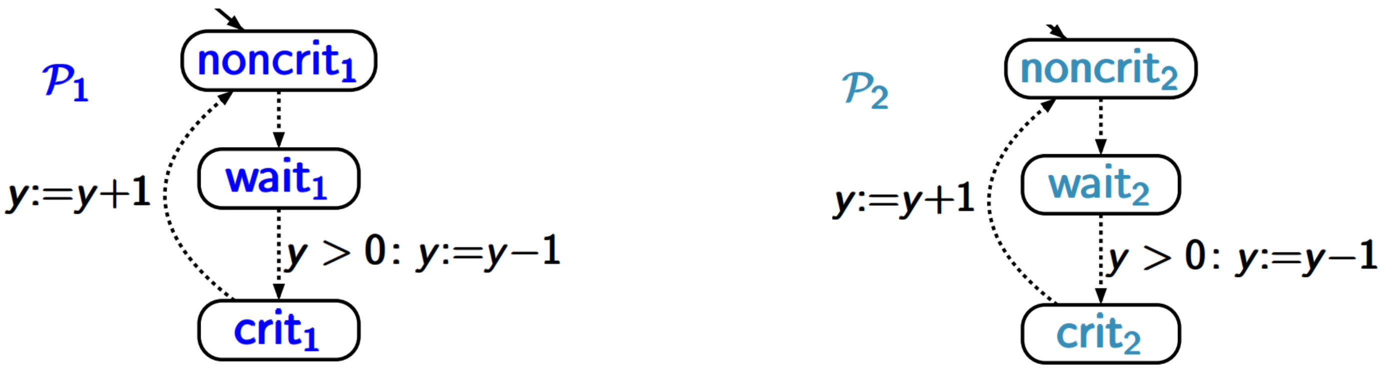
Some abstraction

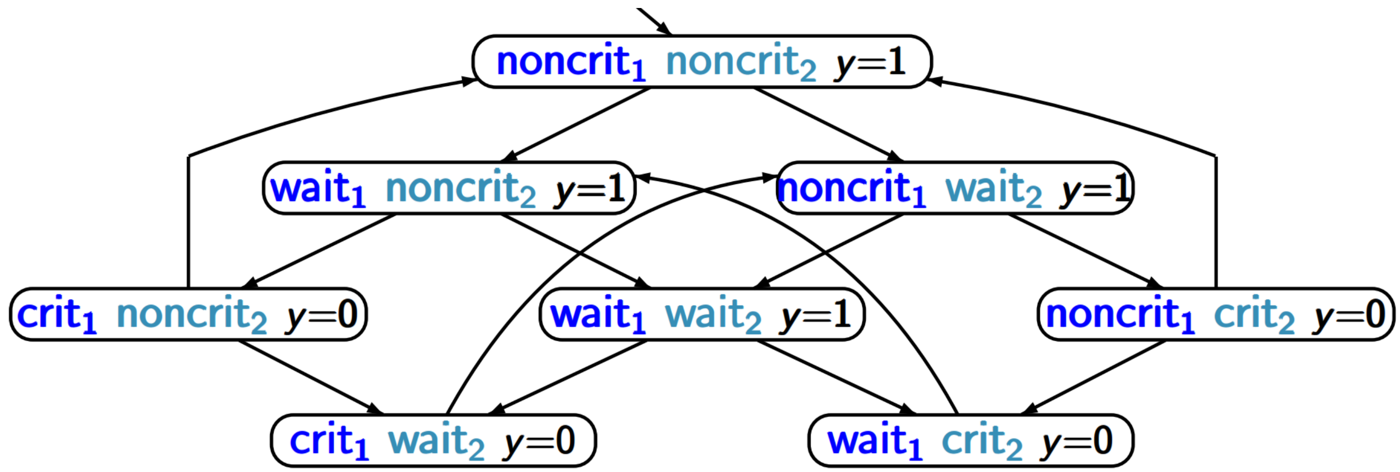
- Can we prove its correctness by using a simpler approach ?

Loop forever
  l_1: non-critical section
    await y>0 do y := y-1;
  l_3: critical section
  y = y+1 ;
End loop



Modeling by transition system





Example: A Security Protocol

- A public-key authentication protocol suggested by Needham and Schroeder
- A(lice) and B(ob) try to establish a common secret key over an insecure channel
  - Both should be convinced of each other's presence
  - An intruder can not get the secret key unless It breaks the encryption algorithm

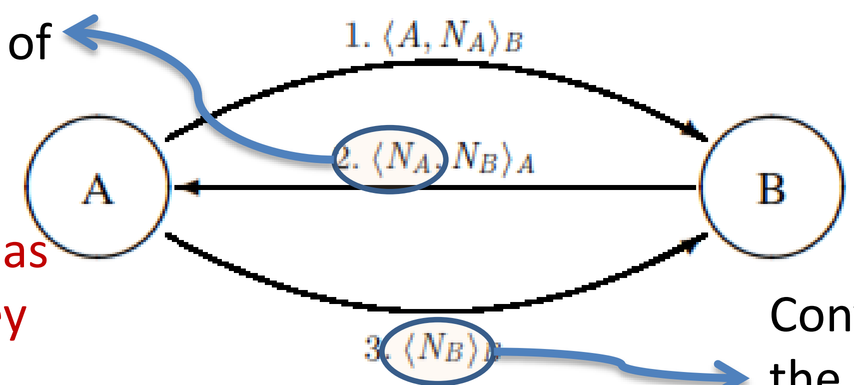
Needham and Schroeder Protocol

- It is based on exchange of three messages between the participating agents.
  - $<M>_C$ denotes that message M is encrypted using agents C’s public key.
  - Assume the encryption algorithm is secure : only agent C can decrypt $<M>_C$ to learn M.
  - All public keys are known to all agents

Needham and Schroeder Protocol (Con.)

- N_A : a random number N_A, called nonce indicating that it should be used only once by any honest agent

Convinces Alice of the authenticity of the massage



Alice accepts <N_A,N_B> as the common secret key

Convinces Bob of the authenticity of the massage

Bob also accepts <N_A,N_B> as the common secret key

Analysis : is the protocol secure ?

- Can an intruder find out the secret key ?
- Attackers can intercept messages, store them and reply them later, initiate runs or respond to runs initiated by honest agents
  - It can only decrypt with his own public key
- The protocol contains a sever flaw , discovered 17 years after its first publish, using model checking!

A PROMELA Model

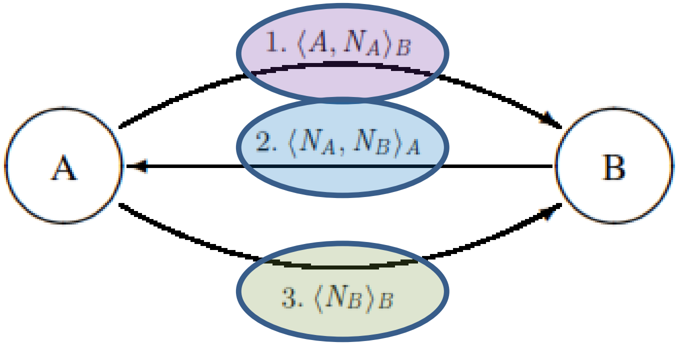
- We need some abstractions :
  - A network of only three agents A, B, I
  - A and B can only participate in one protocol run
  - A act as initiator, B as responder : A and B generate at most one nonce
  - The memory of I is limited to a single message

A PROMELA Model (Con.)

- mtype = { ok, err, msg1, msg2, msg3, keyA, keyB, keyI,agentA, agentB, agentI, nonceA, nonceB, nonceI };
- Encrypted message :
  - typedef Crypt { mtype key, info1, info2};
- Network :
  - chan network = [0] of { mtype, /* msg# */

```
                          mtype, /* receiver */
                          Crypt };
```

A PROMELA Model (Con.)




```
mtype partnerA;
mtype statusA = err;

active proctype Alice() {
      mtype pkey, pnonce;
      Crypt data;

   if /* choose a partner for this run */
   :: partnerA = agentB; pkey = keyB;
   :: partnerA = agentI; pkey = keyI;
   fi;
   network ! (msg1, partnerA, Crypt{pkey, agentA, nonceA});
   network ? (msg2, agentA, data);
   (data.key == keyA) && (data.info1 == nonceA);
   pnonce = data.info2;

   network ! (msg3, partnerA, Crypt{pkey, pnonce, 0});
   statusA = ok;
}
```

A PROMELA Model (Con.)

- Agent I is modeled nondeterministically: we describe the actions that are possible at any given state and let SPIN choose among them

```
bool knows_nonceA, knows_nonceB;

active proctype Intruder() {
    mtype msg, recpt;
    Crypt data, intercepted;
    do
    :: /*intercept or extract*/…
    :: /* Replay or send a message */ …
    Od
}
```

A PROMELA Model (Con.)

```
::/*intercept or extract*/…
   network ? (msg, _, data) ->
      if /* perhaps store the message */
      :: intercepted = data;
      :: skip;
      fi;
      if /* record newly learnt nonces */
      :: (data.key == keyI) ->
         if
         :: (data.info1 == nonceA) || (data.info2 == nonceA)
            -> knows_nonceA = true;
         :: else -> skip;
         fi;
      /* similar for knows_nonceB */
      :: else -> skip;
      fi;
```

```
:: /* Replay or send a message */
   if /* choose message type */
   :: msg = msg1;
   :: msg = msg2;
   :: msg = msg3;
   fi;
   if /* choose recipient */
   :: recpt = agentA;
   :: recpt = agentB;
   fi;
   if /* replay intercepted message or assemble it
   :: data = intercepted;
   :: if
      :: data.info1 = agentA;
      :: data.info1 = agentB;
      :: data.info1 = agentI;
      :: knows_nonceA -> data.info1 = nonceA;
      :: knows_nonceB -> data.info1 = nonceB;
      :: data.info1 = nonceI;
      fi;
   /* similar for data.info2 and data.key */
   fi;
   network ! (msg, recpt, data);
```

Model Checking the Protocol

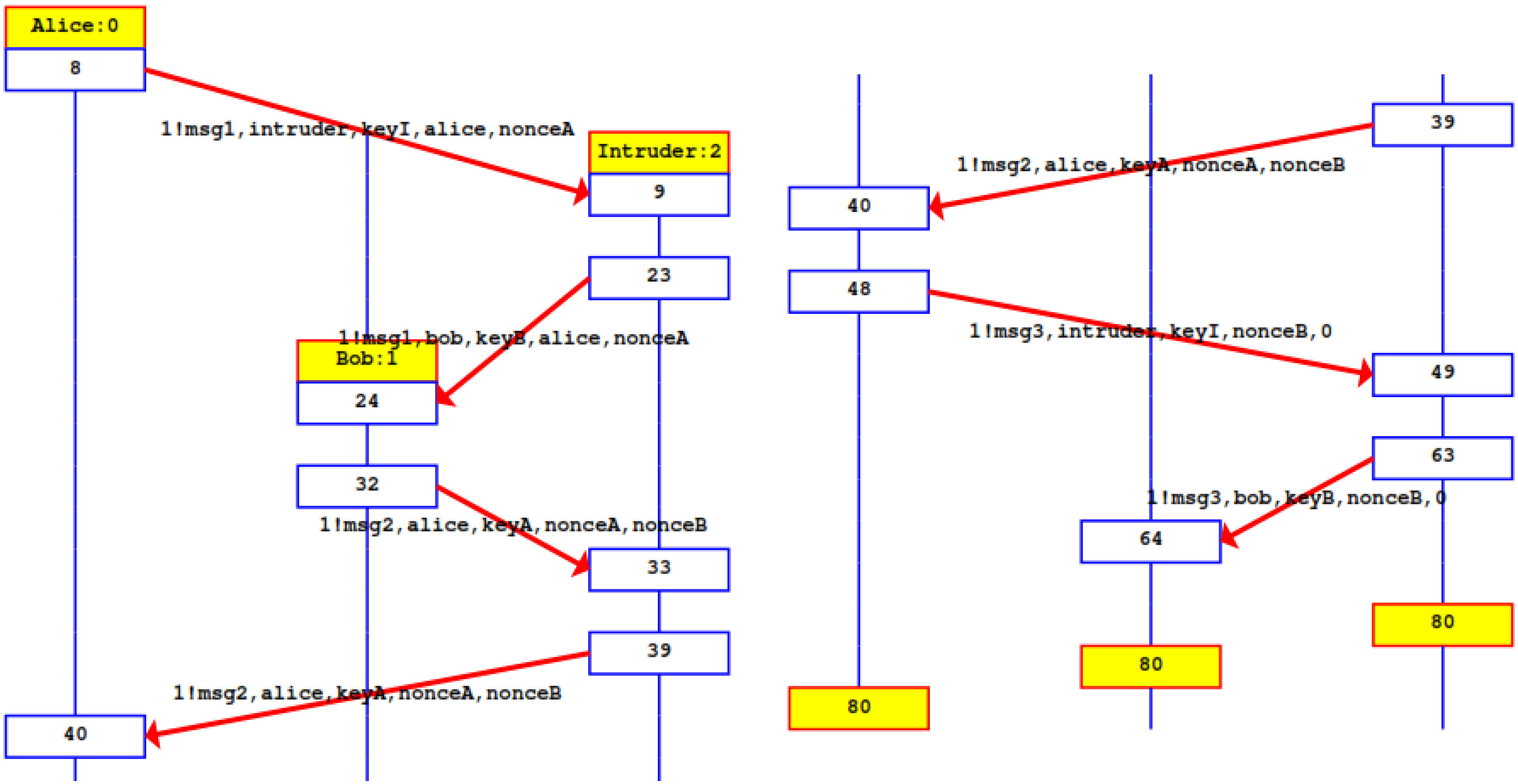
If A successfully completes a run with B then intruder should not have learnt A’s nonce

$$G(\ statusA = ok \land partnetA = agentB \ \Rightarrow \neg\ knows\_nonceA\ )$$

$$G(\ statusB = ok \land partnetB = agentA \ \Rightarrow \neg\ knows\_nonceB\ )$$

✘

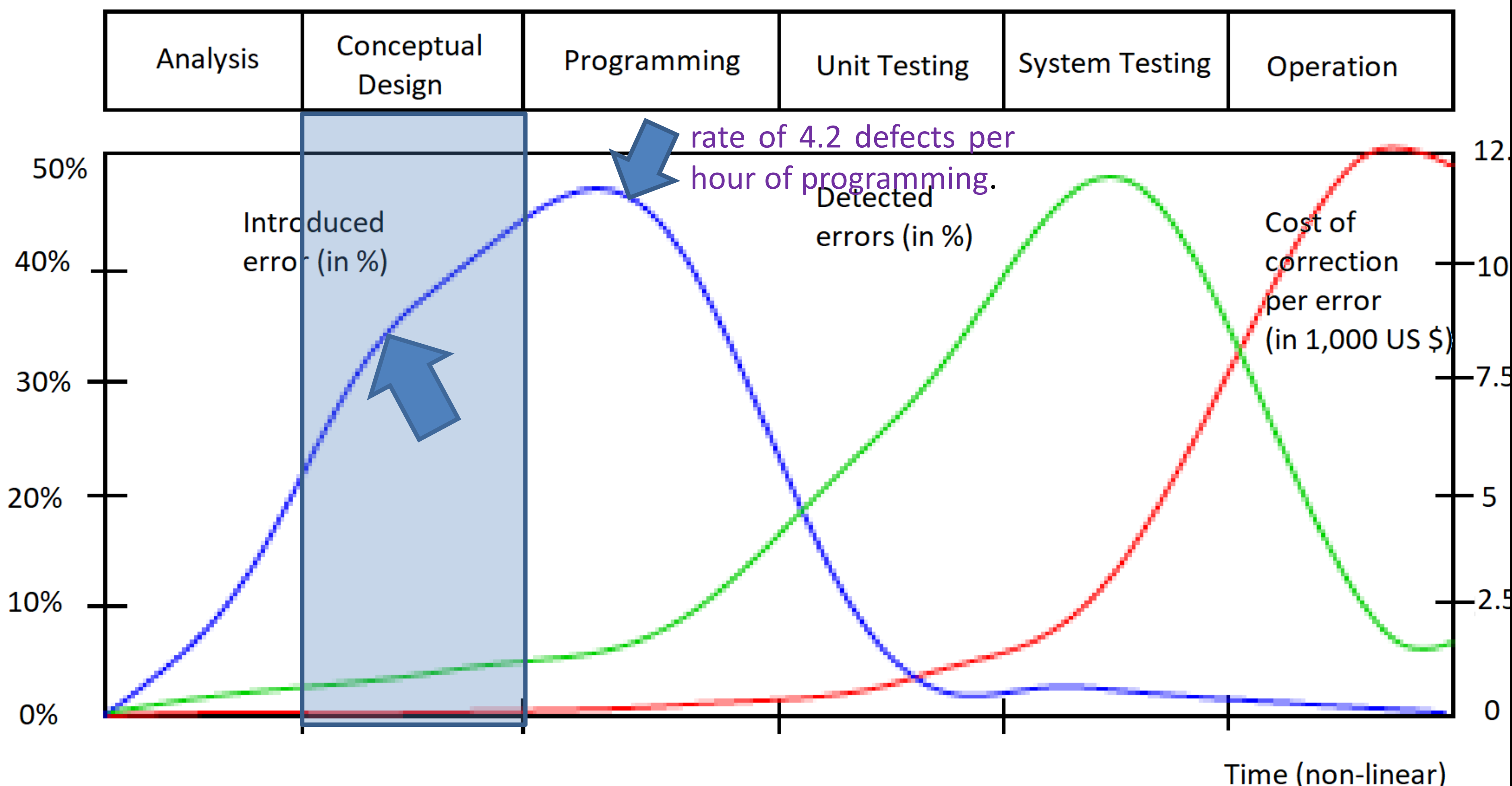
Model Checking the Protocol (Con.)



Advantage and disadvantage

- Advantage
  - We have proved that it is not correct for two agent early at the design time with minimum labor
- Disadvantage
  - If we prove that something is correct for two agent, we cannot prove that it is correct for all the number of agents
  - We cannot be sure that the implemented code conforms to the model

Bug Hunting: the Sooner, the Better



How about more complex protocols?

[Docs] [txt|pdf] [Tracker] [WG] [Email] [Diff1] [Diff2] [Nits]

Versions: (draft-ietf-manet-dymo) 00 01 02 03
04 05 06 07 08 09 10 11 12 13 14 15
16

Mobile Ad hoc Networks Working Group | C. Perkins
Internet-Draft | Futurewei
Intended status: Standards Track | S. Ratliff
Expires: January 23, 2016 | Idirect

J. Dowdell
Airbus Defence and Space
L. Steenbrink
HAW Hamburg, Dept. Informatik
V. Mercieca
Airbus Defence and Space
July 22, 2015

Ad Hoc On-demand Distance Vector (AODVv2) Routing
draft-ietf-manet-aodvv2-11

Abstract

The revised Ad Hoc On-demand Distance Vector (AODVv2) routing protocol is intended for use by mobile routers in wireless, multihop networks. AODVv2 determines unicast routes among AODVv2 routers within the network in an on-demand fashion, offering rapid convergence in dynamic topologies.

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Formal Methods

- Formal methods are the applied mathematics for modelling and analyzing ICT systems
- Formal methods offer a large potential for
  - Obtaining an early integration of verification in the design process
  - Providing more effective verification technique (higher code coverage)
  - Reducing the verification time

Milestones in Formal Verification

- Mathematical program correctness (Turing, 1949)
- Syntax-based technique for sequential programs (Hoare, 1969)
  - for a given input, does a computer program generate the correct output?
  - based on compositional proof rules expressed in predicate logic
- Syntax-based technique for concurrent programs (Pnueli, 1977)
  - handles properties referring to states during the computation
  - based on proof rules expressed in temporal logic
- Automated verification of concurrent programs
  - model-based instead of proof-rule based approach
  - does the concurrent program satisfy a given (logical) property?

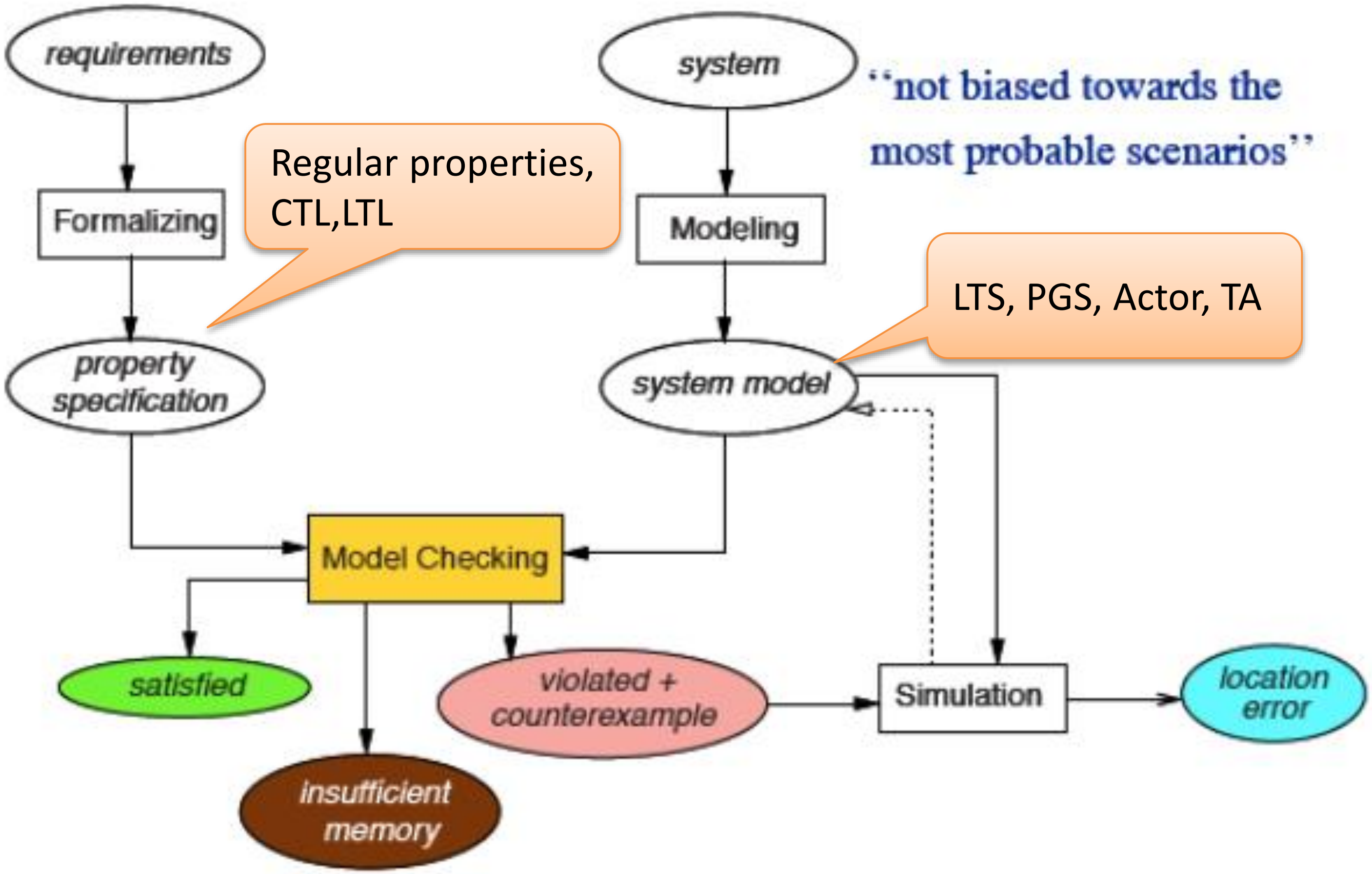
What are the counterparts of model checking technique ?

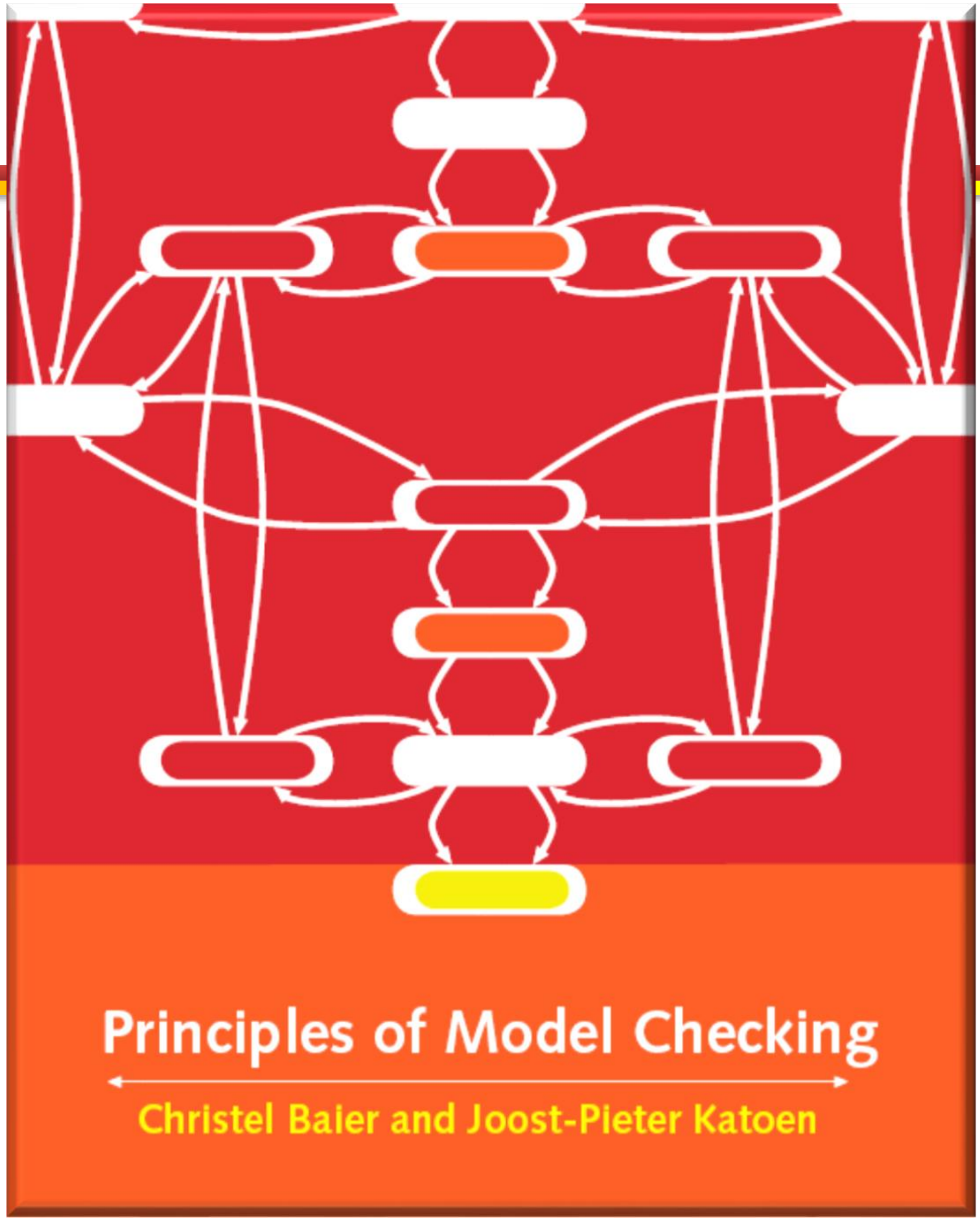
- The model checking question: does the system under the consideration **verifies** the given property ?
  - A System : Model ?
  - A Property
  - A checker

Model Checking Technique

- Model checking is an automated technique that, given a finite-state model of a system and a formal property, systematically checks whether this property holds for (a given state in) that model.

Model Checking Technique (Con.)



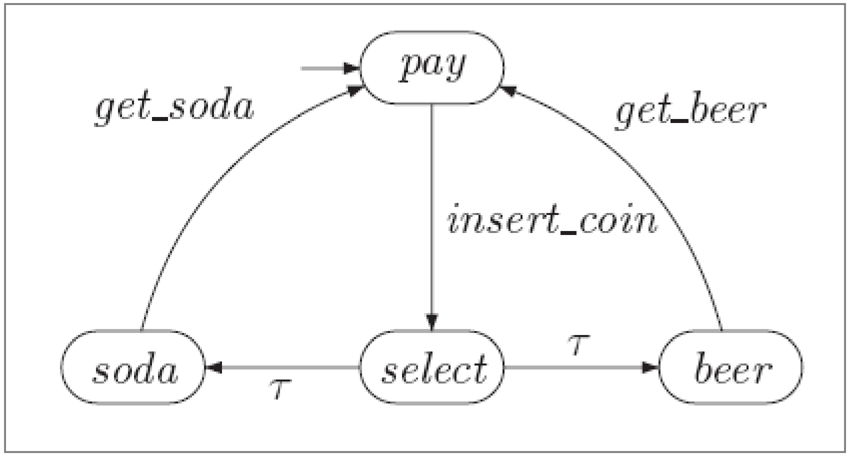

Principles of Model Checking
Christel Baier and Joost-Pieter Katoen

Transition Systems – Formal Def.

Definition 2.1. **Transition System (TS)**

A *transition system* TS is a tuple $(S, Act, \rightarrow, I, AP, L)$ where

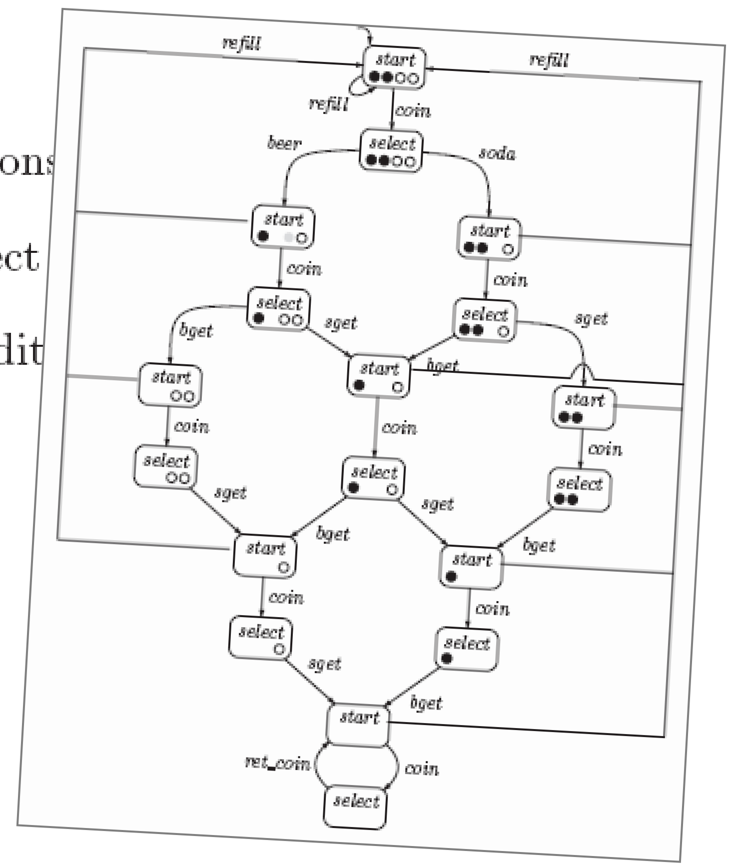
- S is a set of states,
- Act is a set of actions,
- $\longrightarrow \subseteq S \times Act \times S$ is a transition relation,
- $I \subseteq S$ is a set of initial states,
- AP is a set of atomic propositions, and
- $L : S \rightarrow 2^{AP}$ is a labeling function.



Program Graphs – Formal Def.

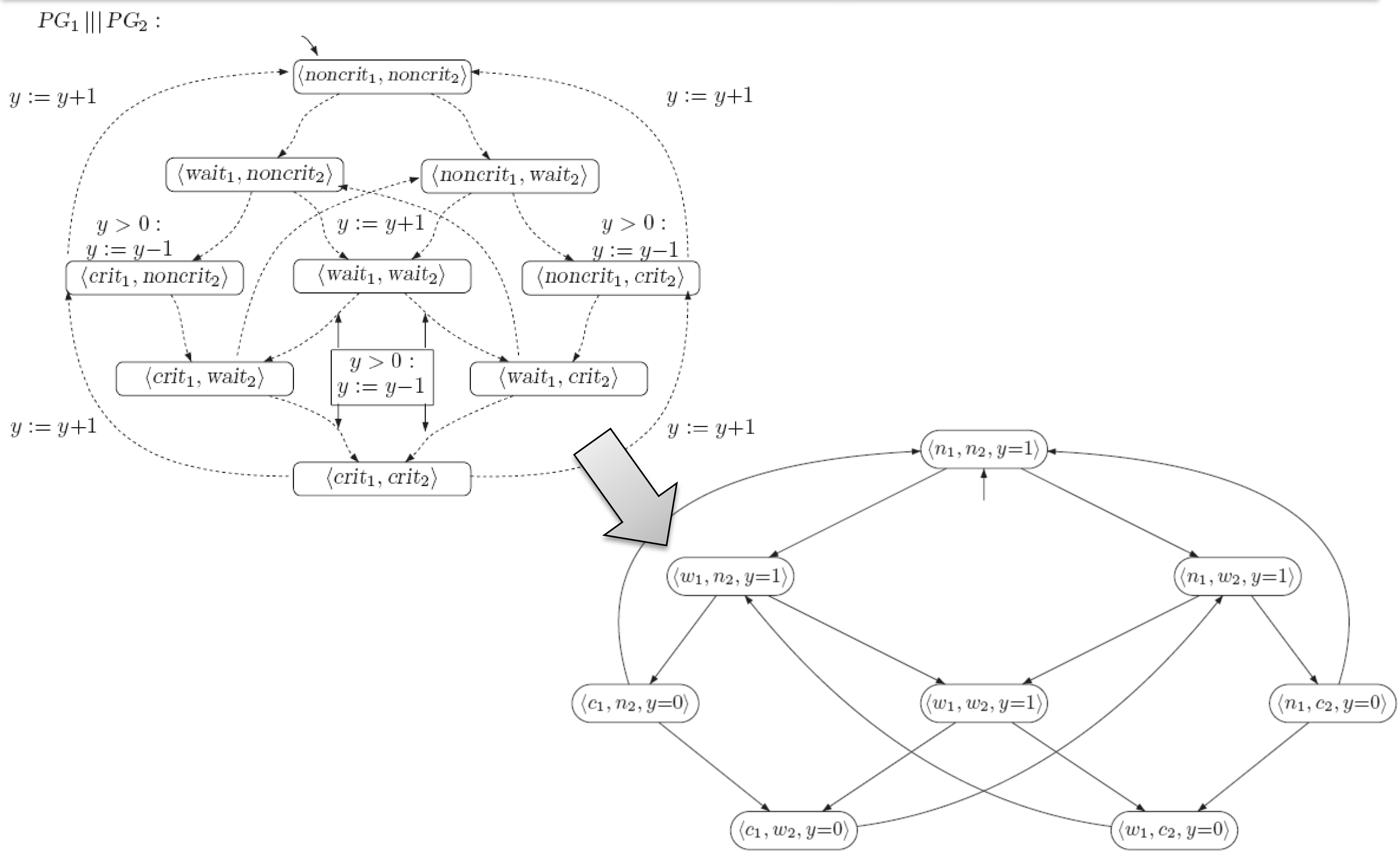
Definition 2.13. Program Graph (PG)

A *program graph PG* over set *Var* of typed variables is a tuple $(Loc, Act, Effect, \hookrightarrow, Loc_0, g_0)$ where



- Loc is a set of locations and Act is a set of actions
- $Effect : Act \times Eval(Var) \rightarrow Eval(Var)$ is the effect
- $\hookrightarrow \subseteq Loc \times Cond(Var) \times Act \times Loc$ is the condit
- $Loc_0 \subseteq Loc$ is a set of initial locations,
- $g_0 \in Cond(Var)$ is the initial condition.

From Program Graph to TS



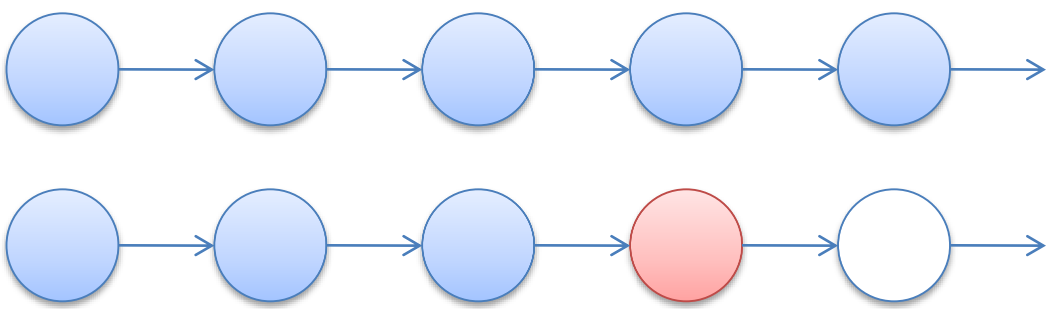
Describing Properties in Linear Temporal Logic

$$\Box\,(\text{ready} \rightarrow (\text{ready}\ \mathsf{U}\ \text{delivered}))$$

- *Globally, If A successfully completes a run with B then intruder should not have learnt the secret key.*

$\Box\, p$

$p\ \mathsf{U}\ \neg q$



There are other types of logics: CTL, Hennesy-Milner, …

Tool Support