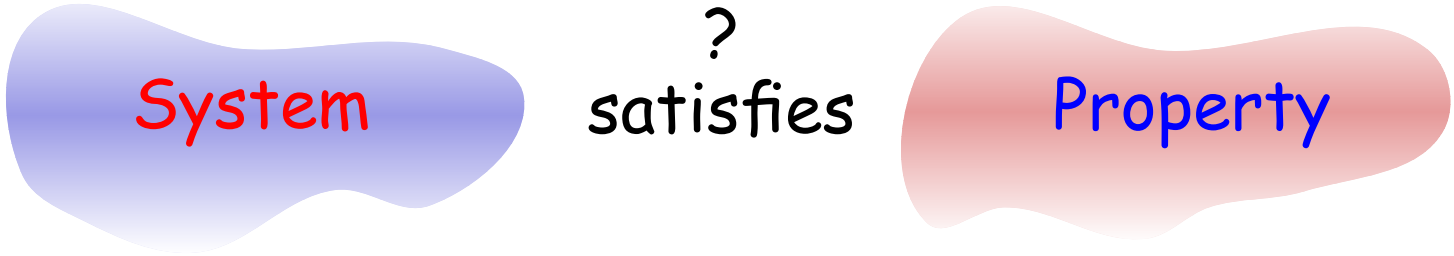
Symbolic Model Checking for Simply Timed Systems

Nicolas MARKEY and Philippe SCHNOEBELEN

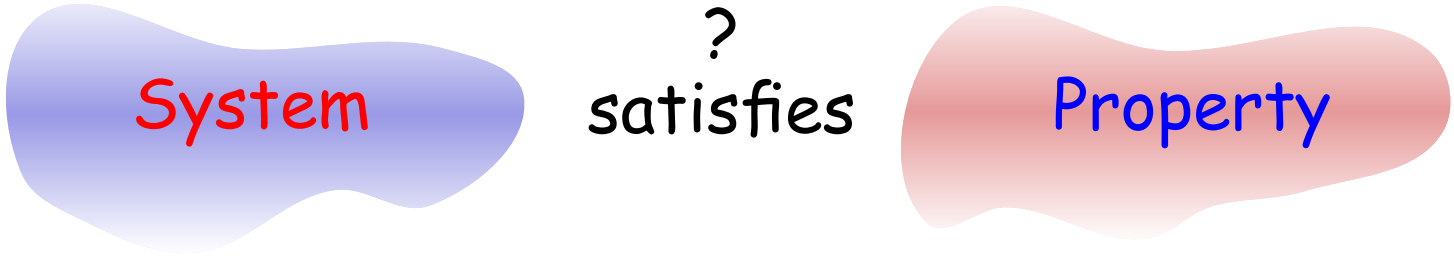
{markey,phs}@lsv.ens-cachan.fr

Lab. Specification & Verification, CNRS & ENS Cachan

Model Checking

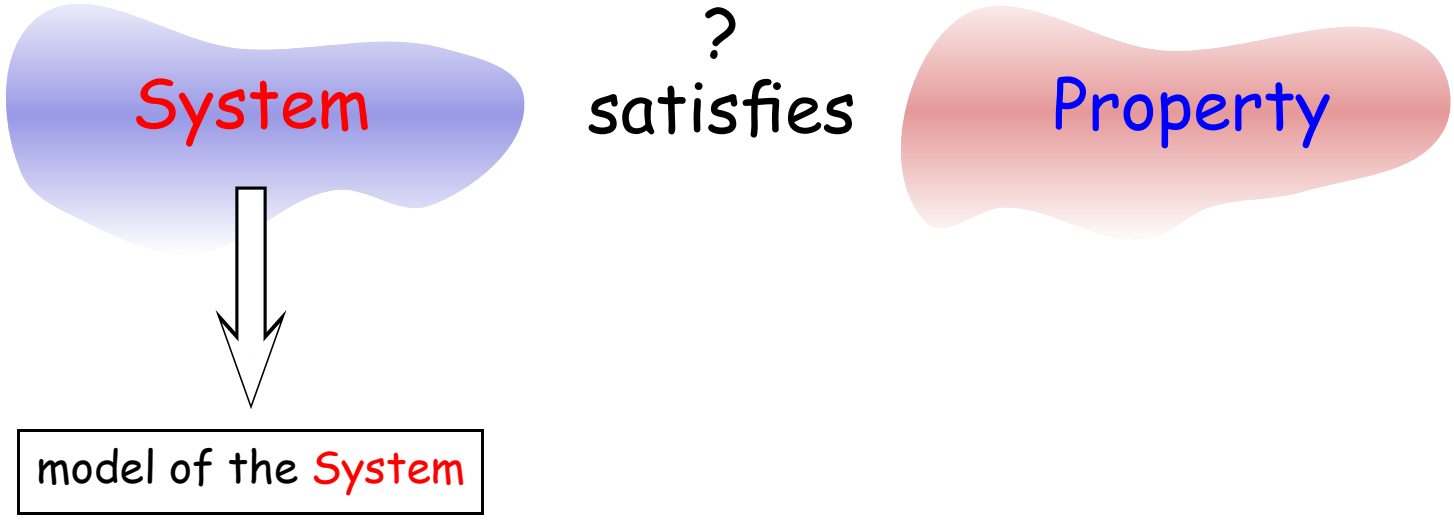


Model Checking



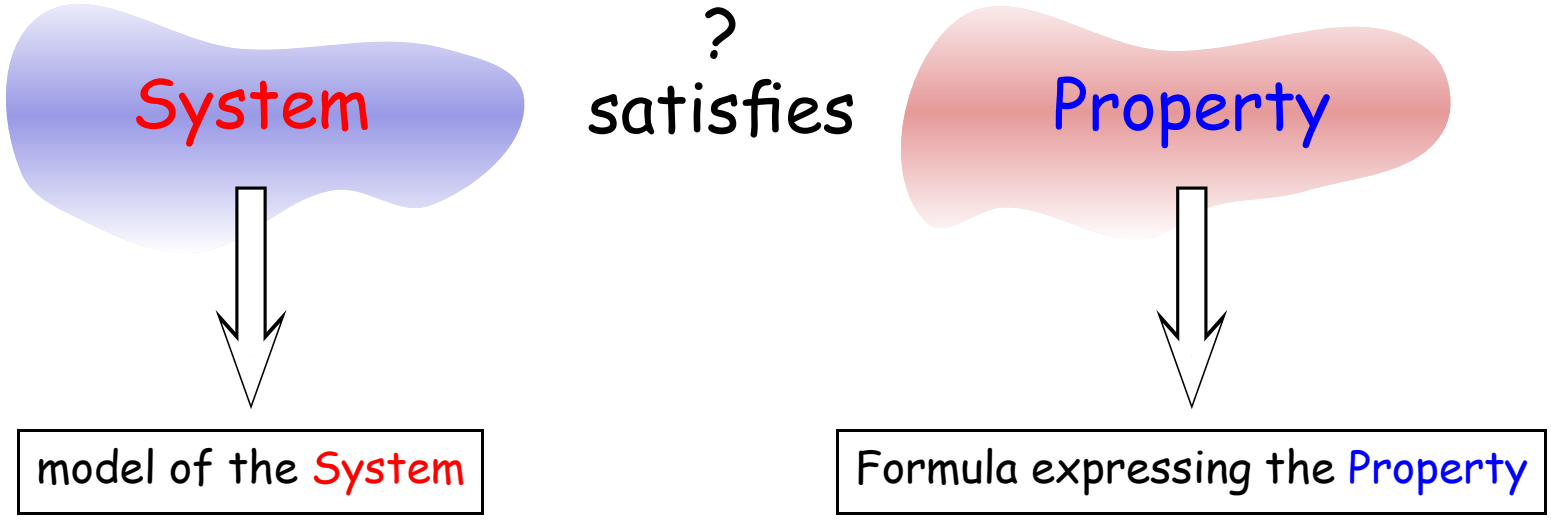
Model Checker

Model Checking



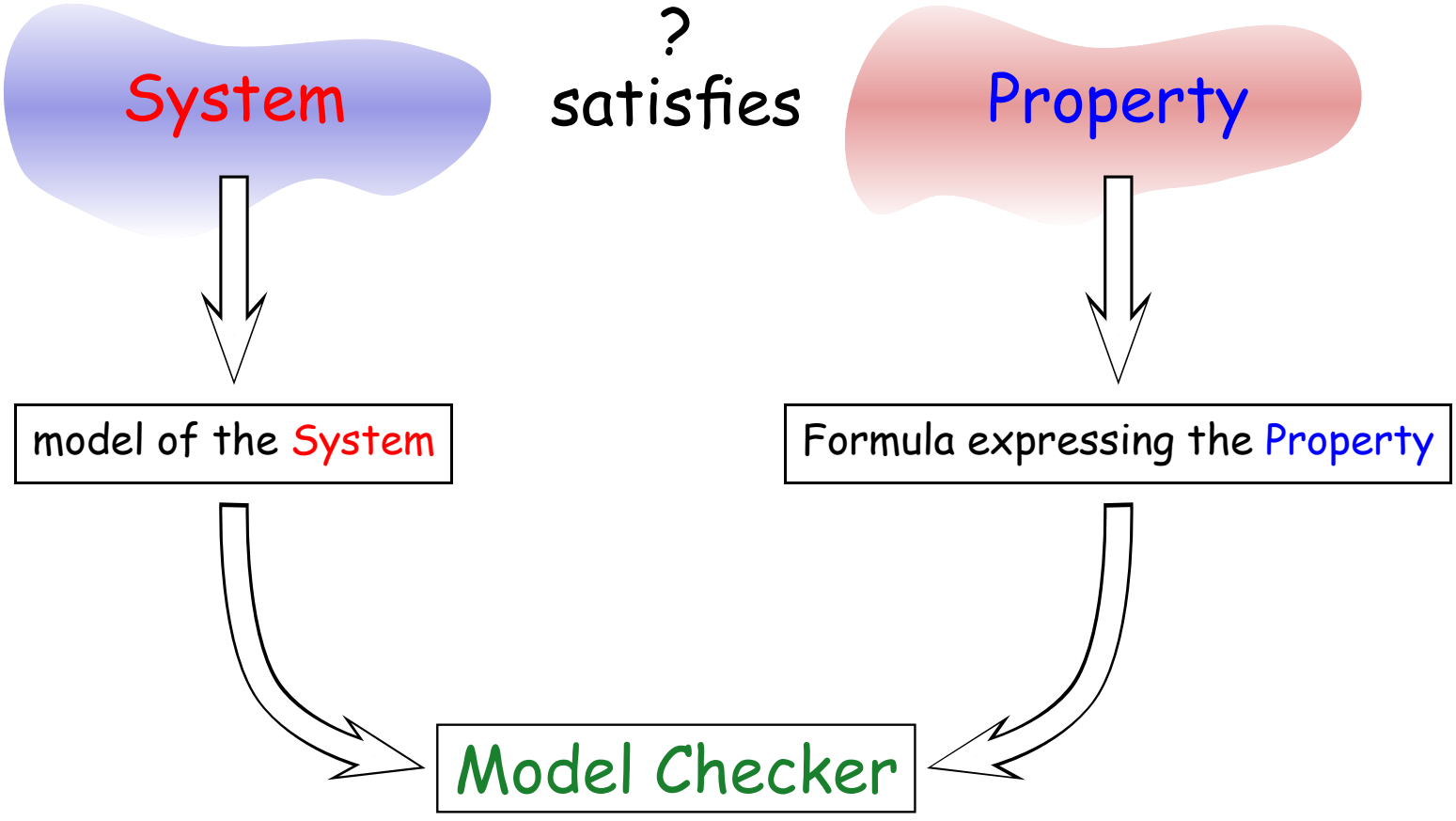
Model Checker

Model Checking

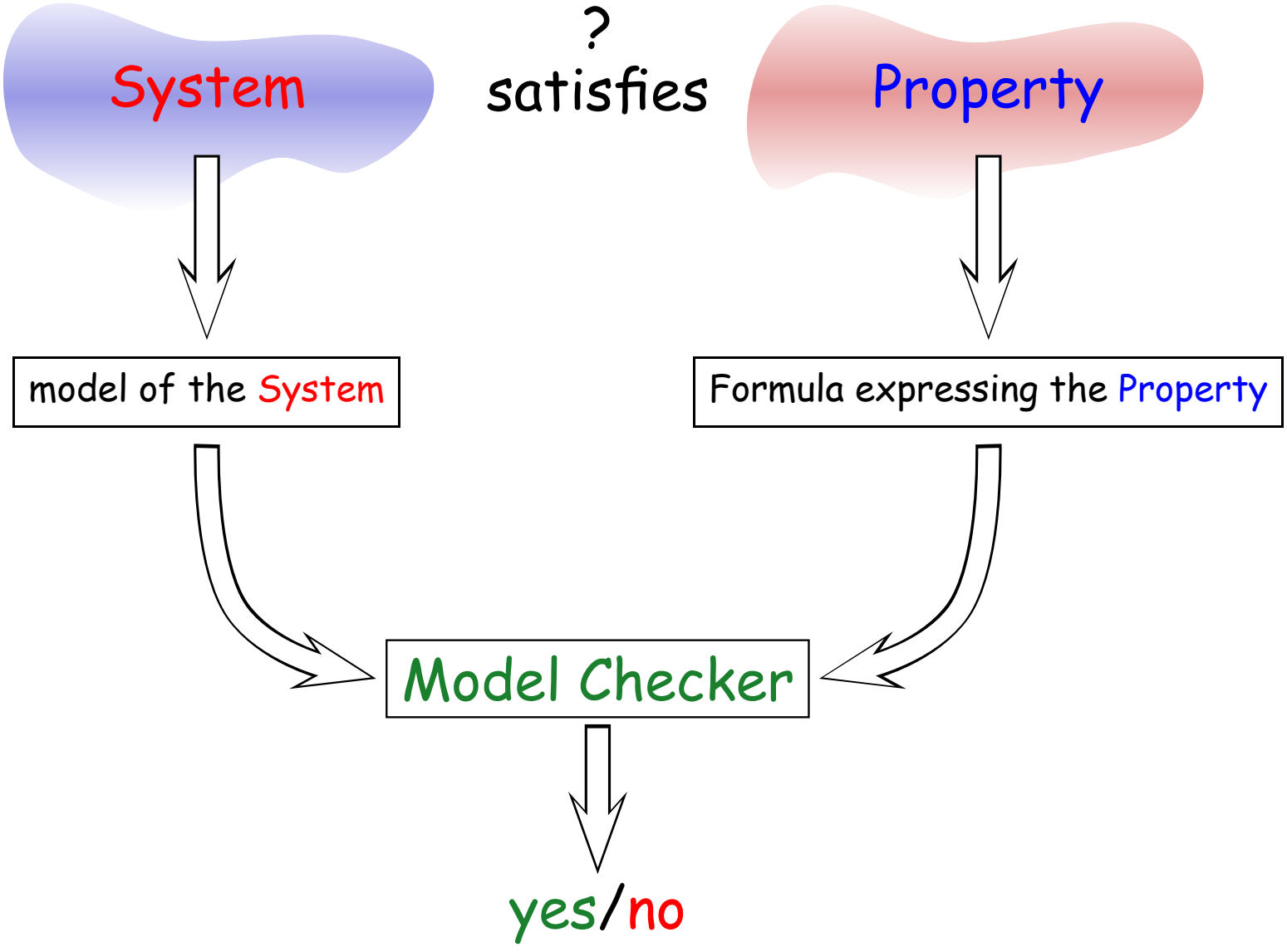


Model Checker

Model Checking



Model Checking



Timed Model Checking

We must sometimes take time into account:

$$AG\ (\text{problem} \rightarrow EF_{\leq 15}\ \text{alarm})$$

Timed Model Checking

We must sometimes take time into account:

$$AG\,(\text{problem} \rightarrow EF_{\leq 15}\,\text{alarm})$$

- Timed automata: Expressive but expensive

Timed Model Checking

We must sometimes take time into account:

$$AG\,(\text{problem} \rightarrow EF_{\leq 15}\,\text{alarm})$$

- Timed automata: Expressive but expensive
- Kripke structures: One time unit per transition.
  $\rightarrow$ more efficient algorithms, but larger models

Timed Model Checking

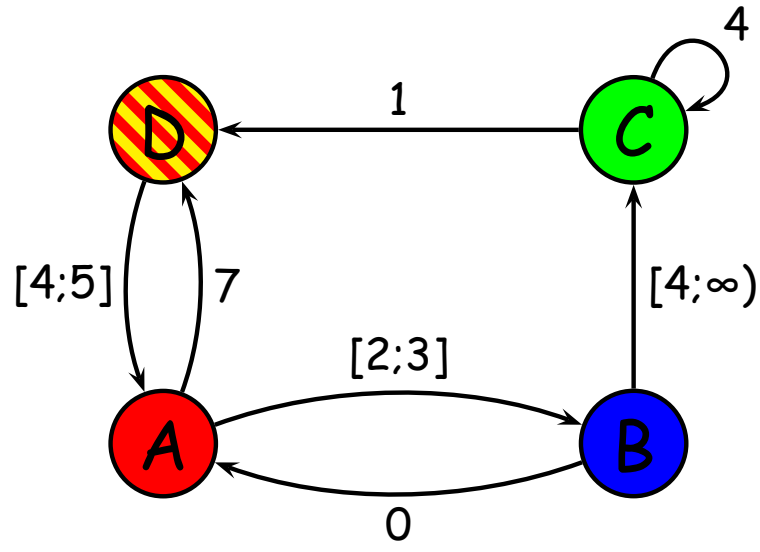
We must sometimes take time into account:

$$AG\,(\text{problem} \rightarrow EF_{\leq 15}\,\text{alarm})$$

- Timed automata: Expressive but expensive
- Kripke structures: One time unit per transition.
  $\rightarrow$ more efficient algorithms, but larger models
- Timed Kripke structures: An interesting alternative...

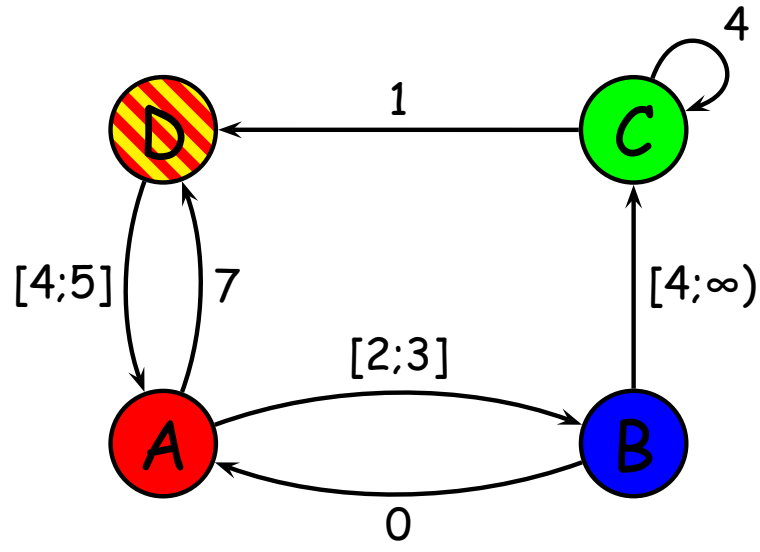
Timed Kripke Structures

Timed Kripke Structure = Kripke Structure with integer durations on transitions:



Timed Kripke Structures

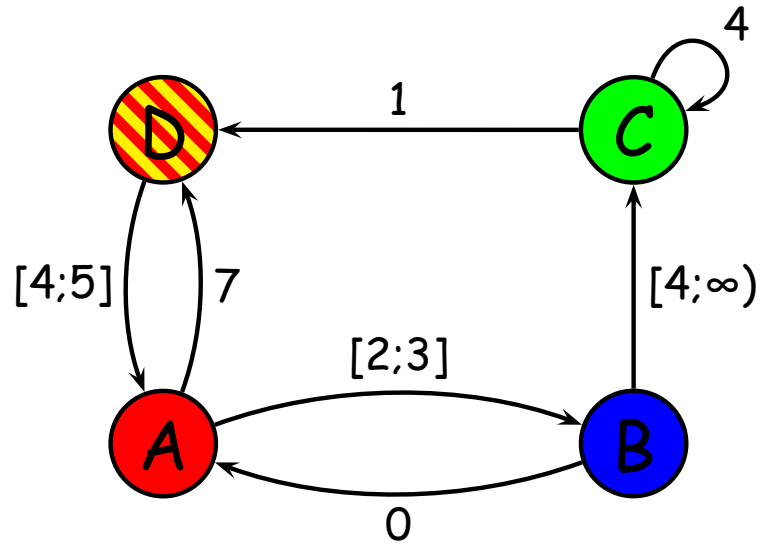
Timed Kripke Structure = Kripke Structure with integer durations on transitions:



Semantics: instantaneous steps, no intermediate state.

Timed Kripke Structures

Timed Kripke Structure = Kripke Structure with integer durations on transitions:

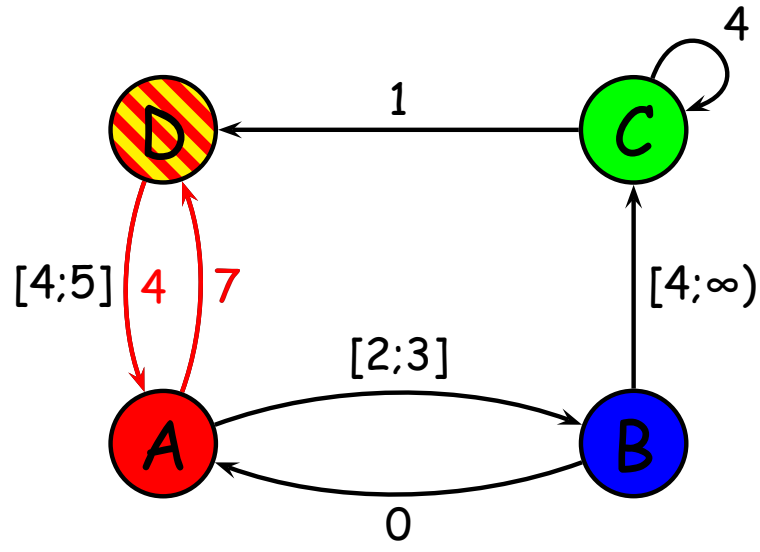


Semantics: instantaneous steps, no intermediate state.

$$A \models E\,(\text{true}\ U_{=11}\ \text{red})$$

Timed Kripke Structures

Timed Kripke Structure = Kripke Structure with integer durations on transitions:

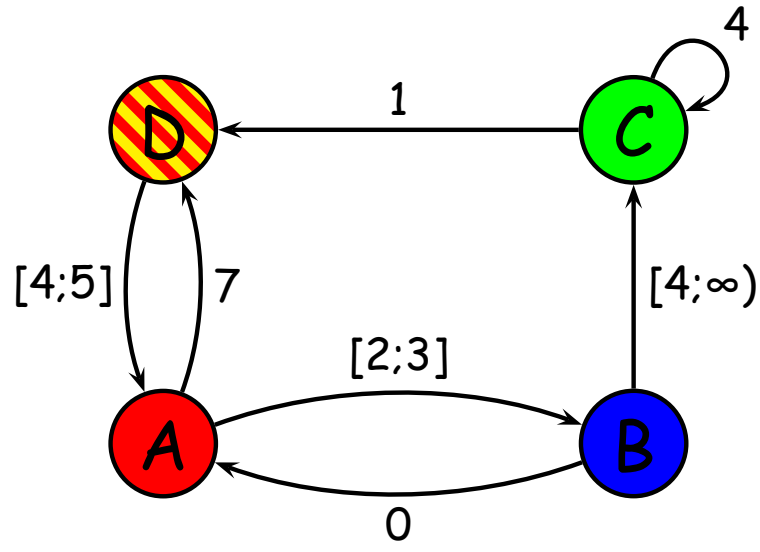


Semantics: instantaneous steps, no intermediate state.

$$A \models E\,(\text{true}\ U_{=11}\ \text{red})$$

Timed Kripke Structures

Timed Kripke Structure = Kripke Structure with integer durations on transitions:

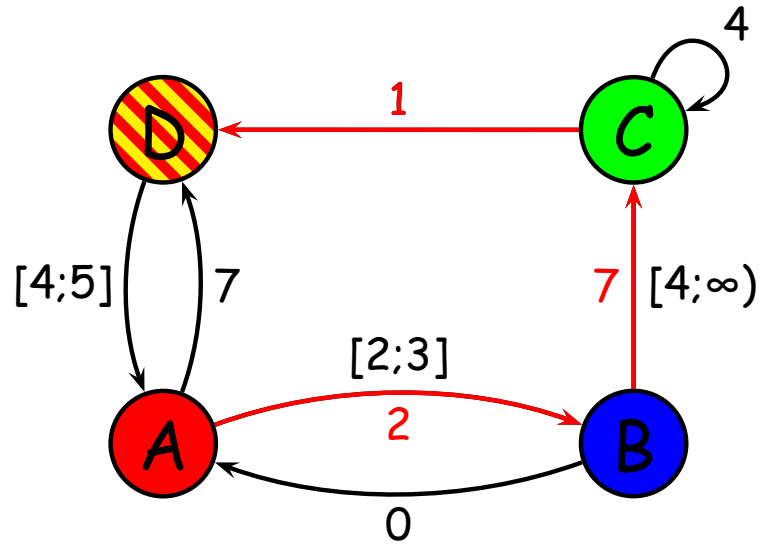


Semantics: instantaneous steps, no intermediate state.

$A \models EF_{=10}$ red

Timed Kripke Structures

Timed Kripke Structure = Kripke Structure with integer durations on transitions:

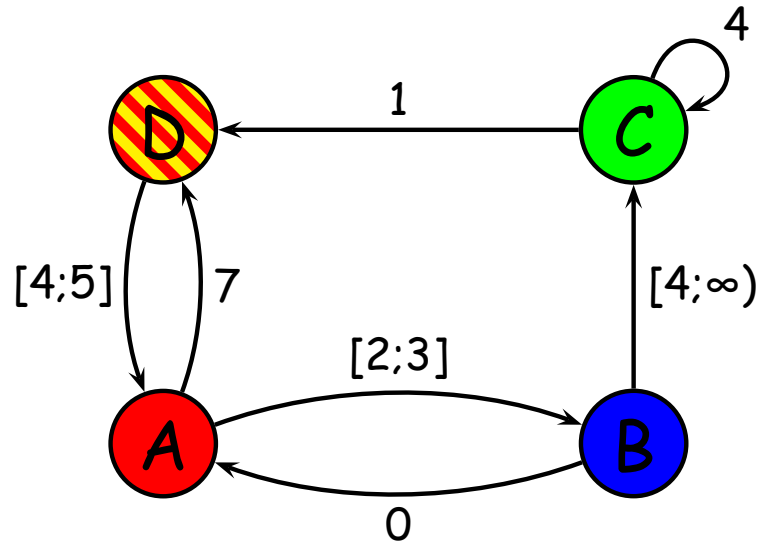


Semantics: instantaneous steps, no intermediate state.

$$A \models EF_{=10}\ red$$

Timed Kripke Structures

Timed Kripke Structure = Kripke Structure with integer durations on transitions:

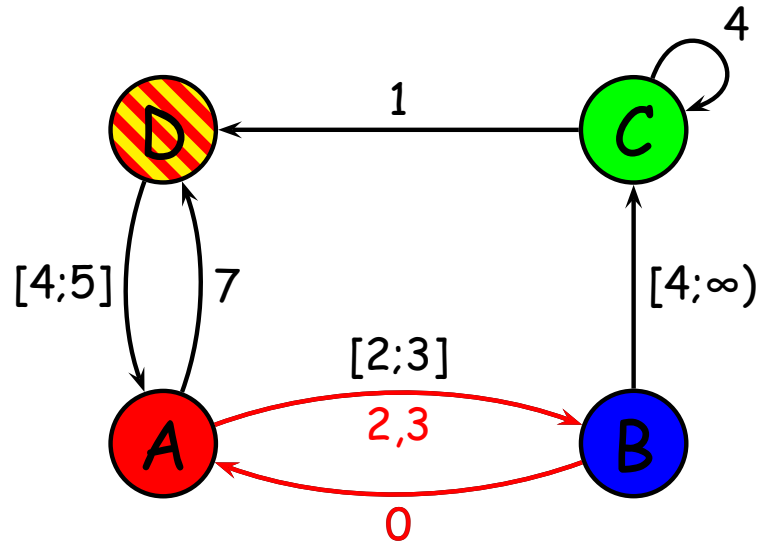


Semantics: instantaneous steps, no intermediate state.

$A \models EF_{=5}$ red

Timed Kripke Structures

Timed Kripke Structure = Kripke Structure with integer durations on transitions:

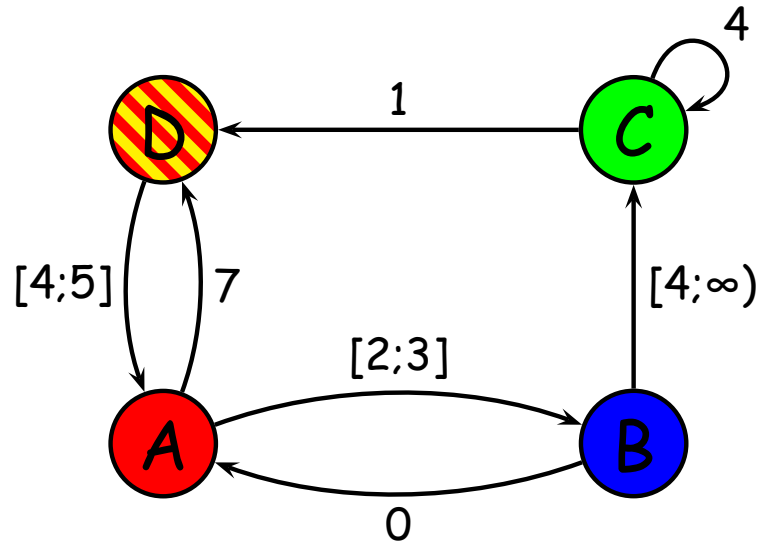


Semantics: instantaneous steps, no intermediate state.

$A \models EF_{=5}$ red

Timed Kripke Structures

Timed Kripke Structure = Kripke Structure with integer durations on transitions:

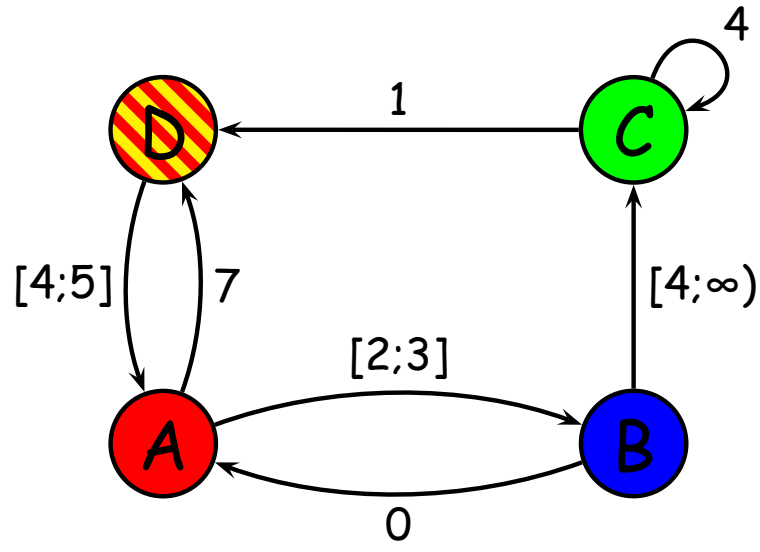


Semantics: instantaneous steps, no intermediate state.

$$A \not\models EF_{=5}\ (red \wedge blue)$$

Timed Kripke Structures

Timed Kripke Structure = Kripke Structure with integer durations on transitions:



Semantics: instantaneous steps, no intermediate state.

$$A \not\models AG\,(EF_{9..10}\ \text{green})$$

Complexity of TCTL over TKS

	KS, TKS 0/1 [EMSS90], [CC01]	TKS [LMS02]
CTL	P-complete	P-complete
TCTL $_{\leq,\geq}$	P-complete	P-complete
TCTL	P-complete	Δ_2^P-complete

$$P \subseteq NP \subseteq \Delta_2^P \stackrel{def}{=} P^{NP} \subseteq PSPACE$$

Outline

- Symbolic algorithms for TCTL model checking
  - for $EU_{=0}$ and $AU_{=0}$
  - for $EU_{[a..b]}$ and $AU_{[a..b]}$
  - special algorithms for $TCTL_{\leq,\geq}$
  - computing minimum and maximum delays
- Implementation on top of NuSMV

Algorithm for $EU_{=0}$ and $AU_{=0}$

For $E[\varphi\ U_{=0}\ \psi]$:

either ψ holds, or φ holds and some successor satisfies $E[\varphi\ U\ \psi]$ in the sub-TKS containing only transitions whose duration is zero.

Algorithm for $EU_{=0}$ and $AU_{=0}$

For $E[\varphi\ U_{=0}\ \psi]$:

either ψ holds, or φ holds and some successor satisfies $E[\varphi\ U\ \psi]$ in the sub-TKS containing only transitions whose duration is zero.

For $EG_{=0}\ \varphi$:

either there is a path where φ always holds, or there is a path verifying φ until a non-zero transition is eventually reached.

Algorithm for $EU_{=0}$ and $AU_{=0}$

For $E[\varphi\ U_{=0}\ \psi]$:

either ψ holds, or φ holds and some successor satisfies $E[\varphi\ U\ \psi]$ in the sub-TKS containing only transitions whose duration is zero.

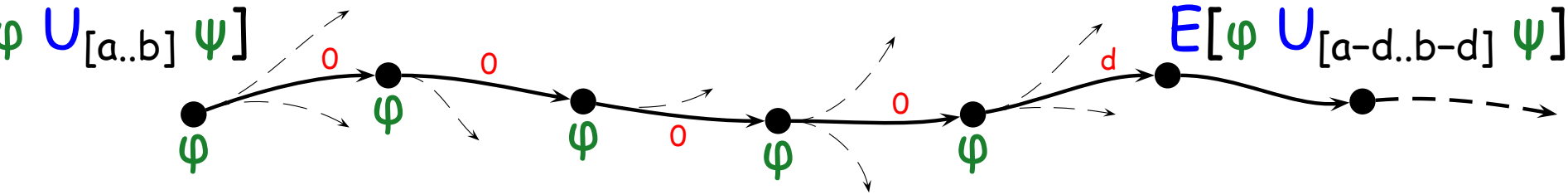
For $EG_{=0}\ \varphi$:

either there is a path where φ always holds, or there is a path verifying φ until a non-zero transition is eventually reached.

For $A[\varphi\ U_{=0}\ \psi]$:

both $A[\varphi\ U\ \psi]$ and $AF_{=0}\ \psi$ must hold.

Algorithm for $EU_{[a..b]}$



At each step i between 0 and b, we iteratively compute max_duration + 1 sets of states s.t.

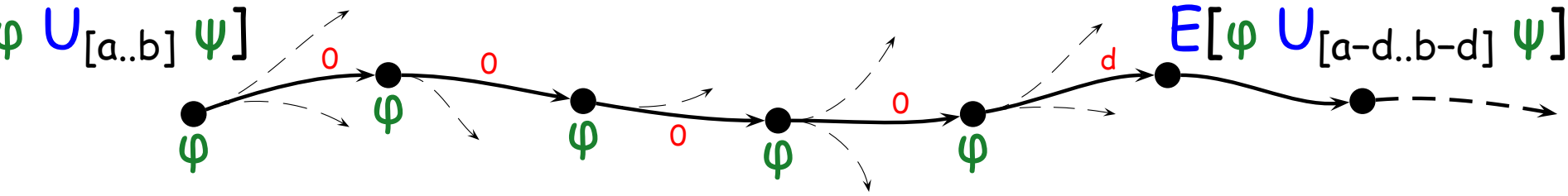
$Set_i[j] = E[(\varphi\ U_{[i+j-1+a-b..i+j-1]}\ \psi) \wedge$ first non-neg. duration $\geq j]$

Thus

$$Set_i[1] = E[(\varphi\ U_{[i+a-b..i]}\ \psi)$$

and

Algorithm for $EU_{[a..b]}$



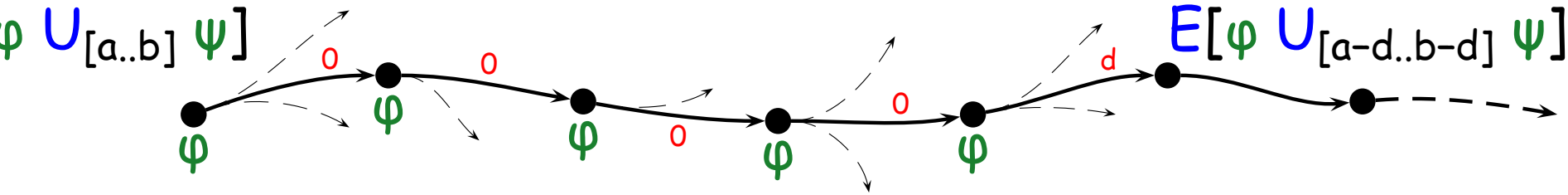
At each step i between 0 and b, we iteratively compute $max_duration + 1$ sets of states s.t.

$Set_i[j] = E[(\varphi\ U_{[i+j-1+a-b..i+j-1]}\ \psi) \wedge$ first non-neg. duration $\geq j]$

First step:

$Set_0[j] = E[(\varphi\ U_{[j-1+a-b..j-1]}\ \psi) \wedge$ first non-neg. duration $\geq j]$

Algorithm for $EU_{[a..b]}$



At each step i between 0 and b, we iteratively compute max_duration + 1 sets of states s.t.

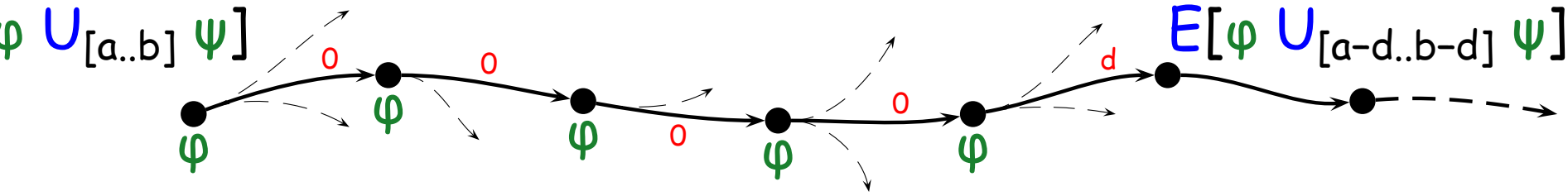
$Set_i[j] = E[(\varphi\ U_{[i+j-1+a-b..i+j-1]}\ \psi) \wedge$ first non-neg. duration $\geq j]$

First step:

$Set_0[j] = E[(\varphi\ U_{[j-1+a-b..j-1]}\ \psi) \wedge$ first non-neg. duration $\geq j]$

$= E[(\varphi\ U_{=0}\ \psi) \wedge$ first non-neg. duration $\geq j]$

Algorithm for $EU_{[a..b]}$



At each step i between 0 and b, we iteratively compute max_duration + 1 sets of states s.t.

$Set_i[j] = E[(\varphi\ U_{[i+j-1+a-b..i+j-1]}\ \psi) \wedge$ first non-neg. duration $\geq j]$

Step i:

$$Set_i[j] = Set_{i-1}[j+1] \cup E[(\varphi\ U_{=0}(\varphi \wedge \xrightarrow{j} \wedge Set_{i-1}[1]))]$$

Algorithm for $EU_{[a..b]}$

```
if (b < a) { return(∅); }

/* Initialization */
for (j = 1; j ≤ max_duration; j++) {
    if (j ≤ b − a + 1) { Set[j] ← E[ φ U=0 (ψ ∧ EX EF=0 duration ≥ j) ] }
    if (j > b − a + 1) { Set[j] ← ⊥ }
}
Res ← E[ φ U=0 ψ ];

/* Loop */
for (i = 1; i ≤ b; i++) {
   for (j = 1; j ≤ max_duration; j++) {
      Set[j] ← Set[j + 1] ∪ E[ φ U=0 (φ ∧ EX (Res ∧ duration = j)) ]
   }
   Res ← Set[1];
}
if (i ≤ b − a) { Res ← Res ∪ ψ; }

/* End */
return(Res);
```

Algorithm for $EU_{\leq a}$

We could use $EU_{[0..a]}$, but it would be expensive: theoretically, $TCTL_{\leq,\geq}$ is easier than TCTL.

Algorithm for $EU_{\leq a}$

We could use $EU_{[0..a]}$, but it would be expensive: theoretically, $TCTL_{\leq,\geq}$ is easier than TCTL.

We design specific algorithms for $TCTL_{\leq,\geq}$, using only the "minimal duration" transition relation:

$$q \xrightarrow{d} q' \quad \text{iff} \quad \begin{cases} q \xrightarrow{d} q' \\ \forall d'.\ (d' \geq d \ \vee \ q \not\xrightarrow{d'} q') \end{cases}$$

This relation can be computed efficiently with BDDs.

Algorithm for $EU_{\leq a}$

```
/* Initialization */
for (j = 1; j ≤ max_duration; j++) {
    Incr[j] ← ⊥;
}
Res ← E[ φ U=0 ψ ];
/* Loop */
for (i = 1; i ≤ a; i++) {
   for (j = 1; j ≤ max_duration; j++) {
      Incr[j] ← Incr[j + 1] ∪ E[ φ U=0 (φ ∧ EX→ (Res ∧ duration = j)) ]
   }
   Res ← Res ∪ Incr[1];
}
/* End */
return(Res);
```

$$Set_i[j] = E[(\varphi\ U_{[j..i+j-1]}\ \psi) \wedge \text{first duration} \geq j]$$

$$Res_i = E[\varphi\ U_{\leq i}\ \psi]$$

Algorithm for $EU_{\geq a}$

```
/* Initialization */
Res ← E[ φ U ψ ];
for (j = 1; j ≤ max_duration; j++) {
    Incr[j] ← E[ φ U=0 (φ ∧ EX⁻ (Res ∧ duration ≥ j)) ];
}
/* Loop */
for (i = 1; i ≤ a; i++) {
   for (j = 1; j ≤ max_duration; j++) {
      Incr[j] ← Incr[j + 1] ∪ E[ φ U=0 (φ ∧ EX⁻ (Res ∧ duration = j)) ]
   }
   Res ← Incr[1];
}
/* End */
return(Res);
```

$$Incr[j] = E[(\varphi\ U_{\geq i+j-1}\ \psi) \wedge \text{first duration} \geq j]$$

$$Res = E[\varphi\ U_{\geq i}\ \psi]$$

Minimum and maximum delays

In order to compute the minimum delay between a set S and a set T, we recursively compute $EF_{\leq a}T$ until it intersects with S or it reaches a fixpoint.

We compute maximum delays by computing $EG_{\leq a}\neg T$, until it does not intersect with S anymore, or until it reaches a fixpoint.

Outline

- Symbolic algorithms for TCTL model checking
  - for $EU_{=0}$ and $AU_{=0}$
  - for $EU_{[a..b]}$ and $AU_{[a..b]}$
  - special algorithms for $TCTL_{\leq,\geq}$
  - computing minimum and maximum delays
- Implementation on top of NuSMV

What is NuSMV?

(Nu)SMV = symbolic CTL model-checker.

What is NuSMV?

(Nu)SMV = symbolic CTL model-checker.

SMV: "Symbolic Model Verifier".
Developped at CMU (Pittsburgh, USA)
by Ken McMillan et al.

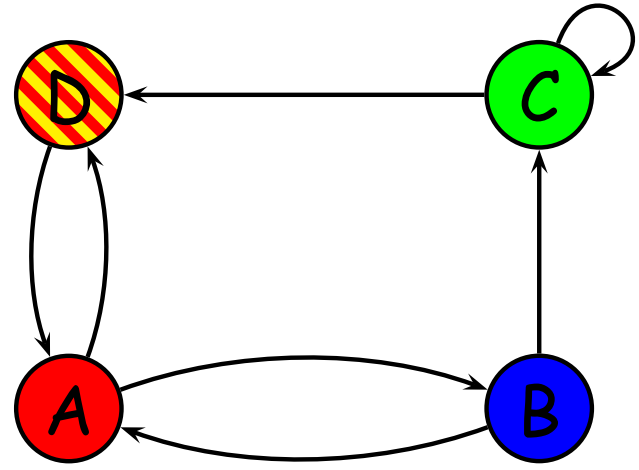
NuSMV: Open-source (LGPL) implementation of SMV
Developped at ITC-IRST (Trento, Italy)
by Alessandro Cimatti et al.

What is NuSMV?

(Nu)SMV = symbolic CTL model-checker.

models defined "symbolically", and encoded using BDD:

```
MODULE automate
  VAR
    red: boolean;
    blue: boolean;
    green: boolean;
    yellow: boolean;
  INIT
    red & !blue & !green & !yellow
  TRANS
    case
       red & !blue & !green & !yellow:
          next(!red &  blue & !green & !yellow) |
          next( red & !blue & !green &  yellow)
      !red &  blue & !green & !yellow:
          next( red & !blue & !green & !yellow) |
          next(!red & !blue &  green & !yellow)
      !red & !blue &  green & !yellow:
          next( red & !blue & !green &  yellow) |
          next(!red & !blue &  green & !yellow)
       red & !blue & !green &  yellow:
          next( red & !blue & !green & !yellow)
```



What is NuSMV?

(Nu)SMV = symbolic CTL model-checker.

$CTL \ni \varphi ::= p \mid \varphi \vee \varphi \mid \neg\varphi \mid EX\,\varphi \mid AX\,\varphi \mid E[\varphi U \varphi] \mid A[\varphi U \varphi]$

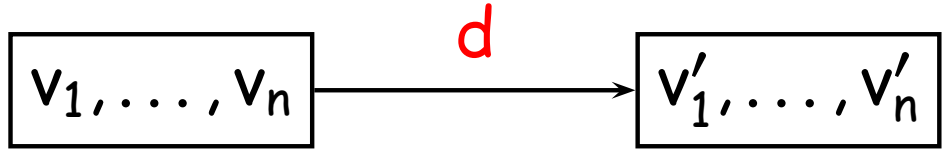
NuSMV also handles TCTL, assuming that one transition takes one time unit.

$TCTL \ni \psi ::= \psi \in CTL \mid E[\psi U_{[c,d]} \psi] \mid A[\psi U_{[c,d]} \psi]$

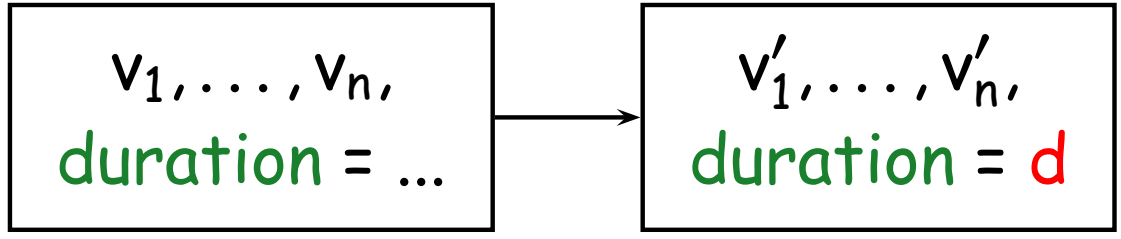
Coding TKS in NuSMV

We add one special integer variable, named duration.

We encode



as



Bridge crossing at night

- 4 persons are to cross the bridge.
- At most two at a time on the bridge.
- They have one lamp, needed when crossing.
- It takes 5, 10, 20 and 25 seconds for each of them to cross.

What is the optimal combination?

Bridge crossing at night

```
MODULE hiker(time,lamp,duration)

VAR
  state: boolean;                -- true = left of the bridge, false = right
  changed: boolean;              -- true if the person just crossed the bridge

INIT
  !state & !changed

TRANS
  ((!state & !lamp) -> ((next(state) & next(duration)>=time & next(changed)) |
                        (next(!state) & next(!changed))))
&
  ((state & lamp) -> ((next(!state) & next(duration)>=time & next(changed)) |
                       (next(state) & next(!changed))))
&
  ((!state & lamp) -> (next(!state) & next(!changed)))
&
  ((state & !lamp) -> (next(state) & next(!changed)))
```

Bridge crossing at night

```
MODULE main

VAR
  v1: hiker(5,lamp,duration);
  v2: hiker(10,lamp,duration);
  v3: hiker(10,lamp,duration);
  v4: hiker(25,lamp,duration);
  lamp: boolean;                -- true = left of the bridge, false = right
  duration: 0..25;

DEFINE
  bridge :=  v1.changed +  v2.changed +  v3.changed +  v4.changed ;
  initial:= (!v1.state) & (!v2.state) & (!v3.state) & (!v4.state);
  final  := ( v1.state) & ( v2.state) & ( v3.state) & ( v4.state);

INIT
  lamp & duration=0

TRANS
  (next(bridge=0)) <-> (next(lamp)=lamp)

COMPUTE MIN[ initial, final ]

SPEC !EF<=60 final
```

Bridge crossing at night

This example can easily be translated into NuSMV, but...

Bridge crossing at night

This example can easily be translated into NuSMV, but...

	NuSMV		TSMV	
	time	memory	time	memory
bridge	0.13 sec.	9 112 KB	0.02 sec.	1 280 KB
bridge × 10	4.14 sec.	18 860 KB	0.04 sec.	1 280 KB
bridge × 20	22.44 sec.	19 508 KB	0.07 sec.	1 280 KB
bridge × 50	176.55 sec.	28 640 KB	0.22 sec.	9 920 KB
bridge × 100	844.25 sec.	50 964 KB	0.45 sec.	11 580 KB
bridge × 200	4889.93 sec.	98 212 KB	1.03 sec.	14 956 KB

TSMV is (more or less) scale-insensitive.

Encouraging results

- relatively easy implementation into NuSMV,
- efficient model checking of timed Kripke structures,
- scale-insensitivity, memory and time consumption equivalent to those of NuSMV on classical Kripke Structure (with duration 1).

Future work

- more examples (NuSMV classical examples, UppAal examples, scheduling examples, ...)
- clean up the code, find better algorithms,
- counter-example generation.