Decision Diagrams to Encode and Manipulate Large Structured Data

Slides made by **Marco Beccuti**

Università degli Studi di Torino

Dipartimento di Informatica

May 2020

Outline

CTL Model Checking

Model Checking CTL

Model checking CTL

It is a formal verification technique such that:

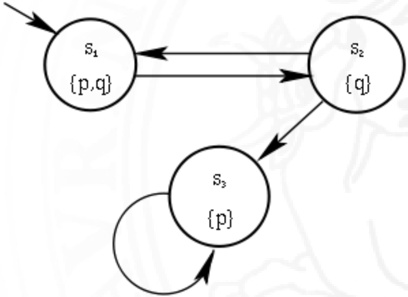
- the **system** is represented as a **Kripke Model** $\mathcal{K}$;
- a **property** is expressed as **Computation Tree Logic (CTL) formula** Φ.

It checks whether the set of state of Kripke model satisfy a CTL formula.

Model checking CTL

A **Kripke model** $\mathcal{K} = (S, \mathcal{N}, L)$ is defined as follows:

- S is a finite set of state;
- $\mathcal{N}$ is a transition relation $S \times 2^S$;
- L is a labeling function $S \times 2^{AP}$, where AP is a set of atomic propositions.



$\mathcal{K}$ can be seen as a tree of executions.

Model checking CTL

CTL formula Φ can be state formula or path formula.

State formula:

- is an atomic proposition, true or false in each state;
- if p and p' are state formulas, then $\neg p$, $p \wedge p'$ and $p \vee p'$ are state formulas;
- if q is a path formula, Eq and Aq are state formulas.

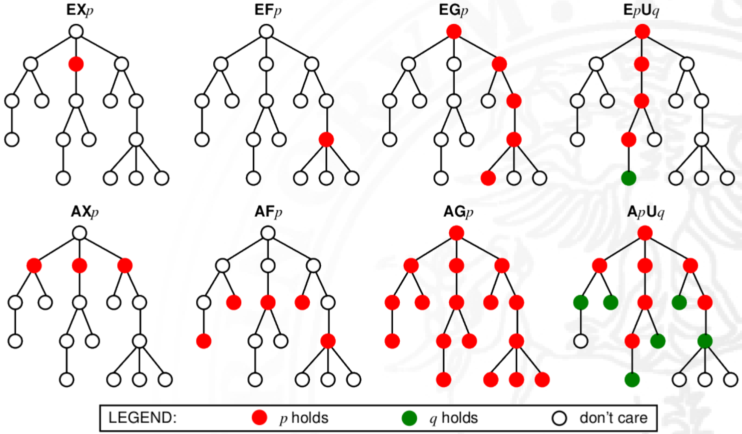
Path formula:

- if p and p' are state formulas, Xp, Fp, Gp, pUp' are path formulas

In CTL:

- a path quantifier, E (*i.e. possibly*) or A (*i.e. inevitably*), must always immediately precede a temporal operator X (*i.e. next*), F (*i.e. finally*), G (*i.e. globally*) and U (*i.e. until*);
- CTL expressions can be nested: $p \vee E\neg pU(\neg p \wedge AXp)$

CTL formula semantics



EX, EU, and EG form a complete set of CTL operators, since:

$\bullet AXp = \neg EX \neg p$ $\bullet$ $AFp = \neg EG \neg p$ $\bullet$ $EFp = E\,true\,Up$

$\bullet AGp = \neg EF \neg p$ $\bullet$ $ApUq = \neg(E \neg q U(\neg p \wedge \neg q)) \wedge \neg EG \neg q$

Examples of CTL statements

- Mutual exclusion:

$$AG\ (\neg(crit_1 \wedge crit_2))$$

- For every computation, it is always possible to return to the initial state:

$$AG\ EF\ initial$$

- Every request will eventually be granted:

$$AG\ (request \Rightarrow AF\ response)$$

- Each process has access to the critical section infinitely often:

$$AG\ AF\ crit_1 \wedge AG\ AF\ crit_2$$

- If a process asks access to the critical region, it eventually obtains it:

$$AG\ request\_critical \Rightarrow AF\ access\_critical$$

CTL model checking algorithm: general idea

The algorithm can be synthesized in **two** macro-steps:

1. Construct the set of states where the formula holds:

$$S_\Phi = \{s \in S : \mathcal{K}, s \models \Phi\}$$

2. compare S_Φ with the set of initial states:

$$S_\Phi \cap S_0 \neq \emptyset$$

CTL model checking algorithm

Since *EX*, *EU*, and *EG* form a complete set of CTL operators then only the following algorithm are sufficient:

- explicit/symbolic *EX* algorithm;
- explicit/symbolic *EU* algorithm;
- explicit/symbolic *EG* algorithm.

EX algorithm for CTL (explicit version)

We assume that all states satisfying p are inserted in S_p and function $\mathcal{N}^{-1}(s_i)$ returns all the states reaching s_i

1: **procedure** $\textsc{ComputeEX}(S_p, S_\Phi)$

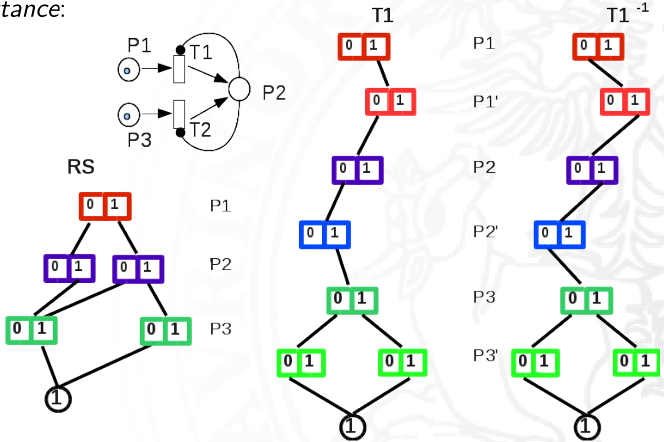
S_p = set of all the states satisfying p
S_Φ = set of the state which satisfies $EX\ p$

2: **for all** $(s \in S_p)$ **do**
3: $S_\Phi.insert(\mathcal{N}^{-1}(s))$;
4: **end for**
5: **end procedure**

EX algorithm for CTL (symbolic version)

All sets of states and relations over sets of states are encoded using MDDs.

For instance:



EX algorithm for CTL (symbolic version)

All sets of states and relations over sets of states are encoded using MDDs. We assume that MDD_{S_p} encoding the set S_p of states satisfying p and $MDD_{\mathcal{N}^{-1}}$ encoding the backward transition relation have been already built.

1: **procedure** $\textsc{ComputeEX}(MDD_{S_p}, MDD_{\mathcal{N}^{-1}}, MDD_{S_\Phi})$

MDD_{S_p} = MDD encoding the set of all the states satisfying p
$MDD_{\mathcal{N}^{-1}}$ = MDD encoding the backwards transition relation
MDD_{S_Φ} = MDD encoding the set of all the states satisfying Φ

2: $MDD_{S_\Phi} = RelationalProduct(MDD_{S_p}, MDD_{\mathcal{N}^{-1}})$;
3: **end procedure**

EU algorithm for CTL (explicit version)

We assume that all states satisfying p are inserted in S_p and those satisfying q are inserted in S_q. Function $\mathcal{N}^{-1}(s_i)$ returns all the states reaching s_i.

procedure COMPUTEEU(S_p, S_q, S_Φ)

S_p = set of all the states satisfying p
S_q = set of all the states satisfying q
S_Φ = set of the state which satisfies $E\ p\ U\ q$

    $S_\Phi = S_q$;
    repeat
        $S_{Curr}.copy(S_\Phi)$;
        for all ($s \in S_{Curr}$) **do**
            $S_{prev} = \mathcal{N}^{-1}(s)$;
            for all ($s' \in S_{prev}$) **do**
                if ($s' \in S_p$) **then**
                    $S_\Phi.insert(s')$;
                end if
            end for
        end for
    until ($S_\Phi \neq S_{Curr}$)
end procedure

EU algorithm for CTL (symbolic version)

All sets of states and relations over sets of states are encoded using MDDs. We assume that MDD_{S_p} encodes the set S_p of states satisfying p, MDD_{S_q} encodes the set S_q of states satisfying q and $MDD_{\mathcal{N}^{-1}}$ encodes the backward transition relation.

procedure $\textsc{ComputeEU}(MDD_{S_p}, MDD_{S_q}, MDD_{\mathcal{N}^{-1}}, MDD_{S_\Phi})$

MDD_{S_p} = MDD encoding the set of all the states satisfying p
MDD_{S_q} = MDD encoding the set of all the states satisfying q
$MDD_{\mathcal{N}^{-1}}$ = MDD encoding the backward transition relation
MDD_{S_Φ} = MDD encoding the set of all the states satisfying Φ

 $MDD_{S_\Phi} = MDD_{S_q}$;
 repeat
  $MDD_{Curr} = MDD_{S_\Phi}$;
  $MDD_{Prev} = RelationalProduct(MDD_{S_\Phi}, MDD_{\mathcal{N}^{-1}})$;
  $MDD_{Prev} = Intersection(MDD_{Prev}, MDD_{S_p})$;
  $MDD_{S_\Phi} = Union(MDD_{Prev}, MDD_{S_\Phi})$;
 until $(MDD_{Curr} \neq MDD_{S_\Phi})$
end procedure

EG algorithm for CTL (explicit version)

We assume that all states satisfying p are inserted in S_p. Function $\mathcal{N}^{-1}(s_i)$ returns all the states reaching s_i. The algorithm relies on finding the **strongly connected components (SCCs)** of a graph.

procedure $\textsc{ComputeEG}(S_p, S_\Phi)$

S_p = set of all the states satisfying p
S_Φ = set of the state which satisfies EGp

    $S_\Phi = ComputeSSC(S_p)$;
    repeat
        $S_{Curr}.copy(S_\Phi)$;
        for all $(s \in S_{Curr})$ **do**
            $S_{prev} = \mathcal{N}^{-1}(s)$;
            for all $(s' \in S_{prev})$ **do**
                if $(s' \in S_p)$ **then**
                    $S_\Phi.insert(s')$;
                end if
            end for
        end for
    until $(S_\Phi \neq S_{Curr})$
end procedure

EG algorithm for CTL (symbolic version)

All sets of states and relations over sets of states are encoded using MDDs. We assume that MDD_{S_p} encodes the set S_p of states satisfying p and $MDD_{\mathcal{N}^{-1}}$ encodes the backward transition relation.

procedure $\textsc{ComputeEG}(MDD_{S_p}, MDD_{\mathcal{N}^{-1}}, MDD_{S_\Phi})$

MDD_{S_p} = MDD encoding the set of all the states satisfying p
$MDD_{\mathcal{N}^{-1}}$ = MDD encoding the backwards transition relation
MDD_{S_Φ} = MDD encoding the set of all the states satisfying Φ

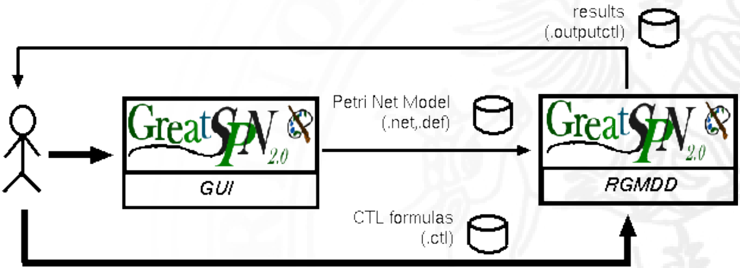
    $MDD_{S_\Phi} = MDD_{S_p}$;
    repeat
        $MDD_{Curr} = MDD_{S_\Phi}$;
        $MDD_{Prev} = RelationalProduct(MDD_{S_\Phi}, MDD_{\mathcal{N}^{-1}})$;
        $MDD_{S_\Phi} = Intersection(MDD_{Prev}, MDD_{S_\Phi})$;
    until $(MDD_{Curr} \neq MDD_{S_\Phi})$
end procedure

Symbolic model checking and GreatSPN

Symbolic model checking and GreatSPN

Symbolic model checking and GreatSPN

The symbolic algorithms have been implemented in GreatSPN using Meddly library (http://meddly.svn.sourceforge.net/)



Symbolic model checking and GretSPN

CTL formula grammar

⟨*CTLformula*⟩	::=	⟨*atomicProposition*⟩ | ⟨*CTLformula*⟩ |
		⟨*CTLformula*⟩ “and” ⟨*CTLformula*⟩ |
		⟨*CTLformula*⟩ “or” ⟨*CTLformula*⟩ |
		“not” ⟨*CTLformula*⟩ | ⟨*CTLformula*⟩ “->” ⟨*CTLformula*⟩ |
		“E” “X” ⟨*CTLformula*⟩ | “E” “G” ⟨*CTLformula*⟩ |
		“E” “[” ⟨*CTLformula*⟩ “U” ⟨*CTLformula*⟩ “]” |
		“A” “X” ⟨*CTLformula*⟩ | “A” “F” ⟨*CTLformula*⟩ |
		“E” “F” ⟨*CTLformula*⟩ | “A” “G” ⟨*CTLformula*⟩ |
		“A” “[” ⟨*CTLformula*⟩ “U” ⟨*CTLformula*⟩ “]”
⟨*atomicProposition*⟩	::=	⟨*inequality*⟩ | ⟨*boolvalue*⟩ | “ndeadlock” | “deadlock” | “en ⟨*var*⟩”
⟨*boolvalue*⟩	::=	“true” | “false”
⟨*inequality*⟩	::=	“(” ⟨*expression*⟩ ⟨*comp_oper*⟩ ⟨*expression*⟩ “)”
⟨*comp_oper*⟩	::=	“<” | “>” | “<=” | “>=” | “=” | “! =”
⟨*expression*⟩	::=	“(” ⟨*expression*⟩ ⟨*arit_oper*⟩ ⟨*expression*⟩ “)” | ⟨*term*⟩ |
		“(” ⟨*number_expr*⟩ “)”
⟨*arit_oper*⟩	::=	“+” | “−” | “∗” | “/”
⟨*term*⟩	::=	⟨*number_expr*⟩ “∗” ⟨*var*⟩ | ⟨*number_expr*⟩ “/” ⟨*var*⟩ | ⟨*var*⟩
⟨*number_expr*⟩	::=	“(” ⟨*number_expr*⟩ ⟨*arit_oper*⟩ ⟨*number_expr*⟩ “)” | ⟨*number*⟩
⟨*var*⟩	::=	[(A-Z)(a-z)][(A-Z)(a-z)(0-9)]*
⟨*numbr*⟩	::=	$\mathbb{R}^+$

Observe that tag ⟨*var*⟩ corresponds to a name of a transition or a place in the input model.

Deadlock (symbolic version)

All sets of states and relations over sets of states are encoded using MDDs. We assume that MDD_{RS} encodes the Rechability Set (RS) and $MDD_{\mathcal{N}^{-1}}$ encodes the backward transition relation.

1: **procedure** COMPUTEDEADLOCK(MDD_{RS}, $MDD_{\mathcal{N}^{-1}}$)

MDD_{RS} = MDD encoding RS
$MDD_{\mathcal{N}-1}$ = MDD encoding the backwards transition relation
$MDD_{S_{\Phi}}$ = MDD encoding the set of all the states satisfying Φ

2: MDD_{Prev}=$RelationalProduct(MDD_{RS}, MDD_{\mathcal{N}-1})$;
3: $MDD_{S_{\Phi}}$=$Difference(MDD_{RS}, MDD_{Prev})$;
4: **end procedure**

Enabled Transition (symbolic version)

All sets of states and relations over sets of states are encoded using MDDs. We assume that MDD_{RS} encodes the Rechability Set (RS) and $MDD^{t}_{\mathcal{N}^{-1}}$ encodes the backward transition relation for transition t. Currently, It works only for PN.

1: **procedure** $\textsc{ComputeEnableT}(MDD_{RS}, t)$

MDD_{RS} = MDD encoding RS
$MDD^{t}_{\mathcal{N}^{-1}}$ encodes the backward transition relation for transition t
MDD_{S_Φ} = MDD encoding the set of all the states satisfying Φ

2: $MDD_{S_\Phi} = RelationalProduct(MDD_{RS}, MDD^{t}_{\mathcal{N}^{-1}})$;
3: **end procedure**

EF algorithm for CTL (symbolic version)

In **GreatSPN**, to improve the efficiency, the *EF* algorithm is implemented directed instead of using $EFp = E\,true\,Up$.
All sets of states and relations over sets of states are encoded using MDDs. We assume that MDD_{S_p} encodes the set S_p of states satisfying p and $MDD_{\mathcal{N}^{-1}}$ encodes the backward transition relation.

procedure ComputeEF($MDD_{S_p}, MDD_{\mathcal{N}^{-1}}, MDD_{S_\Phi}$)

MDD_{S_p} = MDD encoding the set of all the states satisfying p
$MDD_{\mathcal{N}-1}$ = MDD encoding the backwards transition relation
MDD_{S_Φ} = MDD encoding the set of all the states satisfying Φ

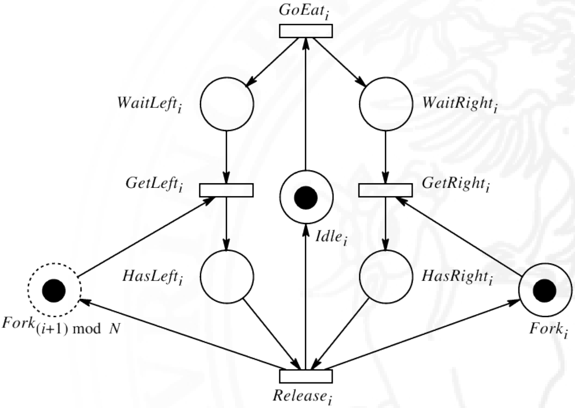
```
    MDD_{S_Φ} = MDD_{S_p};
    repeat
        MDD_{Curr} = MDD_{S_Φ};
        MDD_{Prev} = RelationalProduct(MDD_{S_Φ}, MDD_{N-1});
        MDD_{S_Φ} = Union(MDD_{Prev}, MDD_{S_Φ});
    until (MDD_{Curr} ≠ MDD_{S_Φ})
end procedure
```

Some experimental results using symbolic approach

Some experimental results using symbolic approach

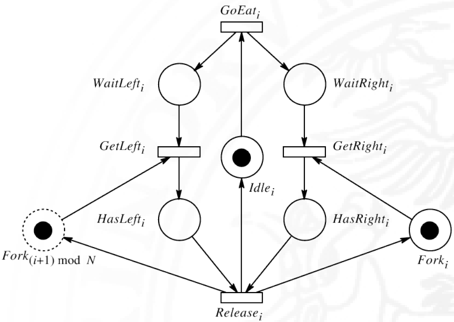
Experiments on Dining Philosophers

A single dining philosopher



Experiments on Dining Philosophers

A single dining philosopher

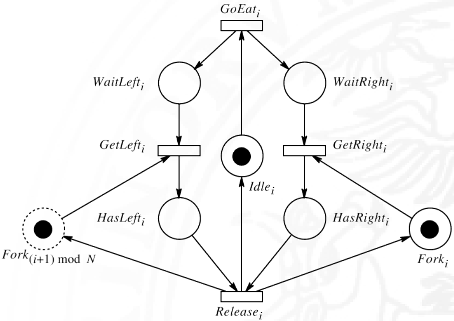


	RS Generation					CTL checking		
|Phi.|	|RS|	T.	Peak MDD	Peak MDD2L	Max. Mem.	T.	Peak DD	Max. Mem
CTL formula:E [(HasLeft1 = 0 or HasRight1 = 0) U (HasLeft2 = 1 and HasRight2 = 1)]								
100	$1.37e^{62}$	0.12s.	119KB.	240KB.	12MB.(9MB.)	1.80s.	250MB.	292MB.(9MB.)
200	$6.82e^{124}$	0.5s.	241KB.	477KB.	21MB.(17MB.)	7.66s.	250MB.	338MB.(17MB.)
316	$3.64e^{197}$	0.02s.	382KB.	934KB.	33MB.(27MB.)	18.31s.	258MB.	652MB.(27MB.)
330	$2.18e^{206}$	0.01s.	399KB.	945KB.	35MB.(28MB.)	19.95s.	281MB.	729MB.(28MB.)

Performed on INTEL CORE i7 with 8Gb of RAM

Experiments on Dining Philosophers

A single dining philosopher

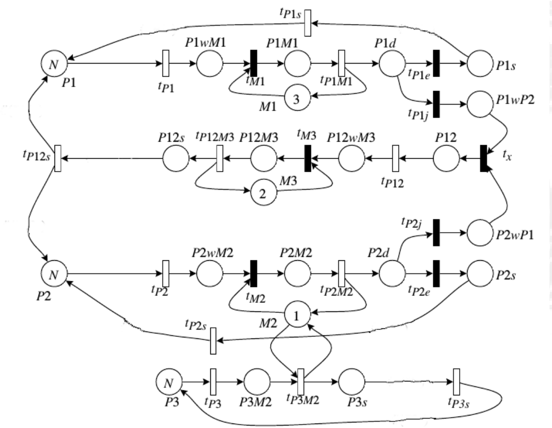


|Phi.|	RS Generation					CTL checking		
	|RS|	T.	Peak MDD	Peak MTMDD	Max. Mem.	T.	Peak DD	Max. Mem
CTL formula: E G (HasLeft1 = 0 or HasRight1 = 0)								
100	$1.37e^{62}$	0.12s.	119KB.	240KB.	12MB.(9MB.)	1.80s.	862KB.	13MB.(9MB.)
200	$6.82e^{124}$	0.5s.	241KB.	477KB.	21MB.(17MB.)	7.66s.	1.7MB.	25MB.(17MB.)
316	$3.64e^{197}$	0.02s.	382KB.	934KB.	33MB.(27MB.)	0.7s.	2.9MB.	39MB.(27MB.)
330	$2.18e^{206}$	0.01s.	399KB.	945KB.	35MB.(28MB.)	0.6s.	3MB.	40MB.(28MB.)

Performed on INTEL CORE i7 with 8Gb of RAM

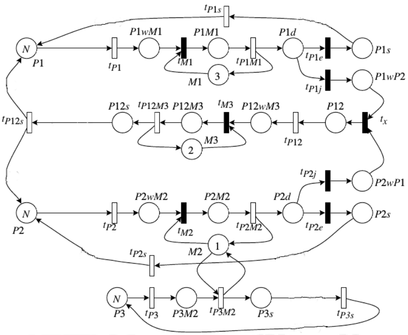
Experiments on Flexible Manufacturing System

FMS



Experiments on Flexible Manufacturing System

FMS

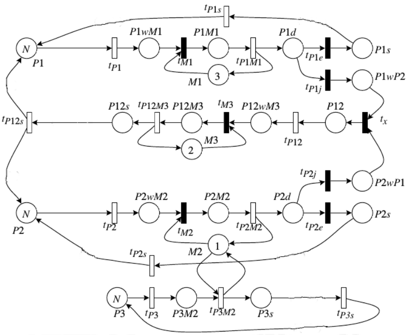


|N|	RS Generation |RS|	T.	Peak MDD	Peak MDD2L	Max. Mem.	CTL checking T.	Peak DD	Max. Mem
CTL formula: E G ¬ (P1s = N and P2s = N and P3s = N))								
30	$2.36e^{12}$	23s.	194MB.	1.2MB.	469MB.(24MB.)	45s.	195MB.	758MB.(24MB.)
40	$3.58e^{13}$	2m.	481MB.	1.9MB.	891MB.(24MB.)	4m.	482MB.	889MB.(24MB.)
42	$5.70e^{13}$	8m.	525MB.	2.2MB.	939MB.(24MB.)	15m.	527MB.	939MB.(24MB.)
45	$1.10e^{14}$	28m.	587MB.	2.5MB.	1.0GB.(24MB.)	34m.	589MB.	1.0GB.(24MB.)

Performed on INTEL CORE i7 with 8Gb of RAM

Experiments on Flexible Manufacturing System

FMS



|N|	|RS|	T.	Peak MDD	Peak MDD2L	Max. Mem.	T.	Peak DD	Max. Mem
	RS Generation					**CTL checking**		
				CTL formula: E F (deadlock)				
30	$2.36e^{12}$	23s.	194MB.	1.2MB.	469MB.(24MB.)	23s.	195MB.	475MB.(24MB.)
40	$3.58e^{13}$	2m.	481MB.	1.9MB.	891MB.(24MB.)	2m.	483MB.	909MB.(24MB.)
42	$5.70e^{13}$	8m.	525MB.	2.2MB.	939MB.(24MB.)	8m.	527MB.	940MB.(24MB.)
45	$1.10e^{14}$	28m.	587MB.	2.5MB.	1.0GB.(24MB.)	28m.	589MB.	1.0GB.(24MB.)

Performed on INTEL CORE i7 with 8Gb of RAM