

# FMSoft

## Lecture 7 — Bounded Model Checking

(lecture version)

---

Tadeusz Litak

November 27, 2018

Informatik 8, FAU Erlangen-Nürnberg

- We have already heard about model checking

- We have already heard about model checking
- Now we focus on (theoretical foundations of) **symbolic** model checking, in particular **bounded** model checking

- We have already heard about model checking
- Now we focus on (theoretical foundations of) **symbolic** model checking, in particular **bounded** model checking
- Why do we care?

- We have already heard about model checking
- Now we focus on (theoretical foundations of) **symbolic** model checking, in particular **bounded** model checking
- Why do we care?
- As it turns out, we find these techniques at the heart of both NuSMV/nuXmv and even more so, the **SCADE suite** <https://de.wikipedia.org/wiki/SCADE> which Christoph may or may not demonstrate. And our Chair is using it elsewhere: we even had a Praktikum for that

- We have already heard about model checking
- Now we focus on (theoretical foundations of) **symbolic** model checking, in particular **bounded** model checking
- Why do we care?
- As it turns out, we find these techniques at the heart of both NuSMV/nuXmv and even more so, the **SCADE suite**  
<https://de.wikipedia.org/wiki/SCADE>  
which Christoph may or may not demonstrate. And our Chair is using it elsewhere: we even had a Praktikum for that
- But what is it all about?

- Model checking with explicit states and transition relations (represented, say, as **adjacency matrix**) has its limits: it runs into **state explosion problem**

- Model checking with explicit states and transition relations (represented, say, as **adjacency matrix**) has its limits: it runs into **state explosion problem**
- We've seen how complex it can be



- Model checking with explicit states and transition relations (represented, say, as **adjacency matrix**) has its limits: it runs into **state explosion problem**
- We've seen how complex it can be
- A 1987 breakthrough: Kenneth McMillan, while doing his PhD at CMU, comes up with the idea of using **symbolic representations** of state transition diagrams in terms of (canonical forms for) **boolean formulas!**

- Model checking with explicit states and transition relations (represented, say, as **adjacency matrix**) has its limits: it runs into **state explosion problem**
- We've seen how complex it can be
- A 1987 breakthrough: Kenneth McMillan, while doing his PhD at CMU, comes up with the idea of using **symbolic representations** of state transition diagrams in terms of (canonical forms for) **boolean formulas!**
- Ordinary propositional logic instead of temporal formalisms

- Model checking with explicit states and transition relations (represented, say, as **adjacency matrix**) has its limits: it runs into **state explosion problem**
- We've seen how complex it can be
- A 1987 breakthrough: Kenneth McMillan, while doing his PhD at CMU, comes up with the idea of using **symbolic representations** of state transition diagrams in terms of (canonical forms for) **boolean formulas!**
- Ordinary propositional logic instead of temporal formalisms
- His canonical forms are **Binary Decision Diagrams (BDDs)**, specifically **Ordered Binary Decision Diagrams (OBDDs)**

- Model checking with explicit states and transition relations (represented, say, as **adjacency matrix**) has its limits: it runs into **state explosion problem**
- We've seen how complex it can be
- A 1987 breakthrough: Kenneth McMillan, while doing his PhD at CMU, comes up with the idea of using **symbolic representations** of state transition diagrams in terms of (canonical forms for) **boolean formulas!**
- Ordinary propositional logic instead of temporal formalisms
- His canonical forms are **Binary Decision Diagrams (BDDs)**, specifically **Ordered Binary Decision Diagrams (OBDDs)**
- His first symbolic model checker based on this technique was SMV—which was followed by Cadence SMV of Cadence Berkeley Labs and NuSMV of IRST in Trento

- In last 20 years or so, OBDDs are part and parcel of standard expositions of model checking techniques  
See, e.g., Chapter 6 of the Huth and Ryan book if you're interested

- In last 20 years or so, OBDDs are part and parcel of standard expositions of model checking techniques  
See, e.g., Chapter 6 of the Huth and Ryan book if you're interested
- But even with these, there is a limit to the number of state variables they can efficiently handle. Still suffer from a form of potential state explosion

- In last 20 years or so, OBDDs are part and parcel of standard expositions of model checking techniques  
See, e.g., Chapter 6 of the Huth and Ryan book if you're interested
- But even with these, there is a limit to the number of state variables they can efficiently handle. Still suffer from a form of potential state explosion
- Bottleneck: finding the right **ordering of state variables**

- In last 20 years or so, OBDDs are part and parcel of standard expositions of model checking techniques  
See, e.g., Chapter 6 of the Huth and Ryan book if you're interested
- But even with these, there is a limit to the number of state variables they can efficiently handle. Still suffer from a form of potential state explosion
- Bottleneck: finding the right **ordering of state variables**
- **Bounded model checking** (Biere/Cimatti/Clarke/Zhu, ETAPS 1999: *Symbolic Model Checking without BDDs*)



- uses ordinary boolean formulas

- uses ordinary boolean formulas
- relies on powerful SAT solving techniques

- uses ordinary boolean formulas
- relies on powerful SAT solving techniques
- Quickly finds counterexamples of minimal size

- uses ordinary boolean formulas
- relies on powerful SAT solving techniques
- Quickly finds counterexamples of minimal size
- uses much less space than OBDDs

- uses ordinary boolean formulas
- relies on powerful SAT solving techniques
- Quickly finds counterexamples of minimal size
- uses much less space than OBDDs
- does not need manual ordering of variables

- Crucial insight allowing encoding of temporal formulas:

- Crucial insight allowing encoding of temporal formulas:
- there are **two main types** of **finite path prefixes**

- Crucial insight allowing encoding of temporal formulas:
- there are **two main types** of **finite path prefixes**
- Those that matter contain a **back loop**  
⇒ can represent an infinite path, thus being a potential witness of satisfiability for a temporal formula



- Crucial insight allowing encoding of temporal formulas:
- there are **two main types** of **finite path prefixes**
- Those that matter contain a **back loop**  
     $\implies$  can represent an infinite path, thus being a potential witness of satisfiability for a temporal formula
- Prefixes without a loop are not representing infinite paths

- Crucial insight allowing encoding of temporal formulas:
- there are **two main types** of **finite path prefixes**
- Those that matter contain a **back loop**  
     $\implies$  can represent an infinite path, thus being a potential witness of satisfiability for a temporal formula
- Prefixes without a loop are not representing infinite paths
- Thus, **bounded semantics** splits into two cases:

- Crucial insight allowing encoding of temporal formulas:
- there are **two main types** of **finite path prefixes**
- Those that matter contain a **back loop**  
⇒ can represent an infinite path, thus being a potential witness of satisfiability for a temporal formula
- Prefixes without a loop are not representing infinite paths
- Thus, **bounded semantics** splits into two cases:
- ...**with a loop**: the earlier state to which there is a back loop is the successor of the last one in the prefix

- Crucial insight allowing encoding of temporal formulas:
- there are **two main types** of **finite path prefixes**
- Those that matter contain a **back loop**  
⇒ can represent an infinite path, thus being a potential witness of satisfiability for a temporal formula
- Prefixes without a loop are not representing infinite paths
- Thus, **bounded semantics** splits into two cases:
- ...**with a loop**: the earlier state to which there is a back loop is the successor of the last one in the prefix
- ...**without a loop**: no formula prefixed with **G** is true and neither is a formula prefixed with **X** at the end of the prefix

- Crucial insight allowing encoding of temporal formulas:
- there are **two main types** of **finite path prefixes**
- Those that matter contain a **back loop**  
 $\implies$  can represent an infinite path, thus being a potential witness of satisfiability for a temporal formula
- Prefixes without a loop are not representing infinite paths
- Thus, **bounded semantics** splits into two cases:
- ...**with a loop**: the earlier state to which there is a back loop is the successor of the last one in the prefix
- ...**without a loop**: no formula prefixed with **G** is true and neither is a formula prefixed with **X** at the end of the prefix
- In order to ensure everything works, we need to restrict attention to formulas in **negation normal form**

## Recall of usual LTL semantics

- note: it's NNF, so we need more primitives
- $\mathcal{M}, \pi \models \mathbf{p}$  if  $\mathbf{p} \in L(\pi(0))$
- $\mathcal{M}, \pi \models \neg \mathbf{p}$  if  $\mathbf{p} \notin L(\pi(0))$
- $\mathcal{M}, \pi \models \phi \wedge \psi$  if  $\mathcal{M}, \pi \models \phi$  and  $\mathcal{M}, \pi \models \psi$
- $\mathcal{M}, \pi \models \phi \vee \psi$  if  $\mathcal{M}, \pi \models \phi$  or  $\mathcal{M}, \pi \models \psi$

## Recall of usual LTL semantics

- note: it's NNF, so we need more primitives
- $\mathcal{M}, \pi \models \mathbf{p}$  if  $\mathbf{p} \in L(\pi(0))$
- $\mathcal{M}, \pi \models \neg \mathbf{p}$  if  $\mathbf{p} \notin L(\pi(0))$
- $\mathcal{M}, \pi \models \phi \wedge \psi$  if  $\mathcal{M}, \pi \models \phi$  and  $\mathcal{M}, \pi \models \psi$
- $\mathcal{M}, \pi \models \phi \vee \psi$  if  $\mathcal{M}, \pi \models \phi$  or  $\mathcal{M}, \pi \models \psi$
- $\mathcal{M}, \pi \models \mathbf{X}\phi$  if  $\mathcal{M}, \pi_1 \models \phi$
- $\mathcal{M}, \pi \models \mathbf{F}\phi$  if  $\exists n \in \mathbb{N}. \mathcal{M}, \pi_n \models \phi$
- $\mathcal{M}, \pi \models \mathbf{G}\phi$  if  $\forall n \in \mathbb{N}. \mathcal{M}, \pi_n \models \phi$
- $\mathcal{M}, \pi \models \phi \mathbf{U} \psi$  if  $\exists n \in \mathbb{N}. \mathcal{M}, \pi_n \models \psi$  and  $\forall i < n. \mathcal{M}, \pi_i \models \phi$
- $\mathcal{M}, \pi \models \phi \mathbf{R} \psi$  if  
either  $\exists n \in \mathbb{N}. \mathcal{M}, \pi_n \models \phi$  and  $\forall i \leq n. \mathcal{M}, \pi_i \models \psi$   
or  $\forall n \in \mathbb{N}. \mathcal{M}, \pi_n \models \psi$

## Bounded semantics

- A path is a *k-loop* if it has no more than  $k$  distinct states



## Bounded semantics

- A path is a *k-loop* if it has no more than  $k$  distinct states
- This means that for some  $l \leq k$ , it enters an infinitely repeating cycle looping back to  $l$

Given a specific  $l$ , we can be more precise and call it *(k, l)-path*

## Bounded semantics

- A path is a *k-loop* if it has no more than  $k$  distinct states
- This means that for some  $l \leq k$ , it enters an infinitely repeating cycle looping back to  $l$   
Given a specific  $l$ , we can be more precise and call it *(k, l)-path*
- We define now *k-semantics* for formulas, notation  $\pi \models_k \phi$ :

## Bounded semantics

- A path is a  **$k$ -loop** if it has no more than  $k$  distinct states
- This means that for some  $l \leq k$ , it enters an infinitely repeating cycle looping back to  $l$

Given a specific  $l$ , we can be more precise and call it  **$(k, l)$ -path**

- We define now  **$k$ -semantics** for formulas, notation  $\pi \models_k \phi$ :
  - if  $\pi$  is a  $k$ -loop, then  $\pi \models_k \phi$  if  $\pi \models \phi$

# Bounded semantics

- A path is a  **$k$ -loop** if it has no more than  $k$  distinct states
- This means that for some  $l \leq k$ , it enters an infinitely repeating cycle looping back to  $l$

Given a specific  $l$ , we can be more precise and call it  **$(k, l)$ -path**

- We define now  **$k$ -semantics** for formulas, notation  $\pi \models_k \phi$ :
  - if  $\pi$  is a  $k$ -loop, then  $\pi \models_k \phi$  if  $\pi \models \phi$
  - otherwise,  $\pi \models_k \phi$  if  $\pi \models_k^0 \phi$

## Bounded semantics

- A path is a  **$k$ -loop** if it has no more than  $k$  distinct states
- This means that for some  $l \leq k$ , it enters an infinitely repeating cycle looping back to  $l$

Given a specific  $l$ , we can be more precise and call it  **$(k, l)$ -path**

- We define now  **$k$ -semantics** for formulas, notation  $\pi \models_k \phi$ :
  - if  $\pi$  is a  $k$ -loop, then  $\pi \models_k \phi$  if  $\pi \models \phi$
  - otherwise,  $\pi \models_k \phi$  if  $\pi \models_k^0 \phi$
- where  $\pi \models_k^i \phi$ , for  $i \leq k$ , is defined to capture the above:

*no formula prefixed with  $G$  is true and neither is a formula prefixed with  $X$  at the end of the prefix*

- $\mathcal{M}, \pi \models_k^i \mathbf{p}$  if  $\mathbf{p} \in L(\pi(i))$

- $\mathcal{M}, \pi \models_k^i \mathbf{p}$  if  $\mathbf{p} \in L(\pi(i))$
- $\mathcal{M}, \pi \models_k^i \neg \mathbf{p}$  if  $\mathbf{p} \notin L(\pi(i))$

- $\mathcal{M}, \pi \models_k^i \mathbf{p}$  if  $\mathbf{p} \in L(\pi(i))$
- $\mathcal{M}, \pi \models_k^i \neg \mathbf{p}$  if  $\mathbf{p} \notin L(\pi(i))$
- $\mathcal{M}, \pi \models_k^i \phi \wedge \psi$  if  $\mathcal{M}, \pi \models_k^i \phi$  and  $\mathcal{M}, \pi \models_k^i \psi$



- $\mathcal{M}, \pi \models_k^i \mathbf{p}$  if  $\mathbf{p} \in L(\pi(i))$
- $\mathcal{M}, \pi \models_k^i \neg \mathbf{p}$  if  $\mathbf{p} \notin L(\pi(i))$
- $\mathcal{M}, \pi \models_k^i \phi \wedge \psi$  if  $\mathcal{M}, \pi \models_k^i \phi$  and  $\mathcal{M}, \pi \models_k^i \psi$
- $\mathcal{M}, \pi \models_k^i \phi \vee \psi$  if  $\mathcal{M}, \pi \models_k^i \phi$  or  $\mathcal{M}, \pi \models_k^i \psi$

- $\mathcal{M}, \pi \models_k^i \mathbf{p}$  if  $\mathbf{p} \in L(\pi(i))$
- $\mathcal{M}, \pi \models_k^i \neg \mathbf{p}$  if  $\mathbf{p} \notin L(\pi(i))$
- $\mathcal{M}, \pi \models_k^i \phi \wedge \psi$  if  $\mathcal{M}, \pi \models_k^i \phi$  and  $\mathcal{M}, \pi \models_k^i \psi$
- $\mathcal{M}, \pi \models_k^i \phi \vee \psi$  if  $\mathcal{M}, \pi \models_k^i \phi$  or  $\mathcal{M}, \pi \models_k^i \psi$
- $\mathcal{M}, \pi \models_k^i X\phi$  if  $i < k$  and  $\mathcal{M}, \pi \models_k^{i+1} \phi$

- $\mathcal{M}, \pi \models_k^i \mathbf{p}$  if  $\mathbf{p} \in L(\pi(i))$
- $\mathcal{M}, \pi \models_k^i \neg \mathbf{p}$  if  $\mathbf{p} \notin L(\pi(i))$
- $\mathcal{M}, \pi \models_k^i \phi \wedge \psi$  if  $\mathcal{M}, \pi \models_k^i \phi$  and  $\mathcal{M}, \pi \models_k^i \psi$
- $\mathcal{M}, \pi \models_k^i \phi \vee \psi$  if  $\mathcal{M}, \pi \models_k^i \phi$  or  $\mathcal{M}, \pi \models_k^i \psi$
- $\mathcal{M}, \pi \models_k^i \mathbf{X}\phi$  if  $i < k$  and  $\mathcal{M}, \pi \models_k^{i+1} \phi$
- $\mathcal{M}, \pi \models_k^i \mathbf{F}\phi$  if  $\exists n. i \leq n \leq k$  and  $\mathcal{M}, \pi \models_k^n \phi$

- $\mathcal{M}, \pi \models_k^i \mathbf{p}$  if  $\mathbf{p} \in L(\pi(i))$
- $\mathcal{M}, \pi \models_k^i \neg \mathbf{p}$  if  $\mathbf{p} \notin L(\pi(i))$
- $\mathcal{M}, \pi \models_k^i \phi \wedge \psi$  if  $\mathcal{M}, \pi \models_k^i \phi$  and  $\mathcal{M}, \pi \models_k^i \psi$
- $\mathcal{M}, \pi \models_k^i \phi \vee \psi$  if  $\mathcal{M}, \pi \models_k^i \phi$  or  $\mathcal{M}, \pi \models_k^i \psi$
- $\mathcal{M}, \pi \models_k^i \mathbf{X}\phi$  if  $i < k$  and  $\mathcal{M}, \pi \models_k^{i+1} \phi$
- $\mathcal{M}, \pi \models_k^i \mathbf{F}\phi$  if  $\exists n. i \leq n \leq k$  and  $\mathcal{M}, \pi \models_k^n \phi$
- it is **never** the case that  $\mathcal{M}, \pi \models_k^i \mathbf{G}\phi$

- $\mathcal{M}, \pi \models_k^i \mathbf{p}$  if  $\mathbf{p} \in L(\pi(i))$
- $\mathcal{M}, \pi \models_k^i \neg \mathbf{p}$  if  $\mathbf{p} \notin L(\pi(i))$
- $\mathcal{M}, \pi \models_k^i \phi \wedge \psi$  if  $\mathcal{M}, \pi \models_k^i \phi$  and  $\mathcal{M}, \pi \models_k^i \psi$
- $\mathcal{M}, \pi \models_k^i \phi \vee \psi$  if  $\mathcal{M}, \pi \models_k^i \phi$  or  $\mathcal{M}, \pi \models_k^i \psi$
- $\mathcal{M}, \pi \models_k^i \mathbf{X}\phi$  if  $i < k$  and  $\mathcal{M}, \pi \models_k^{i+1} \phi$
- $\mathcal{M}, \pi \models_k^i \mathbf{F}\phi$  if  $\exists n. i \leq n \leq k$  and  $\mathcal{M}, \pi \models_k^n \phi$
- it is **never** the case that  $\mathcal{M}, \pi \models_k^i \mathbf{G}\phi$
- $\mathcal{M}, \pi \models_k^i \phi \mathbf{U} \psi$  if  $\exists n. i \leq n \leq k$  and  $\mathcal{M}, \pi \models_k^n \psi$  and  $\forall j$  s.t.  $i \leq j < n. \mathcal{M}, \pi \models_k^j \phi$

- $\mathcal{M}, \pi \models_k^i \mathbf{p}$  if  $\mathbf{p} \in L(\pi(i))$
- $\mathcal{M}, \pi \models_k^i \neg \mathbf{p}$  if  $\mathbf{p} \notin L(\pi(i))$
- $\mathcal{M}, \pi \models_k^i \phi \wedge \psi$  if  $\mathcal{M}, \pi \models_k^i \phi$  and  $\mathcal{M}, \pi \models_k^i \psi$
- $\mathcal{M}, \pi \models_k^i \phi \vee \psi$  if  $\mathcal{M}, \pi \models_k^i \phi$  or  $\mathcal{M}, \pi \models_k^i \psi$
- $\mathcal{M}, \pi \models_k^i \mathbf{X}\phi$  if  $i < k$  and  $\mathcal{M}, \pi \models_k^{i+1} \phi$
- $\mathcal{M}, \pi \models_k^i \mathbf{F}\phi$  if  $\exists n. i \leq n \leq k$  and  $\mathcal{M}, \pi \models_k^n \phi$
- it is **never** the case that  $\mathcal{M}, \pi \models_k^i \mathbf{G}\phi$
- $\mathcal{M}, \pi \models_k^i \phi \mathbf{U} \psi$  if  $\exists n. i \leq n \leq k$  and  $\mathcal{M}, \pi \models_k^n \psi$  and  $\forall j$  s.t.  $i \leq j < n. \mathcal{M}, \pi \models_k^j \phi$
- $\mathcal{M}, \pi \models_k^i \phi \mathbf{R} \psi$  if  $\exists n. i \leq n \leq k$  and  $\mathcal{M}, \pi \models_k^n \phi$  and  $\forall j$  s.t.  $i \leq j \leq n. \mathcal{M}, \pi \models_k^j \psi$

note the “or” clause is not there anymore: it’s like **G**

## **Theorem**

*For any  $\mathcal{M}$ ,  $k$ ,  $\pi$  and  $\phi$  (in NNF),  $\mathcal{M}, \pi \models_k \phi$  implies  $\mathcal{M}, \pi \models \phi$*

## Theorem

For any  $\mathcal{M}$ ,  $k$ ,  $\pi$  and  $\phi$  (in NNF),  $\mathcal{M}, \pi \models_k \phi$  implies  $\mathcal{M}, \pi \models \phi$

## Proof.

Inductively, works because of NNF:  $k$ -satisfaction clauses restrict normal ones and everything is monotone. □



**Theorem**

*For any  $\mathcal{M}$ ,  $k$ ,  $\pi$  and  $\phi$  (in NNF),  $\mathcal{M}, \pi \models_k \phi$  implies  $\mathcal{M}, \pi \models \phi$*

**Proof.**

Inductively, works because of NNF:  $k$ -satisfaction clauses restrict normal ones and everything is monotone. □

**Theorem**

*For any finite  $\mathcal{M}$ ,  $\pi$  and  $\phi$  (in NNF),  $\mathcal{M}, \pi \models \phi$  implies there exists  $k$  s.t.  $\mathcal{M}, \pi \models_k \phi$*

In order to find a concrete bound on this  $k$ , we'd need to resort to previously discussed model checking algorithms

The next step: using bounded semantics to move from LTL to propositional logic

**from LTL to propositional logic**

---

- The notion of  $k$ -satisfiability can be encoded via ordinary propositional calculus

- The notion of  $k$ -satisfiability can be encoded via **ordinary propositional calculus**
- Given a model  $\mathcal{M}$  with a distinguished state  $s^1$  and a formula  $\phi$ , we will construct a propositional formula  $[\mathcal{M}, \phi]_k$  s.t.

$$\mathcal{M}, s^1 \models_k^{\text{CTL}^*} E[\phi] \quad \text{iff} \quad [\mathcal{M}, \phi]_k \text{ is satisfiable}$$

We deliberately switch to indexing from 1: you'll see why

- The notion of  $k$ -satisfiability can be encoded via **ordinary propositional calculus**
- Given a model  $\mathcal{M}$  with a distinguished state  $s^1$  and a formula  $\phi$ , we will construct a propositional formula  $[\mathcal{M}, \phi]_k$  s.t.

$$\mathcal{M}, s^1 \models_k^{\text{CTL}^*} E[\phi] \quad \text{iff} \quad [\mathcal{M}, \phi]_k \text{ is satisfiable}$$

We deliberately switch to indexing from 1: you'll see why

- Combined with above theorems, this will yield

$$\mathcal{M}, s^1 \models^{\text{CTL}^*} E[\phi] \quad \text{iff} \quad [\mathcal{M}, \phi]_k \text{ is satisfiable for some } k$$

and still further

$$\mathcal{M}, s^1 \models^{\text{LTL}} (\neg\phi)^{\text{NNF}} \quad \text{iff} \quad [\mathcal{M}, \phi]_k \text{ is unsatisfiable for all } k$$

Recall:  $\mathcal{M}, s \models^{\text{LTL}} \phi$  iff  $\mathcal{M}, s \models^{\text{CTL}^*} A[\phi]$

- The notion of  $k$ -satisfiability can be encoded via **ordinary propositional calculus**
- Given a model  $\mathcal{M}$  with a distinguished state  $s^1$  and a formula  $\phi$ , we will construct a propositional formula  $[\mathcal{M}, \phi]_k$  s.t.

$$\mathcal{M}, s^1 \models_k^{\text{CTL}^*} E[\phi] \quad \text{iff} \quad [\mathcal{M}, \phi]_k \text{ is satisfiable}$$

We deliberately switch to indexing from 1: you'll see why

- Combined with above theorems, this will yield

$$\mathcal{M}, s^1 \models^{\text{CTL}^*} E[\phi] \quad \text{iff} \quad [\mathcal{M}, \phi]_k \text{ is satisfiable for some } k$$

and still further

$$\mathcal{M}, s^1 \models^{\text{LTL}} (\neg\phi)^{\text{NNF}} \quad \text{iff} \quad [\mathcal{M}, \phi]_k \text{ is unsatisfiable for all } k$$

Recall:  $\mathcal{M}, s \models^{\text{LTL}} \phi$  iff  $\mathcal{M}, s \models^{\text{CTL}^*} A[\phi]$

- In fact, one does not need to scan infinitely many  $k$ : it's always bound by the **diameter** of  $\mathcal{M}$

It's always helpful to have a better bound than the size of  $\mathcal{M}$ .

- Recall your NuSMV experience: states are really vectors of state variables

let's assume all state variables boolean

in principle always doable for enumerative types, if painful

- Recall your NuSMV experience: states are really vectors of state variables
  - let's assume all state variables boolean
  - in principle always doable for enumerative types, if painful
- This is why the size of  $\mathcal{M}$  grows pretty fast



- Recall your NuSMV experience: states are really vectors of state variables

let's assume all state variables boolean

in principle always doable for enumerative types, if painful

- This is why the size of  $\mathcal{M}$  grows pretty fast
- E.g., recall how many states we have in

VAR

```
message : boolean;  
control : boolean;  
success : boolean;
```

- Recall your NuSMV experience: states are really vectors of state variables

let's assume all state variables boolean

in principle always doable for enumerative types, if painful

- This is why the size of  $\mathcal{M}$  grows pretty fast
- E.g., recall how many states we have in

VAR

```
message : boolean;  
control : boolean;  
success : boolean;
```

- Each state is a tuple:

$s = (s(\text{message}), s(\text{control}), s(\text{success}))$

- But labelling is just saying which state variables hold at a given state ...

- But labelling is just saying which state variables hold at a given state ...
- ...and a state is completely determined by its labelling!

- But labelling is just saying which state variables hold at a given state ...
- ...and a state is completely determined by its labelling!
- e.g.  $L(s) = \{\text{message, success}\}$  equivalent to

$$s = (\top, \perp, \top)$$

- But labelling is just saying which state variables hold at a given state ...
- ...and a state is completely determined by its labelling!
- e.g.  $L(s) = \{\text{message, success}\}$  equivalent to

$$s = (\top, \perp, \top)$$

- Still more abstractly: each  $s$  is of the form  $(s_1, s_2, s_3)$  and for this concrete  $s$ ,  $L(s) = \{1, 3\}$   
i.e., atoms are now coordinate numbers

- Now for the transition relation  
 $(s_1, \dots, s_3) \longrightarrow (s'_1, \dots, s'_3)$

- Now for the transition relation  
 $(s_1, \dots, s_3) \longrightarrow (s'_1, \dots, s'_3)$
- Recall

```
next(success) := next(control);
next(control) :=
  case
    success : !control;
    TRUE : control;
  esac;
next(message) :=
  case
    success : {TRUE, FALSE};
    TRUE : message;
  esac;
```



- The transition relation

$$(s_1, \dots, s_n) \longrightarrow (s'_1, \dots, s'_n)$$

is defined in each  $\mathcal{M}$  by a DNF of (in)equalities, e.g.,

$$((s_1 = s'_2) \wedge (s_3 \neq s'_3)) \vee (s_2 = s'_4)$$

- The transition relation

$$(s_1, \dots, s_n) \longrightarrow (s'_1, \dots, s'_n)$$

is defined in each  $\mathcal{M}$  by a DNF of (in)equalities, e.g.,

$$((s_1 = s'_2) \wedge (s_3 \neq s'_3)) \vee (s_2 = s'_4)$$

- From now on, we will write  $T^{\mathcal{M}}(s, s')$  for this formula. It is called the **transition predicate**

- The transition relation

$$(s_1, \dots, s_n) \longrightarrow (s'_1, \dots, s'_n)$$

is defined in each  $\mathcal{M}$  by a DNF of (in)equalities, e.g.,

$$((s_1 = s'_2) \wedge (s_3 \neq s'_3)) \vee (s_2 = s'_4)$$

- From now on, we will write  $T^{\mathcal{M}}(s, s')$  for this formula. It is called the **transition predicate**
- We thus use  $s, s'$  as (meta)variables ranging over vectors. For any  $k$ , fix  $s^1, \dots, s^k$  to be a sequence of such (meta)variables

- The transition relation

$$(s_1, \dots, s_n) \longrightarrow (s'_1, \dots, s'_n)$$

is defined in each  $\mathcal{M}$  by a DNF of (in)equalities, e.g.,

$$((s_1 = s'_2) \wedge (s_3 \neq s'_3)) \vee (s_2 = s'_4)$$

- From now on, we will write  $T^{\mathcal{M}}(s, s')$  for this formula. It is called the **transition predicate**
- We thus use  $s, s'$  as (meta)variables ranging over vectors. For any  $k$ , fix  $s^1, \dots, s^k$  to be a sequence of such (meta)variables
- For each atom  $\mathbf{p}$  (which is now identified with a coordinate number), we can write “ $\mathbf{p}(s^i)$ ” to abbreviate “ $s_{\mathbf{p}}^i = \top$ ”

- The transition relation

$$(s_1, \dots, s_n) \longrightarrow (s'_1, \dots, s'_n)$$

is defined in each  $\mathcal{M}$  by a DNF of (in)equalities, e.g.,

$$((s_1 = s'_2) \wedge (s_3 \neq s'_3)) \vee (s_2 = s'_4)$$

- From now on, we will write  $T^{\mathcal{M}}(s, s')$  for this formula. It is called the **transition predicate**
- We thus use  $s, s'$  as (meta)variables ranging over vectors. For any  $k$ , fix  $s^1, \dots, s^k$  to be a sequence of such (meta)variables
- For each atom  $\mathbf{p}$  (which is now identified with a coordinate number), we can write “ $\mathbf{p}(s^i)$ ” to abbreviate “ $s_{\mathbf{p}}^i = \top$ ”
- Atoms either of the form “ $s_{\mathbf{p}}^i = s_{\mathbf{q}}^j$ ” or of the form “ $s_{\mathbf{p}}^i = \top$ ”

- We already have the first conjunct of our  $[[\mathcal{M}, \phi]]_k$

- We already have the first conjunct of our  $[[\mathcal{M}, \phi]]_k$
- It is  $[[\mathcal{M}]]_k := \bigwedge_{i=1}^{k-1} T^{\mathcal{M}}(s^i, s^{i+1})$   
see why I didn't want to start from 0?

- We already have the first conjunct of our  $[\mathcal{M}, \phi]_k$
- It is  $[\mathcal{M}]_k := \bigwedge_{i=1}^{k-1} T^{\mathcal{M}}(s^i, s^{i+1})$   
see why I didn't want to start from 0?
- Now we need to encode the bounded semantics



## Recall the loopless variant of $k$ -semantics ...

- $\mathcal{M}, \pi \models_k^i \mathbf{p}$  if  $\mathbf{p} \in L(\pi(i))$
- $\mathcal{M}, \pi \models_k^i \neg \mathbf{p}$  if  $\mathbf{p} \notin L(\pi(i))$
- $\mathcal{M}, \pi \models_k^i \phi \wedge \psi$  if  $\mathcal{M}, \pi \models_k^i \phi$  and  $\mathcal{M}, \pi \models_k^i \psi$
- $\mathcal{M}, \pi \models_k^i \phi \vee \psi$  if  $\mathcal{M}, \pi \models_k^i \phi$  or  $\mathcal{M}, \pi \models_k^i \psi$
- $\mathcal{M}, \pi \models_k^i \mathbf{X}\phi$  if  $i < k$  and  $\mathcal{M}, \pi \models_k^{i+1} \phi$
- $\mathcal{M}, \pi \models_k^i \mathbf{F}\phi$  if  $\exists n. i \leq n \leq k$  and  $\mathcal{M}, \pi_n \models_k^i \phi$
- it is **never** the case that  $\mathcal{M}, \pi \models_k^i \mathbf{G}\phi$
- $\mathcal{M}, \pi \models_k^i \phi \mathbf{U}\psi$  if  $\exists n. i \leq n \leq k$  and  $\mathcal{M}, \pi \models_k^n \psi$  and  $\forall j$  s.t.  $i \leq j < n. \mathcal{M}, \pi \models_k^j \phi$
- $\mathcal{M}, \pi \models_k^i \phi \mathbf{R}\psi$  if  $\exists n. i \leq n \leq k$  and  $\mathcal{M}, \pi \models_k^n \phi$  and  $\forall j$  s.t.  $i \leq j \leq n. \mathcal{M}, \pi \models_k^j \psi$

note the “or” clause is not there anymore: it’s like **G**

Now for  $i \leq k$  define ...

- $[\mathbf{p}]_k^i := \mathbf{p}(s^i)$
- $[\neg\mathbf{p}]_k^i := \neg\mathbf{p}(s^i)$
- $[\phi \wedge \psi]_k^i := [\phi]_k^i \wedge [\psi]_k^i$
- $[\phi \vee \psi]_k^i := [\phi]_k^i \vee [\psi]_k^i$
- $[\mathbf{X}\phi]_k^i := [\phi]_k^{i+1}$  if  $i < k$ , else  $\perp$
- $[\mathbf{F}\phi]_k^i := \bigvee_{j=i}^k [\phi]_k^j$
- $[\mathbf{G}\phi]_k^i := \perp$
- $[\phi\mathbf{U}\psi]_k^i := \bigvee_{j=i}^k ([\psi]_k^j \wedge \bigwedge_{n=i}^{j-1} [\phi]_k^n)$
- $[\phi\mathbf{R}\psi]_k^i := \bigvee_{j=i}^k ([\phi]_k^j \wedge \bigwedge_{n=i}^j [\psi]_k^n)$

- For the loop variant, we need to pick  $l \leq k$ : the state variable to which  $k$  loops back

- For the loop variant, we need to pick  $l \leq k$ : the state variable to which  $k$  loops back
- $s^l$  is the successor of  $s^k$ , even though it lies in its past

- For the loop variant, we need to pick  $l \leq k$ : the state variable to which  $k$  loops back
- $s^l$  is the successor of  $s^k$ , even though it lies in its past
- Of course, we don't know where it exactly it loops back to

- For the loop variant, we need to pick  $l \leq k$ : the state variable to which  $k$  loops back
- $s^l$  is the successor of  $s^k$ , even though it lies in its past
- Of course, we don't know where it exactly it loops back to
- We will thus need to form a disjunction of suitably translated formulas for each  $l$  ... but let's not jump ahead

Now for  $i, l \leq k$  define ...

$$\begin{aligned}
 l[\mathbf{p}]_k^i &:= \mathbf{p}(s^i), & l[\neg \mathbf{p}]_k^i &:= \neg \mathbf{p}(s^i) \\
 l[\phi \wedge \psi]_k^i &:= l[\phi]_k^i \wedge l[\psi]_k^i, & l[\phi \vee \psi]_k^i &:= l[\phi]_k^i \vee l[\psi]_k^i \\
 l[\mathbf{X}\phi]_k^i &:= l[\phi]_k^{\text{succ}(i)}, \text{ where } \text{succ}(k) := l \\
 l[\mathbf{F}\phi]_k^i &:= \bigvee_{j=\min(i,l)}^k l[\phi]_k^j \\
 l[\mathbf{G}\phi]_k^i &:= \bigwedge_{j=\min(i,l)}^k l[\phi]_k^j \\
 l[\phi \mathbf{U} \psi]_k^i &:= \bigvee_{j=i}^k (l[\psi]_k^j \wedge \bigwedge_{n=i}^{j-1} l[\phi]_k^n) \vee \\
 &\quad \bigvee_{j=l}^{i-1} (l[\psi]_k^j \wedge \bigwedge_{n=i}^k l[\phi]_k^n \wedge \bigwedge_{n=l}^{j-1} l[\phi]_k^n) \\
 l[\phi \mathbf{R} \psi]_k^i &:= \bigvee_{j=i}^k (l[\phi]_k^j \wedge \bigwedge_{n=i}^j l[\psi]_k^n) \vee \bigwedge_{j=\min(i,l)}^k l[\psi]_k^j \vee \\
 &\quad \bigvee_{j=l}^{i-1} (l[\phi]_k^j \wedge \bigwedge_{n=i}^k l[\psi]_k^n \wedge \bigwedge_{n=l}^j l[\psi]_k^n)
 \end{aligned}$$

- Each element of the family  $\{l[\phi]_k^1\}_{l \leq k}$  should give rise to a disjunct of (some subformula of)  $[\mathcal{M}, \phi]_k$



- Each element of the family  $\{l[\phi]_k^1\}_{l \leq k}$  should give rise to a disjunct of (some subformula of)  $[\mathcal{M}, \phi]_k$
- But clearly, each such disjunct should also state that  $k$  loops back precisely to  $l$  rather than some other  $l' \leq k$

- Each element of the family  $\{ {}_l \llbracket \phi \rrbracket_k^1 \}_{l \leq k}$  should give rise to a disjunct of (some subformula of)  $\llbracket \mathcal{M}, \phi \rrbracket_k$
- But clearly, each such disjunct should also state that  $k$  loops back precisely to  $l$  rather than some other  $l' \leq k$
- We already know how to say it:  ${}_l L_k^{\mathcal{M}} := T^{\mathcal{M}}(s^k, s^l)$   
recall this was the DNF of (in-)equalities defining  $\rightarrow$  in  $\mathcal{M}$

- Each element of the family  $\{l[[\phi]]_k^1\}_{l \leq k}$  should give rise to a disjunct of (some subformula of)  $[[\mathcal{M}, \phi]]_k$
- But clearly, each such disjunct should also state that  $k$  loops back precisely to  $l$  rather than some other  $l' \leq k$
- We already know how to say it:  ${}_l L_k^{\mathcal{M}} := T^{\mathcal{M}}(s^k, s^l)$   
recall this was the DNF of (in-)equalities defining  $\rightarrow$  in  $\mathcal{M}$
- Hence, by the way, we also know how to say that the path is a  $k$ -loop:  $L_k^{\mathcal{M}} := \bigvee_{l=1}^k {}_l L_k^{\mathcal{M}}$

- Each element of the family  $\{l[\phi]_k^1\}_{l \leq k}$  should give rise to a disjunct of (some subformula of)  $[\mathcal{M}, \phi]_k$
- But clearly, each such disjunct should also state that  $k$  loops back precisely to  $l$  rather than some other  $l' \leq k$
- We already know how to say it:  ${}_l L_k^{\mathcal{M}} := T^{\mathcal{M}}(s^k, s^l)$   
recall this was the DNF of (in-)equalities defining  $\rightarrow$  in  $\mathcal{M}$
- Hence, by the way, we also know how to say that the path is a  $k$ -loop:  $L_k^{\mathcal{M}} := \bigvee_{l=1}^k {}_l L_k^{\mathcal{M}}$
- Finally, we can define

$$[\mathcal{M}, \phi]_k := [\mathcal{M}]_k \wedge ((\neg L_k^{\mathcal{M}} \wedge [\phi]_k^1) \vee \bigvee_{l=1}^k ({}_l L_k^{\mathcal{M}} \wedge l[\phi]_k^1))$$

- Each element of the family  $\{l[\phi]_k^1\}_{l \leq k}$  should give rise to a disjunct of (some subformula of)  $[\mathcal{M}, \phi]_k$
- But clearly, each such disjunct should also state that  $k$  loops back precisely to  $l$  rather than some other  $l' \leq k$
- We already know how to say it:  ${}_l L_k^{\mathcal{M}} := T^{\mathcal{M}}(s^k, s^l)$   
recall this was the DNF of (in-)equalities defining  $\rightarrow$  in  $\mathcal{M}$
- Hence, by the way, we also know how to say that the path is a  $k$ -loop:  $L_k^{\mathcal{M}} := \bigvee_{l=1}^k {}_l L_k^{\mathcal{M}}$
- Finally, we can define

$$[\mathcal{M}, \phi]_k := [\mathcal{M}]_k \wedge ((\neg L_k^{\mathcal{M}} \wedge [\phi]_k^1) \vee \bigvee_{l=1}^k ({}_l L_k^{\mathcal{M}} \wedge l[\phi]_k^1))$$

- This is a propositional formula, whose atoms are either of the form “ $s_{\mathbf{p}}^i = s_{\mathbf{q}}^j$ ” or of the form “ $s_{\mathbf{p}}^i = \top$ ”

## Theorem

Given a model  $\mathcal{M}$  with a distinguished state  $s^1$  and a formula  $\phi$ , we have

$$\mathcal{M}, s^1 \models_k^{\text{CTL}^*} E[\phi] \quad \text{iff} \quad [[\mathcal{M}, \phi]]_k \text{ is satisfiable}$$

## Corollary

$$\mathcal{M}, s^1 \models^{\text{CTL}^*} E[\phi] \quad \text{iff} \quad [[\mathcal{M}, \phi]]_k \text{ is satisfiable for some } k$$

## Corollary

$$\mathcal{M}, s^1 \models^{\text{LTL}} (\neg\phi)^{\text{NNF}} \quad \text{iff} \quad [[\mathcal{M}, \phi]]_k \text{ is unsatisfiable for all } k$$

Recall:  $\mathcal{M}, s \models^{\text{LTL}} \phi$  iff  $\mathcal{M}, s \models^{\text{CTL}^*} A[\phi]$

- Recall again: in fact, one does not need to scan infinitely many  $k$ . It's always bound by the **diameter** of  $\mathcal{M}$

- Recall again: in fact, one does not need to scan infinitely many  $k$ . It's always bound by the **diameter** of  $\mathcal{M}$
- This allow us to replace a potentially infinite disjunction over all possible  $k$ 's with a finite one

As noted by Biere et al. 1999, when a Kripke structure comes with an **explicit state representation**, diameter computed by an **easy graph algorithm**, but with a **boolean representation**, encoding that  $n$  is the diameter may require a **QBF**. Less tight constraints, like being the **recurrence diameter**, easier to encode



- Recall again: in fact, one does not need to scan infinitely many  $k$ . It's always bound by the **diameter** of  $\mathcal{M}$
- This allow us to replace a potentially infinite disjunction over all possible  $k$ 's with a finite one

As noted by Biere et al. 1999, when a Kripke structure comes with an **explicit state representation**, diameter computed by an **easy graph algorithm**, but with a **boolean representation**, encoding that  $n$  is the diameter may require a **QBF**. Less tight constraints, like being the **recurrence diameter**, easier to encode

- Another important complexity observation: the size of  $[\mathcal{M}, \phi]_k$  can be made polynomial in  $\phi$  using the technique of **sharing common subformulas**

- We can now use very powerful SAT-solving techniques. Some, like **Stalmarck's algorithm**, patented: you cannot write a commercial tool using it

- We can now use very powerful SAT-solving techniques. Some, like **Stalmarck's algorithm**, patented: you cannot write a commercial tool using it
- In fact, the company owning it is Prover Technologies: the engine behind the SCADE tool

- We can now use very powerful SAT-solving techniques. Some, like **Stalmarck's algorithm**, patented: you cannot write a commercial tool using it
- In fact, the company owning it is Prover Technologies: the engine behind the SCADE tool
- Bounded model checking extended with ***k*-induction** and various heuristics

for some, see *Generating Property-Directed Potential Invariants By Backward Analysis* by Champion, Delmas, Dierkes; *Instantiation-Based Invariant Discovery* by Kahsai, Ge, Tinelli

- We can now use very powerful SAT-solving techniques. Some, like **Stalmarck's algorithm**, patented: you cannot write a commercial tool using it
- In fact, the company owning it is Prover Technologies: the engine behind the SCADE tool
- Bounded model checking extended with ***k*-induction** and various heuristics

for some, see *Generating Property-Directed Potential Invariants By Backward Analysis* by Champion, Delmas, Dierkes; *Instantiation-Based Invariant Discovery* by Kahsai, Ge, Tinelli

- Standard SAT-solving techniques, e.g., the **Davis-Putnam-Logemann-Loveland (DPLL) algorithm** (1962) ...