

# Software Testing and Validation

A.A. 2022/2023

Corso di Laurea in Informatica

## The NuSMV Model Checker

Igor Melatti

Università degli Studi dell'Aquila

Dipartimento di Ingegneria e Scienze dell'Informazione e Matematica



UNIVERSITÀ  
DEGLI STUDI  
DELL'AQUILA



DISIM  
Dipartimento di Ingegneria  
e Scienze dell'Informazione  
e Matematica

# CTL (and LTL) Model Checking

- We saw the theoretical algorithm for CTL model checking
  - we said it was not effective, as it required  $S$  and  $R$  to be in RAM
- Actually, there are methodologies which are able to fit  $S$  and  $R$  in RAM, also for industrial-sized models
- The “father” of the model checkers using such technologies is SMV
  - Symbolic Model Verifier
  - it has then been refactored as NuSMV
- This set of techniques is referred to as *symbolic model checking*
  - Murphi and SPIN style is dubbed *explicit model checking*



UNIVERSITÀ  
DEGLI STUDI  
DELL'AQUILA



DISIM  
Dipartimento di Ingegneria  
e Scienze dell'Informazione  
e Matematica

# CTL (and LTL) Model Checking

- In order to understand how symbolic model checking works, we need some preliminaries
- ROBDDs
  - needed to actually fit  $S$  and  $R$  in RAM
- $\mu$ -calculus
  - together with fixpoint computation
  - extension of  $\lambda$ -calculus
  - needed to efficiently implement CTL and LTL model checking using ROBDDs



UNIVERSITÀ  
DEGLI STUDI  
DELL'AQUILA



DISIM  
Dipartimento di Ingegneria  
e Scienze dell'Informazione  
e Matematica

- Reduced Ordered (Complemented Edges) Binary Decision Diagrams
  - sometimes called simply OBDDs, and even BDDs
  - here we stick to the precise notation, by also outlining the differences
- Let us start with the basis: BDD
- A BDD is a data structure representing a boolean function
  - of course, OBDDs and ROBDDs are data structure as well
  - we will define them in the following



# Boolean Functions

- In our setting a boolean function is  $f : \mathbb{B}^n \rightarrow \mathbb{B}$ 
  - where  $\mathbb{B} = \{0, 1\}$  is the set of boolean values
  - 0 stands for false, 1 for true
  - thus, our boolean functions have  $n$  boolean variables as arguments
  - and return a single boolean value
- Examples:
  - 0 and 1 are boolean functions with  $n = 0$
  - complementation ( $f(x) = \neg x$ ) and identity ( $f(x) = x$ ) are boolean functions with  $n = 1$
  - AND ( $f(x, y) = x \wedge y$ ), OR ( $f(x, y) = x \vee y$ ) are boolean functions with  $n = 2$
  - generally speaking, there are  $2^{2^n}$  different boolean functions of  $n$  boolean variables



UNIVERSITÀ  
DEGLI STUDI  
DELL'AQUILA



DISIM  
Dipartimento di Ingegneria  
e Scienze dell'Informazione  
e Matematica

# Boolean Functions Representation

- Roughly speaking, if you have  $f(x) = x + 1$  with  $x \in \mathbb{R}$ , you can only represent  $f$  through its computation
  - rules s.t., given  $x$ , you compute  $x + 1$
- For boolean functions, the explicit tabular representation is also possible (*truth table*)
  - a table with  $n + 1$  columns
  - first  $n$  columns are for variables values
  - last column is for function value
  - of course, you need  $2^n$  rows



UNIVERSITÀ  
DEGLI STUDI  
DELL'AQUILA



DISIM  
Dipartimento di Ingegneria  
e Scienze dell'Informazione  
e Matematica

# All Boolean Functions of 2 Variables

$p$	$q$	$F^0$	NOR <sup>1</sup>	$\leftrightarrow^2$	$\neg p^3$	$\leftrightarrow^4$	$\neg q^5$	XOR <sup>6</sup>	NAND <sup>7</sup>	AND <sup>8</sup>	XNOR <sup>9</sup>	$q^{10}$	$\rightarrow^{11}$	$p^{12}$	$\leftarrow^{13}$	OR <sup>14</sup>	$\neg^{15}$
T	T	F	F	F	F	F	F	F	F	T	T	T	T	T	T	T	T
T	F	F	F	F	F	T	T	T	T	F	F	F	F	T	T	T	T
F	T	F	F	T	T	F	F	T	T	F	F	T	T	F	F	T	T
F	F	F	T	F	T	F	T	F	T	F	T	F	T	F	T	F	T

- A truth table must take into account all possible values for all its  $n$  arguments
- Which leads to a  $O(2^n)$  RAM required
  - even with optimizations (e.g., only 1 column is actually needed)
- One could represent functions with CNF or DNF, but they are difficult to manipulate
  - furthermore, not canonical: there may be multiple CNFs or DNFs for the same function



UNIVERSITÀ  
DEGLI STUDI  
DELL'AQUILA



DISIM  
Dipartimento di Ingegneria  
e Scienze dell'Informazione  
e Matematica

# Boolean Functions for Model Checking

- In Model Checking algorithms, the following operations are needed:
  - compute the returned value for a given tuple of values  $b_1, \dots, b_n$ 
    - could be ok for truth tables
  - test of equivalence between boolean functions  $f_1 \equiv f_2$ 
    - not ok for truth tables
    - needs canonicity
  - compute the representation of a logical combination of boolean functions
    - e.g.: given the representation of  $f_1, f_2$ , compute the representation of  $f_1 \wedge f_2$
    - not ok for truth tables
- Goal: find a representation able to fulfill such requirements
  - while possibly requiring less than  $O(2^n)$



UNIVERSITÀ  
DEGLI STUDI  
DELL'AQUILA



DISIM  
Dipartimento di Ingegneria  
e Scienze dell'Informazione  
e Matematica



# Binary Decision Diagrams

- Roughly speaking, it is a connected DAG (Directed Acyclic Graph), i.e., a tree
  - only one root
  - each internal node has two successors
  - nodes are labeled by boolean variables
  - edges are labeled by boolean values
  - only two leaves, labeled with boolean values

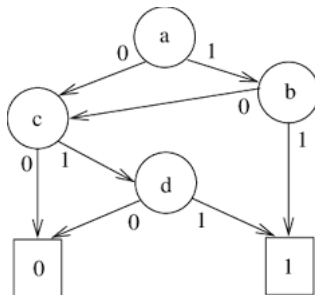


UNIVERSITÀ  
DEGLI STUDI  
DELL'AQUILA



DISIM  
Dipartimento di Ingegneria  
e Scienze dell'Informazione  
e Matematica

# Binary Decision Diagrams



Represented function:  $f(a, b, c, d) = ab + \bar{a}cd + \bar{a}bcd$

- recall that  $+$  is OR,  $\cdot$  is AND,  $\bar{\phantom{x}}$  is negation



UNIVERSITÀ  
DEGLI STUDI  
DELL'AQUILA



DISIM  
Dipartimento di Ingegneria  
e Scienze dell'Informazione  
e Matematica

# BDDs: Formal Definition

- A BDD is a tuple  $\mathcal{B} = \langle V, E, r, \mathcal{V}, \text{var}, \text{low}, \text{high} \rangle$  where:
  - $V$  is a finite set of nodes containing two special nodes **0** and **1**
  - $E \subseteq V \times V$  is a set of edges s.t.:
    - there are no cycles, i.e., for all path  $\pi = v_0, \dots, v_n$ , where  $\forall i = 0, \dots, n. v_i \in V$  and  $\forall i = 0, \dots, n-1. (v_i, v_{i+1}) \in E$ , we have that  $i \neq j$  implies  $v_i \neq v_j$
    - let  $S(v) = \{w \in V \mid (v, w) \in E\}$  be the set of successors of  $v$
    - each internal node has exactly two successors, i.e.,  $\forall v \in V \setminus \{0, 1\}. |S(v)| = 2$
    - **0** and **1** are *terminal nodes*, i.e.,  $\forall v \in \{0, 1\}. |S(v)| = 0$
  - $r \in V$  is the root (i.e.,  $\forall v \in V. (v, r) \notin E$ )
  - $\text{low}, \text{high} : V \rightarrow V$  is the labeling of edges
    - the labeling must be consistent with  $E$ , i.e.,  $\forall v \in V. \text{low}(v), \text{high}(v) \in S(v)$



# BDDs: Formal Definition

- A BDD is a tuple  $\mathcal{B} = \langle V, E, r, \mathcal{V}, \text{var}, \text{low}, \text{high} \rangle$  where:
  - $\mathcal{V}$  is a finite set of boolean variables
    - thus, the boolean function represented by  $\mathcal{B}$  will depend on variables in  $\mathcal{V}$
    - it may be a subset of  $\mathcal{V}$
  - $\text{var} : V \rightarrow \mathcal{V}$  is the labeling of nodes
- A *maximal path* in  $\mathcal{B}$  starts from  $r$  and ends up either in **0** or **1**
- The *semantics* of  $\mathcal{B}$  is the boolean function represented by  $\mathcal{B}$ 
  - intuitively, we follow all maximal paths which end up in **1**
  - formally: next slide



UNIVERSITÀ  
DEGLI STUDI  
DELL'AQUILA



DISIM  
Dipartimento di Ingegneria  
e Scienze dell'Informazione  
e Matematica

# BDDs: Semantics

- Given a BDD  $\mathcal{B} = \langle V, E, r, \mathcal{V}, \text{var}, \text{low}, \text{high} \rangle$ , we recursively define the semantics of each node  $v \in V$ 
  - each node may be seen as the root of a subtree...
  - notation:  $\llbracket v \rrbracket_{\mathcal{B}}$ , or simply  $\llbracket v \rrbracket$  when  $\mathcal{B}$  is understood
- Terminal nodes denote the boolean constants:  
 $\llbracket 0 \rrbracket = \text{false}$ ,  $\llbracket 1 \rrbracket = \text{true}$
- For internal nodes  $v \in V \setminus \{0, 1\}$ , semantics is defined as  
 $\llbracket v \rrbracket = \text{var}(v) \llbracket \text{high}(v) \rrbracket + \overline{\text{var}(v)} \llbracket \text{low}(v) \rrbracket$ 
  - recall that  $+$  is OR,  $\cdot$  is AND,  $\bar{\cdot}$  is negation
- The semantics of  $\mathcal{B}$  is of course  $\llbracket r \rrbracket$



# Canonicity of BDDs

- For a given BDD  $\mathcal{B}$ , we have a unique represented boolean function
- Given a boolean function  $f$ , there is a BDD  $\mathcal{B}$  representing  $f$ , i.e.,  $\llbracket r \rrbracket_{\mathcal{B}} = f$
- However, there may be a BDD  $\mathcal{B}' \neq \mathcal{B}$  s.t.  $\llbracket r' \rrbracket'_{\mathcal{B}} = f$  as well
  - thus, BDDs are not canonical
- Thus, ROBDDs are introduced: by setting limitations, they achieve canonicity
  - for a boolean function  $f$ , there exists a *unique* ROBDD representing  $f$
- Furthermore, for increasing efficiency, complemented edges are introduced
  - number of nodes is reduced



- An OBDD (Ordered BDD)  
 $\mathcal{B} = \langle V, E, r, \mathcal{V}, \text{var}, \text{low}, \text{high}, \text{ord} \rangle$ , is a BDD with an additional ord function
- Namely,  $\text{ord} : \mathcal{V} \rightarrow \{1, \dots, |\mathcal{V}|\}$
- The following properties must hold
  - ord is injective, i.e.,  $\forall v, w \in \mathcal{V}. \text{ord}(v) = \text{ord}(w) \rightarrow v = w$
  - note that this implies that ord is indeed bijective...
  - defines an *ordering* on variables in  $\mathcal{V}$ , e.g., if  $\text{ord}(v) = 10$  then  $v$  is the tenth variable
  - Given a path  $\pi$  on  $\mathcal{B}$ , variables on nodes follow ord
  - i.e.,  $\forall \pi = v_0, \dots, v_n$  s.t.  $\forall i = 0, \dots, n. v_i \in V$  and  $\forall i = 0, \dots, n-1. (v_i, v_{i+1}) \in E$  and  $v_n \notin \{0, 1\}$ , we have that  $i < j$  implies  $\text{ord}(\text{var}(v_i)) < \text{ord}(\text{var}(v_j))$



# COBDDs

- A COBDD (Complemented edges OBDD)  
 $\mathcal{B} = \langle V, E, r, \mathcal{V}, \text{var}, \text{low}, \text{high}, \text{ord}, \text{flip} \rangle$ , is a BDD with an additional  $\text{flip} : V \rightarrow \{0, 1\}$
- For an internal node  $v$ , if  $\text{flip}(v)$  holds then the *else edge* of  $v$  is complemented
- There is now only one terminal node **1**
  - **0** is not needed because of complementation
- Semantics changes, also a flipping bit  $b \in \{0, 1\}$  is necessary
- Terminal node denote the boolean constants:  $\llbracket \mathbf{1}, b \rrbracket = \bar{b}$
- For internal nodes  $v \in V \setminus \{\mathbf{1}\}$ , semantics is defined as  $\llbracket v, b \rrbracket = \text{var}(v) \llbracket \text{high}(v), b \rrbracket + \overline{\text{var}(v)} \llbracket \text{low}(v), b \oplus \text{flip}(v) \rrbracket$
- Semantics of  $\mathcal{B}$  is  $\llbracket r, \text{flip}(r) \rrbracket$





# ROBDDs

- A ROBDD (Reduced OBDD)  $\mathcal{B}$  is a COBDD with the least number of nodes
  - among the ones representing the same boolean function
- From now on, as usual in the literature, we will use OBDD as synonym for ROBDD
- Efficient algorithms ( $O(n)$ , being  $n$  the number of nodes) exist to compute the AND and the OR of two OBDDs
  - negation is  $O(1)$ : just complement  $\text{flip}(r)$ !
- Typically implemented with hash tables of already computed ROBDDs
  - speedup computations
  - equality check is  $O(1)$ : just compare  $r$  and  $r'$
- Furthermore: multi-rooted DAG can be used to represent multiple functions, sharing some nodes



UNIVERSITÀ  
DEGLI STUDI  
DELL'AQUILA



DISIM  
Dipartimento di Ingegneria  
e Scienze dell'Informazione  
e Matematica

## Other Important OBDD Operations

- Application: given the OBDD for  $f(x_1, \dots, x_i, \dots, x_n)$ , compute the OBDD for  $f(x_1, \dots, 0, \dots, x_n)$  or  $f(x_1, \dots, 1, \dots, x_n)$ 
  - sometimes also written  $f(x_1, \dots, x_n)|_{x_i=0}$  or  $f(x_1, \dots, x_n)|_{x_i=1}$
  - Shannon expansion: for every boolean function  $f$ ,  
$$f(x_1, \dots, x_n) = \bar{x}_i f(x_1, \dots, x_n)|_{x_i=0} + x_i f(x_1, \dots, x_n)|_{x_i=1}$$
- Given  $f(x, y)$ , compute the OBDD for:
  - existentialization:  $\exists x : f(x, y) \equiv f(0, y) + f(1, y)$
  - universalization:  $\forall x. f(x, y) \equiv f(0, y) \cdot f(1, y)$
  - both generalized to multiple variables  $x_1, \dots, x_n$
- Given  $f(x), g(x), h(x)$ , compute the OBDD for  $ITE(f, g, h)$ 
  - ITE stands for if-then-else
  - thus,  $ITE(f, g, h) = fg + \bar{f}h$



# OBDD and Model Checking

- OBDDs extremely good in representing *characteristic functions* of finite sets
  - the characteristic function  $\chi : U \rightarrow \{0, 1\}$  of a set  $X \subseteq U$  is defined as

$$\chi(x) = \begin{cases} 1 & \text{if } x \in X \\ 0 & \text{otherwise} \end{cases}$$

- If  $U$  is finite, then each element  $x \in U$  may be encoded using  $n = \lceil \log(|X|) \rceil$  boolean variables  $x_1, \dots, x_n$
- Thus,  $\chi$  may be represented by an OBDD on  $x_1, \dots, x_n$ 
  - as for Model Checking, we may represent  $S$ ,  $\text{Reach}(S)$ ,  $R$ , ...
  - $R$  will need  $2n$  variables!
  - CTL Model Checking algorithm becomes feasible!
    - for many interesting real-sized systems,  $S$ ,  $\text{Reach}(S)$ ,  $R$  will now fit in RAM



UNIVERSITÀ  
DEGLI STUDI  
DELL'AQUILA



DISIM  
Dipartimento di Ingegneria  
e Scienze dell'Informazione  
e Matematica

# OBDD and Model Checking

- The most difficult part is to derive the OBDD for  $R$  directly from the model specification
  - i.e., from the model checker input language
  - it would be rather difficult to do it with SPIN
    - especially because it has a dynamic state space
  - also the one for Murphi would require some effort
  - $S$  is easy, you only have to look at global variables
    - not in SPIN...
- NuSMV input language is tailored to be easily translated into OBDDs
  - also into CNF, as we will see...



UNIVERSITÀ  
DEGLI STUDI  
DELL'AQUILA



DISIM  
Dipartimento di Ingegneria  
e Scienze dell'Informazione  
e Matematica

- SMV (Symbolic Model Verifier): McMillan implementation of the ideas in the famous paper “Symbolic model checking:  $10^{20}$  states and beyond”
  - McMillan PhD dissertation about SMV is one of the most important dissertations in Computer Science
- SMV has been then re-written and standardized by the research group in Trento (also Genova and CMU collaborated), thus becoming NuSMV
  - the engine is still McMillan’s work
  - code has been nearly entirely commented, and made more readable
  - some features has been added: interactive mode, bounded model checking
  - OBDDs are handled via the CUDD library (by E. Somenzi at Colorado University)



# NuSMV Input Language

Taken from `examples/smv-dist/short.smv`

```
MODULE main
```

```
VAR
```

```
  request : {Tr, Fa}; -- same as saying boolean
                      -- (stand for True and False)
```

```
  state : {ready, busy};
```

```
ASSIGN
```

```
  init(state) := ready;
```

```
  next(state) := case
```

```
    state = ready & (request = Tr): busy;
```

```
    1 : {ready, busy};
```

```
  esac;
```

```
SPEC
```

```
  AG((request = Tr) -> AF state = busy)
```



UNIVERSITÀ  
DEGLI STUDI  
DELL'AQUILA



DISIM  
Dipartimento di Ingegneria  
e Scienze dell'Informazione  
e Matematica

# NuSMV Input Language

- One *module*, there may be more, but one of them must be named `main`
- Module variables are those declared with `VAR`
- Base types are like Murphi ones: enumerations and integer subranges, plus the `word` type (i.e., an array of bits)
- Arrays are possible, but can be indexed only with constants
- Structures are modeled through modules
  - That is, each module has its variables (fields of a structure) and may be instantiated many times



UNIVERSITÀ  
DEGLI STUDI  
DELL'AQUILA



DISIM  
Dipartimento di Ingegneria  
e Scienze dell'Informazione  
e Matematica

# NuSMV Input Language

- ASSIGN section specifies (indirectly; it is also possible to it directly, as we will see) the set  $I$  (via `init`) and the relation  $R$  (via `next`)
  - as in Murphi, there expressions which are essentially guard/action
  - differently from Murphi, each action deals with *one variable only*
    - the guard may be defined on any other variable (and it is typically the case)
  - if something is not specified, then it is understood to be non-deterministic



UNIVERSITÀ  
DEGLI STUDI  
DELL'AQUILA



DISIM  
Dipartimento di Ingegneria  
e Scienze dell'Informazione  
e Matematica



# NuSMV Input Language: ASSIGN

- E.g., in `short.smv` initial states are those in which state is ready and request may be either Tr or Fa
- Thus, there are 2 initial states  $I = \{\langle \text{ready}, \text{Tr} \rangle, \langle \text{ready}, \text{Fa} \rangle\}$ , which may be represented with  $\langle \text{ready}, \perp \rangle$
- Also `next(request)` is not specified; before analyzing what does this mean, let us see `next(state)`
- The case expression works as follows: the first condition  $C$  which is evaluated to true is fired, other true guards possibly following  $C$  are ignored



UNIVERSITÀ  
DEGLI STUDI  
DELL'AQUILA



DISIM  
Dipartimento di Ingegneria  
e Scienze dell'Informazione  
e Matematica

# NuSMV Input Language: ASSIGN

- This allows to put 1 (i.e., true) as the last guard, representing the “default” case
- NuSMV also checks if a case expression is exhaustive in its conditions, as this allows it to assume that  $T$  is total
- Note that the last condition on state leads to a non-deterministic transition: if the first guard is false, then state may take any value between ready e busy, that is any value in its domain
- In general, any subset of the variable domain may be used



UNIVERSITÀ  
DEGLI STUDI  
DELL'AQUILA



DISIM  
Dipartimento di Ingegneria  
e Scienze dell'Informazione  
e Matematica

# NuSMV Input Language: ASSIGN

- request is completely non-deterministic, as it does not occur in any next
- I.e., if other rules tells that the system may go from  $s$  to  $t$  and  $(\text{request} = \text{Fa}) \in L(t)$ , then there exists a transition from  $s$  to  $t'$  with  $(\text{request} = \text{Tr}) \in L(t')$  and  $L(t) \setminus \{(\text{request} = \text{Fa})\} = L(t') \setminus \{(\text{request} = \text{Tr})\}$
- Simply stated, if the system may go from  $s$  to  $t$  and request has a value  $v$  in  $t$ , then the system may also go from  $s$  to  $t'$  s.t.  $t$  and  $t'$  only differ in the value of request, which is different from  $v$
- By combining all non-determinism in this example, the Kripke structure defined here excludes just two transitions

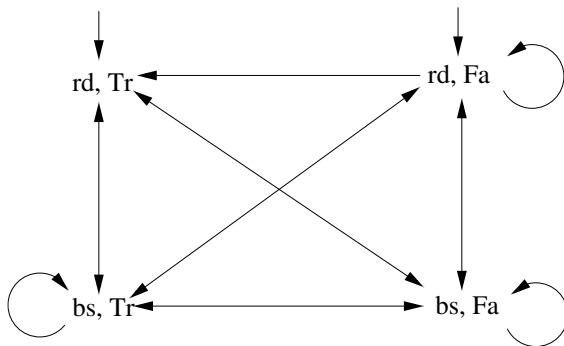


UNIVERSITÀ  
DEGLI STUDI  
DELL'AQUILA



DISIM  
Dipartimento di Ingegneria  
e Scienze dell'Informazione  
e Matematica

## Automata for short.smv: $I$ and $R$



UNIVERSITÀ  
DEGLI STUDI  
DELL'AQUILA



DISIM  
Dipartimento di Ingegneria  
e Scienze dell'Informazione  
e Matematica

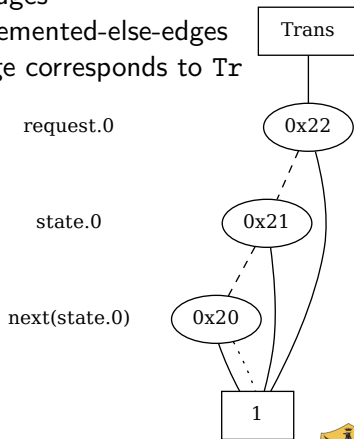
# OBDDs for short.smv: $R$

Straight lines are then-edges

Dashed lines are else-edges

Dotted lines are complemented-else-edges

request.0 “false” edge corresponds to Tr



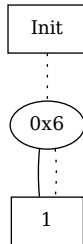
UNIVERSITÀ  
DEGLI STUDI  
DELL'AQUILA



DISIM  
Dipartimento di Ingegneria  
e Scienze dell'Informazione  
e Matematica

# OBDDs for short.smv: /

state.0



UNIVERSITÀ  
DEGLI STUDI  
DELL'AQUILA



DISIM  
Dipartimento di Ingegneria  
e Scienze dell'Informazione  
e Matematica

# NuSMV Input Language

Taken from `examples/smv-dist/short.smv`

```
MODULE main
```

```
VAR
```

```
  request : {Tr, Fa}; -- same as saying boolean
                        -- (stand for True and False)
```

```
  state : {ready, busy};
```

```
ASSIGN
```

```
  init(state) := ready;
```

```
  next(state) := case
```

```
    state = ready & (request = Tr): busy;
```

```
    1 : {ready, busy};
```

```
  esac;
```

```
SPEC
```

```
  AG((request = Tr) -> AF state = busy)
```



UNIVERSITÀ  
DEGLI STUDI  
DELL'AQUILA



DISIM  
Dipartimento di Ingegneria  
e Scienze dell'Informazione  
e Matematica

# NuSMV Input Language

```
MODULE main
VAR
  request : {Tr, Fa};
  state : {ready, busy};
ASSIGN
  init(state) := ready;
  next(state) := case
    state = ready & (request = Tr): busy;
    TRUE : ready;
  esac;
SPEC
  AG((request = Tr) -> AF state = busy)
```



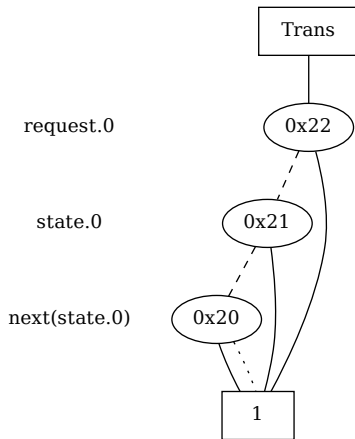
UNIVERSITÀ  
DEGLI STUDI  
DELL'AQUILA



DISIM  
Dipartimento di Ingegneria  
e Scienze dell'Informazione  
e Matematica



# OBDDs for short.smv: $R$

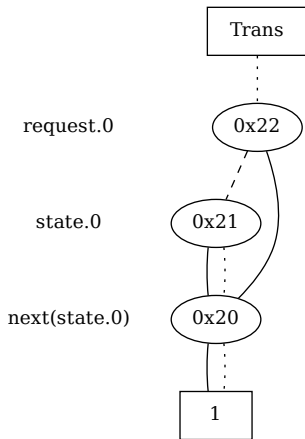


UNIVERSITÀ  
DEGLI STUDI  
DELL'AQUILA



DISIM  
Dipartimento di Ingegneria  
e Scienze dell'Informazione  
e Matematica

# OBDDs for short.soloready.smv: $R$



UNIVERSITÀ  
DEGLI STUDI  
DELL'AQUILA



DISIM  
Dipartimento di Ingegneria  
e Scienze dell'Informazione  
e Matematica

# NuSMV Input Language

```
MODULE main
VAR
  request : {Tr, Fa};
  state : {ready, busy};
ASSIGN
  init(state) := ready;
  next(state) := case
    state = ready & (request = Tr): busy;
    TRUE : ready;
  esac;
  next(request) := request;
SPEC
  AG((request = Tr) -> AF state = busy)
```

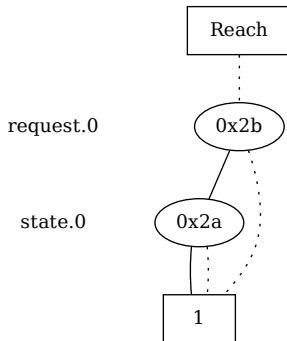


UNIVERSITÀ  
DEGLI STUDI  
DELL'AQUILA



DISIM  
Dipartimento di Ingegneria  
e Scienze dell'Informazione  
e Matematica

# OBDDs for `short.soloready.req_const.smv`: Reach



UNIVERSITÀ  
DEGLI STUDI  
DELL'AQUILA



DISIM  
Dipartimento di Ingegneria  
e Scienze dell'Informazione  
e Matematica

# OBDDs Pros and Cons

```
MODULE main
VAR
  m1 : 0..15; -- m1.0 is MSB!
  m2 : 0..15;
  m3 : 0..30;
ASSIGN
  next(m3) := m1 + m2;

SPEC
  AG(m3 <= 30);
```



UNIVERSITÀ  
DEGLI STUDI  
DELL'AQUILA



DISIM  
Dipartimento di Ingegneria  
e Scienze dell'Informazione  
e Matematica

# OBDDs Pros and Cons

```
MODULE main
VAR
  m1 : 0..15;
  m2 : 0..15;
  m3 : 0..30;
ASSIGN
  next(m3) := case
    m1*m2 <= 30: m1*m2;
    TRUE: m3;
  esac;

SPEC
  AG(m3 <= 30);
```



UNIVERSITÀ  
DEGLI STUDI  
DELL'AQUILA



DISIM  
Dipartimento di Ingegneria  
e Scienze dell'Informazione  
e Matematica

# OBDDs for Adder and Multiplier: /

This is a set with  $16 \cdot 16 \cdot 31 = 7936$  elements  
Just one node to represent it...

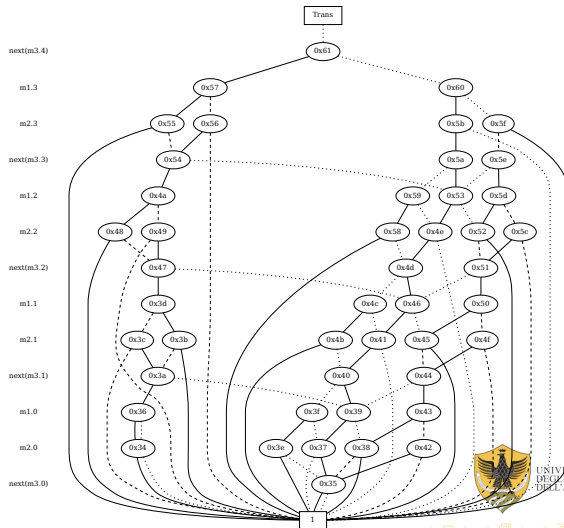


UNIVERSITÀ  
DEGLI STUDI  
DELL'AQUILA



DISIM  
Dipartimento di Ingegneria  
e Scienze dell'Informazione  
e Matematica

# OBDDs for Adder: $R$



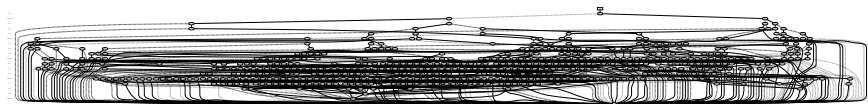
UNIVERSITÀ  
DEGLI STUDI  
DELL'AQUILA



DISIM  
Dipartimento di Ingegneria  
e Scienze dell'Informazione  
e Matematica



# OBDDs for Multiplier: $R$



UNIVERSITÀ  
DEGLI STUDI  
DELL'AQUILA



DISIM  
Dipartimento di Ingegneria  
e Scienze dell'Informazione  
e Matematica

# OBDDs Pros and Cons

- Number of variables is 13 for both models
  - 4 each for  $m_1$  and  $m_2$ , plus 5 for  $m_3$
- Number of BDD nodes:
  - adder: 47
  - multiplier: 538



UNIVERSITÀ  
DEGLI STUDI  
DELL'AQUILA



DISIM  
Dipartimento di Ingegneria  
e Scienze dell'Informazione  
e Matematica

# OBDDs Pros and Cons

- No magic: SAT could be solved using OBDDs
  - just represent the instance with an OBDD and check if it is different from 0
  - very roughly speaking: if it were possible to solve it “efficiently” in this way,  $P=NP$ ...
- Thus, there are boolean functions for which OBDDs representation is exponential, regardless of variable ordering
  - one example is the multiplier seen above
- It is not possible to say if OBDDs will be a good way to represent a problem, before trying it
  - for the adder, it is much more efficient
- Furthermore, finding a variable order in order to minimize the OBDD representation for a given function is an NP-complete problem



UNIVERSITÀ  
DEGLI STUDI  
DELL'AQUILA



DISIM  
Dipartimento di Ingegneria  
e Scienze dell'Informazione  
e Matematica

# OBDDs Pros and Cons

- This also holds for Model Checking in general
- Not possible to say a-priori if a system will fit in the available resources when using a model checker
  - RAM and computation time
- Also, it is not possible to decide which model checker is better
  - explicit (Murphi-or-SPIN like) or symbolic (NuSMV like)?
- However, we are going to see some guidelines
  - as for OBDDs: a good ordering is to interleave present and future variables
  - variable ordering: if OBDDs grow, the model checker can try a different variable ordering



UNIVERSITÀ  
DEGLI STUDI  
DELL'AQUILA



DISIM  
Dipartimento di Ingegneria  
e Scienze dell'Informazione  
e Matematica

# NuSMV Input Language

```
MODULE counter_cell(carry_in)
VAR value : boolean;
ASSIGN
  init(value) := 0;
  next(value) := (value + carry_in) mod 2;
DEFINE carry_out := value & carry_in;

MODULE main
VAR
  bit0 : counter_cell(1);
  bit1 : counter_cell(bit0.carry_out);
  bit2 : counter_cell(bit1.carry_out);

SPEC AG(!bit3.carry_out)
```

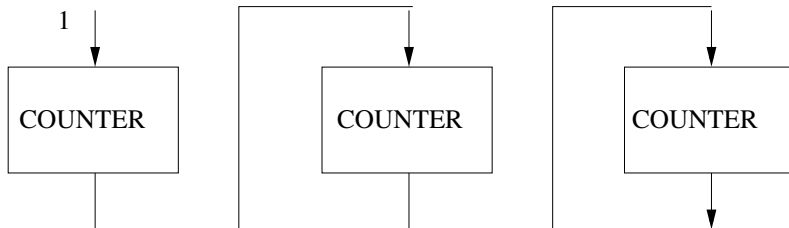


UNIVERSITÀ  
DEGLI STUDI  
DELL'AQUILA



DISIM  
Dipartimento di Ingegneria  
e Scienze dell'Informazione  
e Matematica

# Counter Cell



UNIVERSITÀ  
DEGLI STUDI  
DELL'AQUILA



DISIM  
Dipartimento di Ingegneria  
e Scienze dell'Informazione  
e Matematica

# NuSMV Input Language

- 2 modules, `main` and `counter_cell`
- Main *instantiates* the module `counter_cell` for 3 times
- This is an hardware-like instantiation: the `main` module contains 3 equal copies of the `counter_cell` module, the only difference being the lines in input
- Note that this means the module `main` will have 3 copies of variable `value`



UNIVERSITÀ  
DEGLI STUDI  
DELL'AQUILA



DISIM  
Dipartimento di Ingegneria  
e Scienze dell'Informazione  
e Matematica

# NuSMV Input Language

- Note that `carry_out` (being inside a `DEFINE` section) is *not* a variable, as it is only a shortcut for the expression it defines
  - i.e., there will not be a corresponding variable in the OBDD
  - and indeed, it is not declared as a variable...
- Hence, `bit0` will always sum 1 to its internal variable, and `bit1` will sum 1 only if `bit0` will generate a carry
- The `main` module defines a counter from 0 to 7



UNIVERSITÀ  
DEGLI STUDI  
DELL'AQUILA



DISIM  
Dipartimento di Ingegneria  
e Scienze dell'Informazione  
e Matematica



# NuSMV Input Language

```
MODULE user(semaphore)
VAR
  state : {idle, entering, critical, exiting};
ASSIGN
  init(state) := idle;
  next(state) :=
    case
      state = idle: entering;
      state = entering & !semaphore: critical;
      state = critical: {critical, exiting};
      state = exiting: idle;
    TRUE : state;
esac;
```



UNIVERSITÀ  
DEGLI STUDI  
DELL'AQUILA



DISIM  
Dipartimento di Ingegneria  
e Scienze dell'Informazione  
e Matematica

# NuSMV Input Language

```
next(semaphore) :=  
  case  
    state = entering: TRUE;  
    state = exiting: FALSE;  
    TRUE: semaphore;  
  esac;
```



UNIVERSITÀ  
DEGLI STUDI  
DELL'AQUILA



DISIM  
Dipartimento di Ingegneria  
e Scienze dell'Informazione  
e Matematica

# NuSMV Input Language

```
MODULE main
```

```
VAR
```

```
    semaphore : boolean;
```

```
    proc1 : process user(semaphore);
```

```
    proc2 : process user(semaphore);
```

```
ASSIGN
```

```
    init(semaphore) := FALSE;
```

```
SPEC
```

```
    AG(!(proc1.state = critical & proc2.state = critical))
```

```
LTLSPEC
```

```
    G F proc1.state = critical
```



UNIVERSITÀ  
DEGLI STUDI  
DELL'AQUILA



DISIM  
Dipartimento di Ingegneria  
e Scienze dell'Informazione  
e Matematica

# NuSMV Input Language

- In the previous examples, all variables were evolving at the same time
- There is a global clock as in a synchronous digital circuit: given the current value for all variables in the current clock tick, in the next clock tick all variables may change their variables at the same time (synchronously: hardware parallel execution)
- In this example, instead, instantations are *processes*
- I.e., just one variable at a time may change; other variables are forced to stay fixed
- No dynamic process spawning as in SPIN: the number of processes is known from the beginning



UNIVERSITÀ  
DEGLI STUDI  
DELL'AQUILA



DISIM  
Dipartimento di Ingegneria  
e Scienze dell'Informazione  
e Matematica

# NuSMV Input Language

- *Synchronous vs. asynchronous* systems
- In asynchronous systems, there is essentially one (implicit) additional module, which acts as a scheduler
- This is indeed what the verification algorithm does
- Each process is automatically provided with an additional variable `running` which is true iff that process is currently running



UNIVERSITÀ  
DEGLI STUDI  
DELL'AQUILA



DISIM  
Dipartimento di Ingegneria  
e Scienze dell'Informazione  
e Matematica

# NuSMV Input Language

```
MODULE inverter(input)
VAR
  output : boolean;
ASSIGN
  init(output) := 0;
  next(output) := !input;
```

```
MODULE main
VAR
  gate1 : process inverter(gate3.output);
  gate2 : process inverter(gate1.output);
  gate3 : process inverter(gate2.output);
```

```
SPEC
  AG(!gate2.output | !gate3.output)
```

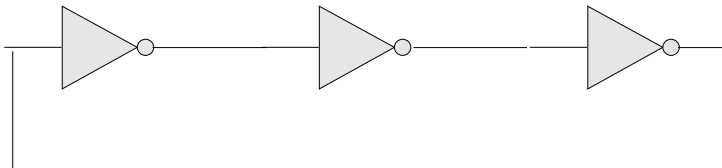


UNIVERSITÀ  
DEGLI STUDI  
DELL'AQUILA



DISIM  
Dipartimento di Ingegneria  
e Scienze dell'Informazione  
e Matematica

# Inverter Cell



UNIVERSITÀ  
DEGLI STUDI  
DELL'AQUILA



DISIM  
Dipartimento di Ingegneria  
e Scienze dell'Informazione  
e Matematica

# NuSMV Input Language

Using Direct specification it is possible to define non-total transition relations or empty initial states set

```
MODULE inverter(input)
VAR
    output : boolean;
INIT
    output = 0
TRANS
    next(output) = !input
```



UNIVERSITÀ  
DEGLI STUDI  
DELL'AQUILA



DISIM  
Dipartimento di Ingegneria  
e Scienze dell'Informazione  
e Matematica



# NuSMV Input Language

Without processes, is it equivalent?

```
MODULE inverter(input)
VAR
  output : boolean;
ASSIGN
  init(output) := 0;
  next(output) := !input union output;
                -- or {!input, output}
```

```
MODULE main
VAR
  gate1 : inverter(gate3.output);
  gate2 : inverter(gate1.output);
  gate3 : inverter(gate2.output);
```



UNIVERSITÀ  
DEGLI STUDI  
DELL'AQUILA



DISIM  
Dipartimento di Ingegneria  
e Scienze dell'Informazione  
e Matematica

# NuSMV As A Tool

- NuSMV is provided with an interactive shell, as there are many tasks it may accomplish (simulation, many verification options); see user manual from chapter 3, especially Figure 3.1 at page 87
- Differently from explicit model checkers, no need to give separate commands to generate a file to be compiled and executed: all is represented as OBDDs, you only have to use them properly
- Executing a non-interactive verification in NuSMV is the same as giving the following list of interactive commands
- 1. `read_model` reads and stores the syntactic structure of the input model
  - no OBDDs here: tree-like structure, but representing the syntactic structure of the input (abstract syntax tree)



UNIVERSITÀ  
DEGLI STUDI  
DI CROTONE



DISIM  
Dipartimento di Ingegneria  
e Scienze dell'Informazione  
e Matematica

# NuSMV As A Tool

- 2. `flatten_hierarchy` (recursively) brings inside `main` all modules instantiated by `main`
  - very similar to the unfolding we mentioned for Murphi and SPIN: for such explicit model checkers, this was only needed for theoretical purposes, in order to define the Kripke structure of an input model
  - here, it must be actually performed in the source code of NuSMV, in order to then be able to encode  $R$  and  $I$  as OBDDs
  - to this aim, there must be only one module, the `main`, containing all variables coming from the modules it instantiates (to be applied recursively)
  - note that, again, this resembles digital circuits, where such a flattening is a natural operation
  - this could entail adding a scheduler module if processes are used



UNIVERSITÀ  
DEGLI STUDI  
DELL'AQUILA



DISIM  
Dipartimento di Ingegneria  
e Scienze dell'Informazione  
e Matematica

# NuSMV As A Tool

- 3. `encode_variables` for each variable  $x$  with domain  $D$  s.t.  $|D| > 2$ , NuSMV defines  $x_1 \dots, x_m$  *boolean* variables with  $m = \lfloor \log_2 |D| \rfloor + 1$ ; it also defines the encoding for constants used in the input models
- 4. `build_flat_model` combines the result of the preceding operations to obtain the flattened and booleanized syntactic structure which represents the Kripke structure defined by the input model
- 5. `build_model` from the syntactic structure to OBDDs for  $R$  and  $I$  (plus other ones)
- 6. `check_ctl`spec (or `check_ltl`spec, or both, depending on what you have to verify); it starts the actual verification
  - we will be back soon on these last 2 steps



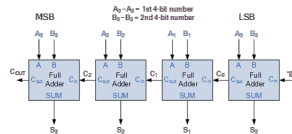
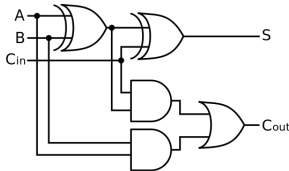
UNIVERSITÀ  
DEGLI STUDI  
DELL'AQUILA



DISIM  
Dipartimento di Ingegneria  
e Scienze dell'Informazione  
e Matematica

# From Syntactic Structure to OBDDs

- How does `build_model` work?
- All operations must be implemented bitwise (*bit-vector*)
  - this means that we have to build the corresponding digital circuit, remember the Digital Systems Design course?
  - if we have to implement a sum between two variables encoded with maximum 4 bits (note that result is on 5 bits):



UNIVERSITÀ  
DEGLI STUDI  
DELL'AQUILA



DISIM  
Dipartimento di Ingegneria  
e Scienze dell'Informazione  
e Matematica

# From Syntactic Structure to OBDDs

- Analogously, you can represent other arithmetic operations (subtract, multiply, divide)
- With other simple digital circuits, also equality and ordering can be easily implemented
  - e.g.,  $\text{next}(a) = b + c$  is translated in this way:
    - multiple OBDDs are used to sum all bits of  $b$  and  $c$
    - an OBDD  $B$  is created which is true iff all variables of  $\text{next}(a)$  are equal to such OBDDs
  - e.g.,  $\text{next}(a) = \text{case } a < b: b + c; \text{TRUE} : a$  is translated in this way:
    - again we have  $B$  as before, plus an OBDD  $C$  which is true if  $a < b$
    - then, NuSMV computes the OBDD  $\text{ITE}(C, B, a)$



UNIVERSITÀ  
DEGLI STUDI  
DELL'AQUILA



DISIM  
Dipartimento di Ingegneria  
e Scienze dell'Informazione  
e Matematica

# From a NuSMV Description to KS

- From a NuSMV model  $\mathcal{M}$  (defined with the ASSIGN section) to the corresponding Kripke structure  $\mathcal{S} = (S, I, R, L)$ 
  - $V = \langle v_1, \dots, v_n \rangle$  is the set of variables defined inside the `main` module of  $\mathcal{M}$ , with domains  $\langle D_1, \dots, D_n \rangle$ 
    - note that each  $D_i$  may be the instantiation of other modules
    - in which case, again, all variables must be considered as *unfolded*
    - that is, if a variable  $v$  is the instantiation of a module with  $k$  variables, then  $v$  counts as  $k$  variables instead of one
    - if one of such  $k$  variables is another instantiation, this procedure must be recursively repeated
    - NuSMV calls this operation *hierarchy flattening*
    - essentially, it is the same as for records in Murphi
    - simple types are the recursion base step



# From a NuSMV Description to KS

- $S = D_1 \times \dots \times D_n$  (as in Murphi)
- $I$  is defined by looking at `init` predicates
  - $s \in I$  iff, for all variables  $v \in V$ ,  $s(v) \in \text{init}(v)$ 
    - note that, by NuSMV syntax, each  $\text{init}(v)$  is actually a set (possibly a singleton)
  - if  $\text{init}(v)$  is not specified in  $\mathcal{M}$ , then any value for  $v$  is ok: in this case, formally, if  $s \in I$ , then also  $s' \in I$  being  $s'(v') = s(v') \forall v' \neq v$



UNIVERSITÀ  
DEGLI STUDI  
DELL'AQUILA



DISIM  
Dipartimento di Ingegneria  
e Scienze dell'Informazione  
e Matematica



# From a NuSMV Description to KS

- $R$  is defined by looking at `next` predicates
  - we assume all `next` predicates to be defined by the case construct (if not, simply assume it is the case construct with just one TRUE condition)
  - for each (flattened) variable  $v$ , we name  $g_1(v), \dots, g_{k_v}(v)$  the conditions (guards) of the case for `next( $v$ )`, and  $a_1(v), \dots, a_{k_v}(v)$  the resulting values (actions) of the case for `next( $v$ )`
  - note that, by NuSMV syntax, each  $a_i(v)$  is actually a set (possibly a singleton)
  - $(s, s') \in R$  iff, for all variables  $v \in V$ , if  $g_i(s(v)) \wedge \forall j < i \neg g_j(s(v))$  then  $s'(v) \in a_i(v)$
  - that is,  $s$  may go in  $s'$  iff, for all variables  $v$ , if the values of  $v$  in  $s$  satisfy the guard  $g_i$  (and none of the preceding guards for the same variable), then the value of  $v$  in  $s'$  is one of the values specified by the case for guard  $g_i$
  - note that, in doing this, you also have to resolve inputs for modules



# From a NuSMV Description to KS

- $AP = \{(v = d) \mid v = v_i \in V \wedge d \in D_i\}$
- $(v = d) \in L(s)$  iff variable  $v$  has value  $d$  in  $s$
- If, instead, the NuSMV model  $\mathcal{M}$  is defined with the TRANS section, then
  - $V = \langle v_1, \dots, v_n \rangle$  is the set of variables as above and  $S = D_1 \times \dots \times D_n$
  - $I$  is defined by looking at INIT section
    - $s \in I$  iff, for all variables  $v \in V$  and for all INIT sections  $\mathcal{I}$ ,  $\mathcal{I}(s(v))$  holds
  - $R$  is defined by looking at TRANS section
    - $(s, s') \in R$  iff, for all variables  $v \in V$  and TRANS sections  $T$ ,  $T(s(v), s'(v))$  holds



UNIVERSITÀ  
DEGLI STUDI  
DELL'AQUILA



DISIM  
Dipartimento di Ingegneria  
e Scienze dell'Informazione  
e Matematica

# $\lambda$ -calculus: Representing Functions

- In a nutshell: using  $f(x)$  has some drawbacks
  - you are forced to name a function ( $f$  in the example above)
  - it is not always clear if a letter is a parameter or an argument
  - it is not computationally clear what happens for multiple inputs
    - $f(x, y)$ : do you have to provide both  $x, y$ , otherwise you get an error?
    - as an alternative, you may provide just one argument, and obtain a new function
    - e.g.  $f(x, y) = x + y$ , we have that  $f(x, 4)$  is a function on  $x$



UNIVERSITÀ  
DEGLI STUDI  
DELL'AQUILA



DISIM  
Dipartimento di Ingegneria  
e Scienze dell'Informazione  
e Matematica

# $\lambda$ -calculus: Representing Functions

- Instead of writing  $f(x) = E(x)$ , for some expression  $E(x)$ , we write  $\lambda x.E(x)$ 
  - if you *want*, you can name a function  $f(x) = \lambda x.E(x)$
- $\lambda(x, y).x + y$ : both argument must be given, otherwise it is an error
- $\lambda x \lambda y.x + y$ : if you provide  $x = 4$  only, you get a function  $\lambda y.4 + y$
- If an OBDD contains variables  $x_1, \dots, x_n$ , then it represent some function  $\lambda x_1 \dots \lambda x_n. E(x_1, \dots, x_n)$



UNIVERSITÀ  
DEGLI STUDI  
DELL'AQUILA



DISIM  
Dipartimento di Ingegneria  
e Scienze dell'Informazione  
e Matematica

# $\mu$ -calculus: Fixpoints

- In a nutshell: we have a set  $L$  with an ordering  $\leq$ 
  - $\leq$  could be partial, i.e., not defined on some pair  $(l_1, l_2) \in L \times L$
  - $L, \leq$  is a *complete lattice* if any subset  $A \subseteq L$  has a greatest lower bound and a least upper bound in  $L$
  - $\sup A = \min\{x \in L \mid \forall a \in A. a \leq x\}$
  - $\inf A = \max\{x \in L \mid \forall a \in A. x \leq a\}$



UNIVERSITÀ  
DEGLI STUDI  
DELL'AQUILA



DISIM  
Dipartimento di Ingegneria  
e Scienze dell'Informazione  
e Matematica

# $\mu$ -calculus: Fixpoints

- Let  $I = \{0, \dots, 10\}$ , then  $2^I, \subseteq$  is a complete lattice
  - e.g.,  $\{0, 1, 2\} \leq \{0, 1, 2, 3\}$ , whilst  $\{0, 1, 2\}, \{0, 1, 3\}$  cannot be compared
  - $\sup\{\{0, 1, 2\}, \{0, 1, 3\}\} = \min\{x \in 2^I \mid \forall a \in \{0, 1, 2\}, \{0, 1, 3\}.a \subseteq x\} = \min\{\{0, 1, 2, 3\}, \dots, I\} = \{0, 1, 2, 3\}$
  - $\inf\{\{0, 1, 2\}, \{0, 1, 3\}\} = \max\{x \in 2^I \mid \forall a \in \{0, 1, 2\}, \{0, 1, 3\}.x \subseteq a\} = \max\{\{0, 1\}, \dots, \emptyset\} = \{0, 1\}$
- $2^I, \subseteq$  is always a complete lattice, if  $I$  is a finite set
  - $\sup J = \bigcup_{x \in J} x$ ,  $\inf J = \bigcap_{x \in J} x$
  - at the worst,  $\sup J = I$  and  $\inf J = \emptyset$



# $\mu$ -calculus: Fixpoints

- Suppose you have a function  $T : L \rightarrow L$ . An element  $x \in L$  is a *fixpoint of  $T$*  iff  $T(x) = x$
- Given a  $T$ , there may be several fixpoints: we are interested in the maximum or the minimum of such fixpoints
  - notation  $\mu T$  and  $\nu T$
  - where typically  $T$  is expressed with a  $\lambda$  notation
  - $\mu T \equiv x$  s.t.  $T(x) = x \wedge \forall y \in L. T(y) = y \rightarrow x \leq y$
  - $\nu T \equiv x$  s.t.  $T(x) = x \wedge \forall y \in L. T(y) = y \rightarrow y \leq x$



UNIVERSITÀ  
DEGLI STUDI  
DELL'AQUILA



DISIM  
Dipartimento di Ingegneria  
e Scienze dell'Informazione  
e Matematica

# $\mu$ -calculus: Fixpoints

- Let  $T : 2^I \rightarrow 2^I$  be defined as  $T(x) = x$ , or better  $T \equiv \lambda x.x$ 
  - we have  $\mu T = \emptyset, \nu T = I$
- Let  $T \equiv \lambda x.\emptyset$ 
  - we have  $\mu T = \nu T = \emptyset$
- Let  $T \equiv \lambda x.x \cup \{10\}$ 
  - we have  $\mu T = \{10\}, \nu T = I$
- Let  $T \equiv \lambda x.x \setminus \{10\}$ 
  - we have  $\nu T = \{0, \dots, 9\}, \mu T = \emptyset$



UNIVERSITÀ  
DEGLI STUDI  
DELL'AQUILA



DISIM  
Dipartimento di Ingegneria  
e Scienze dell'Informazione  
e Matematica



# $\mu$ -calculus: Fixpoints

- We define sets by their characteristic function, thus let us rewrite the previous examples
  - thus the  $x$  in  $\lambda x$  is a function  $x : I \rightarrow \{0, 1\}$
  - it represents a set  $X$ , thus  $x(y) = 1$  iff  $y \in X$
- $T \equiv \lambda x. x$  is ok also if  $x$  is a characteristic function
- $T \equiv \lambda x. \emptyset$  could be rewritten as  $T \equiv \lambda x. \lambda y. 0$
- $T \equiv \lambda x. x \cup \{10\}$  could be rewritten as
$$T \equiv \lambda x. \lambda y. [y = 10 \rightarrow 1] \wedge [y \neq 10 \rightarrow x(y)]$$
- $T \equiv \lambda x. x \setminus \{10\}$  could be rewritten as
$$T \equiv \lambda x. \lambda y. [y = 10 \rightarrow 0] \wedge [y \neq 10 \rightarrow x(y)]$$
- In this last example, we have
$$\nu T. \lambda x. \lambda y. [y = 10 \rightarrow 0] \wedge [y \neq 10 \rightarrow x(y)] \equiv \lambda y. [y \neq 10]$$



# $\mu$ -calculus: Fixpoints

- We deal with monotonic (i.e., increasing or decreasing)  $T$ , thus fixpoints always exists
  - $x \leq y \rightarrow T(x) \leq T(y)$ ,  $T$  monotonically increasing
  - $x \leq y \rightarrow T(y) \leq T(x)$ ,  $T$  monotonically decreasing
- Previous examples are all monotonic
- By Knaster-Tarski theorem,  $\mu T = \inf\{x \mid T(x) \leq x\}$ 
  - analogously,  $\nu T = \sup\{x \mid T(x) \geq x\}$



UNIVERSITÀ  
DEGLI STUDI  
DELL'AQUILA



DISIM  
Dipartimento di Ingegneria  
e Scienze dell'Informazione  
e Matematica

# Computation of Fixpoints in CTL Model Checking

- Given a KS  $\mathcal{S} = (S, I, R, L)$ , we want to label states, thus  $L = 2^S$  and  $\leq$  is  $\subseteq$ 
  - in the following,  $x = x_1, \dots, x_n$  with  $n = \lceil \log |S| \rceil$
  - characteristic functions of subsets of  $S$
  - thus,  $T : 2^S \rightarrow 2^S$
- For  $k \geq 1$ , let  $T^k(x) = T(T^{k-1}(x))$ , with  $T^1 = T$
- For least fixpoints, start with  $\emptyset$ , and apply  $T$  since  $T^k(\emptyset) = T^{k-1}(\emptyset)$
- For greatest fixpoints, start with  $S$ , and apply  $T$  since  $T^k(S) = T^{k-1}(S)$
- At most,  $k = |S|$



UNIVERSITÀ  
DEGLI STUDI  
DELL'AQUILA



DISIM  
Dipartimento di Ingegneria  
e Scienze dell'Informazione  
e Matematica

# $\mu$ -calculus: Fixpoints

- The “really interesting” fixpoints are those which are recursively defined
  - typically, basing on some other already defined sets, i.e., characteristic functions
  - e.g.,  $T \equiv \lambda x. f(x) \vee T(x)$ , where  $f : S \rightarrow \{0, 1\}$  is known
  - thus, the compactly-written least and greatest fixpoints are  $\mu T. \lambda x. f(x) \vee T(x)$  and  $\nu T. \lambda x. f(x) \vee T(x)$
  - e.g.,  $T \equiv \lambda x. f(x) \wedge T(x)$
  - e.g.,  $T \equiv \lambda x. T(x)$
- Not immediately clear what they do, but by the Knaster-Tarski theorem and the previous reasoning, we may apply the following algorithms
  - least fixpoints  $\mu$  are computed for increasing  $T$
  - greatest fixpoints  $\nu$  are computed for decreasing  $T$
  - viceversa are trivial:  $\mu T$  is  $\lambda x. 0$  for decreasing  $T$  and  $\nu T$  is  $\lambda x. 1$  for increasing  $T$



UNIVERSITÀ  
DEGLI STUDI  
DELL'AQUILA



DISIM  
Dipartimento di Ingegneria  
e Scienze dell'Informazione  
e Matematica

# Computation of Least (Minimum) Fixpoint

```
OBDD lfp(MuFormula T) /*  $\mu Z.T(Z)$  */  
{  
  Q =  $\lambda x.0$ ;  
  Q' = T(Q);  
  /* T clearly says where Q must be replaced */  
  /* e.g.: if  $\mu Z.\lambda x.f(x) \vee Z(x)$ , then  
    Q' =  $\lambda x.f(x) \wedge Q(x)$  */  
  while (Q  $\neq$  Q') {  
    Q = Q';  
    Q' = T(Q);  
  }  
  return Q; /* or Q', they are the same... */  
}
```



UNIVERSITÀ  
DEGLI STUDI  
DELL'AQUILA



DISIM  
Dipartimento di Ingegneria  
e Scienze dell'Informazione  
e Matematica

# Computation of Greatest (Maximum) Fixpoint

```
OBDD gfp(NuFormula T) /*  $\nu Z.T(Z)$  */  
{  
  Q =  $\lambda x. 1$ ;  
  Q' = T(Q);  
  while (Q  $\neq$  Q') {  
    Q = Q';  
    Q' = T(Q);  
  }  
  return Q;  
}
```



UNIVERSITÀ  
DEGLI STUDI  
DELL'AQUILA



DISIM  
Dipartimento di Ingegneria  
e Scienze dell'Informazione  
e Matematica

# Symbolic Model Checking of $\mathbf{AG}p$

- The idea is to compute the set of reachable states, and check if for all of them  $p$  holds
- $\text{Reach} = \mu Z. \lambda x. [I(x) \vee \exists y : (Z(y) \wedge R(y, x))]$ 
  - of course, we get an OBDD on  $x$  as a result
  - recall that  $x$  (and  $y$ ) is a vector of all boolean variables
- $\forall x \in S. \text{Reach}(x) \rightarrow p(x)$ 
  - computationally easier: check that  $\text{Reach}(x) \wedge \neg p(x) = 0$
  - otherwise, we have a reachable state for which  $p$  does not hold...



UNIVERSITÀ  
DEGLI STUDI  
DELL'AQUILA



DISIM  
Dipartimento di Ingegneria  
e Scienze dell'Informazione  
e Matematica

# Symbolic CTL Model Checking

- All CTL formulas can be reduced to 3: **EXf**, **f EU g**, **EGf**
  - all other formulas may be reduced to these three, using negation and other boolean combinations
  - with OBDDs, we can do all such things!
- Given OBDDs for  $f$  (and  $g$ ), we compute the OBDD representing **EXf**, **f EU g**, **EGf**
  - that is, the OBDD for the set  $X = \{s \in S \mid \mathcal{S}, s \models \mathbf{EX}f\}$  etc
- Let it be  $B$ : then, simply check  $\neg B(x) \wedge I(x) = 0$ 
  - recall that  $\mathcal{S} \models \Phi$  iff  $\forall s \in I. \mathcal{S}, s \models \Phi$
- **EXf** does not require a fixpoint computation: it is equivalent to (the OBDD representing)  $\lambda x. \exists y : R(x, y) \wedge f(y)$





# Symbolic CTL Model Checking

- For  $f \text{ EU } g$ , recall that it is equivalent to the CTL formula  $g \vee (f \wedge \text{EX}(f \text{ EU } g))$
- Thus,  $f \text{ EU } g = \mu Z. \lambda x. g(x) \vee (f(x) \wedge \text{EX}Z(x)) = \mu Z. \lambda x. g(x) \vee (f(x) \wedge (\exists y : R(x, y) \wedge Z(y)))$ 
  - note that  $g(x) \vee (f(x) \wedge \text{EX}Z(x))$  is increasing, i.e. for  $Z_1 \subseteq Z_2$  we have that  $(g(x) \vee (f(x) \wedge \text{EX}Z_1(x))) \rightarrow (g(x) \vee (f(x) \wedge \text{EX}Z_2(x)))$
- Analogously:  $\text{EG}f = f \wedge \text{EX}(\text{EG}f)$ , thus  $\text{EG}f = \nu Z. \lambda x. f(x) \wedge \text{EX}Z(x) = \nu Z. \lambda x. f(x) \wedge (\exists y : R(x, y) \wedge Z(y))$ 
  - note that  $f(x) \wedge \text{EX}Z(x)$  is decreasing, i.e. for  $Z_1 \subseteq Z_2$  we have that  $(f(x) \wedge \text{EX}Z_2(x)) \rightarrow (f(x) \wedge \text{EX}Z_1(x))$



# CTL Model Checking

```
bool checkCTL(KS S, CTL  $\varphi$ ) {  
    let  $S = \langle S, I, R, L \rangle$ ;  
     $B = \text{Lb1St}(\varphi)$ ;  
    return  $\lambda x. I(x) \wedge \neg B(x) = \lambda x. 0$ ;  
}  
  
OBDD Lb1St(CTL  $\varphi$ ) { /* also  $S = \langle S, I, R, L \rangle$  */  
    if ( $\exists p \in AP. \varphi = p$ ) return  $\lambda x. p(x)$ ;  
    else if ( $\varphi = \neg\phi$ ) return  $\lambda x. \neg \text{Lb1St}(\phi)(x)$ ;  
    else if ( $\varphi = \phi_1 \wedge \phi_2$ )  
        return  $\lambda x. \text{Lb1St}(\phi_1)(x) \wedge \text{Lb1St}(\phi_2)(x)$ ;  
    else if ( $\varphi = \mathbf{EX}\phi$ )  
        return  $\lambda x. \exists y : R(x, y) \wedge \text{Lb1St}(\phi)(y)$ ;  
    else if ( $\varphi = \mathbf{EG}\phi$ )  
        return  $\text{gfp}(\nu Z. \lambda x. \text{Lb1St}(\phi)(x) \wedge (\exists y : R(x, y) \wedge Z(y)))$ ;  
    else if ( $\varphi = \phi_1 \mathbf{EU} \phi_2$ )  
        return  $\text{lfp}(\mu Z. \lambda x. \text{Lb1St}(\phi_2)(x) \vee$   
             $(\text{Lb1St}(\phi_1)(x) \wedge (\exists y : R(x, y) \wedge Z(y))))$ ;  
}
```

