

# Explicit Model Checking Binary Decision Diagrams

Julien Schmaltz

Institute for Computing and Information Sciences  
Radboud University Nijmegen  
The Netherlands  
julien@cs.ru.nl

April 6, 2009

## Agenda coming lectures ...

- ✓ Part I: Linear Time
- ✓ Part II: Branching Time
- ✓ Part III: Comparison
- Part IV: Binary Decision Diagrams and Symbolic Model Checking
- Part V: The SAL tool
- Part VI: Bounded Model Checking with SAT
- Part VII:  $k$ -induction, SMT, and disjunctive invariants

# Agenda for today

- Part IV: Binary Decision Diagrams and Symbolic Model Checking

More precisely:

- LTL and CTL explicit model checking
- BDDs and symbolic model checking

## Part I

# Explicit Model Checking

- 1 Introduction
- 2 Explicit Model Checking
- 3 Automata-Based LTL Model Checking
  - Principles
  - From LTL to NBA
  - Summary
- 4 Explicit CTL Model Checking
  - Basic formulae
  - Check EU
  - Check EG

# Bug hunting

"The rare glitch project" (Clarke)

- Main goal of model checking is to find bugs !
- Automatic technique
- Ability to generate counter-examples
- The main issue is the infamous state-explosion problems

# The Model Checking Problem

- Let  $M$  be a state-transition graph representing some computing system
- Let  $S$  be its specification express as formulae in some temporal logic
- MC = Does  $M$  satisfy  $S$  ?
  - Yes !
  - No ! and here is a counter-example ...
- Based on the traversal of the state-space
  - Explicit
  - Symbolic

# Explicit Model Checking: Agenda

- Automata-based LTL model checking
- CTL model checking

# LTL Model-Checking Problem

- Let  $TS$  be a transition system
- Let  $\varphi$  be an LTL-formula

Basic idea:

try to *disprove*  $TS \models \varphi$  by *looking for a path*  $\pi$  s.t.  
 $\pi \not\models \varphi$

- If no such path exists, then  $TS \models \varphi$
- If such a path exists, then we know that  $TS \not\models \varphi$  and this path is a counter-example !

Basic algorithm

- Visit all states
- Terminate because finite state-space

# Remember LTL

model logic over infinite sequences introduced in CS by Pnueli in  
1977

- Propositional logic
  - Atomic propositions:  $a \in AP$
  - Boolean connectives:  $\neg a$  and  $\varphi \wedge \psi$
- Temporal operators
  - "Next" noted  $X\varphi$  or  $\bigcirc\varphi$
  - "Until" noted  $\varphi U\psi$  or  $\varphi \cup \psi$
- Derived operators
  - $F\varphi$  (also noted  $\diamond\varphi$ )  $\equiv \top U\varphi$  "eventually  $\varphi$ "
  - $G\varphi$  (also noted  $\square\varphi$ )  $\equiv \neg F\neg\varphi$  "globally  $\varphi$ "

# Towards model-checking ...

Last week we wrote:

$$TS \models \varphi \quad \text{iff} \quad \text{Traces}(TS) \subseteq \text{Words}(\varphi)$$

This can be rewritten as:

$$TS \models \varphi \quad \text{iff} \quad \text{Traces}(TS) \cap ((2^{AP})^\omega \setminus \text{Words}(\varphi)) = \emptyset$$

By definition, this gives the following:

$$TS \models \varphi \quad \text{iff} \quad \text{Traces}(TS) \cap \text{Words}(\neg\varphi) = \emptyset$$

## Model-checking LTL formulae

$$TS \models \varphi \quad \text{iff} \quad \text{Traces}(TS) \cap \text{Words}(\neg\varphi) = \emptyset$$

To prove  $TS \models \varphi$ , do the following:

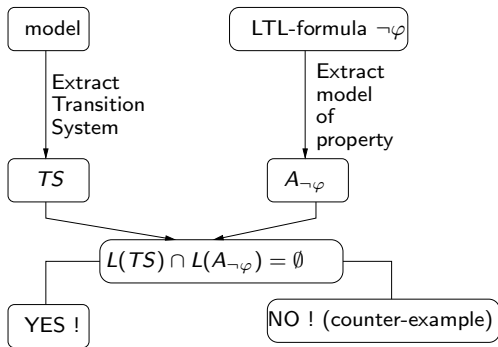
- Construct a representation of  $\neg\varphi$
- Check whether  $TS$  contains words of  $\neg\varphi$
- This is **not the case**, then answer "Yes" !
- If this **is the case**, then answer "No" !
  - So, you just found a path  $\pi$  such that  $\pi \not\models \varphi$
  - Extract a finite prefix of this path
  - Output this counter-example !

# The Automata-Based Approach

[Vardi and Wolper 1986]

- LTL formulae
- Büchi automata recognize  $\omega$ -regular languages
- Possible to encode any LTL-formula as a (non-deterministic) Büchi automata
- Algorithm
  - Construct automaton  $A_{\neg\varphi}$  for negation of LTL-formula  $\varphi$
  - Make the product of  $A_{\neg\varphi}$  with  $TS$
  - Is the language accepted by this product empty ?
  - If yes, then the product does not contain any word of  $Words(\neg\varphi)$ , answer **Yes !**
  - Otherwise, answer **No !** and output finite prefix of word as **counter-example**

# Model-Checking Overview



# Non-Deterministic Büchi Automata

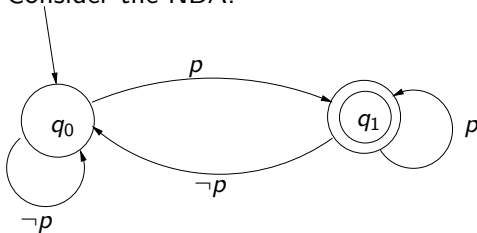
Automata with **accept states**. It is defined as a tuple  $(Q, \Sigma, \delta, Q_0, F)$ , where

- $Q$  is a finite set of states
- $\Sigma$  is a (finite) alphabet
- $\delta : Q \times \Sigma \rightarrow 2^Q$  is a (finite) transition relation
- $Q_0 \subseteq Q$  is a (finite) set of initial states
- $F \subseteq Q$  is a set of **accept** (or final) states, called the **acceptance set**

A run is **accepting** if it visits an accept state **infinitely often**.

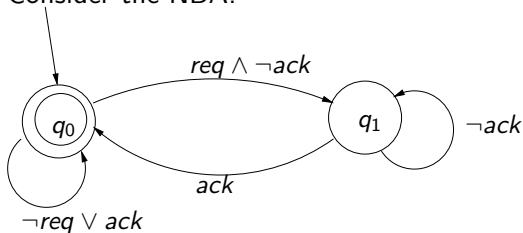
## NBA and LTL: "infinitely often"

- Consider the formula **GF** $p$  "infinitely often  $p$ "
- Consider the NBA:



## NBA and LTL: Request Acknowledge

- Consider the formula  $\mathbf{G}(req \Rightarrow \mathbf{F}ack)$  "all requests must be acknowledged"
- Consider the NBA:



# LTL Model Checking

- LTL is a logic for formalizing **path**-based properties
- LTL-formula  $\varphi$  can be transformed algorithmically into NBA  $A_\varphi$ 
  - Possible exponential blow-up
- Satisfiability and validity of LTL amounts to NBA emptiness-check
- Complexity: **PSPACE**-complete

- 1 Introduction
- 2 Explicit Model Checking
- 3 Automata-Based LTL Model Checking
  - Principles
  - From LTL to NBA
  - Summary
- 4 Explicit CTL Model Checking
  - Basic formulae
  - Check EU
  - Check EG

# Computational Tree Logic (CTL)

modal logic over infinite **trees** [Clarke & Emerson 1981]

- State formulae containing path quantifiers
  - atomic proposition:  $a \in AP$
  - Boolean connectives:  $\neg\varphi$  and  $\varphi \wedge \psi$
  - there exists a path satisfying  $\varphi$ :  $\mathbf{E}\varphi$  or  $\exists\varphi$
  - all paths satisfy  $\varphi$ :  $\mathbf{A}\varphi$  or  $\forall\varphi$
- Paths formulae containing temporal operators
  - Next  $\varphi$ :  $\mathbf{X}\varphi$  or  $\bigcirc\varphi$
  - $\varphi$  until  $\psi$ :  $\varphi\mathbf{U}\psi$
- In a CTL formula path and state formulae alternate

## Derived Operators

- Potentially  $\varphi$ :  $\mathbf{EF}\varphi = \mathbf{E}(\mathbf{TU}\varphi)$
- Inevitably  $\varphi$ :  $\mathbf{AF}\varphi = \mathbf{A}(\mathbf{TU}\varphi)$
- Potentially always  $\varphi$ :  $\mathbf{EG}\varphi = \neg\mathbf{AF}\neg\varphi$
- Invariantly  $\varphi$ :  $\mathbf{AG}\varphi = \neg\mathbf{EF}\neg\varphi$
- Weak until
  - $\mathbf{E}(\varphi\mathbf{W}\psi) = \neg\mathbf{A}((\varphi \wedge \neg\psi)\mathbf{U}(\neg\varphi \wedge \neg\psi))$
  - $\mathbf{A}(\varphi\mathbf{W}\psi) = \neg\mathbf{E}((\varphi \wedge \neg\psi)\mathbf{U}(\neg\varphi \wedge \neg\psi))$

# Weak-Until

- In LTL, we can define weak-until as follows:

$$\varphi \mathbf{W} \psi \triangleq \varphi \mathbf{U} \psi \vee \mathbf{G} \varphi$$

- We then have the following duality between **U** and **W**

- $\neg(\varphi \mathbf{U} \psi) = (\varphi \wedge \neg\psi) \mathbf{W} (\neg\varphi \wedge \neg\psi)$
- $\neg(\varphi \mathbf{W} \psi) = (\varphi \wedge \neg\psi) \mathbf{U} (\neg\varphi \wedge \neg\psi)$

- $\mathbf{E}(\varphi \mathbf{W} \psi) \triangleq \varphi \mathbf{U} \psi \vee \mathbf{G} \varphi$  is not in CTL

- Using these equivalences and the duality between universal and existential quantifications, we can define weak-until in CTL as follows:

- $\mathbf{E}(\varphi \mathbf{W} \psi) = \neg \mathbf{A}((\varphi \wedge \neg\psi) \mathbf{U} (\neg\varphi \wedge \neg\psi))$
- $\mathbf{A}(\varphi \mathbf{W} \psi) = \neg \mathbf{E}((\varphi \wedge \neg\psi) \mathbf{U} (\neg\varphi \wedge \neg\psi))$

# Semantics of **EW**

$$\mathbf{E}(\varphi \mathbf{W} \psi) = \neg \mathbf{A}((\varphi \wedge \neg \psi) \mathbf{U} (\neg \varphi \wedge \neg \psi))$$

So,  $s \models \mathbf{E}(\varphi \mathbf{W} \psi)$  if and only if there exists a path  $\pi$  from  $s$  such that:

$$\pi \not\models (\varphi \wedge \neg \psi) \mathbf{U} (\neg \varphi \wedge \neg \psi)$$

Such a path exists if and only if:

- either  $s_j \models \varphi \wedge \neg \psi$  for all  $j \geq 0$ , i.e.,  $\pi \models \mathbf{G}(\varphi \wedge \neg \psi)$  or

# Semantics of EW

- there exists an index  $j$  such that
  - $s_j \not\models \varphi \wedge \neg\psi$  and  $s_j \not\models \neg\varphi \wedge \neg\psi$ , i.e.,  $s_j \models \psi$  and
  - $s_i \models \varphi \wedge \neg\psi$  for all  $0 \leq i < j$

this is equivalent to  $\pi \models \varphi \mathbf{U} \psi$

At the end, we get that  $\pi \models \varphi \mathbf{W} \psi$

- if and only if  $\pi \models \varphi \mathbf{U} \psi$  or  $\pi \models \mathbf{G}(\varphi \wedge \neg\psi)$
- if and only if  $\pi \models \varphi \mathbf{U} \psi$  or  $\pi \models \mathbf{G}\varphi$

# Semantics of **AW**

"do het zelf" ...

# Operators

- Basic operators: **EX**, **EG**, **EU**
- Derived operators:
  - $\mathbf{AX}\varphi = \neg\mathbf{EX}(\neg\varphi)$
  - $\mathbf{EF}\varphi = \mathbf{E}(\mathbf{TU}\varphi)$
  - $\mathbf{AG}\varphi = \neg\mathbf{EF}(\neg\varphi)$
  - $\mathbf{AF}\varphi = \neg\mathbf{EG}(\neg\varphi)$
  - $\mathbf{A}(\varphi\mathbf{U}\psi) = \neg\mathbf{E}(\neg\psi\mathbf{U}(\neg\varphi \wedge \neg\psi)) \wedge \neg\mathbf{EG}\neg\psi$

# Existential Normal Form (ENF)

Any CTL formula can be expressed in terms of

- $\neg, \vee, \mathbf{EX}, \mathbf{EG}, \mathbf{EU}$

All other operators can be derived from these 5 ones

- $\mathbf{AX}p \equiv \neg\mathbf{EX}\neg p$
- $\mathbf{A}(p \mathbf{U} q) \equiv \neg\mathbf{E}(\neg q \mathbf{U} (\neg q \wedge \neg q)) \wedge \neg\mathbf{EG}\neg q$

Model checking algorithm for these 5 cases + atomic proposition

# Transition System Semantics

- For CTL-formula  $\Phi$ , the **satisfaction set**  $Sat(\Phi)$  is:

$$Sat(\Phi) = \{ s \in S \mid s \models \Phi \}$$

- $TS$  satisfies  $\Phi$  iff  $\Phi$  holds in all its initial states:

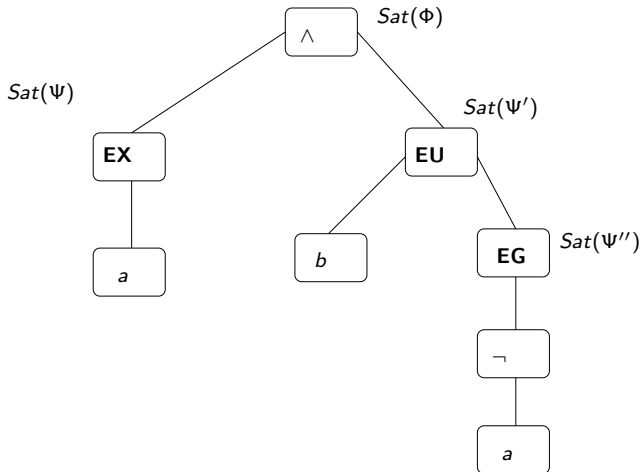
$$TS \models \Phi \text{ iff } \forall s_0 \in I : s_0 \models \Phi$$

- This is equivalent to  $I \subseteq Sat(\Phi)$

## Basic Model Checking algorithm

- Label all states of  $TS$  that satisfy  $\Phi$
- Check that all initial states have been labelled
- Checking CTL-formula: bottom-up labelling of states

# Example

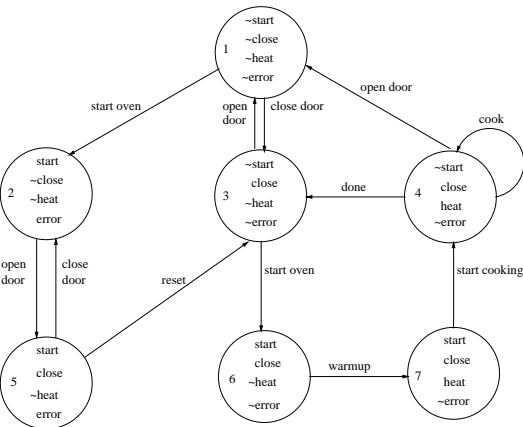


$$\Phi = \text{EX}a \wedge \text{E}(b \text{ U } \text{EG}\neg c)$$

## Checking basic formulae

- Checking  $\neg p$ 
  - Label states that are not labelled with  $p$
- Checking  $p \vee q$ 
  - Label states that are either labelled with  $p$  or with  $q$
- Checking **EX** $p$ 
  - Label each state that has some successor labelled with  $p$

# Checking basic formulae: Example



$Sat(\text{start}) = ?$   
 $Sat(\neg\text{heat}) = ?$   
 $Sat(\mathbf{EX}\text{heat}) = ?$   
 $Sat(\mathbf{EX}\text{start}) = ?$

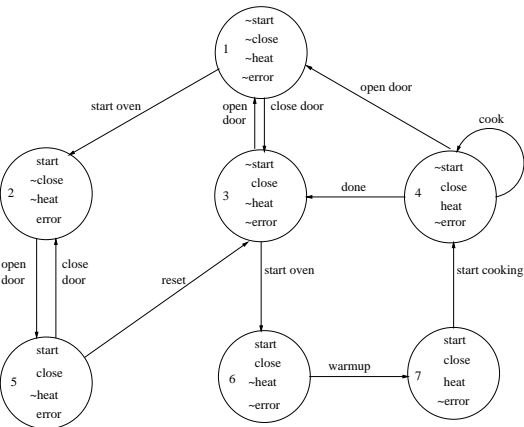
# Check EU

Let  $\phi = \mathbf{E}(f_1 \mathbf{U} f_2)$  and assume that  $f_1$  and  $f_2$  have already been treated.

- 1 Find all states that are labelled with  $f_2$ .
- 2 Go backward to find all states that can be reached by a path where  $f_1$  always holds

All such states satisfy  $\phi$

# Check EU: Example



$Sat(\mathbf{E}(start\mathbf{U}heat)) = ?$   
 $Sat(\mathbf{EF}(heat)) = ?$

## Check EG: Lemma

- Based on the decomposition of the graph into non-trivial **strongly connected components** (SCC's)
- An SCC  $C$  is
  - a **maximal** sub-graph such that
  - every node in  $C$  is reachable from every other node in  $C$  in path with nodes in  $C$
- A non-trivial SCC iff
  - Either it has more than one node
  - Or it contains one node with a self-loop

Let  $TS[\phi]$  be  $TS$  restricted to states that satisfy  $\phi$ . Then:

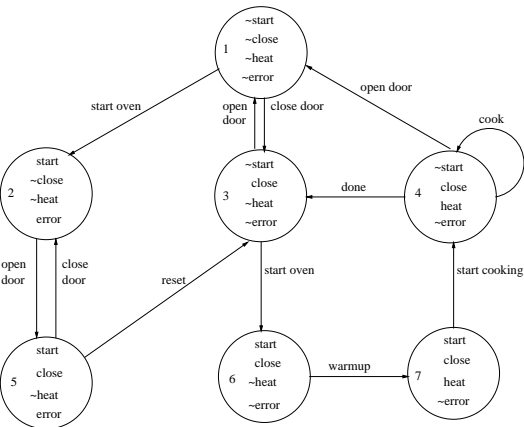
$s \models \mathbf{EG}\phi$  iff  $s \models \phi$  and there is a non-trivial SCC reachable from  $s$

## Procedure for Check EG

To check **EG** $\phi$

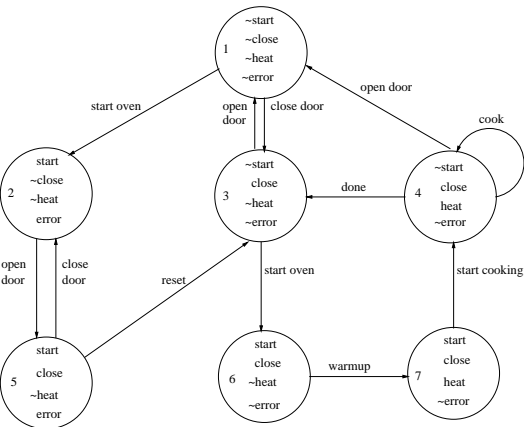
- 1 Construct  $TS[\phi]$
- 2 Find SCC's
- 3 Get states that belong to non-trivial SCC's
- 4 Proceed backward to find all states that can be reached by a path with all states labelled with  $\phi$

# Check EG: Example



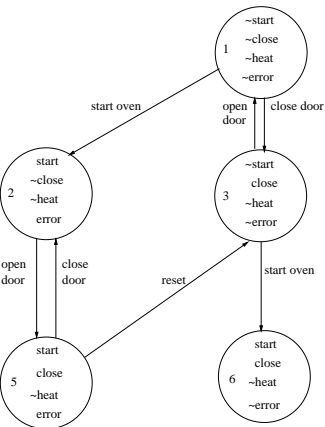
Let's check **EG**( $\neg$ heat)

# Check EG: Example



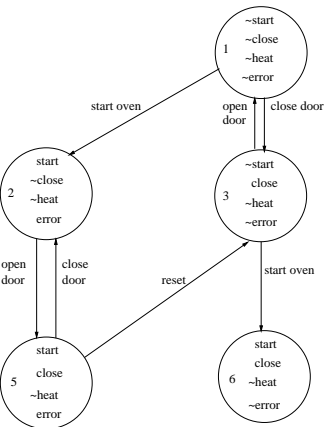
Let's check  $\mathbf{EG}(\neg\text{heat})$   
 $TS[\neg\text{heat}] = ?$

# Check EG: Example



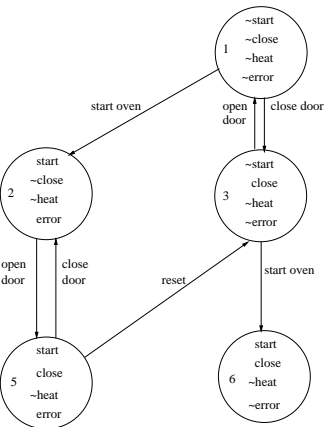
Let's check  $\mathbf{EG}(\neg\text{heat})$   
 $TS[\neg\text{heat}] = ?$

## Check EG: Example



Let's check  $\mathbf{EG}(\neg\text{heat})$   
 $TS[\neg\text{heat}] = ?$   
 $\text{SCC} =$

## Check EG: Example



Let's check  $\mathbf{EG}(\neg\text{heat})$

$TS[\neg\text{heat}] = ?$

$\text{SCC} =$

$\text{Sat}(\mathbf{EG}(\neg\text{heat})) = ?$

## Complete example

Check **AG**(*start*  $\Rightarrow$  **A F***heat*)

## Part II

# Symbolic Model Checking with BDDs

# Symbolic Model Checking

- No explicit enumeration of the states
- Symbolic representation of the state space
- Termination: fixpoint characterization of CTL operators
- Fixpoints detected as isomorphism between symbolic representations of sets of states

# (RO)BDD's (Reduced Ordered) Binary Decision Diagrams

[Bryant 1986]

- Canonical form representation for Boolean functions
- Substantially more compact than CNF or DNF
- Efficient manipulation of BDD's

# Shannon and Binary Decision Trees

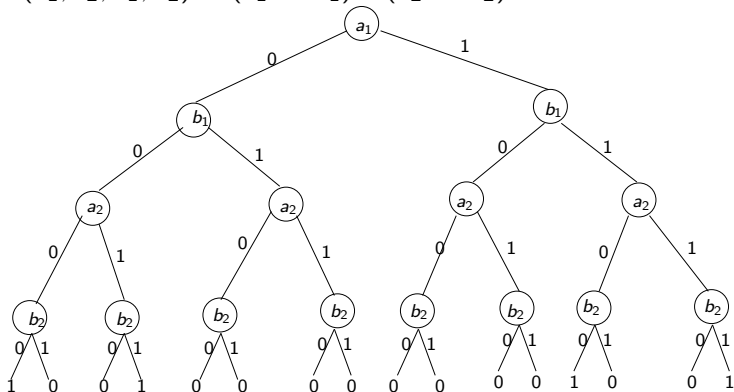
- Shannon expansion for Boolean function  $f$

$$f = (\neg a \wedge f|_{a=0}) \vee (a \wedge f|_{a=1})$$

- Using this expansion and a variable ordering, one can build a **binary decision tree**
- Binary Decision Trees are not very compact (same size as truth tables)

# Binary Decision Tree for a 2-bit comparator

$$f(a_1, a_2, b_1, b_2) = (a_1 \Leftrightarrow b_1) \wedge (a_2 \Leftrightarrow b_2)$$



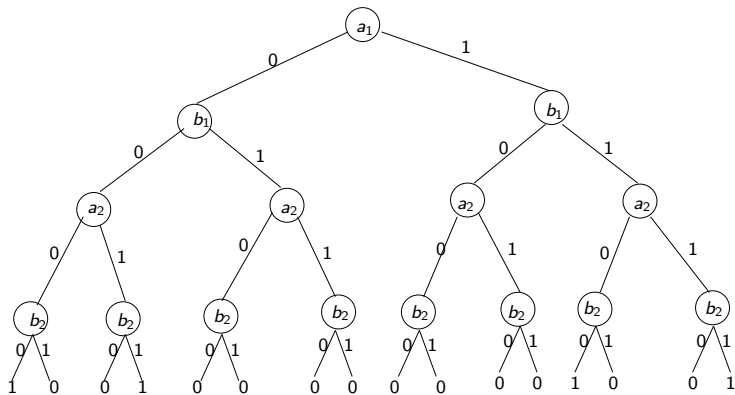
# Reduced Ordered Binary Decision Diagram (ROBDD)

[Bryant 1986]

- Trees are neither canonical nor really concise
- Reductions rules (Bryant's function **Reduce**)
  - Rule 1: **Remove duplicate terminal nodes**
  - Rule 2: **Remove duplicate non-terminals**
  - Rule 3: **Remove redundant tests**
- Total ordering on the variables
- Canonical form
  - Equivalence = isomorphism (constant time)
  - Satisfiability = checking equivalence with BDD with one '0'
- Efficient implementation of Boolean operations (Bryant's function **Apply**)

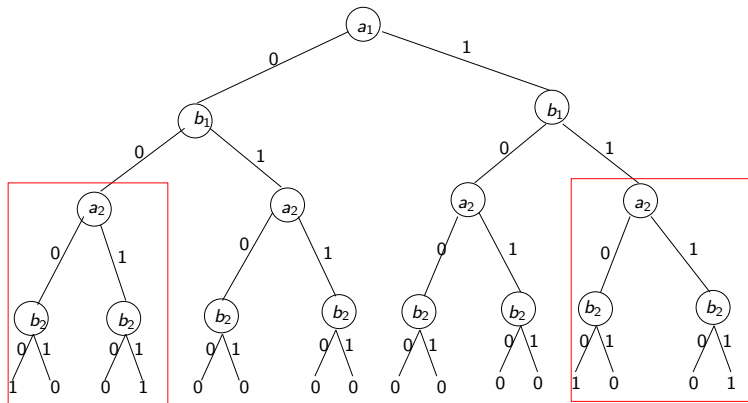
# ROBDD for 2-bit comparator with ordering

$$a_1 < b_1 < a_2 < b_2$$



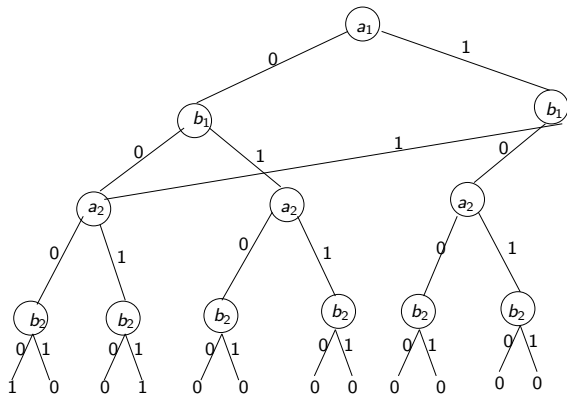
# ROBDD for 2-bit comparator with ordering

$$a_1 < b_1 < a_2 < b_2$$



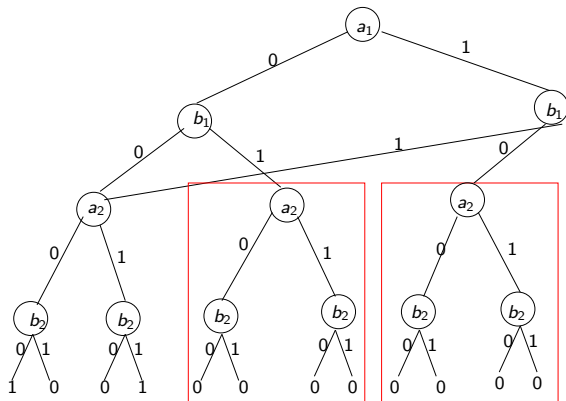
# ROBDD for 2-bit comparator with ordering

$$a_1 < b_1 < a_2 < b_2$$



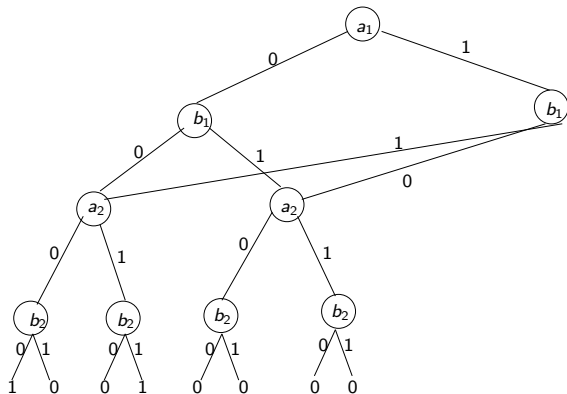
# ROBDD for 2-bit comparator with ordering

$$a_1 < b_1 < a_2 < b_2$$



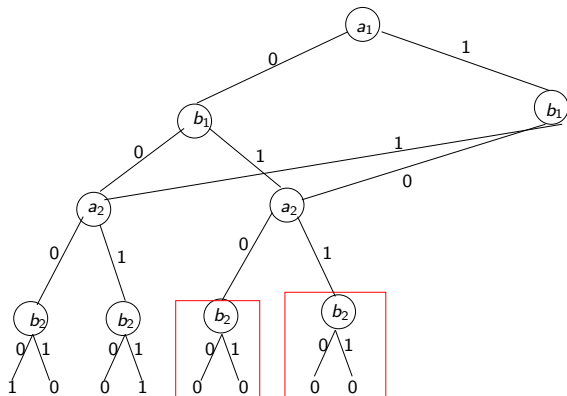
# ROBDD for 2-bit comparator with ordering

$$a_1 < b_1 < a_2 < b_2$$



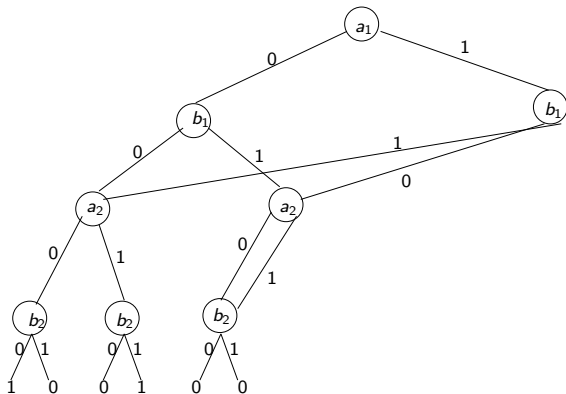
# ROBDD for 2-bit comparator with ordering

$$a_1 < b_1 < a_2 < b_2$$



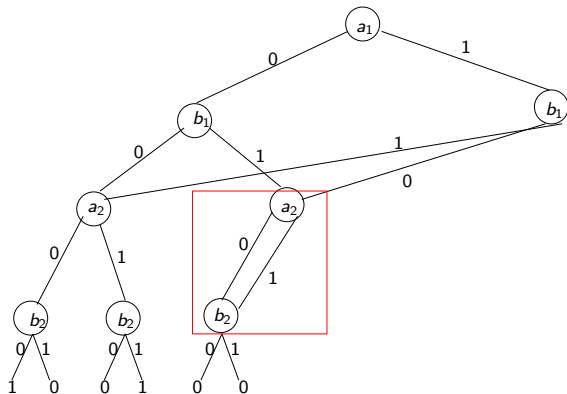
# ROBDD for 2-bit comparator with ordering

$$a_1 < b_1 < a_2 < b_2$$



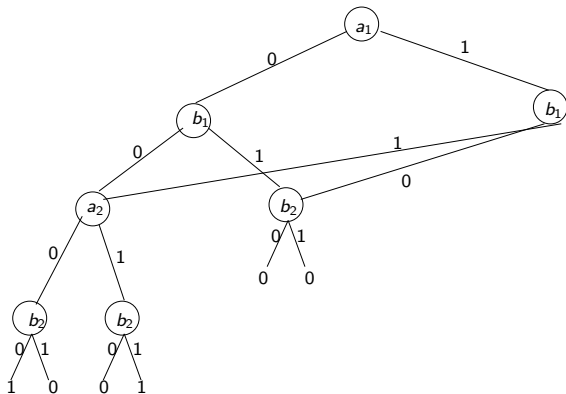
# ROBDD for 2-bit comparator with ordering

$$a_1 < b_1 < a_2 < b_2$$



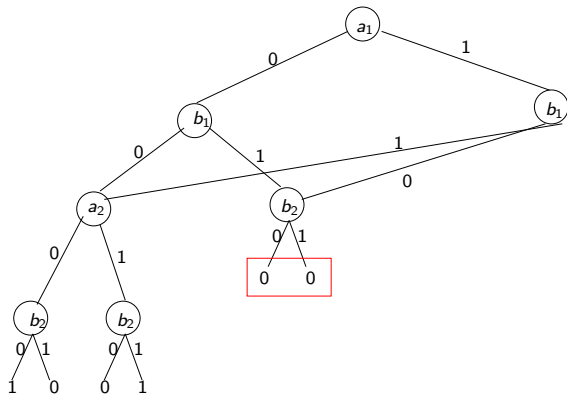
# ROBDD for 2-bit comparator with ordering

$$a_1 < b_1 < a_2 < b_2$$



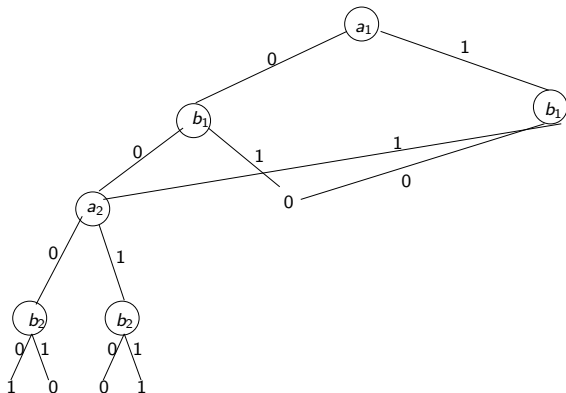
# ROBDD for 2-bit comparator with ordering

$$a_1 < b_1 < a_2 < b_2$$



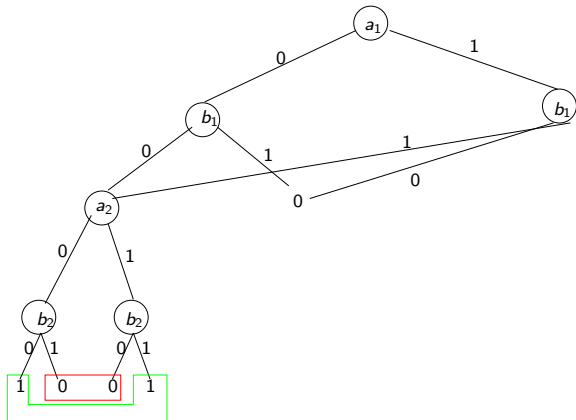
# ROBDD for 2-bit comparator with ordering

$$a_1 < b_1 < a_2 < b_2$$



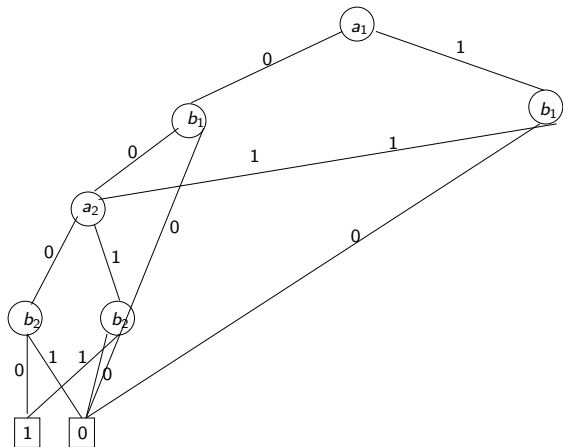
# ROBDD for 2-bit comparator with ordering

$$a_1 < b_1 < a_2 < b_2$$



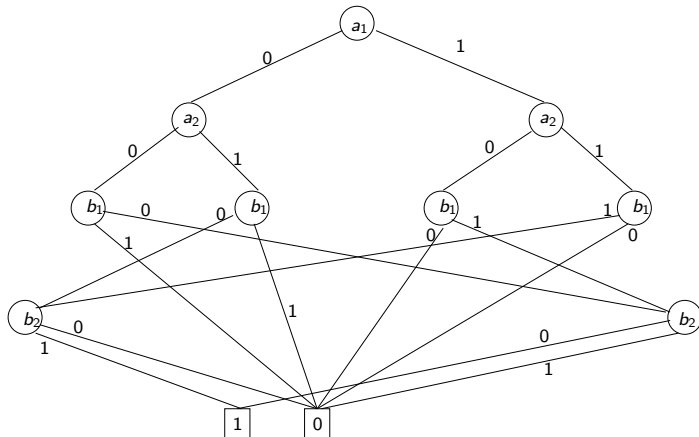
# ROBDD for 2-bit comparator with ordering

$$a_1 < b_1 < a_2 < b_2$$



# ROBDD for 2-bit comparator with ordering

$$a_1 < a_2 < b_1 < b_2$$



## Variable ordering problem

- The size of a BDD **largely depends** on the variable ordering
  - With  $a_1 < b_1 < a_2 < b_2 \dots < a_n < b_n$ , the number of vertices is  $3 \cdot n + 2$
  - With  $a_1 < a_2 \dots < a_n < b_1 < b_2 \dots < b_n$ , the number of vertices is  $3 \cdot 2^n - 1$
- Checking that an ordering is optimal is NP-hard
- There might be **no optimal ordering**
  - Boolean function for the middle output of a multiplier
- Heuristics and dynamic ordering

# Logical operations on ROBDD's (1)

- Logical **negation**  $\neg f(a, b, c, d)$   
 Replace each leaf by its negation
- Logical **conjunction**  $f(a, b, c, d) \wedge g(a, b, c, d)$ 
  - Use **Shannon's expansion** as follows

$$f \wedge g = \neg a \wedge (f|_{\neg a} \wedge g|_{\neg a}) \vee a \wedge (f|_a \wedge g|_a)$$

to break the problem into **two sub-problems**. Solve sub-problems recursively.

- Always **combine isomorphic subtrees** and **eliminate redundant nodes**
- Hash tables stores previously computed sub-problems
- Number of sub-problems bounded by  $|f| \cdot |g|$

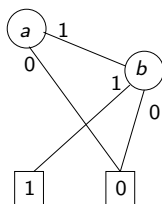
# Examples

Do  $a \wedge b$  then  $\neg a \vee \neg b$  with ordering  $b < a$

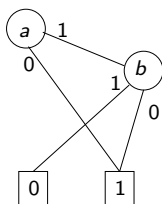
Do  $c$  and  $\neg c$

Do  $(\neg a \vee \neg b) \wedge c$  with ordering  $b < a < c$

# Logical operations on ROBDD's (1): Negation

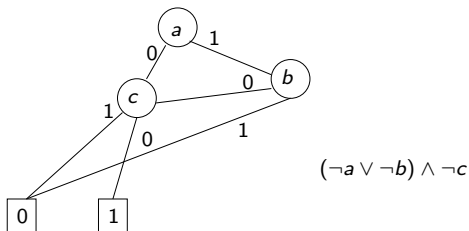
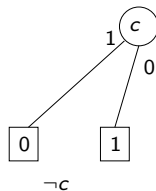
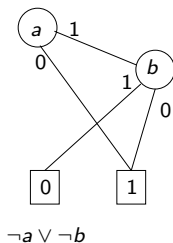


$a \wedge b$



$\neg(a \wedge b)$

# Logical operations on ROBDD's (1): Conjunction



## Logical operations on OBDD's (2)

- **Boolean quantification:**  $\exists a : f(a, b, c, d)$

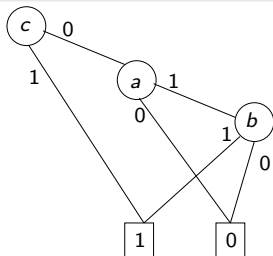
- By definition

$$\exists a : f = f|_{\neg a} \vee f|_a$$

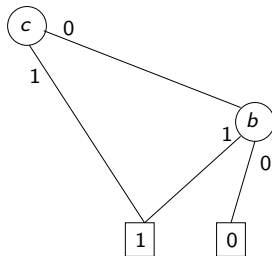
- $f(a, b, c, d)|_{\neg a}$ : replace all  $a$ -nodes by left sub-tree
- $f(a, b, c, d)|_a$ : replace all  $a$ -nodes by right sub-tree

From all these basic operations, we can build ROBDD's for complex functions

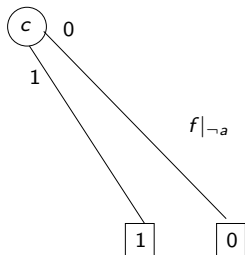
## Logical operations on ROBDD's (2): Quantification



$a \wedge b \vee c$



$f|_a$



$f|_{\neg a}$

# Representing Transition Systems with BDD's

Assume system behaviour given by  $n$  Boolean state variables  $v_1, v_2, \dots, v_n$ .

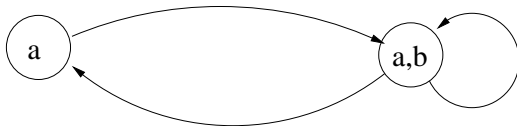
Transition relation  $T$  as Boolean formula of the state variables:

$$T(v_1, v_2, \dots, v_n, v'_1, v'_2, \dots, v'_n)$$

where  $v_1, v_2, \dots, v_n$  is the current state and  $v'_1, v'_2, \dots, v'_n$  the next state.

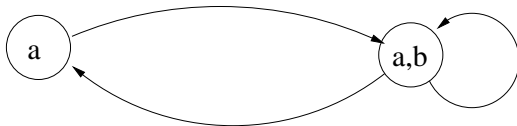
Express  $T$  as a ROBDD !

## Encoding Transition Systems as BDD's: Example



Boolean formulae for transition relation = characteristic function  
= ?

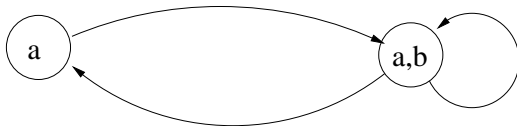
# Encoding Transition Systems as BDD's: Example



Boolean formulae for transition relation = characteristic function  
= ?

$$(a \wedge \neg b \wedge a' \wedge b') \vee (a \wedge b \wedge a' \wedge b') \vee (a \wedge b \wedge a' \wedge \neg b')$$

# Encoding Transition Systems as BDD's: Example



Boolean formulae for transition relation = characteristic function  
= ?

$$(a \wedge \neg b \wedge a' \wedge b') \vee (a \wedge b \wedge a' \wedge b') \vee (a \wedge b \wedge a' \wedge \neg b')$$

Represent as a ROBDD !