3-Valued Abstraction and Its Applications in Model Checking

Orna Grumberg
Technion, Israel

Summerschool at Marktoberdorf, 2009

1

Outline

- Introduction to Model Checking and Abstraction
  - Temporal logic and model checking
  - Abstraction
  - 3-Valued abstraction
- 3-Valued abstraction for compositional verification
- 3-Valued abstraction in (Bounded) Model Checking for hardware

2

Why (formal) verification?

- safety-critical applications: Bugs are unacceptable!
  - Air-traffic controllers
  - Medical equipment
  - Cars
- Bugs found in later stages of design are expensive, e.g. Intel's Pentium bug in floating-point division
- Hardware and software systems grow in size and complexity: Subtle errors are hard to find by testing
- Pressure to reduce time-to-market

Automated tools for formal verification are needed

3

Formal Verification

Given

- a model of a (hardware or software) system and
- a formal specification

does the system model satisfy the specification?

Not decidable!

To enable automation, we restrict the problem to a decidable one:

- **Finite-state** reactive systems
- **Propositional** temporal logics

4

Finite state systems - examples

- Hardware designs
- Controllers (elevator, traffic-light)
- Communication protocols (when ignoring the message content)
- High level (abstracted) description of non finite state systems

5

Properties in temporal logic - examples

- **mutual exclusion:**
  always $\neg$($cs_1 \wedge cs_2$)
- **non starvation:**
  always (request $\Rightarrow$ **eventually** granted)
- **communication protocols:**
  ($\neg$ get-message) **until** send-message

6

Model Checking [EC81,QS82]

An efficient procedure that receives:

- A finite-state model describing a system
- A temporal logic formula describing a property

It returns
yes, if the system has the property
no + Counterexample, otherwise

7

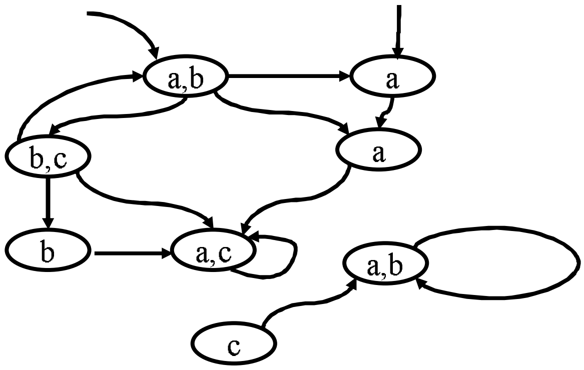
Model Checking

- Emerging as an industrial standard tool for verification of hardware designs: Intel, IBM, Cadence, …
- Recently applied successfully also for software verification: SLAM (Microsoft), Java PathFinder and SPIN (NASA), BLAST (EPFL), CBMC (Oxford),…

8

Model of a system

Kripke structure / transition system



9

Propositional temporal logic

In **Negation Normal Form**

AP – a set of atomic propositions

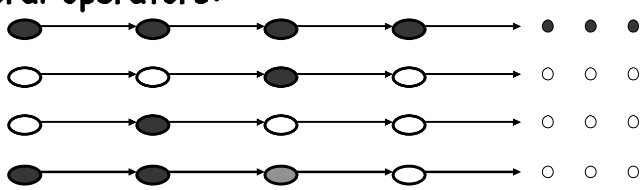
Temporal operators:

Gp

Fp

Xp

pUq



Path quantifiers: ***A*** for **all** path

E there **exists** a path

10

CTL/CTL*

- **CTL*** - Allows any combination of temporal operators and path quantifiers
- **CTL** – a useful sub-logic of CTL*

ACTL / ACTL*

The **universal** fragments of CTL/CTL* with only universal path quantifiers

11

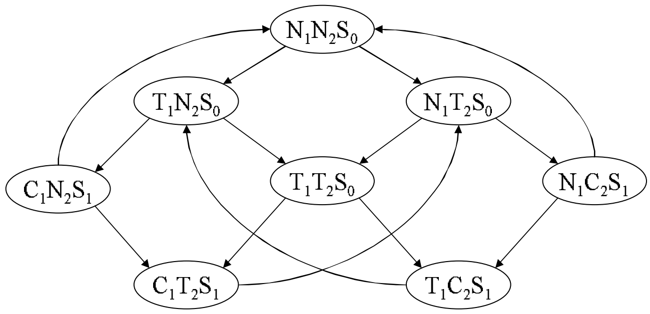
Mutual Exclusion Example

- Two process mutual exclusion with shared semaphore
- Each process has three states
  - Non-critical (N)
  - Trying (T)
  - Critical (C)
- Semaphore can be available (S_0) or taken (S_1)
- Initially both processes are in the Non-critical state and the semaphore is available --- $N_1 N_2 S_0$

$$\begin{array}{lcl} N_1 & \rightarrow & T_1 \\ T_1 \wedge S_0 & \rightarrow & C_1 \wedge S_1 \\ C_1 & \rightarrow & N_1 \wedge S_0 \end{array} \quad || \quad \begin{array}{lcl} N_2 & \rightarrow & T_2 \\ T_2 \wedge S_0 & \rightarrow & C_2 \wedge S_1 \\ C_2 & \rightarrow & N_2 \wedge S_0 \end{array}$$

12

Mutual Exclusion Example



$M \models AG\,EF\,(N_1 \land N_2 \land S_0)$

No matter where you are there is always a way to get to the initial state

Main limitation

The state explosion problem:
Model checking is efficient in time but suffers from high space requirements:

The number of states in the system model grows exponentially with

- the number of variables
- the number of components in the system

Solutions to the state-explosion problem

Symbolic model checking:
The model is represented symbolically

- BDD-based model checking
- SAT-based Bounded Model Checking (BMC)
- SAT-based Unbounded Model Checking

15

Other solutions to the state explosion problem

Small models replace the full, concrete model:

- Abstraction
- Compositional verification
- Partial order reduction
- Symmetry

16

Relations between small models and concrete models

Equivalence strongly preserves CTL*

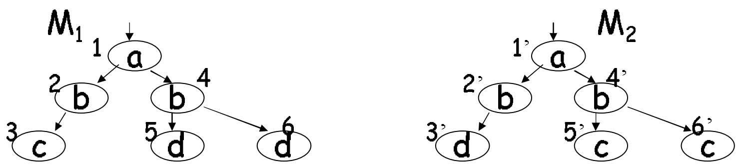
If $M_1 \equiv M_2$ then for every **CTL*** formula φ,

$M_1 \models \varphi \Leftrightarrow M_2 \models \varphi$

17

Bisimulation equivalence

$M_1 \equiv M_2$



Both models satisfy the CTL formula:

$EX(b \wedge AXc) \wedge EX(b \wedge AXd)$

18

Relations between small models and concrete models

preorder weakly preserves ACTL*

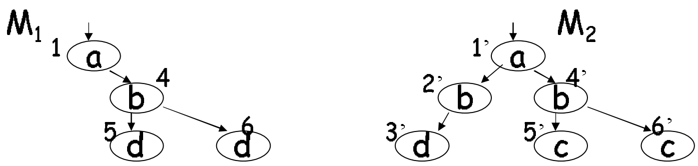
If $\mathbf{M_2} \geq \mathbf{M_1}$ then for every **ACTL*** formula φ,

$\mathbf{M_2} \models \varphi \Rightarrow \mathbf{M_1} \models \varphi$

19

Simulation preorder

$\mathbf{M_1} \leq \mathbf{M_2}$



ACTL formula $\varphi = AG(b \rightarrow (AXc \vee AXd))$

$M_2 \models \varphi \Rightarrow M_1 \models \varphi$

20

2-valued CounterExample-Guided Abstraction Refinement (CEGAR)

[CGJLV02]

21

Abstraction-Refinement

- **Abstraction**: removes or simplifies details that are irrelevant to the property under consideration, thus reducing # of states
- **Refinement** might be needed

22

Abstraction preserving ACTL/ACTL*

Existential Abstraction:

The abstract model is an **over-approximation** of the concrete model:

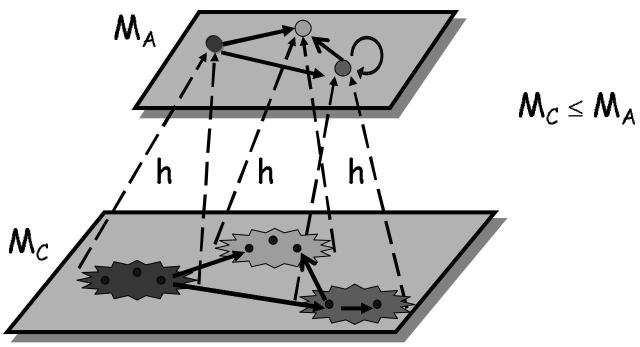
- The abstract model has **more behaviors**
- But no concrete behavior is lost

- Every ACTL/ACTL* property true in the abstract model is also true in the concrete model

23

Existential Abstraction

Given an abstraction function $h : S \rightarrow S_A$, the concrete states are grouped and mapped into abstract states :



24

Widely used Abstractions (S_h, h)

- Localization reduction: each variable either keeps its concrete behavior or is fully abstracted (has free behavior) **[Kurshan94]**
- Predicate abstraction: concrete states are grouped together according to the set of predicates they satisfy **[GS97,SS99]**
- Data abstraction: the domain of each variable is abstracted into a small abstract domain **[CGL94,LONG94]**

25

Logic preservation Theorem

- **Theorem** $M_C \leq M_A$, therefore for every ACTL* formula φ,

$$M_A \models \varphi \Rightarrow M_C \models \varphi$$

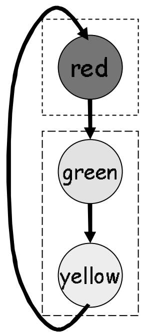
- However, the reverse may not be valid.

26

Traffic Light Example

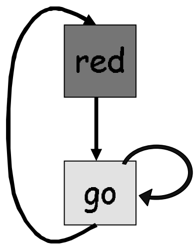
Property:
φ =**AG AF** ¬ (state=red)

Abstraction function h maps green, yellow to go.



M_C

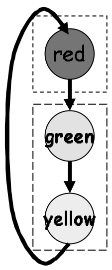
$M_C \models \varphi \Leftarrow M_A \models \varphi$



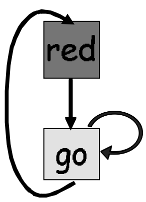
M_A

Traffic Light Example (Cont)

If the abstract model invalidates a specification, the actual model may still satisfy the specification.



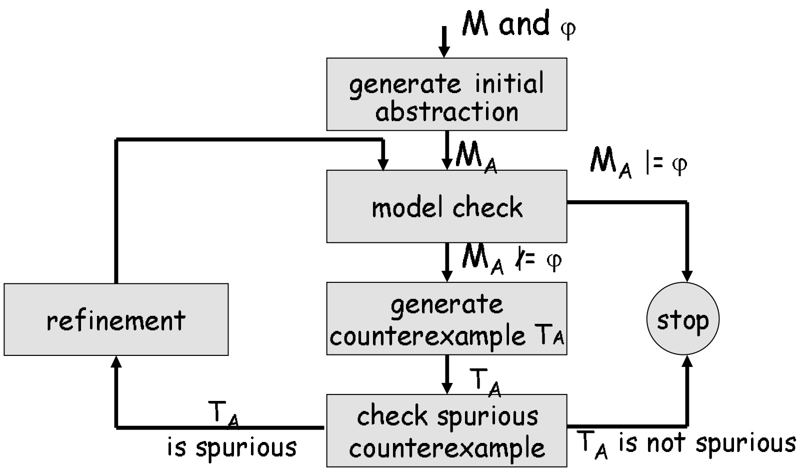
M_C



M_A

- Property:
  φ =**AG AF** (state=red)
- $M_C \models \varphi$ but $M_A \not\models \varphi$
- Spurious Counterexample:
  $\langle$red,go,go, ...$\rangle$

The CEGAR Methodology



29

3-Valued Abstraction for Full CTL*

30

Abstract Models for CTL*

- Two transition relations [LT88]
- Kripke Modal Transition System (KMTS)
- $M = (S, S_0, Rmust, Rmay, L)$
  - Rmust: an under-approximation
  - Rmay: an over-approximation
  - $Rmust \subseteq Rmay$

31

Abstract Models for CTL* (cont.)

Labeling function :

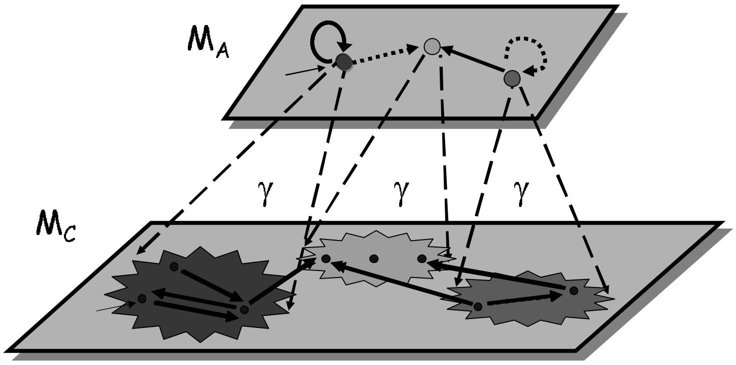
- $L: S \rightarrow 2^{Literals}$
- $Literals = AP \cup \{\neg p \mid p \in AP\}$
- At most one of p and $\neg$p is in L(s).
  - Concrete: exactly one of p and $\neg$p is in L(s).
  - KMTS: possibly none of them is in L(s).

32

Abstract Models for CTL* (cont.)

must and may transitions:

may: over approximation (∃∃)



33

3-Valued Semantics

tt, **ff** are definite

- Additional truth value: ⊥ (indefinite)
- Abstraction preserves both **truth** and **falsity**
- (abstract) s_a represents (concrete) s_c:
  - φ is **true** in $s_a \Rightarrow$ φ is **true** in s_c
  - φ is **false** in $s_a \Rightarrow$ φ is **false** in s_c
  - φ is ⊥ in $s_a \Rightarrow$ the value of φ in s_c is **unknown**

[BG99]

34

3-Valued Semantics

- Universal properties (**A**ψ) :
  - **Truth** is examined along *all* **may**-successors
  - **Falsity** is shown by a *single* **must**-successor
- Existential properties (**E**ψ) :
  - **Truth** is shown by a *single* **must**-successor
  - **Falsity** is examined along *all* **may**-successors

35

Compositional Verification and 3-Valued Abstraction Join Forces [SG07]

36

We describe

- How to join forces of two popular solutions:
  - **Abstraction-Refinement**
  - **Compositional reasoning**

In order to obtain

- fully automatic
- compositional model checking
- for the full μ-calculus

37

Compositional Verification

The system is composed of $M_1 \| \ldots \| M_n$

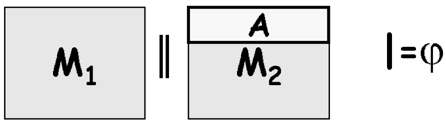
- "**divide and conquer**" approach: try to verify each component separately
- Problem: usually impossible due to dependencies
  - a component is typically designed to satisfy its requirements in *specific* **environments** (contexts)

➔ More sophisticated schemes are needed

38

Assume-Guarantee (AG) paradigm

Introduces **assumptions** representing a component's environment



1. check if a component M_1 **guarantees** φ when it is a part of a system satisfying **assumption *A***.
2. **discharge** assumption: show that the remaining components (the environment) satisfy ***A***

Main challenge: **How to construct assumptions?**

39

Automatic Compositional Framework

- Previous work: based on the **Assume-Guarantee (AG)** paradigm and on assumption generation via **learning**, for **universal safety** properties [CGP03, AMN05, CCST05,…]
- Our approach: based on techniques from **3-valued** model checking, applicable to the **full mu-calculus**

40

General Idea

- View M_i as a 3-valued **abstraction** $\mathbf{M_i\uparrow}$ of $\mathbf{M_1 \| \ldots \| M_n}$ and check each $M_i\uparrow$ separately using a **3-valued semantics**:
  - **tt** and **ff** are definite: hold also in $M_1 \| \ldots \| M_n$
  - $\perp$ is indefinite: value in $M_1 \| \ldots \| M_n$ is unknown
- If no $M_i\uparrow$ returned a definite result, **identify the parts which are indefinite** and compose only them

41

Composition of Models $M_1 \| M_2$

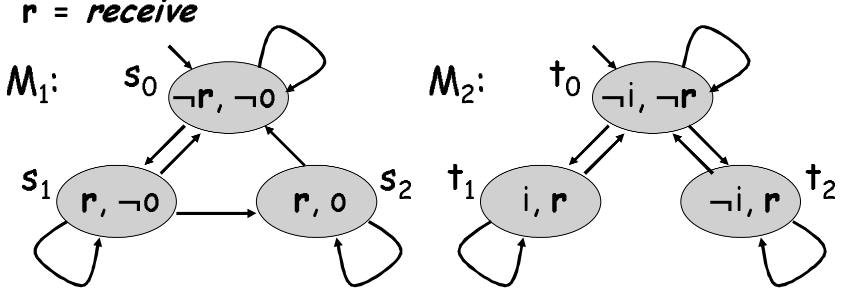
$M_1 = (AP_1, S_1, s^0_1, R_1, L_1)$, $M_2 = (AP_2, S_2, s^0_2, R_2, L_2)$

- Components **synchronize** on the **joint labelling of the states** $\mathbf{AP_1 \cap AP_2}$

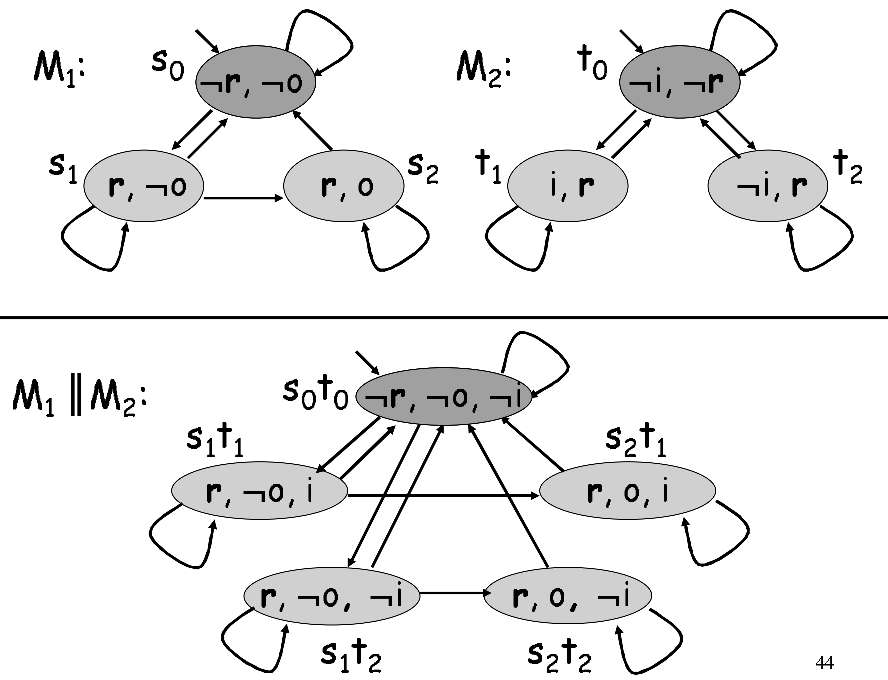
42

- o = *output*
- i = *input*
- **r** = ***receive***

Example



43



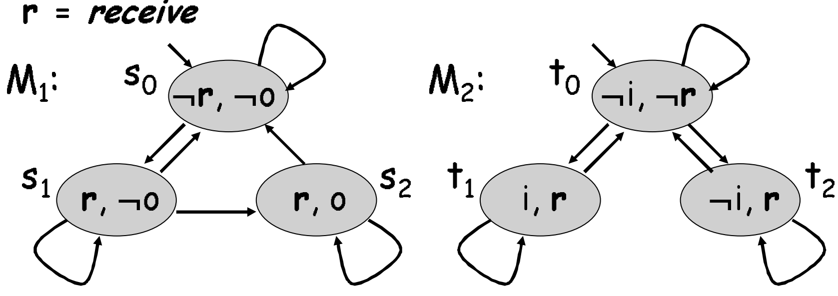
44

- This composition is suitable for describing hardware designs
- Same ideas are applicable to other synchronization models, e.g. **Labeled Transition Systems (LTS)** that synchronize on the joint actions and interleave the local transitions

45

- o = *output*
- i = *input*
- **r** = ***receive***

Example



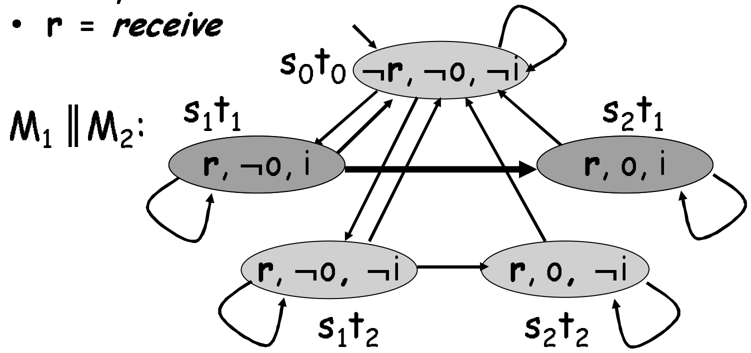
$\Box(\neg i \vee \Diamond o)$:

in all successors, ***input*** signal implies that there exists a successor producing the ***output*** signal

46

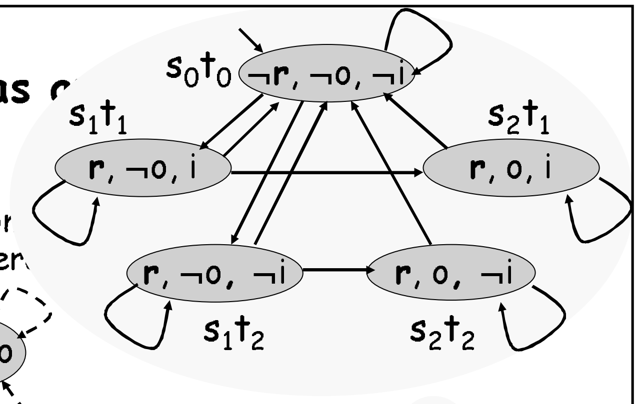
Example

- o = *output*
- i = *input*
- **r** = ***receive***

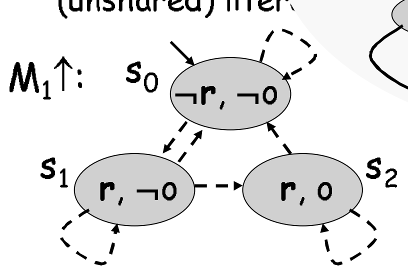


□(¬i ∨ ◊o) :
in all successors, ***input*** signal implies that there exists a successor producing the ***output*** signal

View M_i as a



Transitions becor
(unshared) liter

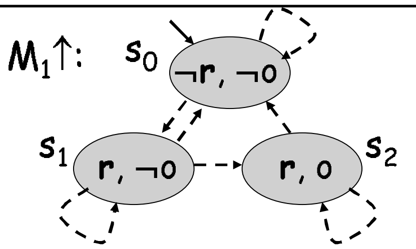


- Value of *inp* nknown

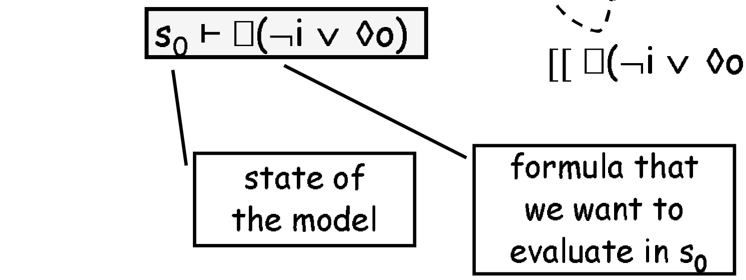
- s_i in M_1↑ abstracts **all** states **(s_i,t)** in $M_1 \| M_2$
- Model check using **3-valued MC-based** [SG03, GLLS05, GLLS07]

Model Checking

MC Graph:

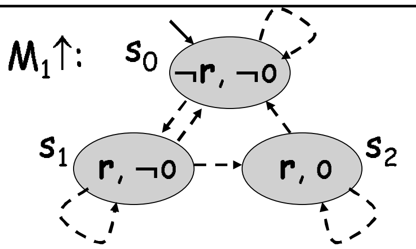


$[[\ \Box(\neg i \vee \Diamond o)\ ]](s_0) = ?$

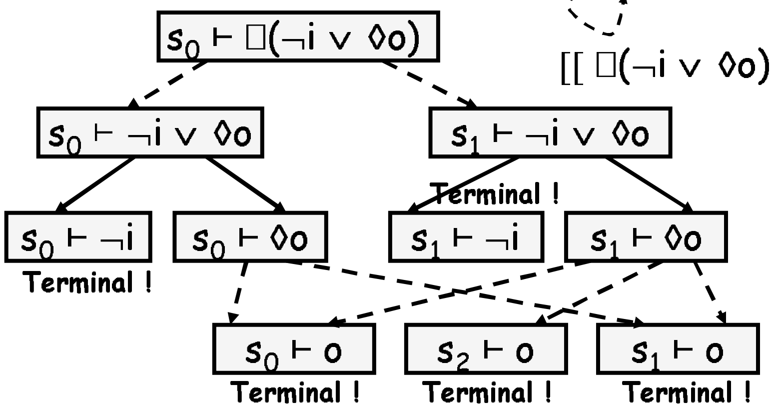


Model Checking

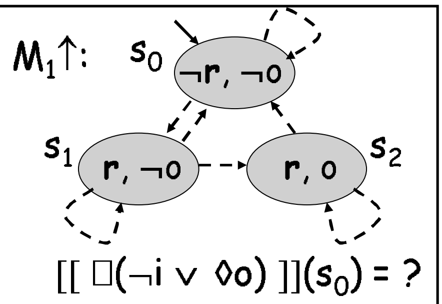
MC Graph:



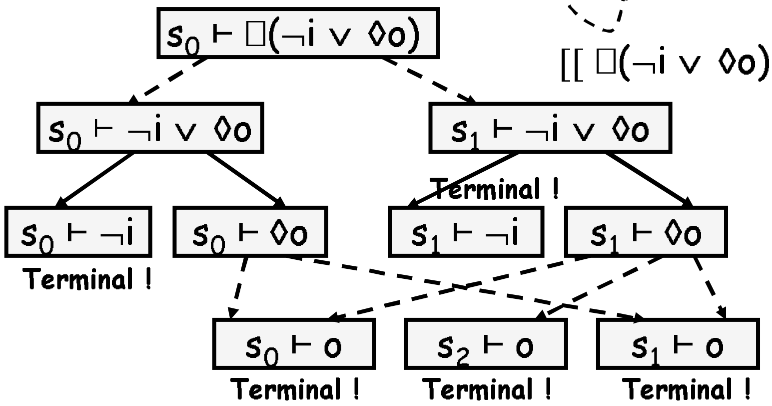
$[[\ \Box(\neg i \vee \Diamond o)\ ]](s_0) = ?$



Model Checking



Bottom-up coloring:

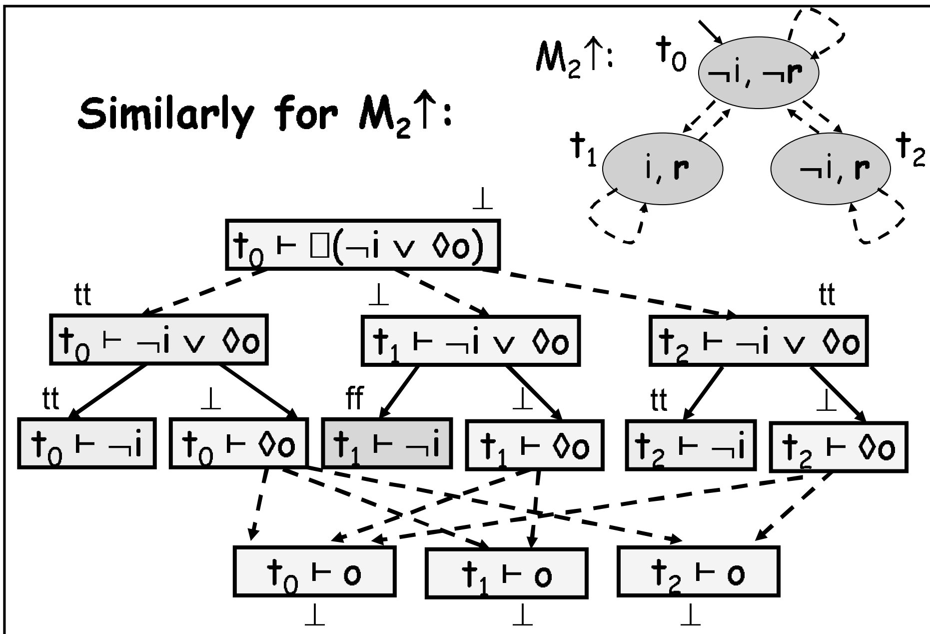


3-Valued Model Checking Results

- **tt** and **ff** are definite:
  hold in the concrete model as well
  ➔ In our case: hold in $M_1 \| M_2$
- $\perp$ is indefinite

➔ result on M_1 is indefinite

Similarly for $M_2\uparrow$:



53

3-Valued Model Checking Results

- ⊥ on both components
  ⇒ **Refinement** is needed

➔ consider the composition

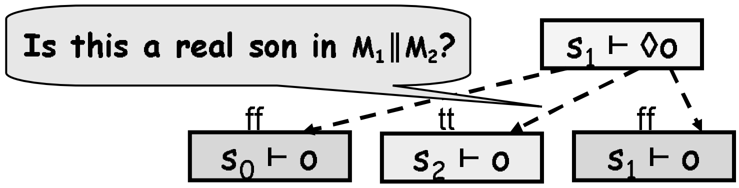
! **But** – **only** the parts of the abstract models for which the **model checking result is** ⊥ are identified and composed

54

Identify the indefinite parts:

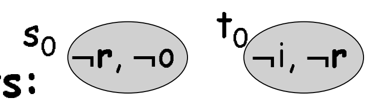
- Construct **?-subgraph**, top-down
- For each ⊥-node keep only **witnessing sons**:
  - ∨, ◊ : keep **tt-sons** + **⊥-sons**
  - ∧, □ : keep **ff-sons** + **⊥-sons**

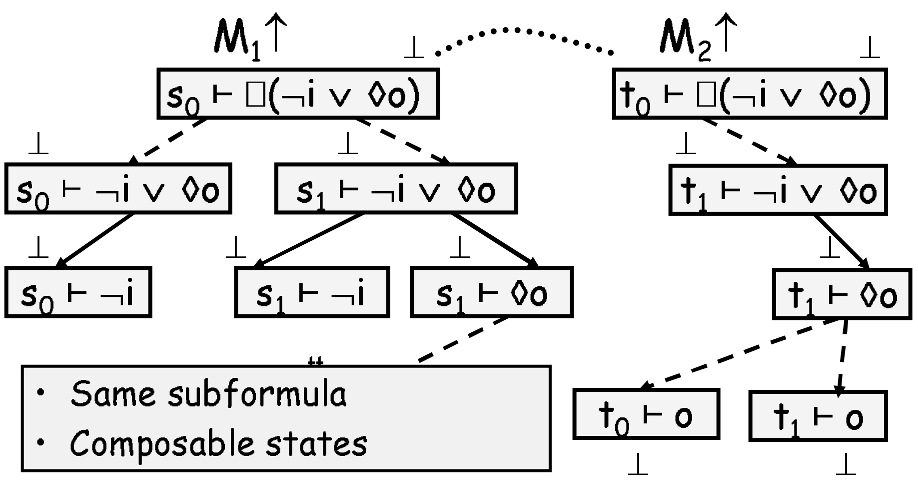
Remaining sons suffice to determine result



55

Compose indefinite parts:

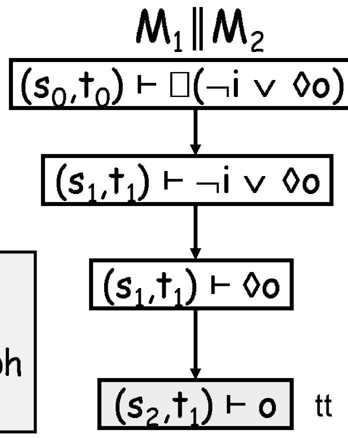




56

Product Graph:

All transitions are real transitions



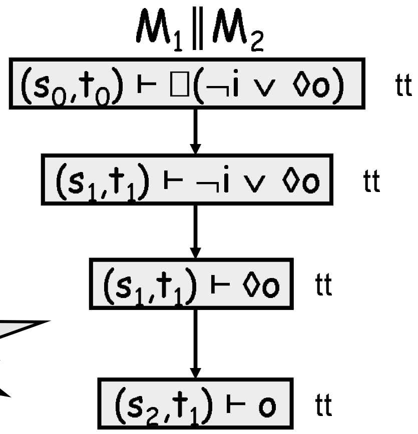
Substantially **smaller than full** game-graph for $M_1 \| M_2$

Terminal nodes are all colored **tt** or **ff**

57

Color Product Graph:

All transitions are real transitions



Terminal nodes are all colored **tt** or **ff**

58

Compositional Model Checking

For each i = 1,2:
- Lift M_i to $M_i\uparrow$
- Construct **model checking graph for $M_i\uparrow$**
- Apply **3-valued** coloring

If both results are indefinite:
- Construct **?-subgraphs**
- Compose ?-subgraphs to obtain **product graph**
- **Color** product graph

59

Summary

- New ingredient to **compositional model checking**: uses a MC graph to **identify and focus** on the parts of the components where their composition in necessary.
  - orthogonal to the AG approach
- **Automatic compositional abstraction-refinement** framework, which is **incremental**.
- Applicable to the **full mu-calculus**.

60

More background:
SAT-Based Bounded Model Checking (BMC)
[BCCFZ99]

61

SAT-based model checking

- Translates the model and the specification to a propositional formula
- Uses efficient tools for solving the satisfiability problem

Since the satisfiability problem is **NP-complete**, SAT solvers are based on **heuristics**.

62

SAT tools

- Using heuristics, SAT tools can solve very large problems fast.
- They can handle systems with thousands of variables that create formulas with a few millions variables.

GRASP (Silva, Sakallah)
Prover (Stalmark)
Chaff (Malik)
MiniSAT

63

Bounded model checking for checking AGp

- Unwind the model for k levels, i.e., construct all computation of length k
- If a state satisfying $\neg$p is encountered, then produce a counter example

The method is suitable for **falsification**, not verification

64

Bounded model checking with SAT

- Construct a formula $f_{M,k}$ describing all possible computations of M of length k
- Construct a formula f_φ expressing φ=EF¬p
- Check if $f = f_{M,k} \wedge f_\varphi$ is satisfiable

If f is satisfiable then M $\not\models$ AGp
The satisfying assignment is a **counterexample**

65

Example – shift register

Shift register of 3 bits: <x, y, z>
Transition relation:
$R(x,y,z,x',y',z') = \quad x'=y \wedge y'=z \wedge z'=1$
(**error** marks $z'=1$)

Initial condition:
$I(x,y,z) = x=0 \vee y=0 \vee z=0$

Specification: *AG* ($x=0 \vee y=0 \vee z=0$)

66

Propositional formula for k=2

$f_M = (x_0=0 \lor y_0=0 \lor z_0=0) \land$
$(x_1=y_0 \land y_1=z_0 \land z_1=1) \land$
$(x_2=y_1 \land y_2=z_1 \land z_2=1)$

$f_\varphi = V_{i=0,..2} (x_i=1 \land y_i=1 \land z_i=1)$

Satisfying assignment: 101 011 111
This is a counter example!

67

3-Valued Abstraction in (Bounded) Model Checking for Hardware [YFGL09]

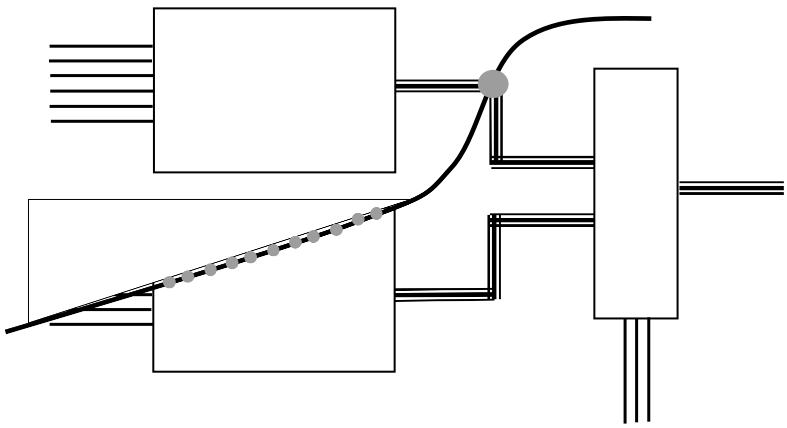
68

Motivation

- Increase capacity of (Bounded) Model Checking
  - By abstracting out parts of the model
- "Smart" abstraction
  - Automatic or manual
- "Easy" abstraction
  - Abstract out inputs or critical nodes
- Holy Grail: Change the level of BMC

69

Abstraction in Model Checking



70

Localization reduction

Over-approximating abstraction:
Abstract model contains more behaviors

- Property is true on abstract model $\Rightarrow$
  Property is true on the concrete model
- Property is false: counterexample might be spurious
- Refinement is needed (CEGAR)

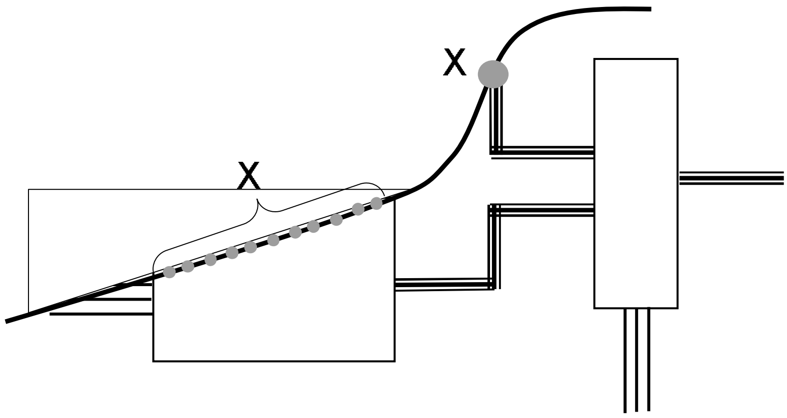
71

- Finding cutpoints:
  computationally expensive or needs human expertise
- False negative results:
  overhead in checking if counterexample is spurious

72

3-Valued Abstraction

- Add a third value "X" ("Unknown")



73

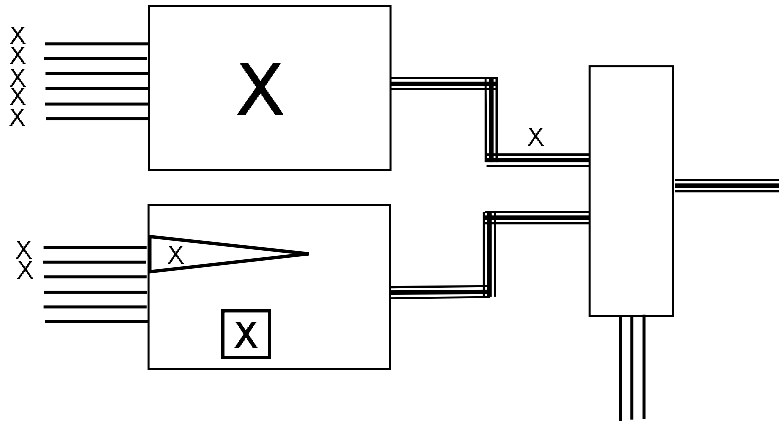
Introducing X ("Unknown")

- Property is true on abstract model $\Rightarrow$ Property is true on the concrete model
- Property is false on abstract model $\Rightarrow$ Property is false on the concrete model
- Property is X $\Rightarrow$ needs refinement

74

3-Valued Abstraction

- Add a third value "X"



75

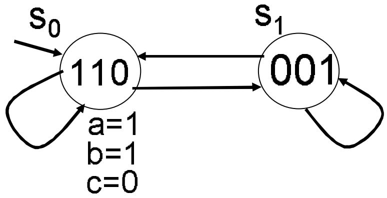
Outline

- Model Checking – Automata Approach
  - Kripke Structures
  - LTL
  - Büchi Automata
  - BMC
- 3-Valued Abstraction
- 3-Valued BMC (X-BMC)

76

Kripke Structure

- $M=(S,s_0,R,L)$ over AP
- $L:S\rightarrow(AP\rightarrow\{0,1\})$ $\quad L:S\rightarrow\{0,1\}^{AP}$
- Can describe hardware circuits

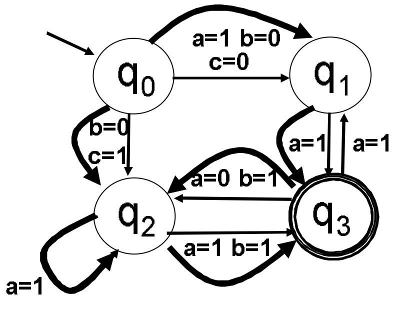


$AP=\{a,b,c\}$

77

Büchi Automata

- $B=(\Sigma,Q,q_0,\rho,\alpha)$ $\quad \rho:Q\times\Sigma\rightarrow 2^Q$ $\quad w\in\Sigma^\omega$
- Accepts w iff there is an accepting run for w
  - Such that α is met infinitely often



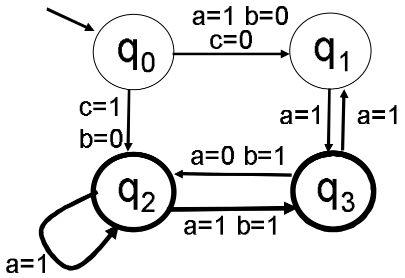
$\Sigma=\{0,1\}^{\{a,b,c\}}, \quad \alpha=\{q_3\}$

- 100,100,010,110,010,110,010,110...
- ✖ 010,010,010,010...
- ✖ 001,100,100,100...

78

Büchi Automata

- ρ can be represented as a function $F:Q\times\Sigma\times N\rightarrow Q$
  - $q'=F(q,\sigma,nd)$



$\rho(q_2,110)=\{q_2,q_3\}$

$F(q_2,110,0)=q_2$

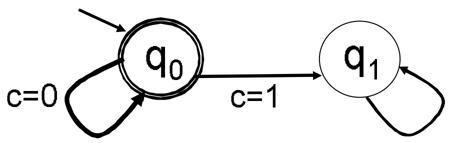
$F(q_2,110,1)=q_3$

79

Büchi for LTL

- Given $\varphi=A\psi$, build an automaton $B_{\neg\psi}$ for $\neg\psi$
- $\Sigma=\{0,1\}^{AP}$

$P=AFc$



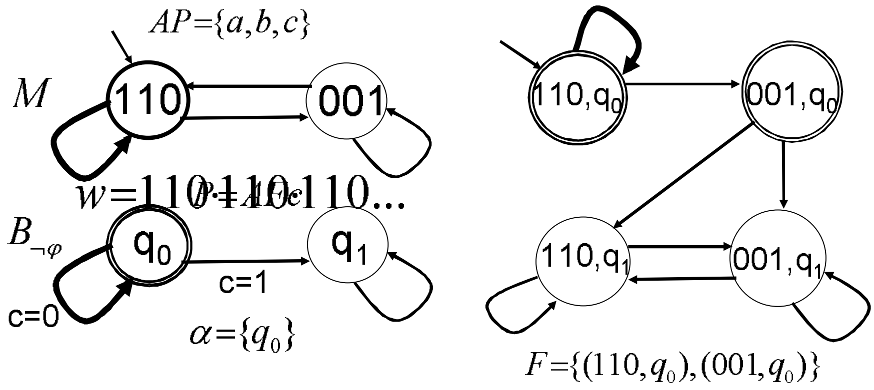
$\alpha=\{q_0\}$

$\pi=q_0,q_0,q_0,q_0\cdots$

80

Model Checking

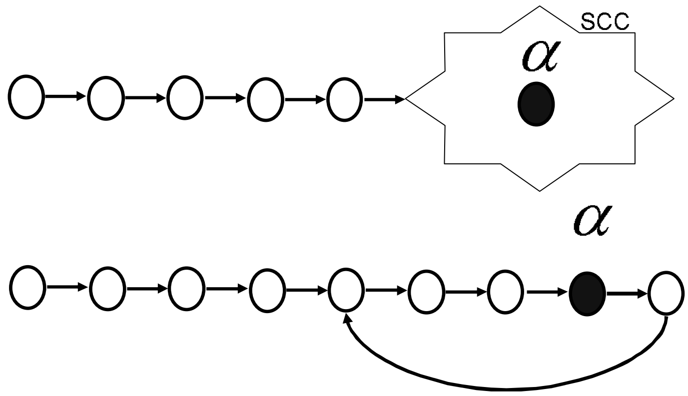
- Let $E = M \times B$ $\quad F = S \times \alpha$
- Reduce Model Checking to Emptiness of E



81

Model Checking

- Fair Paths in E



82

Bounded Model Checking (BMC)

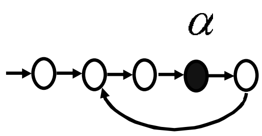
- Build a propositional representation of E
  - Describe paths of bounded length

$$\varphi_M^i(\overline{v}_0 \ldots \overline{v}_i) = I_0^M(\overline{v}_0) \wedge \bigwedge_{0 \le j < i} R_M(\overline{v}_j, \overline{v}_{j+1})$$

$$\varphi_B^i(\overline{v}_0 \ldots \overline{v}_i) = I_0^B(\overline{v}_0) \wedge \bigwedge_{0 < j < i} R_B(\overline{v}_j, \overline{v}_{j+1}) \wedge fair_i$$

$$fair_i(\overline{v}_0 \ldots \overline{v}_i) = \bigvee_{0 \le l < i} \left((\overline{v}_l = v_i) \wedge \bigvee_{l \le j \le i} \alpha_E(\overline{v}_j) \right)$$

$$\varphi_i(\overline{v}_0 \ldots \overline{v}_i) = \varphi_M^i \wedge \varphi_B^i$$



83

BMC

- Check finite paths in E

$BMC(M, P)$

$i \leftarrow 0$

$while(true)$ {

if SAT (φ_i) return $false$

$inc(i)$

}

84

3-Valued logic

- Ternary domain *D* = { 0, 1, *X*}
  - X is "unknown" (not "don't care")

	0	1	X
¬	1	0	X

∧	0	1	X
0	0	0	0
1	0	1	X
X	0	X	X

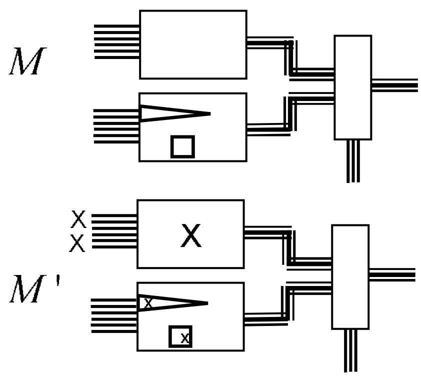
∨	0	1	X
0	0	1	X
1	1	1	1
X	X	1	X

  - Ternary operators agree with Boolean operators on Boolean values

85

3-Valued Abstraction

- Ternary domain *D* = { 0, 1, *X*}
  - X is "unknown" (not "don't care")

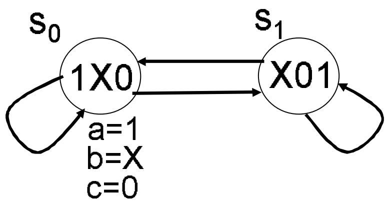


$$[M' \models P]=1 \;\Rightarrow\; [M \models P]=1$$
$$[M' \models P]=0 \;\Rightarrow\; [M \models P]=0$$

86

3-Valued Kripke Structure

- $M' = (S', s'_0\, R', L')$ over AP
- $L' : S' \rightarrow \{0,1,X\}^{AP}$



$AP = \{a,b,c\}$

87

3-Valued LTL

- Over AP
- $P = A\psi$

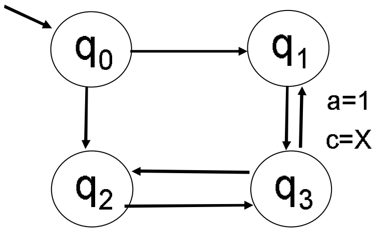
$\pi |= \psi \;\in \{0,1,X\}$

$$[M' |= P] = \begin{cases} 1 & \forall \pi, [\pi |= \psi] = 1 \\ 0 & \exists \pi, [\pi |= \psi] = 0 \\ X & otherwise \end{cases}$$

88

3-Valued Büchi

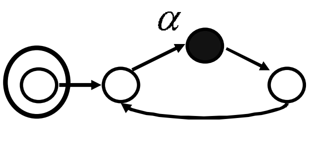
- $\Sigma=\{0,1,X\}^{AP}$
- 3-Valued transition function F' for ρ
  - $F':Q\times\Sigma\times N\rightarrow Q$
  - Ternary variables and operators



$F'(q_3,11X,0)=q_1$

89

3-Valued Model Checking



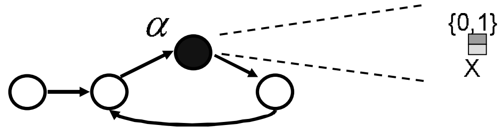
$E'=M'\times B'$

{0,1}	{0,1}	{0,1}	{0,1}	{0,1}	{0,1}
X	X	X	X	X	X

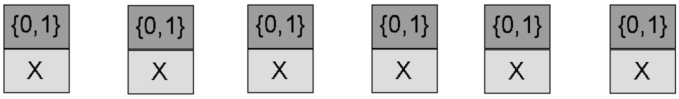
- A short loop is a witness for a long concrete loop
  - Lower the bound required for finding bugs

90

3-Valued Model Checking



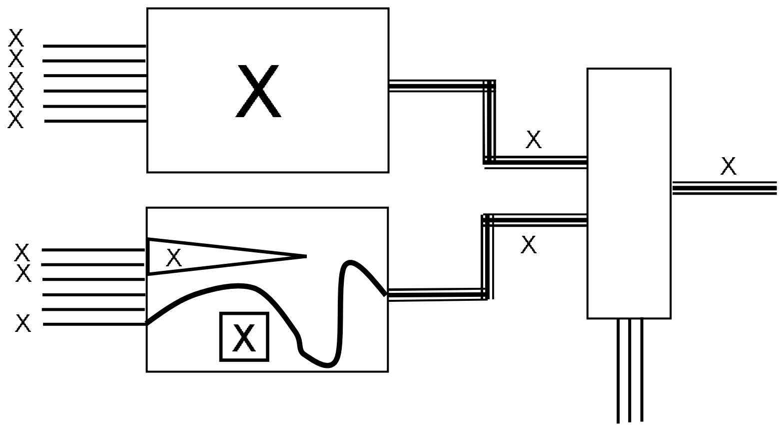
$$\alpha_E(\overline{v}_j) \in \{0,1,X\}$$



- Checking might yield an "unknown" result.

91

X-BMC



92

BMC - Reminder

$$\varphi^{i}_{M}(\overline{v}_0 \ldots \overline{v}_i) = I^{M}_{0}(\overline{v}_0) \wedge \bigwedge_{0 \le j < i} R_M(\overline{v}_j, \overline{v}_{j+1})$$

$$\varphi^{i}_{B}(\overline{v}_0 \ldots \overline{v}_i) = I^{B}_{0}(\overline{v}_0) \wedge \bigwedge_{0 < j < i} R_B(\overline{v}_j, \overline{v}_{j+1}) \wedge fair_i$$

$$fair_i(\overline{v}_0 \ldots \overline{v}_i) = \bigvee_{0 < l < i} \left((\overline{v}_l = \overline{v}_j) \wedge \bigvee_{l < j < i} \alpha_E(\overline{v}_j) \right)$$

93

X-BMC

- Create 3-Valued propositional formulae (dual rail)

$BMC(M', \psi)$ {

$i \leftarrow 0$

$while(true)$ {

if $SAT(\varphi'^{i}_{M'} = 1 \wedge \varphi'^{i}_{B} = 1)$ return $false$

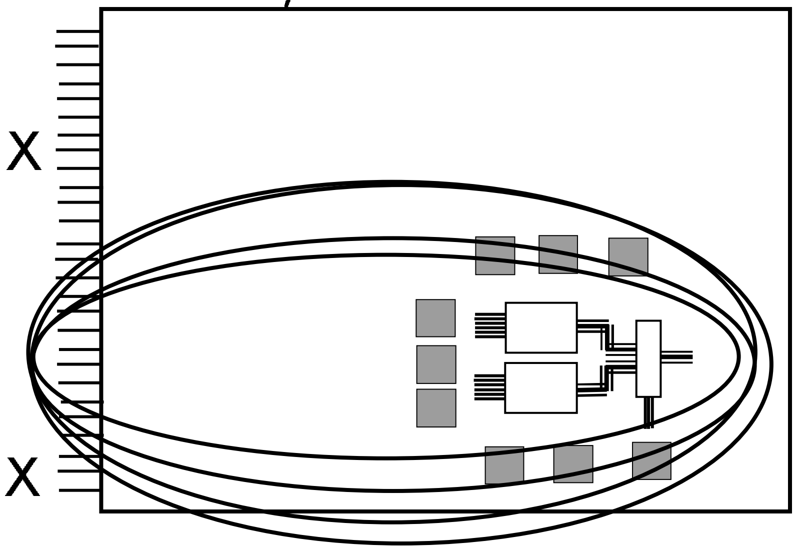
if $SAT(\varphi'^{i}_{M'} = 1 \wedge \varphi'^{i}_{B} = X)$ return X

$inc(i)$

}

94

Holy Grail - Revisited



95

Experimental Results (EXE Cluster)

		Model	EXE	Abs 1	Abs 2	Abs 3	Abs 4	Abs 5
		# Latches	133K	132K	115K	108K	74K	71K
		# Gates	6.1M	6.0M	5.9M	5.8M	0.6M	0.5M
	Property	Result	Run Time (s)					
XBMC	P1	fail	266	281	270	254	103	105
	P2	pass	262	271	265	244	212	205
	P3	fail	264	280	249	282	285	103
	P4	pass	412	365	342	323	X	X
	P5	fail	278	267	252	264	110	108
	P6	pass	654	640	631	615	587	552
BMC	P1	fail	M/O	M/O	M/O	12280	525	168
	P2	pass	M/O	M/O	M/O	479	411	235
	P3	fail	M/O	M/O	M/O	M/O	M/O	408
	P4	F/N	M/O	M/O	M/O	M/O	F/N	F/N
	P5	fail	M/O	M/O	M/O	M/O	908	632
	P6	pass	M/O	M/O	M/O	M/O	2241	199

96

Conclusion

- 3-Valued Abstraction
  - Models, specification and automata
  - Automatic or manual abstraction
  - Abstraction of inputs to the model
- 3-Valued Bounded Model Checking
  - Enhanced performance
  - Increased capacity
  - Reduced counterexample lengths
  - Insensitive to size of irrelevant parts of the model
  - Allows checking higher level models
    - Change in methodology (!)
- Unbounded Model Checking (Induction)
- Automatic Refinement

97

Conclusion (Final)

We introduced 3-valued abstraction and demonstrated its usefulness in two different applications:

- Compositional verification
- (Bounded) model checking for hardware

3-valued abstract models are:

- More precise
- Enable verification and falsification
- Avoid false negative results

98

Thank You

99

- **BDDs:**
  R. E. Bryant, Graph-based Algorithms for Boolean Function Manipulation, IEEE transactions on Computers, 1986

- **BDD-based model checking:**
  J.R. Burch, E.M. Clarke, K.L. McMillan, D.L. Dill, L.J. Hwang, Symbolic Model Checking: 10^20 States and Beyond, LICS'90

- **SAT-based Bounded model checking:**
  Symbolic model checking using SAT procedures instead of BDDs, A. Biere, A. Cimatti, E. M. Clarke, M. Fujita, Y. Zhu, DAC'99

100

- **Existential abstraction + data abstraction:**
  E. M. Clarke, O. Grumberg, D. E. Long, Model Checking and Abstraction, TOPLAS, 1994.

- **Localization reduction:**
  R. P. Kurshan, Computer-Aided Verification of coordinating processes – the automata theoretic approach, 1994

101

- **Predicate abstraction:**
  S. Graf and H. Saidi, Construction of abstract state graphs with PVS, CAV'97

  H. Saidi and N. Shankar, Abstract and Model Check while you Prove, CAV'99

- **BDD-based CEGAR:**
  Clarke, Grumberg, Jha, Lu, Veith, Counterexample-Guided Abstraction Refinement, CAV2000, JACM2003

102

- 3-Valued Abstraction-Refinement:
  S. Shoham and O. Grumberg, A Game-Based Framework for CTL Counterexamples and Abstraction-Refinement, CAV'03

- 3-Valued BMC:
  A. Yadgar, A. Flaisher, O. Grumberg, and M. Lifshits, High Capacity (Bounded) Model Checking Using 3-Valued Abstraction

103