AVACS Autumn School @ Oldenburg

Precision of BlackBox Verification Techniques: Hardness and Technology

Paolo Marin

Chair of Computer Architecture
University of Freiburg

October 1st, 2015

Acknowledgements

- *Bernd Becker*
- *Karina Gitina*
- *Stefan Kupferschmid*
- *Christian Miller*
- *Sven Reimer*
- *Christoph Scholl*
- *Ralf Wimmer*
- *...*

Why Verification?

Safety & Security

- Human life
- Money risk
- A project’s development cost

Our Target

- Incomplete designs
- Discrete systems

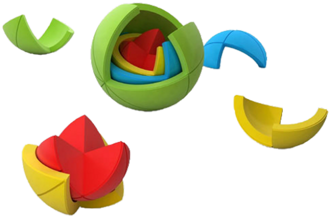
AVACS TP-S1: Systems of Systems

- **Automatic verification methods**
- Distributed systems
- Statically connected components

→ Compositional approach ←

- Miniaturization
- Reuse of components

→ Embedded Systems
System on Chip ←



AVACS TP-S1: Systems of Systems

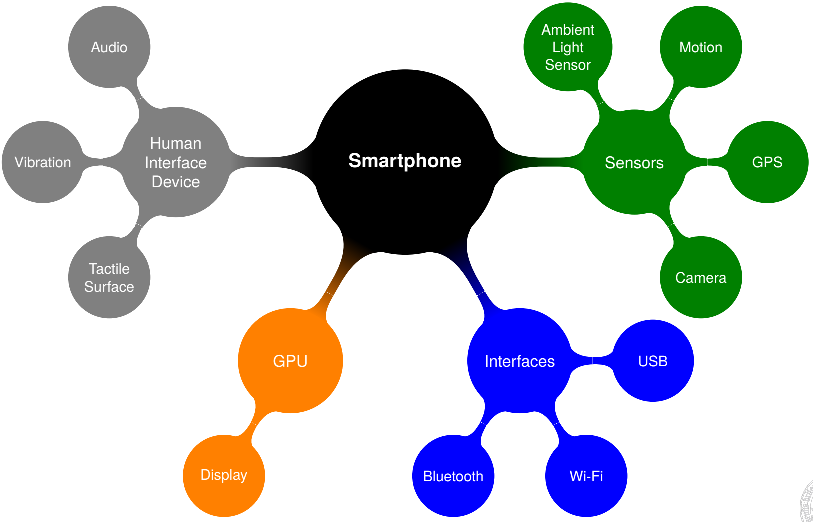
- **Automatic verification methods**
- Distributed systems
- Statically connected components

$\rightarrow$ Compositional approach $\leftarrow$

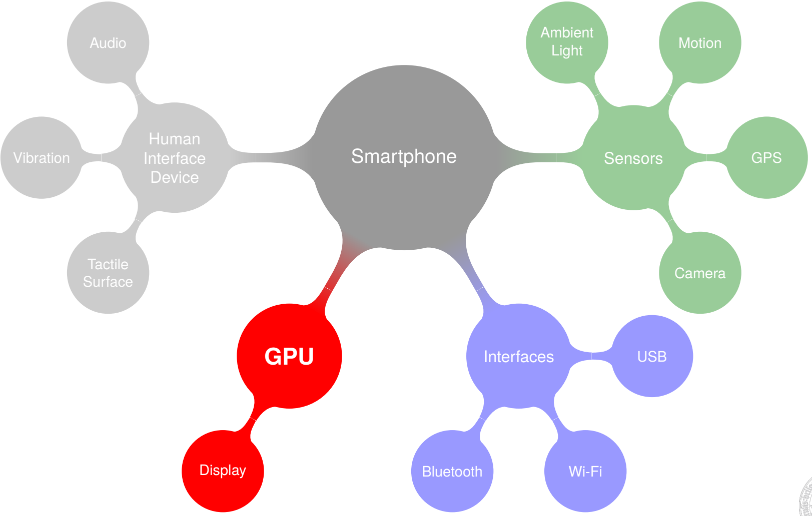
- Miniaturization
- Reuse of components

$\rightarrow$ Embedded Systems
System on Chip $\leftarrow$

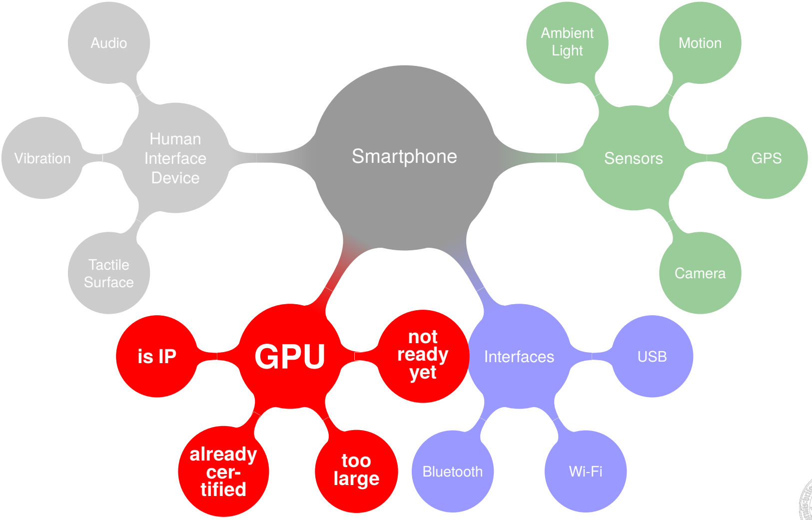
A Very Simple Example



A Very Simple Example



A Very Simple Example



Our Challenge

Verification of Incomplete Designs

Combinational Circuits

Is there any input vector that makes the given system produce a different output from the specification?
Validation

Sequential Circuits

Is there a sequence of inputs so that eventually the system output does not fulfill the specification?
Property checking

What happens if our design is incomplete?

Our Challenge

Verification of Incomplete Designs

Combinational Circuits

Is there any input vector that makes the given system produce a different output from the specification?

Validation $\longrightarrow$ Equivalence Checking

Sequential Circuits

Is there a sequence of inputs so that eventually the system output does not fulfill the specification?

Property checking $\longrightarrow$ Model Checking

What happens if our design is incomplete?

Our Challenge

Verification of Incomplete Designs

Combinational Circuits

Is there a Blackbox implementation that makes the implementation fulfill the specification?

Realizability $\longrightarrow$ Equivalence Checking

Sequential Circuits

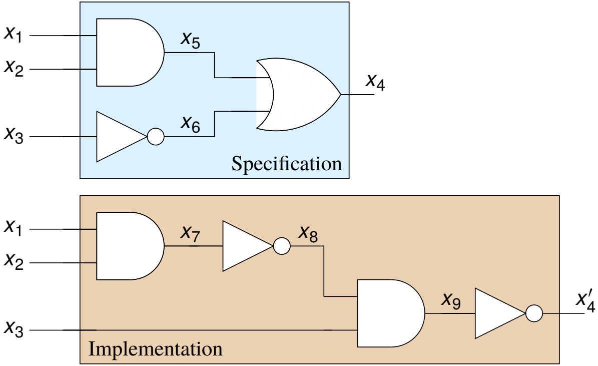
Is there a sequence of inputs so that eventually the system output does not fulfill the specification?

Property checking $\longrightarrow$ Bounded Model Checking

What happens if our design is incomplete?

Formal Verification for Combinational Circuits

Let the specification and the implementation of a combinational circuit be defined as follows



Question: are specification and implementation equivalent?

Circuit Equivalence Checking

- Formally prove whether the two circuits differ
- Construction of a BDD via symbolic simulation

  High memory requirements

  Optimizations possible (e.g. computation of equivalent sub-circuits via simulation)

- Solving a satisfiability problem

Focus

SAT based equivalence checking

Propositional Logic: Syntax

Definition

Let $x_1, \ldots, x_n$ be a set of Boolean variables. A propositional logic formula is defined inductively as:

- A variable x_i is a formula.
- For every formulas F_1 and F_2
  - the conjunction $(F_1 \wedge F_2)$ and
  - the disjunction $(F_1 \vee F_2)$ are also formulas.
- For every formula F, its negation $(\neg F)$ is a formula.
- A formula F is in conjunctive normal form (CNF) iff
  - F is the conjunction of $n \geq 0$ clauses $(C_1 \wedge C_2 \wedge \ldots \wedge C_n)$
  - which are the disjunction of $m \geq 0$ literals $(l_1 \vee l_2 \vee \ldots \vee l_m)$
  - and a literal is a variable x or its negation $\neg x$

Transformation into CNF requires linear time.

Propositional Logic: SAT Problem

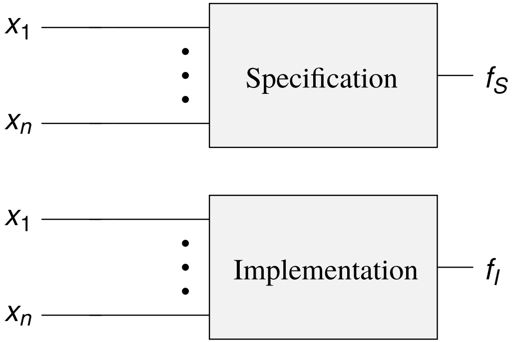
Definition

- A propositional logic formula F is satisfiable iff there exists an assignment $\mathscr{A}(F) = 1$.
- It is common to say that one of these kinds of assignments, also called Model, satisfies the formula F, and is represented with $\mathscr{A} \models F$.
- On the other hand, if there exist no assignment $\mathscr{A}$ such that $\mathscr{A}(F) = 1$, then F is unsatisfiable. For every such assignment $\mathscr{A}$ then $\mathscr{A} \not\models F$.

SAT for the Verification of Combinational Circuits

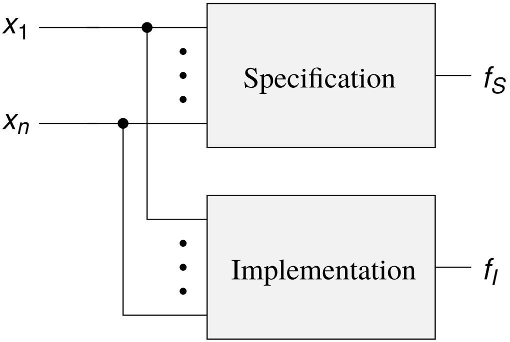
- Given
  - Specification and implementation of a combinational circuit
- Question
  - Are the specification and the implementation equivalent?
- Approach for SAT-based equivalence checking
  - Generate a so-called miter-circuit joining specification and implementation
  - Build a Boolean formula from the miter representation
  - Solve the formula with a SAT algorithm
- The specification and the implementation of a combinatorial circuit are equivalent iff the Boolean formula generated from the miter is unsatisfiable

Construction of the Miter Circuit



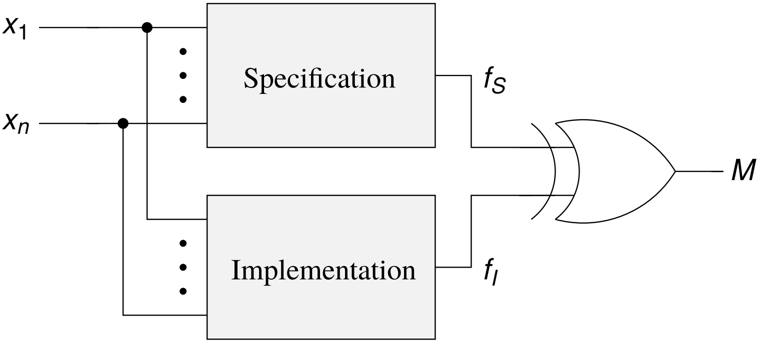
$\Rightarrow$ Connect corresponding inputs

Construction of the Miter Circuit



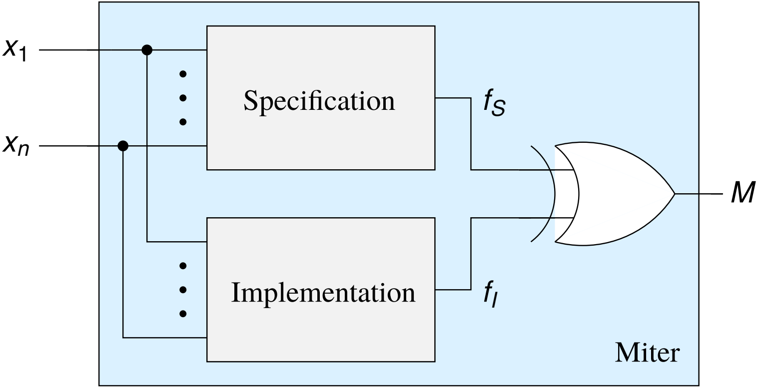
$\Rightarrow$ Link corresponding outputs by EXOR gates

Construction of the Miter Circuit



$\Rightarrow$ Miter circuit

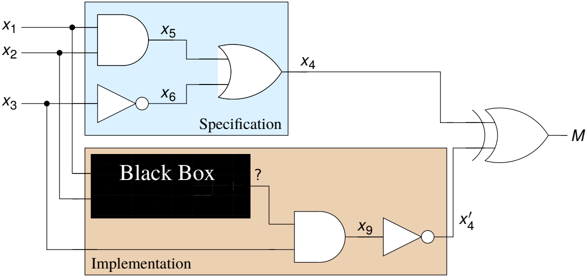
Construction of the Miter Circuit



$\Rightarrow M = 1 \Leftrightarrow$ Specification & Implementation not equivalent

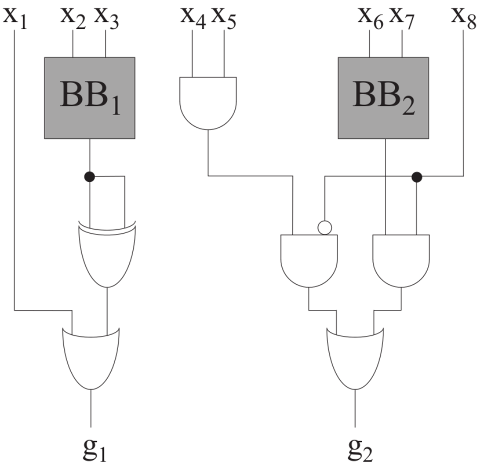
Partial Equivalence Checking [Scholl, Becker 2001]

- Part of the design replaced by a Blackbox
- Output modeled as an Unknown value
- `01X`-Logic 3-valued signals [Jain 2000]



Realizability problem: If unrealizability is returned, it **may** depend on a too coarse approximation

Coarse Approximation: Example



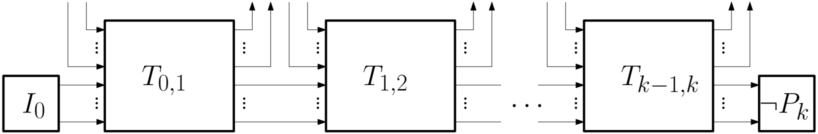
Constant 0 `XOR` gate output not detectable using `01X` logic.

Formal Verification for Sequential Systems

- Functional equivalence of two sequential circuits can be proved
- A specification which cannot be expressed as a sequential circuit or a deterministic finite state automaton cannot be proved
  - safety properties
  - liveness properties
- The resulting problem must be decidable
  - temporal structure
  - temporal logics $\longrightarrow$ e.g. **CTL**
  - proof system
- We restrict properties to **invariants**

Bounded Model Checking

Sequential designs: behaviour depending on inputs and time
$\longrightarrow$ Model checking: conversion into a combinational system



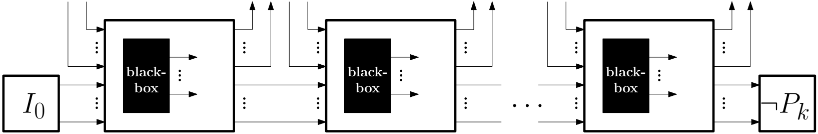
Iteratively unfold the system k times. The SAT-based BMC formula

$$I_0 \wedge T_{0,1} \wedge \ldots \wedge T_{k-1,k} \wedge \neg P_k$$

evaluates to $\top$ iff there exists a counterexample of length k that violates the safety property.

Bounded Model Checking for Incomplete Designs

Sequential designs: behaviour depending on inputs and time
$\longrightarrow$ Model checking: conversion into a combinational system



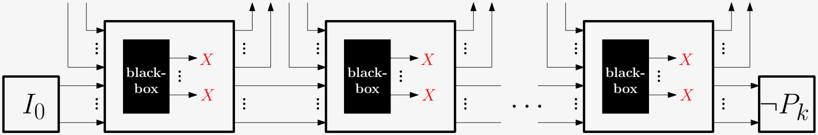
Iteratively unfold the system k times. The SAT-based BMC formula

$$I_0 \wedge T_{0,1} \wedge \ldots \wedge T_{k-1,k} \wedge \neg P_k$$

evaluates to $\top$ iff there exists a counterexample of length k that violates the safety property **regardless of the implementation of the blackbox.**

BB-BMC: Limits [Herbstritt et al. 2006]

`01X`**-modeling**: apply the value X to all blackbox outputs



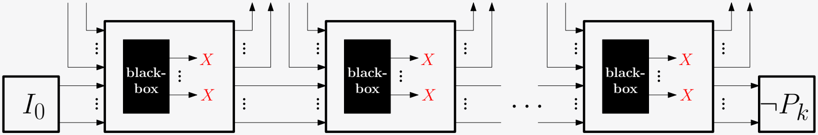
- 3-valued encoding [Jain 2000]
- transformation to CNF [Tseitin 1968]

$\Rightarrow$ SAT problem

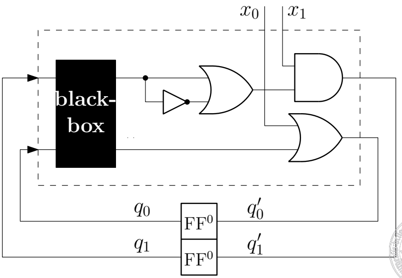
- X may “propagate” to $\neg P_k$

BB-BMC: Limits [Herbstritt et al. 2006]

`01X`-**modeling**: apply the value X to all blackbox outputs

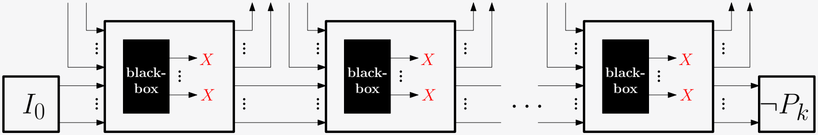


- $\neg P_k$: $(q_0 \wedge q_1)$
- q_1 evaluates to 0 or X
- counterexample not found

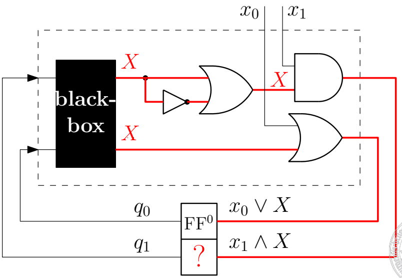


BB-BMC: Limits [Herbstritt et al. 2006]

`01X`**-modeling**: apply the value X to all blackbox outputs



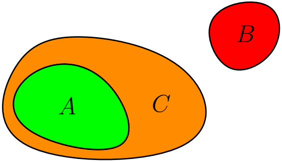
- $\neg P_k$: $(q_0 \wedge q_1)$
- q_1 evaluates to 0 or X
- counterexample not found



BMC and Craig Interpolation [McMillan 2003]

$$\underbrace{I_0 \wedge T_{0,1}}_{A} \wedge \underbrace{T_{1,2} \wedge \ldots \wedge T_{k-1,k} \wedge \neg P_k}_{B}$$

- Craig interpolant C of A and B
  - Over-approximation of the reachable states
  - Implied by A
  - Contains only AB-common variables (here: latches)
  - Unsatisfiable in conjunction with B

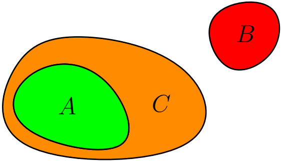


- If a fixpoint of the reachable states reached
  $\Rightarrow$ unsatisfiable for every unfolding depth $\Rightarrow$ `01X`-hard

BMC and Craig Interpolation [McMillan 2003]

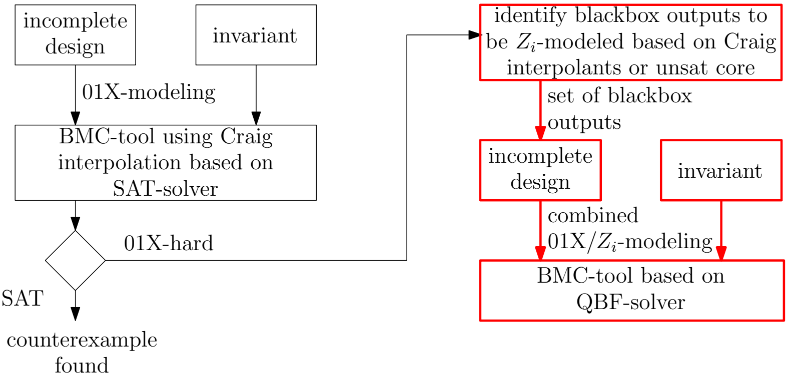
$$\underbrace{I_0 \wedge T_{0,1}}_{A} \wedge \underbrace{T_{1,2} \wedge \ldots \wedge T_{k-1,k} \wedge \neg P_k}_{B}$$

- Craig interpolant C of A and B
  - Over-approximation of the reachable states
  - Implied by A
  - Contains only AB-common variables (here: latches)
  - Unsatisfiable in conjunction with B



- If a fixpoint of the reachable states reached
  $\Rightarrow$ unsatisfiable for every unfolding depth $\Rightarrow$ `01X`-hard

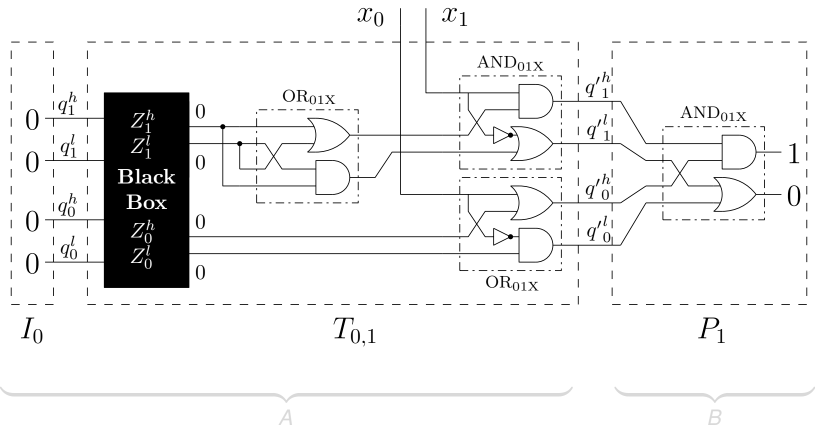
BB-BMC Workflow [Miller et al, 2010]



Heuristics for Identifying Blackbox Outputs

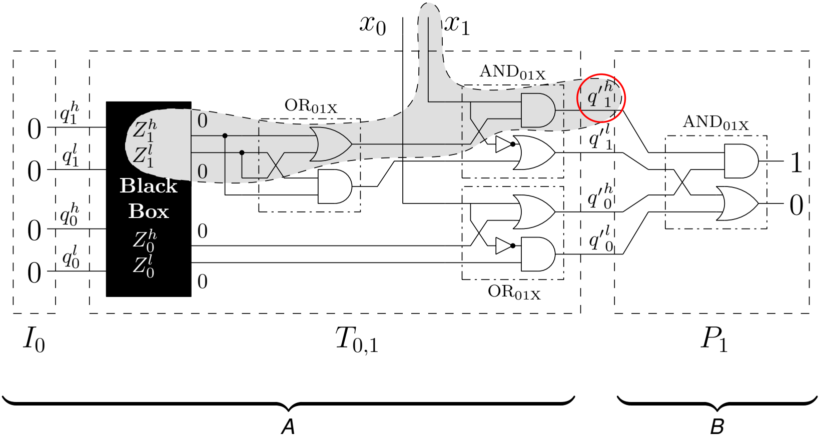
- **Exploiting Craig interpolant**
  - Analyze last computed Craig interpolant C
  - Perform cone-of-influence analysis on all latches in C
  - Model all blackbox outputs influencing these latches using Z_i
- **Exploiting unsatisfiable core**
  - Determine unsatisfiable core at unfolding depth where the fixed-point was found
  - Blackbox outputs included in this unsatisfiable core directly influence the unsatisfiability of the problem
  - Model these blackbox outputs using Z_i

Exploiting Craig Interpolants



- Derived Craig interpolant $C = \neg(q'^h_1)$
- Z_1 has influence on latch in C.
- Model Z_1 using Z_i and Z_0 using 01X.

Exploiting Craig Interpolants



- Derived Craig interpolant $C = \neg(q'^h_1)$
- Z_1 has influence on latch in C.
- Model Z_1 using Z_i and Z_0 using 01X.

Logic of Quantified Boolean Formulas

Syntax of QBF

Let $x_1, \ldots, x_n$ be a set of variables. A QBF logic is defined through the following inductive process:

- Every propositional logic formula and every variable x_i are QBF formulas.
- The constants **true** (1, $\top$) and **false** (0, $\bot$) are QBF formulas.
- For every QBF formula F, $\exists x F$ and $\forall x F$ are QBF formulas.
- For every formula F_1 and F_2, $\neg F_1$, $(F_1 \wedge F_2)$, and $(F_1 \vee F_2)$ are QBF formulas.

Logic of Quantified Boolean Formulas

Definition (Semantics of QBF Logic)

An assignment $\mathscr{A}_x : \{x_1, \ldots, x_n\} \rightarrow \{0, 1\}$ is a mapping that assigns either the value 0 or 1 to a variable of the formula and satisfies the following conditions:

- For each variable x_i contained in F:
  - $\mathscr{A}(x_i) = \mathscr{A}_x(x_i)$.
- For each constant formula 0 or 1:
  - $\mathscr{A}(0) = 0$, $\mathscr{A}(1) = 1$
- For each subformula F_1 and F_2 of F:
  - $\mathscr{A}(F_1 \wedge F_2) = 1 \Leftrightarrow \mathscr{A}(F_1) = 1$ and $\mathscr{A}(F_2) = 1$.
  - $\mathscr{A}(F_1 \vee F_2) = 1 \Leftrightarrow \mathscr{A}(F_1) = 1$ or $\mathscr{A}(F_2) = 1$.
- For each subformula F' of F:
  - $\mathscr{A}(\neg F') = 1 \Leftrightarrow \mathscr{A}(F') = 0$.
  - $\mathscr{A}(\exists x_i F') = 1 \Leftrightarrow \mathscr{A}_x(F') \text{or} \mathscr{A}_{\neg x}(F') = 1$.
  - $\mathscr{A}(\forall x_i F') = 1 \Leftrightarrow \mathscr{A}_x(F') = \mathscr{A}_{\neg x}(F') = 1$.

Logic of Quantified Boolean Formulas

The prefix defines the dependencies among the variables in a linear way. Given a formula F whose prefix is $Q_1 x_1 Q_2 x_2 \ldots Q_n x_n$

Definition (Quantifier alternations)

We define quantifier alternation as the number of switchings between $\forall$ and $\exists$ quantifiers reading the prefix from left to right.

Definition (Level of a variable)

The level of a variable x_i is 1 plus the number of alternations that precede it. It is common to use the terms outermost quantifier level to indicate the level 0, and innermost quantifier level to indicate that "beside" the matrix.

Logic of Quantified Boolean Formulas

Definition (Prenex Conjunctive Normal Form, PCNF)

A QBF formula F is in prenex conjunctive normal form (PCNF) iff it is a prefix and a matrix, where the matrix is a conjunction of clauses:

$$F = Q_1 X_1 Q_2 X_2 \dots Q_n X_n \bigwedge_{j=1}^{m} C_j \qquad \text{with } C_1, \dots, C_m \text{ Clauses}$$

- Example: $\exists x_1 \exists x_2 \forall x_3 \exists x_4 (x_1 \vee \neg x_2 \vee x_3) \wedge (x_2 \vee x_4)$
- An assignment $\mathscr{A}$ satisfies a CNF formula F iff every clause in F is satisfied. Wrong! We have to follow the semantics imposed by the prefix.

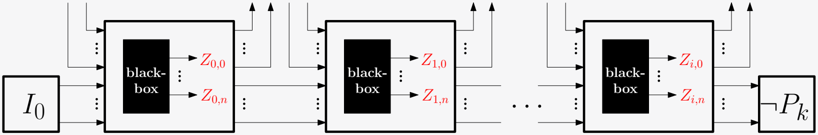
Z_i-Encoding: PEC

- Blackbox outputs act as additional primary inputs
- Encoded as universal variables
- "No matter what the Blackbox does" the rest must hold
- Exact if the system includes 1 Blackbox
- Additional constraints to make the Blackbox's output consistent (combinational)
- Compromise between precision and speed

$$\exists \overline{X} \forall Z_i, \exists \overline{Y}, M.(\phi(\overline{X}, Z_i, \overline{Y}, M) \wedge (M \equiv 1))_{CNF} \quad ?$$

Z_i-Encoding: BB-BMC

Z_i-modeling: use one ∀-variable for each blackbox output



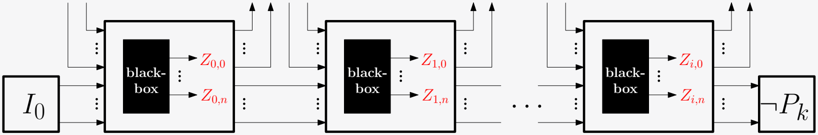
- Blackbox outputs are **universally quantified**
- Tseitin transformation
- Prefix generation (see next slide)

⇒ QBF problem

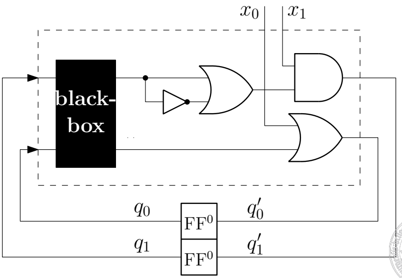
- more precise

Z_i-Encoding: BB-BMC

$\mathbf{Z_i}$-modeling: use one $\forall$-variable for each blackbox output

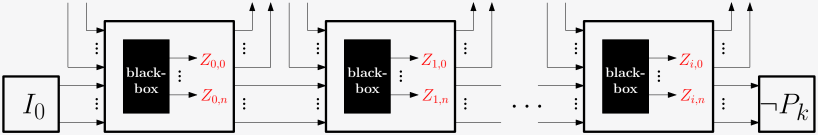


- $\neg P_k$: $(q_0 \wedge q_1)$
- $\exists x_0 x_1 \ \forall \mathbf{Z_0 Z_1} \ \exists \vec{H}$ CNF
- satisfied for $x_0 = 1, x_1 = 1$

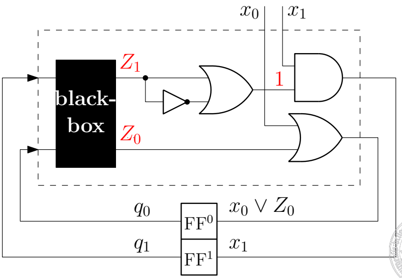


Z_i-Encoding: BB-BMC

$\mathbf{Z_i}$-modeling: use one $\forall$-variable for each blackbox output



- $\neg P_k$: $(q_0 \wedge q_1)$
- $\exists x_0 x_1 \;\; \forall \mathbf{Z_0 Z_1} \;\; \exists \vec{H}$ CNF
- satisfied for $x_0 = 1, x_1 = 1$



Z_i-Encoding: BB-BMC

Non-uniform quantifier prefix (*pref*$_1$):

$$\underbrace{\exists x_{0,0},\ldots,x_{n,0}\ \forall Z_{0,0},\ldots,Z_{m,0}\ \exists H_0}_{\text{depth 0}}\ldots\underbrace{\exists x_{0,k},\ldots,x_{n,k}\ \forall Z_{0,k},\ldots,Z_{m,k}\ \exists H_k}_{\text{depth k}}$$

- inputs can “react” to the values of the blackbox outputs
- $2\cdot(k+1)$ quantifier alternations

Uniform quantifier prefix (*pref*$_2$):

$$\underbrace{\exists x_{0,0},\ldots,x_{n,k}}_{\substack{\text{primary inputs}\\ \text{depth 0...k}}}\ \underbrace{\forall Z_{0,0},\ldots,Z_{m,k}}_{\substack{\text{blackbox outputs}\\ \text{depth 0...k}}}\ \underbrace{\exists H_0,\ldots,H_k}_{\substack{\text{Tseitin}\\ \text{depth 0...k}}}$$

- exactly one input sequence
- 2 quantifier alternations
- $pref_2 \implies pref_1$

Incremental SAT [Een 2003]

- Incremental SAT Problem: within a loop, the input formula is augmented by new sub-expressions
- Advantages: reuse of conflict clauses and decision heuristic scores
- Assumptions to unconstrain running formula

Use of Assumptions

$\varphi = (x \vee y) \wedge (\neg x \vee y \vee z)$	non incremental
$\varphi_{0/\neg w} = (x \vee y) \wedge (\neg x \vee y \vee z \vee w)$	incr. step 0
$\varphi_{1/w} = (x \vee y) \wedge (\neg x \vee y \vee z \vee w) \wedge (\ldots)$	incr. step 1

Incremental QBF Solving Problem

- step 0: $\Phi_0 = Q_1 X_1^0 \ldots Q_n X_n^0 \ \phi_0$
- ...
- step i: $\Phi_i = Q_1 X_1^0 \cdots X_1^i \ldots Q_n X_n^0 \cdots X_n^i \ \phi_{i-1} \setminus \phi_i^- \ \wedge \phi_i^+$

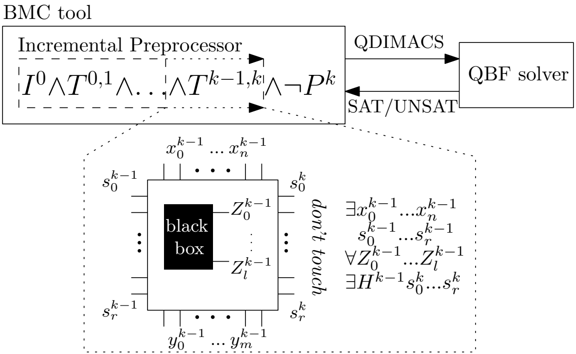
Incremental QBF Solving Problem

- step 0: $\Phi_0 = Q_1 X_1^0 \ldots Q_n X_n^0 \; \phi_0$
- $\ldots$
- step i: $\Phi_i = Q_1 X_1^0 \cdots X_1^i \ldots Q_n X_n^0 \cdots X_n^i \; \phi_{i-1} \setminus \phi_i^- \wedge \phi_i^+$

Howto

- Assumption-based solving
- Add new variables to existing quantifier blocks
- Add new quantifier blocks
- Add and delete clauses
- Avoid memory reallocation and fragmentation
- Keep learned clauses, learned solution cubes only in special cases

Incremental QBF Preprocessing [Miller et al., 2012]



- Idea: keep a compact (preprocessed) representation of current unfolding $I_0 \wedge T_{0,1} \wedge \ldots \wedge T_{k-1,k}$ in the incremental preprocessor.
- Preserve interface using *dont-touch/frozen* variables.

Standard/Incremental BMC Procedures

BMC tool		QBF solver		
$I_0 \wedge \neg P_0$	$\rightarrow$	preprocess	$\rightarrow$	solve
$I_0 \wedge T_{0,1} \wedge \neg P_1$	$\rightarrow$	preprocess	$\rightarrow$	solve
$I_0 \wedge T_{0,1} \wedge T_{1,2} \wedge \neg P_2$	$\rightarrow$	preprocess	$\rightarrow$	solve
$\vdots$		$\vdots$		$\vdots$
$I_0 \wedge T_{0,1} \wedge T_{1,2} \wedge \ldots \wedge T_{k-1,k} \wedge \neg P_k$	$\rightarrow$	preprocess	$\rightarrow$	solve

Standard BMC

- For each unfolding, the QBF formula is directly passed to the QBF solver.
- The QBF solver may invoke a preprocessor before solving the formula.
- At every step the whole formula is preprocessed!

Standard/Incremental BMC Procedures

BMC tool			QBF solver
$I_0 \wedge \neg P_0$	$\rightarrow$	preprocess $\rightarrow$	solve
			$\downarrow$
$I_0 \wedge T_{0,1} \wedge \neg P_1$	$\rightarrow$	preprocess $\rightarrow$	solve
			$\downarrow$
$I_0 \wedge T_{0,1} \wedge T_{1,2} \wedge \neg P_2$	$\rightarrow$	preprocess $\rightarrow$	solve
			$\downarrow$
$\vdots$		$\vdots$	$\vdots$
			$\downarrow$
$I_0 \wedge T_{0,1} \wedge T_{1,2} \wedge \ldots \wedge T_{k-1,k} \wedge \neg P_k$	$\rightarrow$	preprocess $\rightarrow$	solve

Incremental Solving

- Reuse learnt information during solving process.
- Preprocessor may eliminate variables and delete/merge/add clauses.
  - $\rightarrow$ Learnt information not valid anymore $\rightarrow$ deactivate preprocessing
  - $\rightarrow$ At least pre-preprocess the transition relation

Standard/Incremental BMC Procedures

BMC tool				QBF solver
$I_0 \wedge \neg P_0$	$\rightarrow$	preprocess	$\rightarrow$	solve
		$\downarrow$		
$I_0 \wedge T_{0,1} \wedge \neg P_1$	$\rightarrow$	preprocess	$\rightarrow$	solve
		$\downarrow$		
$I_0 \wedge T_{0,1} \wedge T_{1,2} \wedge \neg P_2$	$\rightarrow$	preprocess	$\rightarrow$	solve
		$\downarrow$		
$\vdots$		$\vdots$		$\vdots$
		$\downarrow$		
$I_0 \wedge T_{0,1} \wedge T_{1,2} \wedge \ldots \wedge T_{k-1,k} \wedge \neg P_k$	$\rightarrow$	preprocess	$\rightarrow$	solve

Incremental Preprocessing

- Move preprocessor to the BMC tool.
- Reuse the preprocessed QBF formula for the construction of the next unfolding.

Standard/Incremental BMC Procedures

Incremental Reasoning

- Incremental preprocessing can become slow
- Hybrid way: eventually switch from incremental preprocessing to incremental solving

Standard/Incremental BMC Procedures

Incremental Reasoning

- Because of the linear prefix a QBF formula can be incrementally extended “to the left”, “to the right”, or keeping the initial quantifier alternations
- Backwards incremental solving more efficient (solution learning)
- Forwards incremental preprocessing is more effective (Tseitin auxiliary variables elimination)
- Matter of heuristics

`01X` vs Z_i

- Uniform faster
- `01X` fastest
- Uniform more precise
- Non-uniform most precise
- ...
- The “right solver” can speed up the process
- How to select the right encoding for a specific Blackbox?

Is Z_i-QBF Precise Enough?

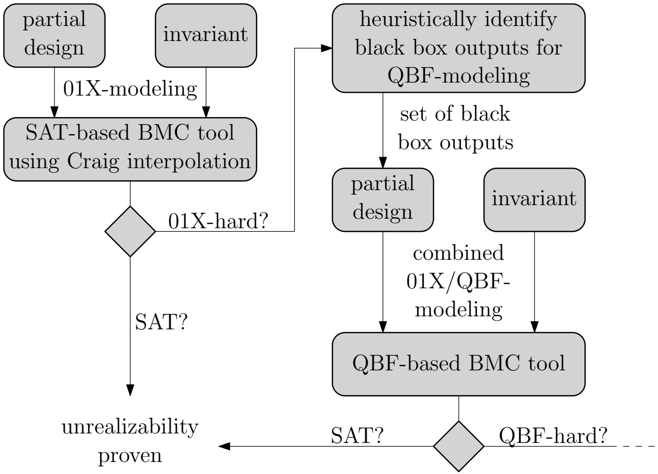
PEC

A Z_i-encoded PEC problem is exact iff its underapproximation (BB-outputs depending on all inputs) and its overapproximation (some or all BB-outputs are independent on some primary inputs) return the same result.

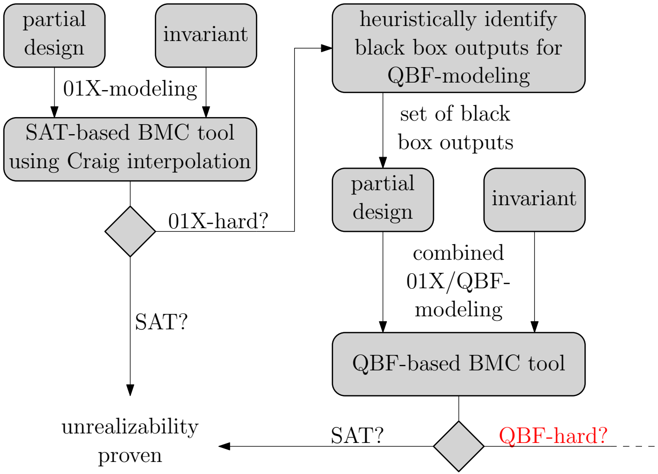
BB-BMC

Approximate if multiple Blackboxes do not have complete knowledge about the inputs

BB-BMC Verification Workflow



BB-BMC Verification Workflow



BB-BMC: Is Z_i-QBF precise enough?

QBF-Hardness [Miller et al., 2013]

A partial design is QBF-hard iff the (pure) Z_i-modeled BMC problem is unsatisfiable for all unfoldings and the property is definitely realizable.

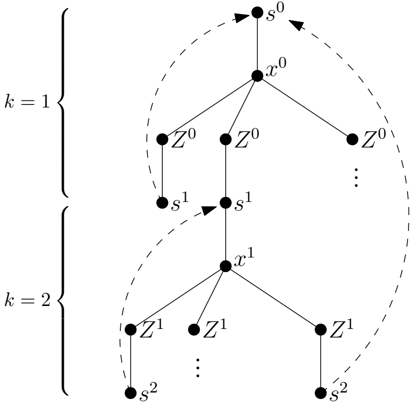
$\Rightarrow$ Prove QBF-hardness using the following iterative procedure…

Proving QBF-Hardness

Iteratively search for graph with the following properties:

(1) s^0 fulfills P

(2) For each x^i there exists a Z^i such that s^{i+1}
- is either equivalent to a state which was explored before
- or it fulfills P and (2) for next i



- Formulate procedure as QBF problems
- If such graph exists the design is QBF-hard
- Otherwise need “higher” logic

Dependency Quantified Boolean Formulas

- Generalization of QBF
- Allow arbitrary partially ordered dependencies
- Dependencies of existential variables on universal ones explicitly stated
- Variable order in the prefix irrelevant

Example

$$\forall x_1 \forall x_2 \exists y_1(x_1) \exists y_2(x_2) : \varphi$$

- y_1 depends only on x_1
- y_2 depends only on x_2

Dependency Quantified Boolean Formulas

- Generalization of QBF
- Allow arbitrary partially ordered dependencies
- Dependencies of existential variables on universal ones explicitly stated
- Variable order in the prefix irrelevant

Example

$$\forall x_1 \forall x_2 \exists y_1(x_1) \exists y_2(x_2) : \varphi$$

- y_1 depends only on x_1
- y_2 depends only on x_2

Semantics of QBF and DQBF

QBF:

$$\forall x_1 \exists y_1 \forall x_2 \exists y_2 : \varphi$$

is satisfied iff there are functions s_{y_1} and s_{y_2} such that replacing y_1 with $s_{y_1}(x_1)$ and y_2 with $s_{y_2}(x_1, x_2)$ yields a tautology.

Semantics of QBF and DQBF

QBF:

$$\forall x_1 \exists y_1 \forall x_2 \exists y_2 : \varphi$$

is satisfied iff there are functions s_{y_1} and s_{y_2} such that replacing y_1 with $s_{y_1}(x_1)$ and y_2 with $s_{y_2}(x_1, x_2)$ yields a tautology.

DQBF:

$$\forall x_1 \forall x_2 \exists y_1(x_1) \exists y_2(x_2) : \varphi$$

is satisfied iff there are functions s_{y_1} and s_{y_2} such that replacing y_1 with $s_{y_1}(x_1)$ and y_2 with $s_{y_2}(x_2)$ yields a tautology.

Semantics of QBF and DQBF

QBF:

$$\forall x_1 \exists y_1 \forall x_2 \exists y_2 : \varphi$$

is satisfied iff there are functions s_{y_1} and s_{y_2} such that replacing y_1 with $s_{y_1}(x_1)$ and y_2 with $s_{y_2}(x_1, x_2)$ yields a tautology.

DQBF:

$$\forall x_1 \forall x_2 \exists y_1(x_1) \exists y_2(x_2) : \varphi$$

is satisfied iff there are functions s_{y_1} and s_{y_2} such that replacing y_1 with $s_{y_1}(x_1)$ and y_2 with $s_{y_2}(x_2)$ yields a tautology.

$\Rightarrow$ s_{y_1} and s_{y_2} are called **Skolem functions.**

Semantics of QBF and DQBF

QBF:

$$\forall x_1 \exists y_1 \forall x_2 \exists y_2 : \varphi$$

is satisfied iff there are functions s_{y_1} and s_{y_2} such that replacing y_1 with $s_{y_1}(x_1)$ and y_2 with $s_{y_2}(x_1, x_2)$ yields a tautology.

DQBF:

$$\forall x_1 \forall x_2 \exists y_1(x_1) \exists y_2(x_2) : \varphi$$

is satisfied iff there are functions s_{y_1} and s_{y_2} such that replacing y_1 with $s_{y_1}(x_1)$ and y_2 with $s_{y_2}(x_2)$ yields a tautology.

$\Rightarrow$ s_{y_1} and s_{y_2} are called **Skolem functions.**

Easy example of cyclic dependency.

Complexity

- **SAT:**
  Deciding satisfiability of SAT is NP-complete
- **QBF:**
  Deciding satisfiability of QBF is PSPACE-complete
- **DQBF:**
  Deciding satisfiability of DQBF is NEXPTIME-complete

Preprocessing: necessary

Inherited from QBF

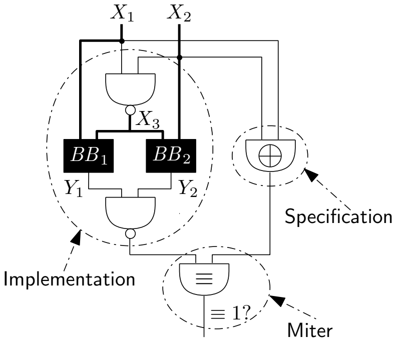
- Syntactic and semantic unit literal elimination
- Pure literal elimination
- Equivalent literals
- Blocked clauses elimination
- Universal variable expansion $\Rightarrow$ simplification into QBF
- Dependency sets can be cyclic!

Newly developed for DQBF [Wimmer et al., 2015]

- More expensive checks are effective
- Variable elimination via D-Q-Resolution more restricted
- Reduction of dependency sets

PEC via DQBF [Gitina et al. 2013]

Are there implementations of the Blackboxes such that implementation and specification become equivalent?



DQBF formulation:

$$\forall X_1 \forall X_2 \forall X_3 \exists Y_1(X_1, X_3) \exists Y_2(X_2, X_3) : \varphi$$

The Blackboxes are in topological order to guarantee that the circuit is combinational

BB-BMC via DQBF

- We can prove which Blackboxes require to be modeled using DQBF
- How to encode the problem is clear
- Some DQBF solvers are available
- A general and precise BB-BMC tool is definitely demanded
- Robustness and good performance are needed

BB-BMC via DQBF

- We can prove which Blackboxes require to be modeled using DQBF
- How to encode the problem is clear
- Some DQBF solvers are available
- A general and precise BB-BMC tool is definitely demanded
- Robustness and good performance are needed
- So far: open problem/future work

Summary

- Formal verification of incomplete discrete systems
- Partial equivalence checking of combinational circuits
- Blackbox-bounded model checking of sequential systems
- SAT: `01X` modeling fastest, well studied core algorithms and data structures, suitable to simplest topologies
  ⋆ NP-Complete
- QBF: Z_i modeling stable technology, industrial acceptance required ⋆ PSPACE-Complete
- DQBF: **explicit dependency** encoding, young and currently under development ⋆ NEXPTIME-Complete