Deductive Methods, Bounded Model Checking

http://d3s.mff.cuni.cz

Department of Distributed and Dependable Systems

Pavel Parízek

CHARLES UNIVERSITY IN PRAGUE
faculty of mathematics and physics
Deductive methods
If you want to know more ...

- Decision Procedures and Verification (NAIL094)
  - Lecturer: Martin Blicha, D3S
  - [http://d3s.mff.cuni.cz/teaching/decision_procedures/](http://d3s.mff.cuni.cz/teaching/decision_procedures/)

Basic terminology (reminder)

- Logic formula
  - syntax, semantics

- Propositional logic

- First-order logic
  - Predicates
  - Quantifiers

- Assignment
  - Partial assignment

- Satisfiability

- Validity (tautology)
Relation between satisfiability and validity

φ is valid $\rightarrow$ φ is satisfiable

φ is valid $\iff$ !φ is unsatisfiable

φ is satisfiable $\iff$ !φ is not valid
Normal forms

- Negation normal form (NNF)
  - syntax: !, |, & and variables
  - Negation only for variables
  - Example: \((a \mid (b \& \neg c)) \& (\neg d)\)

- Conjunctive normal form (CNF)
  - NNF as a conjunction of disjunctions
  - Example: \((a \mid b \mid \neg c) \& (\neg d) \& (e \mid \neg f)\)

- Disjunctive normal form (DNF)
  - NNF as a disjunction of conjunctions
  - Example: \((a \& b \& \neg c) \mid (\neg d) \mid (e \& \neg f)\)
Getting the normal forms

- De Morgan’s law
- Distributive law

Q: Is there a problem with conversion?
Getting the normal forms

- Transformation into an equivalent formula in CNF or DNF

- Problem: exponential blow-up of the size

- Remedy: creating *equisatisfiable* formula
Equisatisfiability

- Equisatisfiable formulas $\phi$, $\psi$
  - both satisfiable or both unsatisfiable

Examples

$\phi$: $!(a \rightarrow b)$  
$\psi$: $a \& \!b$

$\phi$: $a \mid b$
$\psi$: $(a \mid n) \& (\!n \mid b)$

$\phi$: $a \& b \& \!c$
$\psi$: true

$\phi$: $!a \leftrightarrow b$
$\psi$: false
Equisatisfiability

- Equisatisfiable formulas $\phi$, $\psi$
  - both satisfiable or both unsatisfiable

Examples

- $\phi$: !(a $\rightarrow$ b)  $\psi$: a $\&$ !b  \quad$EQ, ES$
- $\phi$: a $|$ b  $\psi$: (a $|$ n) $\&$ (!n $|$ b)  \quad$ES$
- $\phi$: a $\&$ b $\&$ !c  $\psi$: true  \quad$ES$
- $\phi$: !a $\leftrightarrow$ b  $\psi$: false  \quad$-$
• Tseitin’s encoding
  - Widely used algorithm for transforming a given propositional formula $\phi$ into an equisatisfiable formula $\phi'$ in CNF with linear growth only

• Practice: various optimizations applied
SAT solving
SAT solving

• Goal
  ▪ Decide whether a given propositional formula $\phi$ in CNF is satisfiable

• Possible answers
  ▪ Satisfiable + assignment (values, model)
  ▪ Unsatisfiable + core (subset of clauses)

• Satisfiable formula $\phi \iff$ there exists a partial assignment satisfying all clauses in $\phi$
NAIVE brute force solution

- Trying all possible assignments
  - Systematic traversal of a binary tree

DPLL (Davis-Putnam-Loveland-Logemann)

- Motivation: partial assignment can imply values of other variables in the given formula
- Example: from $(\neg a \lor b)$, $v = \{ a \rightarrow 1 \}$ we get $\{ b \rightarrow 1 \}$
- Approach: iterative deduction
  - Inferring value of a particular variable
- Basic algorithm used in modern SAT solvers (with many additional optimizations) $\Rightarrow$ DPLL-based SAT solving
SAT solving: optimizations

- Adding learned clauses (implied)
- Non-chronological backtracking
- Choice of the branching variable
  - Various heuristics on the best choice exist

- Restarts
  - When it takes too long, restart the solver and use other “seeds” for heuristic functions
SAT solving

- Problem size: 10K – 1M variables
  - Typical input formulas have structure
- Worse for random instances
- Hard instances exist (of course)
- Tools are getting better all the time
  - Reason: industry demand, annual competitions
  - [http://www.satcompetition.org/](http://www.satcompetition.org/)

- Other approaches
  - Stochastic search (random walk)
    - Quickly finds solution for satisfiable instances
  - Ordered binary decision diagrams
Propositional logic: semantic X proof

- Semantic domain $\models$
  - Goal: find satisfying assignment for $\varphi$

- We know that: $\models \varphi \iff \vdash \varphi$

- Proof domain $\vdash$
  - Goal: derive the proof
  - axioms, inference rules
Resolution

- Input: CNF formula $\phi$ (a set of clauses)

- Goal: derive empty clause ($false$)

- Iterative process
  - Choose two suitable clauses from the set
    - Requirement: they must have complementary literals $r, \neg r$
  - Apply resolution step on these clauses
    $$(p_1 \lor \ldots \lor p_N \lor r), (q_1 \lor \ldots \lor q_N \lor \neg r) \Rightarrow (p_1 \lor \ldots \lor p_N \lor q_1 \lor \ldots \lor q_N)$$
  - Add the newly derived clause into the set
  - Repeat until we derive $false$ (or fail/stop)
Resolution

• Equivalent statements
  1) CNF formula $\phi$ is unsatisfiable
  2) We can derive empty clause using resolution on the clauses from $\phi$

• Resolution used in practice
  ▪ Checking validity of a first-order logic formula
  ▪ Proof-by-contradiction
    ▪ Add negation of the conjecture into the set
SAT solving and propositional logic

- SAT looks very good, but we need more
  - For program verification, full theorem proving, ...

- First-order logic (predicate logic)
- Interesting theories
  - Linear integer arithmetic ($\mathbb{N}, \mathbb{Z}$)
  - Data structures (arrays, bit vectors)
Decision procedure
Decision procedure

- Algorithm that
  - Always terminates
  - Outputs: YES/NO

- Decision procedure for a particular theory T
  - Always terminates and provides a correct answer for every formula of T
  - Goal: checking validity of logic formulas
Interesting theories

- Equality logic
  - With uninterpreted functions
- Linear arithmetic
  - Integer
  - Rational
- Difference logic
- Arrays
- Bit vectors
Equality logic

- Syntax
  - Atomic formulas
    \[ term = term \mid true \mid false \]
  - Terms
    \[ variable \mid constant \]

- Deciding validity of an equality logic formula is NP-complete problem
- Polynomial algorithm exists for the conjunctive fragment (uses only \& and \( \exists \))
Equality logic with uninterpreted functions

• Syntax
  ▪ Atomic formulas
    \[ \text{term} = \text{term} \mid \text{predicate}(\text{term}, \ldots, \text{term}) \mid \text{true} \mid \text{false} \]
  ▪ Terms
    \[ \text{variable} \mid \text{constant} \mid \text{function}(\text{term}, \ldots, \text{term}) \]

• Semantics
  ▪ No implicit meaning of functions and predicates
  ▪ \[ a_1 = b_1 \land \ldots \land a_N = b_N \rightarrow f(a_1, \ldots, a_N) = f(b_1, \ldots, b_N) \]

• Decision procedure
  ▪ Transform into an equisatisfiable formula in equality logic
Equality logic with uninterpreted functions

- **Purpose:** abstraction
  - Full formula $\Rightarrow$ function semantics defined using axioms
  - Uninterpreted symbols $\Rightarrow$ just equality between arguments
  - $\models \phi^\text{EUF} \rightarrow \models \phi$

- **False answers possible**
  - Example: $add(1,2) \neq add(2,1)$ in EUF

- Formula with UF easier to decide than the “full” formula
Linear arithmetic

- **Syntax**
  - Atomic formulas
    
    \[
    \text{term} = \text{term} \mid \text{term} < \text{term} \mid \text{term} \leq \text{term} \mid \text{true} \mid \text{false}
    \]
  - Terms
    
    \[
    \text{variable} \mid \text{constant} \mid \text{constant variable} \mid \text{term} + \text{term}
    \]

- **Example:** \((3x + 2y \leq 5z) \land (2x - 2y = 0)\)

- **Arithmetic without multiplication** → Presburger arithmetic

- **Decision procedure**
  - General case (full theory): \(2^{2\Omega(n)}\)
  - Conjunctive fragment over \(\mathbb{Q}\)
    - Linear programming: Simplex method (EXP), Ellipsoid method (P)
  - Conjunctive fragment over \(\mathbb{Z}\)
    - Integer linear programming (NP-complete)
Difference logic

• Syntax
  - Atomic formulas
    \[ \text{variable} - \text{variable} < \text{constant} \mid \]
    \[ \text{variable} - \text{variable} \leq \text{constant} \mid \]
    \[ \text{true} \mid \text{false} \]
  - Operators: \(!, \&, \leftarrow, \leftrightarrow\)

• Example: \((x - y < 3) \land (y - z \leq -4) \land (z - x \leq 1)\)

• Decision procedure
  - Conjunctive fragment polynomial for \(\mathbb{Q}\) and \(\mathbb{Z}\)
Data structures

- Array theory
  - Function symbols
    - \texttt{select}(a,i) \quad // \text{read, } a[i]
    - \texttt{store}(a,i,e) \quad // \text{update, } a[i] = e
  - Axiom \textbf{read-over-write}
    - \texttt{select}(\texttt{store}(a,i,e),i) = e

- Bit vectors
  - Motivation: precise computer arithmetic (overflows, ...)
  - Reasoning about individual bits in a finite vector (array)
  - Syntax: operators bitwise-AND, bitwise-OR, bitwise-XOR
  - Decision procedure
    - Typically flattened into a large instance of SAT
    - Many clever optimizations (encoding)
Combining theories

- Goal
  - Formulas that combine multiple theories
  - Example: linear arithmetic + arrays

- Decision procedures
  - Combined under specific constraints

- Nelson-Oppen method
Decision procedures: summary

- Decision procedures
  - Typically work for conjunctive fragments of the respective theories

- But we still need more
  - Formulas with arbitrary boolean structure and interesting theories (linear arithmetic, arrays)
Goal

- Decide satisfiability of a quantifier-free formula that involves constructs of specific theories

Idea

- Using combination of a SAT solver and a decision procedure (DP) for a conjunctive fragment of the respective theory
Naive use of a SAT solver

1. Extract boolean skeleton of the given formula $\phi$
2. Run the SAT solver on the boolean skeleton
   a) unsatisfiable $\Rightarrow$ the input formula is unsatisfiable
   b) satisfiable $\Rightarrow$ we get a satisfying assignment $\nu$
3. Run the DP on the formula derived from the satisfying assignment $\nu$
   a) satisfiable $\Rightarrow$ the input formula is satisfiable
   b) unsatisfiable $\Rightarrow$ add the blocking clause for $\nu$ to the boolean skeleton and continue with the step 2
Approaches to SMT

- **DPLL(T)-based SMT solving**
  - Eagerness: DPLL asks DP for partial assignments during traversal
    - Benefit: earlier conflict discovery
  - Updating the set of clauses given to DP on-the-fly
    - iteration (add), backtracking (remove)
- **Theory-based learning**
  - DP can identify clauses valid/invalid in the given theory T
Available SMT solvers
- Z3, CVC4, Yices, MathSAT 5, OpenSMT, ...

SMT-LIB v2
- Defines common input format
- Big library of SMT problems
- http://www.smt-lib.org/

SMT-COMP
- Competition of SMT solvers
- http://smtcomp.org
SMT solving in practice

• Current state
  ▫ Good performance
  ▫ Highly automated
  ▫ Many applications

• Drawbacks
  ▫ Restricted to specific theories and domains ($\mathbb{Q}$, $\mathbb{Z}$)
  ▫ Very limited support for quantifiers (mostly $\exists$)
  ▫ Much less powerful than full theorem proving
Theorem proving

- **Input**
  - Theory T: set of axioms
  - General formula $\phi$ in predicate logic

- **Goal**
  - Decide validity of the formula $\phi$ in T
    - Semantic domain: show unsatisfiable negation
    - Proof domain: prove $\phi$ from the axioms of T

- Very powerful
- Interactive
  - Partially automated

- Tools: PVS, Isabelle/HOL
Deductive methods: closing remarks

• Approaches
  - DPLL-based SAT solving
  - Decision procedures
  - DPLL(T)-based SMT solving

• Formulas
  - Propositional logic (boolean)
  - Predicate logic with theories
    • Equality with uninterpreted functions
    • Linear arithmetic (difference logic)
    • Data structures (arrays, bit vectors)

• Applications in program verification
Bounded model checking
Bounded model checking

- Goal: Exploring traces with bounded length
  - Options: fixed integer value $K$, iteratively increasing
  - Still remember preemption bounding for threads?

- Approach
  - Encoding bounded program state space and properties into a logic formula $\phi$
  - Find property violations by checking satisfiability of $\phi$

- Challenge
  - Encoding program behavior into the formula $\phi$
Program state space

- Program $P = (S, T, INIT)$
  - $S$ is a set of program states
    - Predicates about values of program variables
    - Program counter (PC)
  - $INIT \subseteq S$ is a set of initial states
  - $T \subseteq S \times S$ is a transition relation

- Single transition
  - Updates program counter and some variables
  - Relating old and new values $(x, x', pc, pc')$
  - Example: $x = 2, x' = x + 1, pc = 5, pc' = pc + 1$
\[(pc = 1) \land (x' = x + 2y) \land (pc' = pc + 1)\]
\[\lor\]
\[(pc = 2) \land (x' = 0) \land (pc' = pc + 6)\]
\[\lor\]
\[\ldots\]
\[\lor\]
\[(pc = N) \land (x' = x - y + 5) \land (pc' = pc + 1)\]
Traces with bounded length

- Transition relation unfolded at most K times
  - Fresh copies of program variables \((x, x', \ldots, x^{(K)})\) used for each unfolding of the transition relation

- Example
  - INIT: \(x = 0, pc = 1\)
  - \(T(K): ( ((pc = 1) \land (x' = x + 2y) \land (pc' = pc + 1)) \lor \ldots \ldots ) \)

- Specific consequences
  - Bounded number of loop iterations (unrolling)
Large formula

\[ \text{INIT}(s_0) \land ( \bigwedge_{i=0..k-1} T(s_i, s_{i+1}) ) \land ( \bigvee_{i=0..k} \neg p(s_i) ) \]

Represents all possible executions of the program with the length bounded by K
BMC: verification procedure

1) Derive formula representing the state space

2) Run the SAT/SMT solver on the formula in CNF

3) Interpret verification results
   - Satisfying assignment $\Rightarrow$ we get a counterexample with the length $\leq K$
   - Unsatisfiable formula $\Rightarrow$ no property violations in program executions of the length $\leq K$
BMC: technical challenges

- Encoding program in a mainstream language into a logic formula
  - heap, allocation, pointers, threads, synchronization

- Example: dynamic heap
  - Use predicate logic with array theory (*select, store*)
  - Array element access \( a[i] \)
    - Separate variables for the element \( a[i] \) and the index \( i \)
  - Pointer access \( (*p) \)
    - Separate variables for dereference \( *p \) and the pointer \( p \)
  - Transitions defined properly
Further reading
