Deductive Methods, Bounded Model Checking

http://d3s.mff.cuni.cz

Department of Distributed and Dependable Systems

D3S

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/

- D. Kroening and O. Strichman. Decision Procedures: An Algorithmic Point of View. Springer, 2008.

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 → φ is satisfiable

φ is valid ↔ !φ is unsatisfiable

φ is satisfiable ↔ !φ is not valid

Normal forms

- Negation normal form (NNF)
  - syntax: !, |, & and variables
  - Negation only for variables
  - Example: (*a* | (*b* & !*c*)) & (!*d*)
- Conjunctive normal form (CNF)
  - NNF as a conjunction of disjunctions
  - Example: (*a* | *b* | !*c*) & (!*d*) & (*e* | !*f*)
- Disjunctive normal form (DNF)
  - NNF as a disjunction of conjunctions
  - Example: (*a* & *b* & !*c*) | (!*d*) | (*e* & !*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 ϕ, ψ
  - both satisfiable or both unsatisfiable

- Examples

ϕ: $!(a \rightarrow b)$	ψ: a & $!b$	??
ϕ: $a \mid b$	ψ: $(a \mid n)$ & $(!n \mid b)$	??
ϕ: a & b & $!c$	ψ: true	??
ϕ: $!a \leftrightarrow b$	ψ: false	??

Department of Distributed and Dependable Systems

Equisatisfiability

- Equisatisfiable formulas ϕ, ψ
  - both satisfiable or both unsatisfiable

- Examples

ϕ: $!(a \rightarrow b)$	ψ: a & $!b$	EQ, ES
ϕ: $a \mid b$	ψ: $(a \mid n)$ & $(!n \mid b)$	ES
ϕ: a & b & $!c$	ψ: true	ES
ϕ: $!a \leftrightarrow b$	ψ: false	–

Equisatisfiability

- Tseitin's encoding
  - Widely used algorithm for transforming a given propositional formula ϕ into an equisatisfiable formula ϕ' in CNF with linear growth only

- Practice: various optimizations applied

SAT solving

SAT solving

- Goal
  - Decide whether a given propositional formula ϕ in CNF is satisfiable
- Possible answers
  - Satisfiable + assignment (values, model)
  - Unsatisfiable + core (subset of clauses)
- Satisfiable formula $\phi \leftrightarrow$ there exists a partial assignment satisfying all clauses in ϕ

SAT solving

- 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 (!*a* | *b*), v = { *a* → 1 } we get { *b* → 1 }
  - Approach: iterative deduction
    - Inferring value of a particular variable
  - Basic algorithm used in modern SAT solvers (with many additional optimizations) → 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/

- Other approaches
  - Stochastic search (random walk)
    - Quickly finds solution for satisfiable instances
  - Ordered binary decision diagrams

Propositional logic: semantic X proof

- Semantic domain ⊨
  - Goal: find satisfying assignment for φ

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

- Proof domain ⊢
  - Goal: derive the proof
  - axioms, inference rules

Resolution

- Input: CNF formula ϕ (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*, !*r*
  - Apply resolution step on these clauses

    (p1 | ... | pN | ***r***), (q1 | ... | qN | **!*r***) → (p1 | ... | pN | q1 | ... | qN)
  - Add the newly derived clause into the set
  - Repeat until we derive *false* (or fail/stop)

Department of Distributed and Dependable Systems

Resolution

- Equivalent statements
  1) CNF formula ϕ is unsatisfiable
  2) We can derive empty clause using resolution on the clauses from ϕ

- 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* | true | false
  - Terms

    variable | *constant*

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

Equality logic with uninterpreted functions

- Syntax
  - Atomic formulas

    term = *term* | ***predicate*(*term*, ..., *term*)** | true | false
  - Terms

    variable | *constant* | ***function*(*term*, ..., *term*)**

- Semantics
  - No implicit meaning of functions and predicates
  - $a1 = b1$ & ... & $aN = bN \rightarrow f(a1,\dots,aN) = f(b1,\dots,bN)$

- Decision procedure
  - Transform into an equisatisfiable formula in equality logic

Equality logic with uninterpreted functions

- Purpose: abstraction
  - Full formula ➔ function semantics defined using axioms
  - Uninterpreted symbols ➔ just equality between arguments
  - $\models \phi^{EUF} \rightarrow \models \phi$
- False answers possible
  - Example: *add*(1,2) != *add*(2,1) in EUF
- Formula with UF easier to decide than the "full" formula

Department of Distributed and Dependable Systems

Linear arithmetic

- Syntax
  - Atomic formulas
    term = *term* | *term* < *term* | *term* ≤ *term* | true | false
  - Terms
    variable | *constant* | *constant variable* | *term* + *term*

- Example: (3x + 2y ≤ 5z) & (2x – 2y = 0)

- Arithmetic without multiplication ➔ Presburger arithmetic

- Decision procedure
  - General case (full theory): $2^{2^{O(n)}}$
  - 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

    variable – *variable* < *constant* |

    variable – *variable* ≤ *constant* |

    true | false
  - Operators: !, &, ←, ↔

- Example: (x – y < 3) & (y – z ≤ -4) & (z – x ≤ 1)

- Decision procedure
  - Conjunctive fragment polynomial for $\mathbb{Q}$ and $\mathbb{Z}$

Department of Distributed and Dependable Systems

Data structures

- Array theory
  - Function symbols

    select(*a*,*i*)   // read, a[i]

    store(*a*,*i*,*e*)   // update, a[i] = e
  - Axiom **read-over-write**

    select(*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)

Department of Distributed and Dependable Systems

Satisfiability Modulo Theory (SMT)

Satisfiability Modulo Theory (SMT)

- 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

Approaches to SMT

- Naive use of a SAT solver
  1. Extract boolean skeleton of the given formula ϕ
  2. Run the SAT solver on the boolean skeleton
     a) **unsatisfiable** → the input formula is unsatisfiable
     b) **satisfiable** → we get a satisfying assignment *v*
  3. Run the DP on the formula derived from the satisfying assignment *v*
     a) **satisfiable** → the input formula is satisfiable
     b) **unsatisfiable** → add the blocking clause for *v* to the boolean skeleton and continue with the step 2

Department of Distributed and Dependable Systems

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

SMT solving in practice

- 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 ϕ in predicate logic
- Goal
  - Decide validity of the formula ϕ in T
    - Semantic domain: show unsatisfiable negation
    - Proof domain: prove ϕ 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 ϕ
  - Find property violations by checking satisfiability of ϕ
- Challenge
  - Encoding program behavior into the formula ϕ

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: $x = 2$, $x' = x + 1$, $pc = 5$, $pc' = pc + 1$

Transition relation

$$(pc = 1) \land (x' = x + 2y) \land (pc' = pc + 1)$$

$$\lor$$

$$(pc = 2) \land (x' = 0) \land (pc' = pc + 6)$$

$$\lor$$

… … …

$$\lor$$

$$(pc = N) \land (x' = x - y + 5) \land (pc' = pc + 1)$$

Traces with bounded length

- Transition relation unfolded at most K times
  - Fresh copies of program variables (x, x', ..., $x^{(K)}$) used for each unfolding of the transition relation
- Example
  - *INIT*: $x = 0$, $pc = 1$
  - T(K): (

$$((pc = 1) \land (x' = x + 2y) \land (pc' = pc + 1)) \lor$$

... ...

$$((pc^{(K-1)} = 1) \land (x^{(K)} = x^{(K-1)} + 2y^{(K-1)}) \land (pc^{(K)} = pc^{(K-1)} + 1)) \lor$$

... ...

)

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

Department of Distributed and Dependable Systems

Encoding program behavior in logic

- Large formula

$$INIT(s_0) \wedge (\bigwedge_{i=0..k-1} T(s_i, s_i+1)) \wedge (\bigvee_{i=0..k} \neg p(s_i))$$

- 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 ➔ we get a counterexample with the length ≤ K
   - Unsatisfiable formula ➔ no property violations in program executions of the length ≤ 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

- D. Kroening and O. Strichman. **Decision Procedures: An Algorithmic Point of View**. Springer, 2008.

- A. Biere, A. Cimatti, E. Clarke, O. Strichman, and Y. Zhu. **Bounded Model Checking**. Advanced in Computers, 58, 2003