Satisfiability Modulo Theories and its applications in (and out of) Formal Verification

Alessandro Cimatti
Fondazione Bruno Kessler, Trento, Italy
cimatti@fbk.eu

The Embedded Systems Unit

- Formal verification methods for complex system design

- Research
  - formal verification, requirements validation
  - safety analysis (fault tree analysis, FMEA)
  - planning/execution/monitoring for on-board autonomy
- Tool development
  - The NuSMV model checker
  - The MathSAT SMT solver
  - The RAT requirements analysis tool
  - The FSAP platform for safety analysis
  - The MBP planner
- Technology transfer
  - European Space Agency
  - European Railway Agency
  - Industrial partners (e.g. process control, railways signaling)

Formal Methods in a Nutshell

- Formal specification
  - representing artifact (specification, design, algorithm) and desired properties by means of a mathematically precise, unambiguous, logical language
- Formal verification
  - prove theorems in corresponding mathematical theory
- Objectives
  - prove correctness
  - find "more" bugs
  - find them "earlier" in the development flow
- Key issues
  - usability, seamless integration in development flow
  - expressiveness vs automation

Outline of the talk

- Traditional Formal Verification
  - Boolean techniques:BDD, SAT
- Satisfiability Modulo Theories
  - Beyond the Boolean case
- SMT for verification (et al.)
  - From SAT-based to SMT-based
  - Software Model Checking
  - Requirements Validation
  - Other applications
- Conclusions

Formal Verification

- Focus on "fully automated" verification

- Reactive System
  - not finite computation program (e.g. sorting)
  - communication protocol, hw design, control software, OS
  - modeled as a state transition system
- Requirements
  - modeled as formulae in a temporal logic
- Does my system satisfy the requirements?

- Model checking
  - search configurations of state transition system
  - detect violation to property, and produce witness of violation
  - conlcude absence of violation

Properties

- Safety properties
  - nothing bad ever happens
    - never (P1.critical & P2.critical)
    - always (P1.critical -> (P1.critical until P1.done))
  - state transition system can't reach a bad configuration
- Liveness properties
  - something good will happen
    - always (P1.trying -> eventually P1.critical)
  - state transition system can not exhibit a bad cycle

Model Checking

- From properties to "monitors"
  - able to recognize violations
  - checking safety properties reduced to reachability analysis
- Given model of reactive system
  - State variables V
  - States S = Pow(V)
  - Initial states $I \subset S$
  - Transition relation R: S -> Pow(S)
  - Bad states $B \subset S$
- Find whether a bad state is reachable
  - $s_0, s_1, \ldots, s_n$ with $s_0 \in I$, $s_{i+1} \in R(s_i)$, $s_n \in B$

Model Checking

- Prove that nothing bad can ever happen
- An "easy" problem
  - linear in size of state space
  - easy?
- State space exponential in the number of variables…
  - not so easy

Explicit State Model Checking

- Each state stored and expanded as individual object
- E.g. model with x, y, z
- Each state represented as a bit vector
  - 000, 001, …, 111
- Vanilla Algorithm
  1. Open := I, Closed := {}
  2. Open := Open \ {s}
  3. If s∈B return "violation"
  4. Closed := Closed U {s}
  5. Open := Open U (R(s) \ Closed)
  6. If Open = {} return "success"
  7. goto step 2.

Explicit State Model Checking

- The SPIN model checker
- Very high degree of technology
  - partial order reduction
  - bit-state hashing
  - heuristic guidance
  - disk storage techniques
- Very effective in certain application domains
  - e.g. communication protocols
- Main limitation: memory consumption
  - "proportional" to number of reachable states

Symbolic Representation

- State variables as variables in a logical language
  - x, y, z, w
- A state is an assignment to state variables
  - The bitvector 0011
  - The assignment x |-> F , y |-> F , z |-> T , w |-> T
  - The formula ¬x & ¬y & z & w
- A set of states is a a set of assignments
  - can be represented by a logical formula
  - x and not y represents {1000 , 1001 , 1010, 1011} or a larger set, if more variables are present
- Set operations represented by logical operations
  - union, intersection, complementation as
  - disjunction, conjunction, negation
- I(X), B(X) are formulae in X
  - Is there a bad initial state?
  - Is I(X) & B(X) satisfiable?

Symbolic Representation

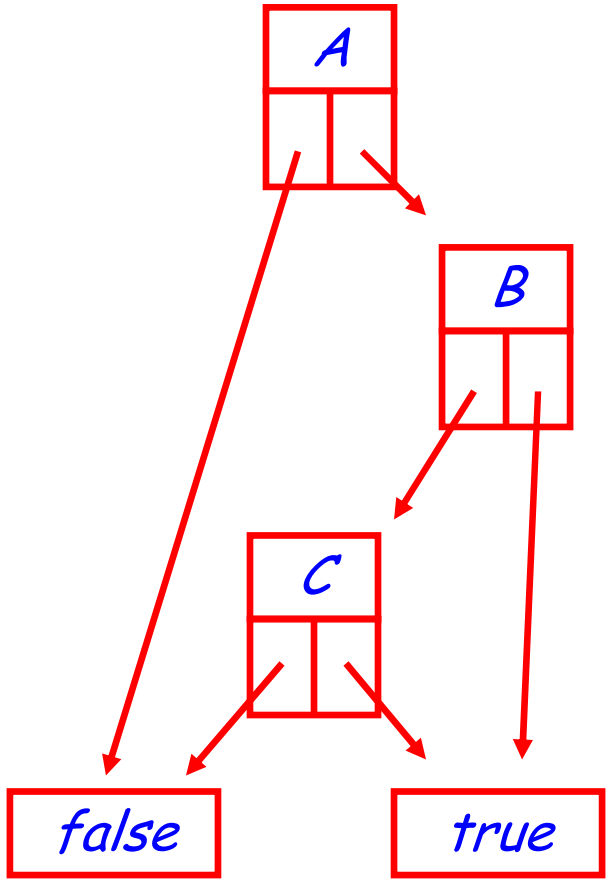
- Symbolic representation of transitions?
- Transition
  - pair of assignments to state variables
- Use two sets of variables
  - current state variables: x, y, z
  - next state variables: x', y', z'
- A formula in current and next state variables
  - represents a set of assignments to X and X'
  - a set of transitions
  - R(X, X')

BDD-based Symbolic Model Checking

- The first form of Symbolic Model Checking
- Based on Binary Decision Diagrams
  - canonical representation for logical formulae
- I(X), R(X, X'), B(X)
  - each represented by a BDD
- Image computation: compute all successors of all states in S(X)
  - based on projection operation
  - exists X.(S(X) and R(X, X'))

Binary Decision Diagrams

- Binary Decision Diagrams
  - canonical representation for boolean functions
  - ITE nodes
  - fixed order on test variables
  - $(A \wedge (B \vee C))$
- Reduction rules
  - only one occurrence of the same subtree
  - if(P, b, b) == b
- Can blow up in space
- Order of variables can make huge difference



More on BDDs

- Core of traditional EDA tools
  - In practice, can be extremely efficient
  - They provide QBF functionalities

    $\exists x.\Phi(x, V) \;==\; \Phi(false, V) \vee \Phi(true, V)$
  - Fundamental operation in model checking

Symbolic Reachability Analysis

```
Visited := False
New :=  I
while (true) {
   if IsSat(New & B) return "violation"
   New := Image(New, R) & ¬Visited
   if New(x) = False return "success"
   Visited := Visited | New
}
```

A symbolic breadth-first search, where each layer is represented by a BDD

Techniques for BDD-based SMC

- Variable orderings
  - dynamically change order to reduce size
- Partitioning
  - list of implicitly conjuncted BDDs rahter than single, monolithic BDD
  - trading one quantification over many variables with multiple quantifications over reduced number of varaibles
- Reachability Algorithms
  - priority-based reachability
  - overapproximations
  - inductive reasoning

Symbolic model checking without BDDs

- [BCCZ99] contained two key insights
- Focus on finding bugs
  - give up proof of correctness
  - try to falsify property, i.e. witness to violation
  - within given resource limit (bound)
- Use SAT solver instead of BDDs

Symbolic Representation

- Vectors of state variables
  - current state X
  - next state X'
- Initial condition I(X)
- Transition relation R(X, X')
- Bug states B(X)

- I, R, B, represented as formulae rather than BDDs
  - much smaller size!

Bounded Model Checking

- State variables replicated K times
  - X_0 , X_1, X_{k-1}, X_k
- Look for bugs of increasing length
  - $I(X_0) \wedge R(X_0, X_1) \wedge \ldots \wedge R(X_{k-1}, X_k) \wedge B(X_k)$
  - bug if satisfiable
  - increase k until …

K-Induction

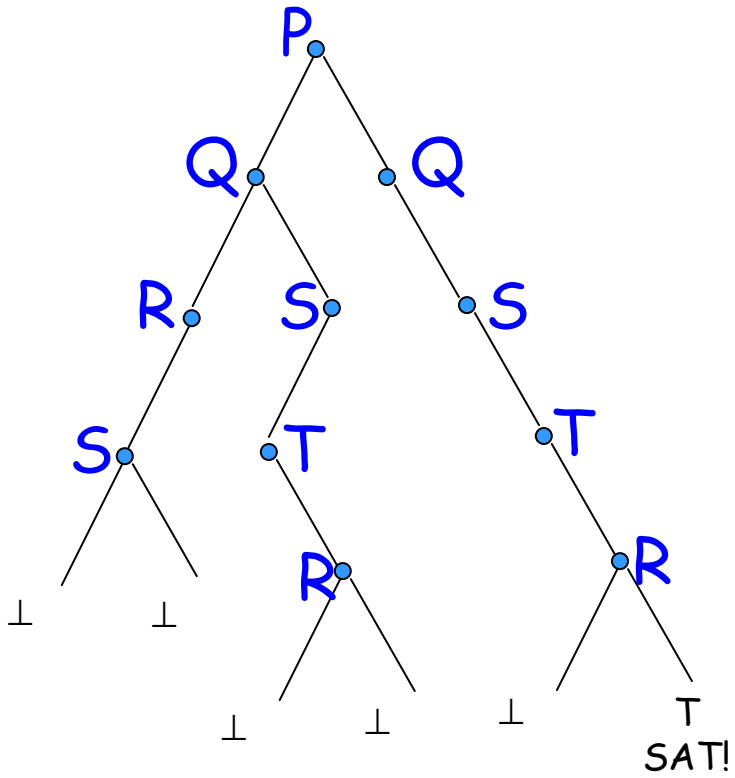
- Prove absence of bugs by induction
  - $I(X_0) \wedge B(X_0)$
  - $I(X_0) \wedge R(X_0, X_1) \wedge \neg B(X_1)$
  - ...
  - $\neg B(X_0) \wedge R(X_0, X_1) \wedge \ldots \wedge \neg B(X_{k-1}) \wedge R(X_{k-1}, X_k) \wedge B(X_k)$
  - proved correct if unsatisfiable (and no bugs until k)

- Important features
  - incremental interface
  - lemmas can be shifted over time
    - from $\Phi(X_0, X_1)$ to $\Phi(X_i, X_j)$

SAT-based vs BDD-based

- BDD-based
  - all models, may blow up in space
  - actually based on QBF operators
  - can easily check fix point
  - uses twice |X| variables
- SAT-based
  - one model
  - may diverge in time
  - much weaker in QBF operators
  - uses k times |X| variables
  - How is it possible?
  - SAT solvers are impressive objects!

Boolean SAT: search space



- The DPLL procedure
- Incremental construction of satisfying assignment
- Backtrack/backjump on conflict
- Learn reason for conflict
- Splitting heuristics

Techniques for SAT-based SMC

- Incrementality/backtrackability
  - bounded model checking problems are similar
  - SAT solver can add and remove clauses
- Unsatisfiable core extraction
  - used for explanation and problem simplification
- Interpolation
  - a whole research line with own algorithms
  - disregarded here for time limits

Beyond Boolean Verification: Satisfiability Modulo Theories

Beyond the Boolean case

- Boolean verification engines are very powerful
- They work at the boolean level

- Why is this a limitation?
  - Boolean representation not expressive enough
    - encoding may not exist, or can "blow up"
  - Boolean reasoning not the "right" level of abstraction
    - important information may be lost during encoding

Some examples

- RTL circuits
  - word w[n] reduced to w.1 … w.n boolean variables
  - booleanization destroys data path structure!
- pipelines
  - function symbols used to abstract blocks
- timed automata
  - real-valued variables for timing
  - difference constraints to express time elapse
- hybrid automata
  - real-valued variables for physical dynamics
  - mathematical constraints to express continuous evolution
- software verification
  - integer-valued variables for proof obligations

Satisfiability Modulo Theory

- Trade off between expressiveness and reasoning
  - SAT solvers: boolean case, automated and very efficient
  - theorem provers: general FOL, limited automation
- Satisfiability Modulo Theories: a sweet spot?
  - retain efficiency of boolean reasoning
  - increase expressiveness
  - decidable fragments of FOL
- Impact on verification:
  - increase capacity by working above the boolean level

Statisfiability Modulo Theories

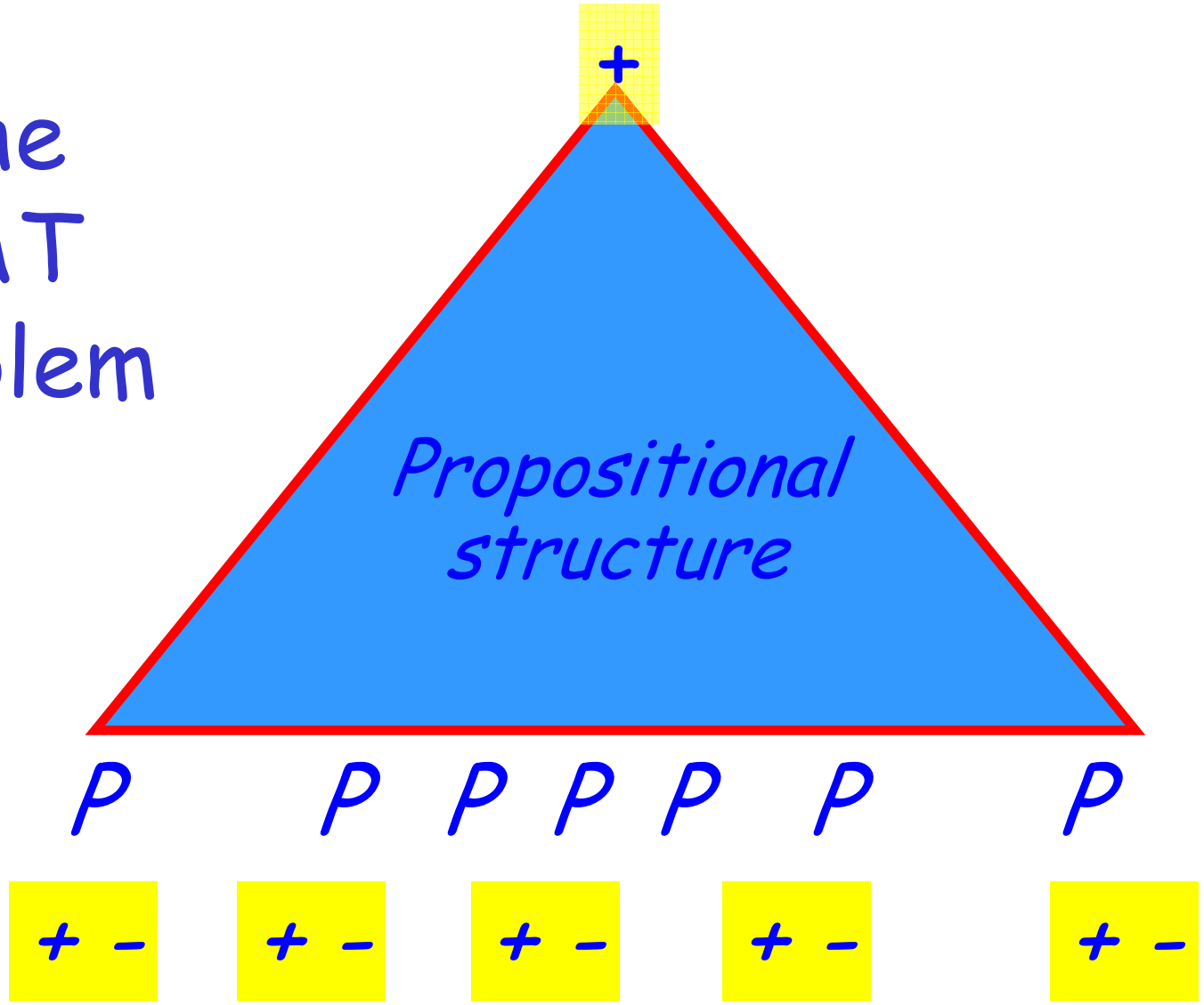
- An extension of boolean SAT
- Some atoms have non-boolean (theory) content
  - A1
  - A2
  - A3
- Theory interpretation for individual variables, constants, functions and predicates
  - if x = 0, y = 20, z = 10
  - then A1 = T, A2 = T, A3 = F
- Interpretations of atoms are constrained
  - A1, A2 and A3 can not be all true at the same time

Theories of Practical Interest

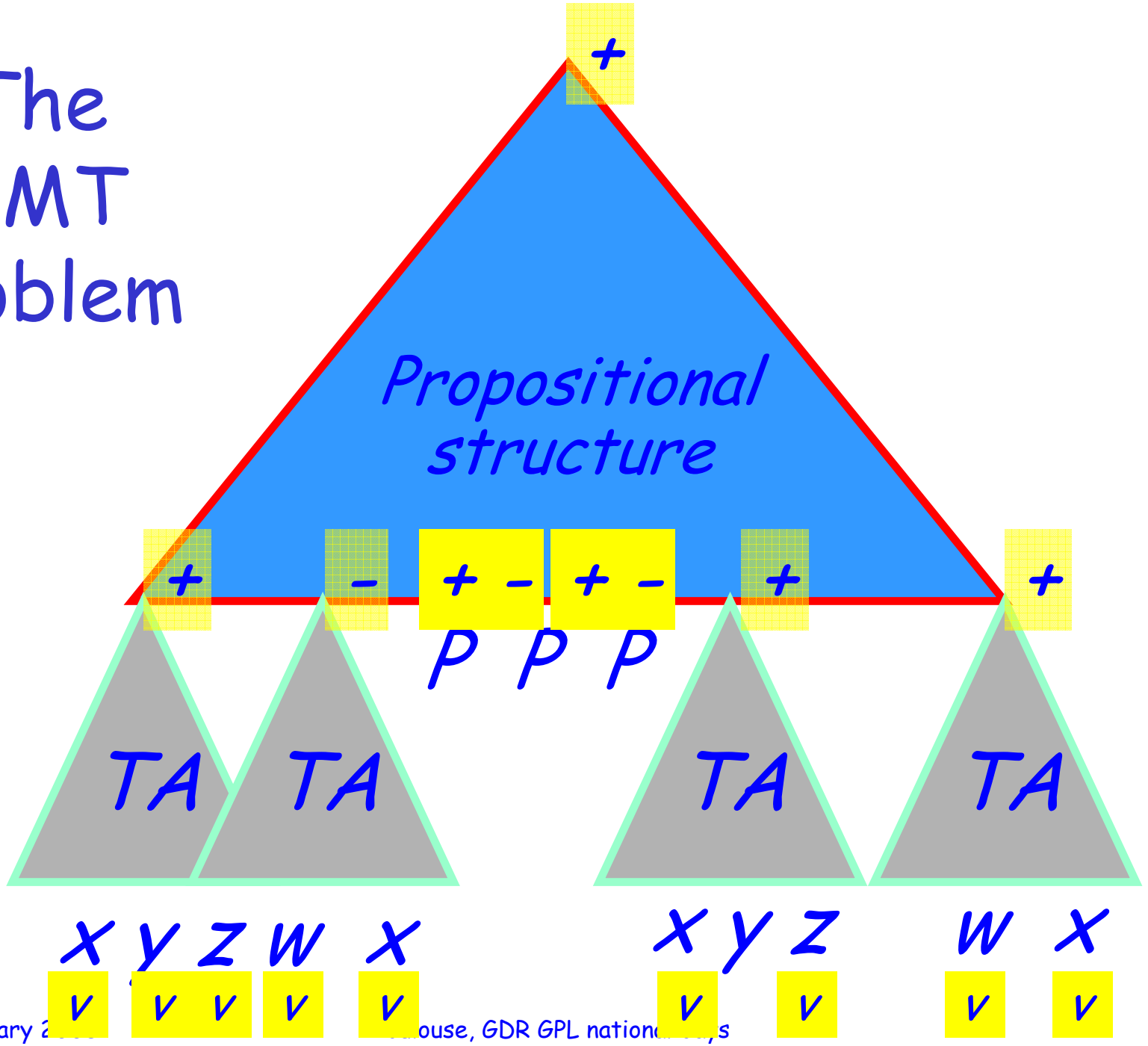
- Equality Uninterpreted Functions (EUF)
  - $x = f(y), h(x) = g(y)$
- Difference constraints (DL)
  - $x - y \leq 3$
- Linear Arithmetic
  - $3x - 5y + 7z \leq 1$
  - reals (LRA), integers (LIA)
- Arrays (Ar)
  - read(write(A, i, v), j)
- Bit Vectors (BV)
- Their combination

SMT solvers

The SAT problem



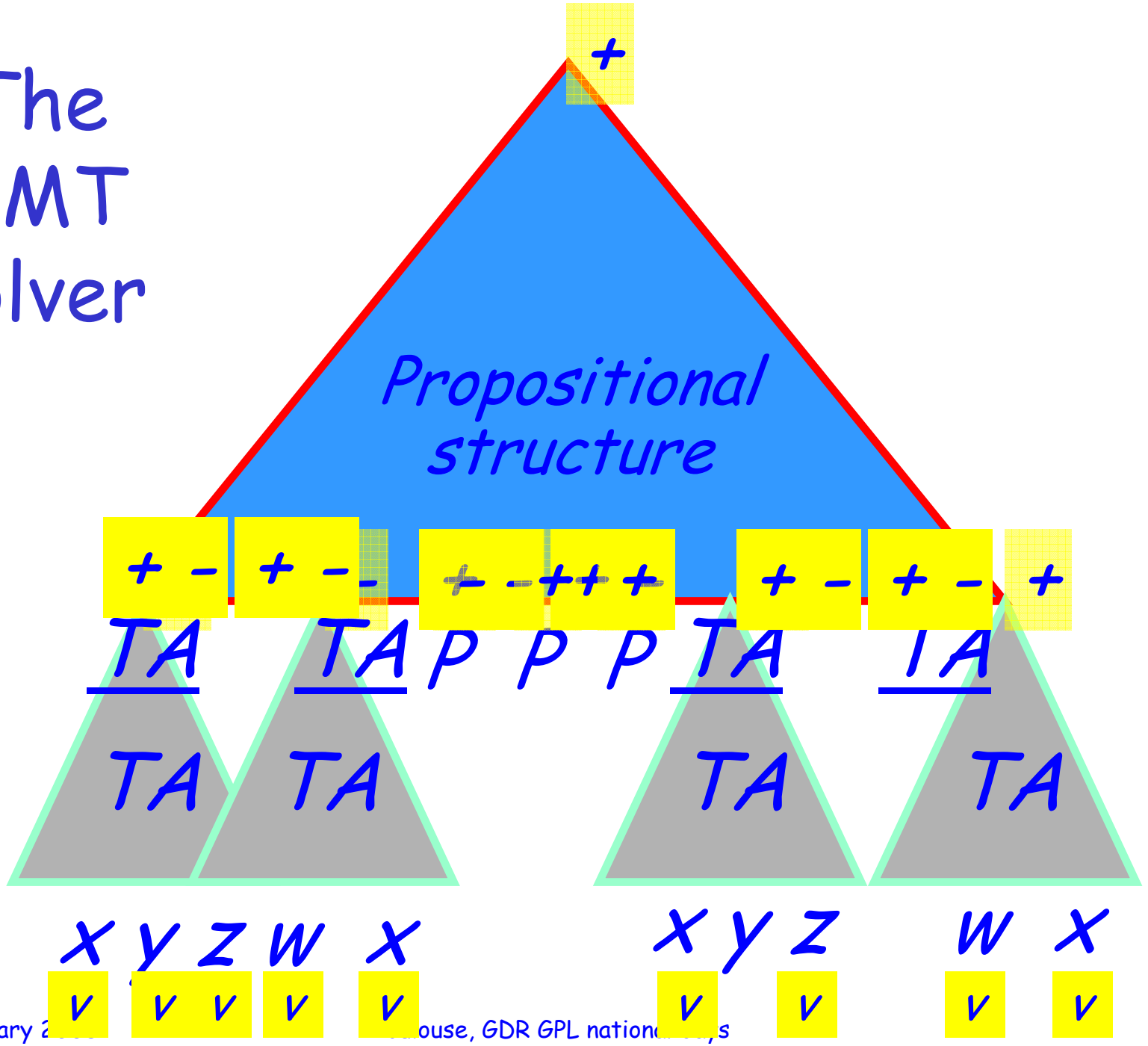
The SMT problem



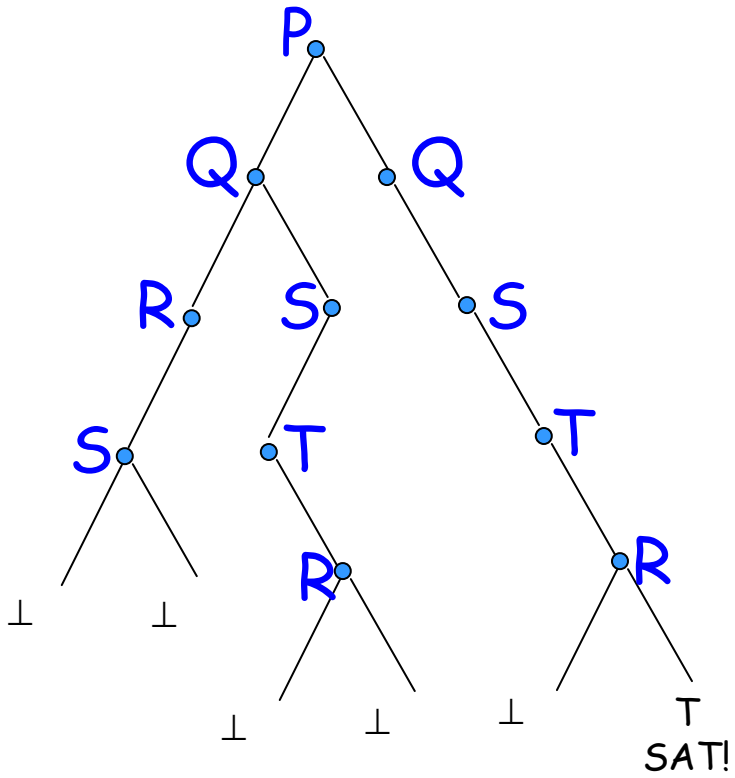
MathSAT: intuitions

- The search combines boolean reasoning and theory reasoning
- Find boolean model
  - theory atoms treated as boolean atoms
  - truth values to boolean and theory atoms
  - model propositionally satisfies the formula
- Check consistency wrt theory
  - set of constraints induced by truth values to theory atoms
  - existence of values to theory variables
- The MathSAT approach
  - Boolean search DPLL
  - theory reasoning

The SMT solver



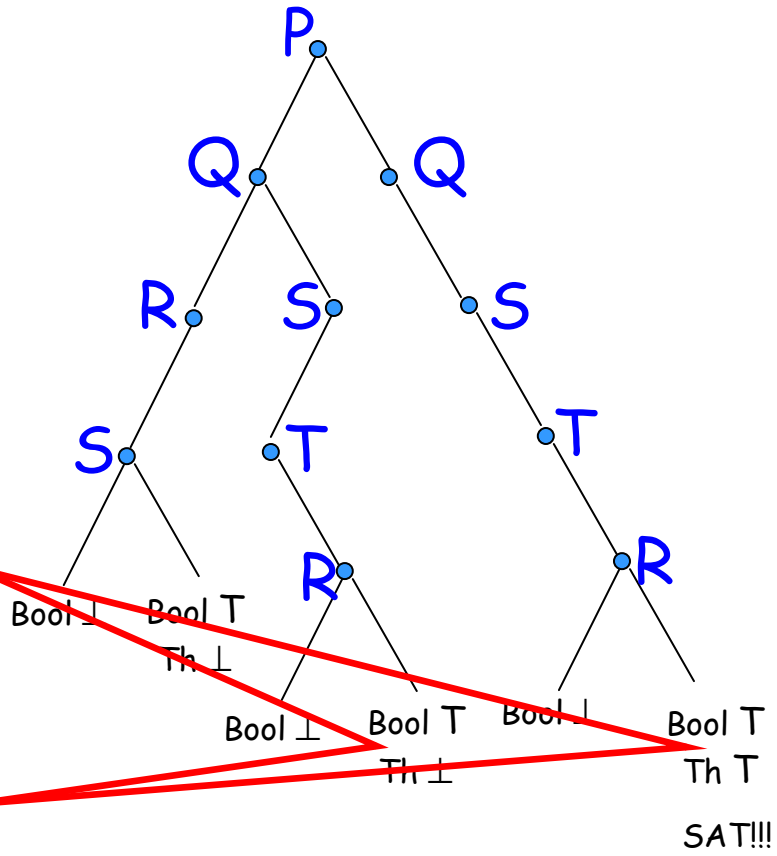
Boolean DPLL



- The DPLL procedure
- Incremental construction of satisfying assignment
- Backtrack/backjump on conflict
- Learn reason for conflict
- Splitting heuristics

MathSAT: search space

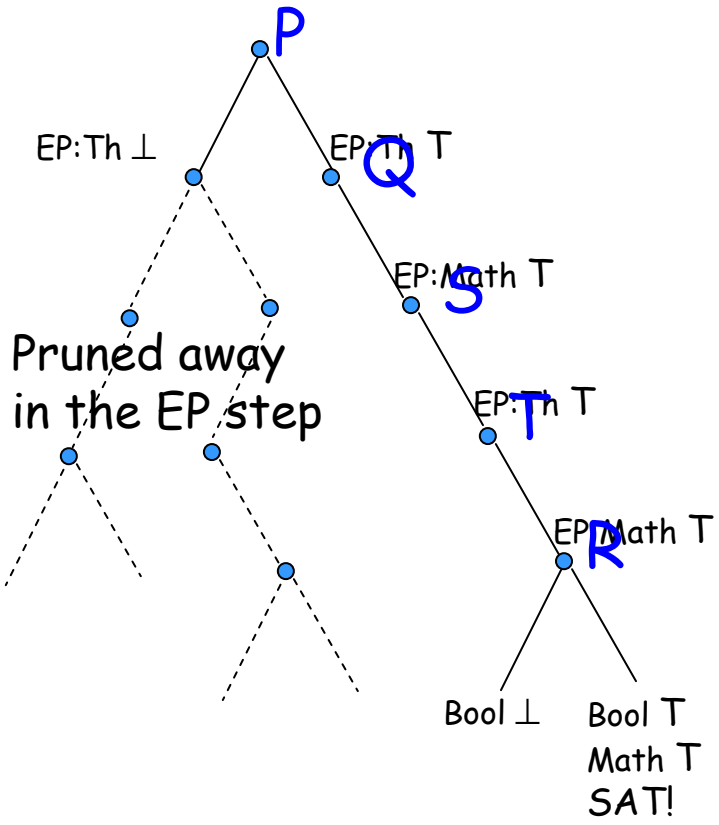
P	T	$x - y \leq 3$
P_1	F	
P_2	T	$y - z = 10$
Q	F	
R	T	$x - z \geq 15$
R_1	F	
S	F	$z - 2*w = 1$
S_1	T	



Many boolean models are not theory consistent!

Early pruning

Check theory consistency of partial assignments



Learning Theory Conflicts

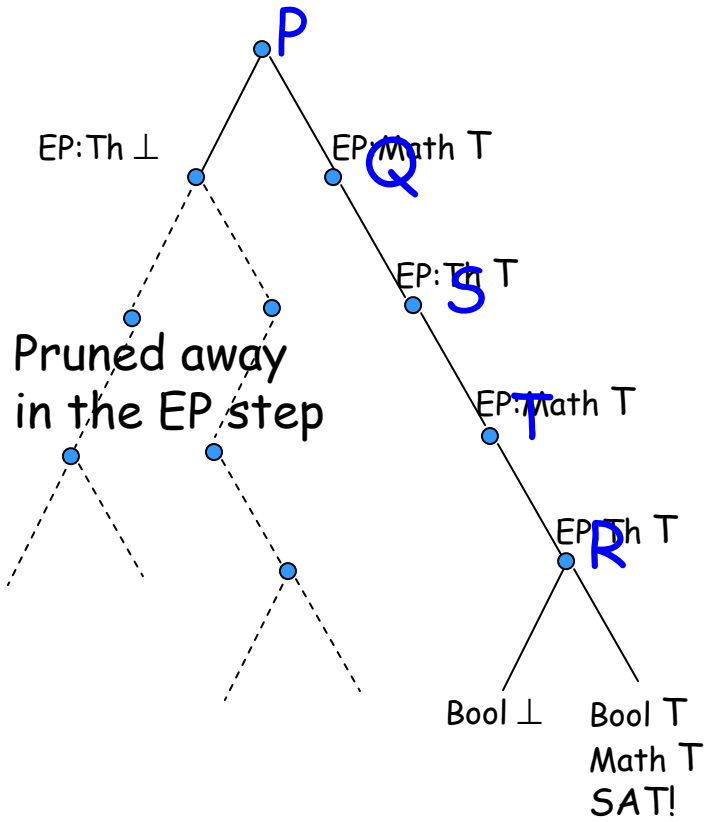
The theory solver can detect a reason for inconsistency

I.e. a subset of the literals that are mutually unsatisfiable

E.g. x = y, y = z, x != z

Learn a conflict clause
x != y or y != z or x = z

By BCP the boolean enumeration will never make same mistake again



Theory Deduction

The theory solver can detect that certain atoms have forced values

E.g. from x = y and x = z
infer that y = z should be true

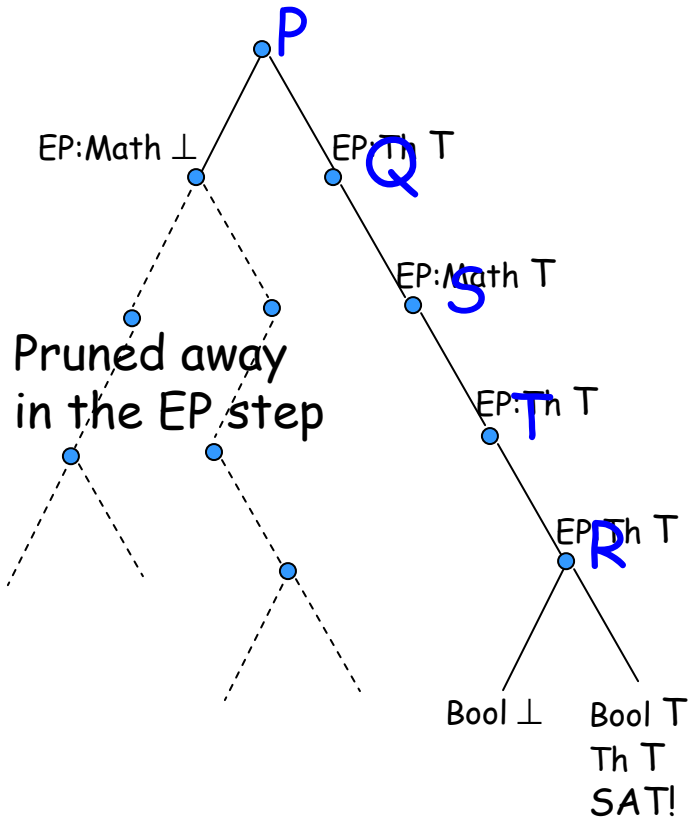
Force deterministic assignments

Theory version of BCP

Furthermore, we can learn the deduction:

x=y & x = z -> y=z

Theory Conflict vs theory deduction



Optimizations

- Incrementality and Backtackability
  - add constraints without restarting from scratch
  - remove constraints without paying too much
- Limiting cost of early pruning
  - filtering, incomplete calls
- Conflict set minimization
  - return T-inconsistent subset of assignment
- Deduction
  - return forced values to unassigned theory atoms
- Static learning
  - precompile obvious theory reasoning reasoning to boolean

State of the art

- Relatively recent field, a lot of interest
- Impressive improvements in the last four years

- Many solvers available
  - Yices, MathSAT, Barcelogic, CVC3, Z3, Boolector, Spear, …
- SMT-LIB
  - unified language
  - wide benchmark set from several application domains
- SMT-COMP
  - held yearly
- SMT Workshop
  - this year at CADE

SMT-based verification

The Role of SMT in verification

- State variables of various types
  - in addition to discrete
  - reals, integers, bitvectors, arrays, …
- Representation
  - higher level
  - structural information is retained

Symbolic Encoding

- Vectors of state variables
  - current state X
  - next state X'
- Initial condition I(X)
- Transition relation R(X, X')
- Bug states B(X)

- Key difference
  - X, X' are not limited to boolean variables
  - I, R, B are STM formulae

SMT-based Algorithms

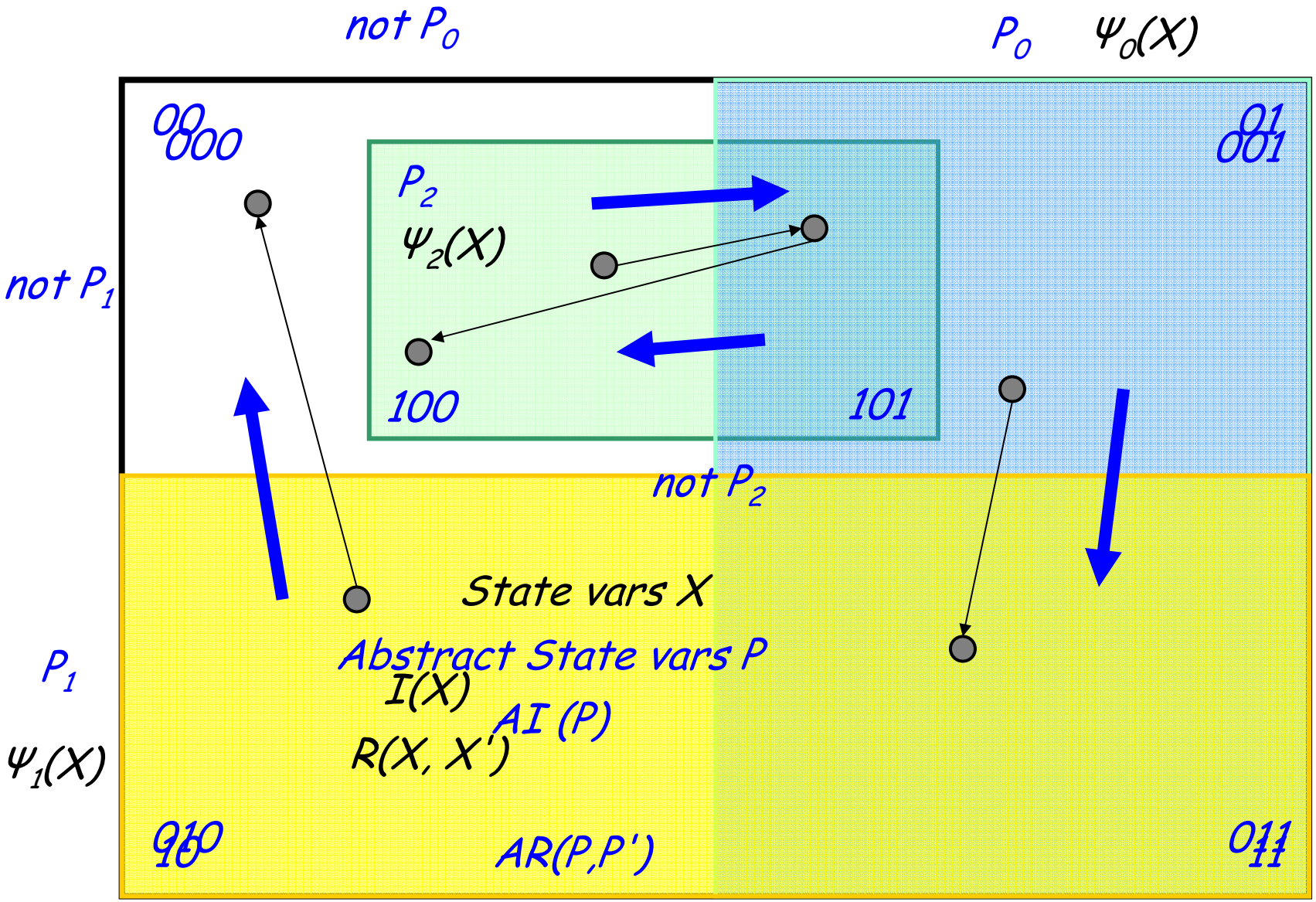
- From SAT-based to SMT-based algorithms
- Simply replace SAT solver with SMT solvers
  - bounded model checking
  - k-induction

BMC and Induction

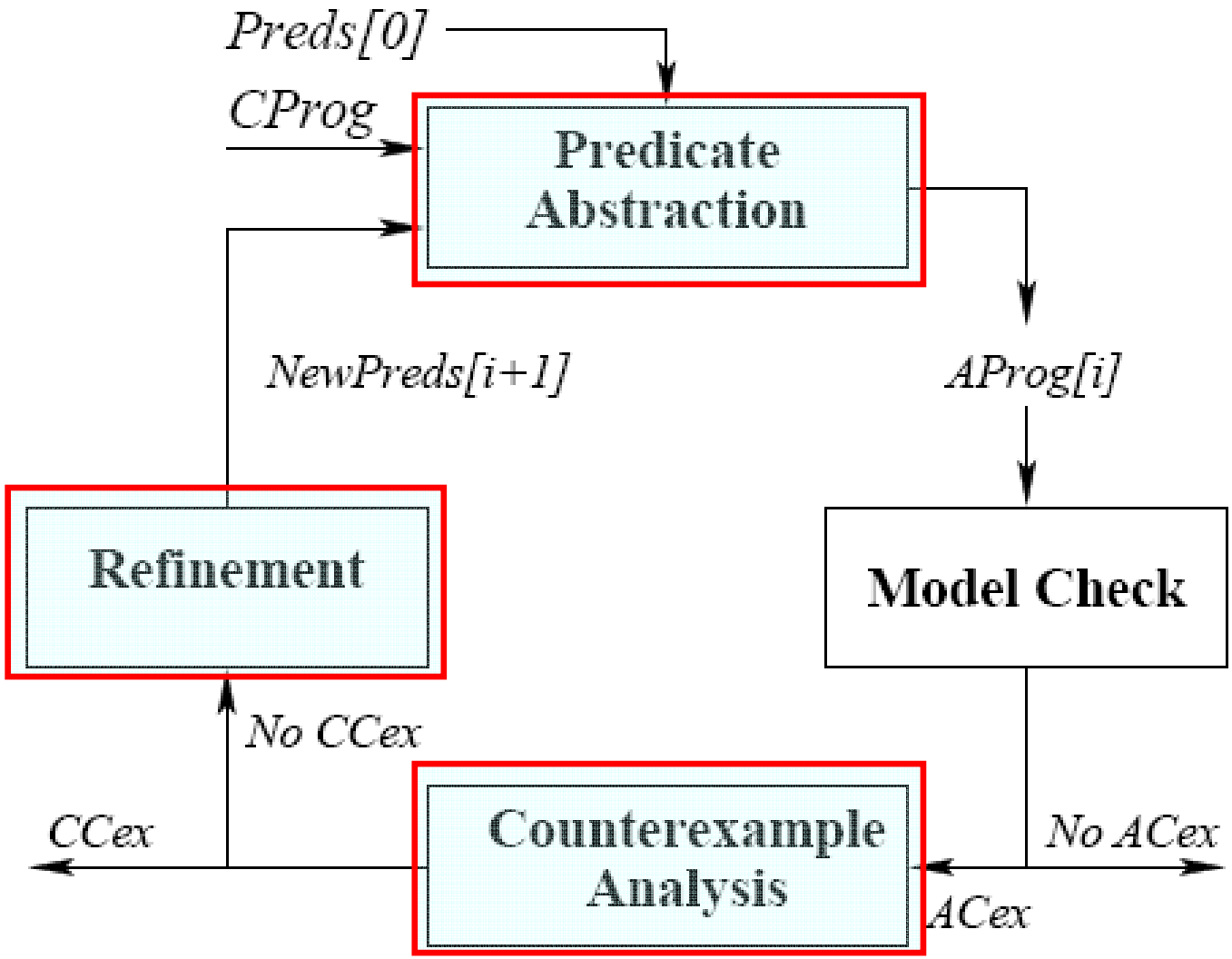
- Look for bugs of increasing length
  - $I(X_0) \wedge R(X_0, X_1) \wedge \ldots \wedge R(X_{k-1}, X_k) \wedge B(X_k)$
  - bug if satisfiable
  - increase k until …
- Prove absence of bugs by induction
  - $\neg B(X_0) \wedge R(X_0, X_1) \wedge \ldots \wedge \neg B(X_{k-1}) \wedge R(X_{k-1}, X_k) \wedge B(X_k)$
  - proved correct if unsatisfiable (and no bugs until k)

- Important features
  - incremental interface
  - theory lemmas should be retained
  - theory lemmas can be shifted over time
    - from $\Phi(X_0, X_1)$ to $\Phi(X_i, X_j)$
  - Unsat core and generation of interpolants
  - Elimination of quantifiers

Counter-Example Guided Abstraction-refinement


not P_0
P_0
$\Psi_0(X)$
000
001
P_2
$\Psi_2(X)$
not P_1
100
101
not P_2
State vars X
Abstract State vars P
P_1
I(X)
AI (P)
$\Psi_1(X)$
R(X, X')
010
011
AR(P,P')

Counter-Example Guided Abstraction-Refinement (CEGAR)



Computing Abstractions

- Given concrete model CI(X), CR(X, X')
- Given set of predicates $\Psi_i(X)$
  each associated to abstract variable P_i
- Obtain the corresponding abstract model
- AR(P, P') is defined by

$$\exists X\, X'.(CR(X, X') \wedge \bigwedge_i P_i \leftrightarrow \Psi_i(X) \wedge \bigwedge_i P_i' \leftrightarrow \Psi_i(X'))$$

- Existential quantification as AllSMT
  - SMT solver extended to generate all satisfying assignment

NuSMV + MathSAT

- Forthcoming: NuSMT
  - tight integration of MathSAT and NuSMV
  - symbolic verification of
    - timed systems
    - hybrid systems
    - high level circuits

- Stay tuned at
  - The NuSMV model checker
    - http://nusmv.fbk.eu/
  - The MathSAT SMT solver
    - http://mathsat4.disi.unitn.it/

Requirements Validation based on SMT

The EuRailCheck project

Focus here: requirements, not model

- In traditional formal verification
  - the *design* is under analysis
  - the requirements are taken as "golden"
  - verification means checking compliance
- Here the goal is to
  - enhance quality of *requirements*
- A much harder task!
  - from informal to formal

Why is it so hard?

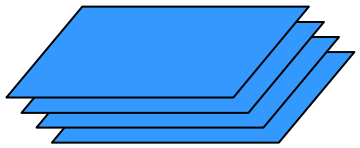
- Requirements analysis is a pervasive problem in nowadays industry
  - In hardware design, standards for languages to represent properties and design intent are emerging (e.g. PLS, SVA)
- Problem 1: Natural language
  - ambiguous
  - degree of automation
  - requires background information
- Problem 2: when are my requirements good?
  - are they too strict? Are some required behaviours being (wrongly) disallowed?
  - are they too weak? Are some undesirable behaviours being (wrongly) allowed?
- The source of the matter is that what is being modeled is informal
  - the design intent that must be captured by the specification is in the head of the specifier

Issues of interest in this project

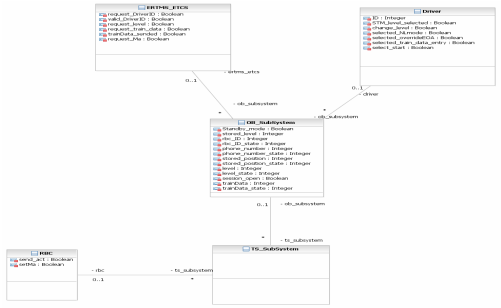
- PB1: Bridging the gap between natural language and formal analysis
- PB2: providing methods for pinpointing flaws in requirements
- And also (as usual) …
  - Integration within requirements engineering flow
  - Usability
    - Avoid intricate formalisms
    - Hide formal methods with semiformal representations
  - Automation of the verification process
    - Model checking

From Informal to Formal

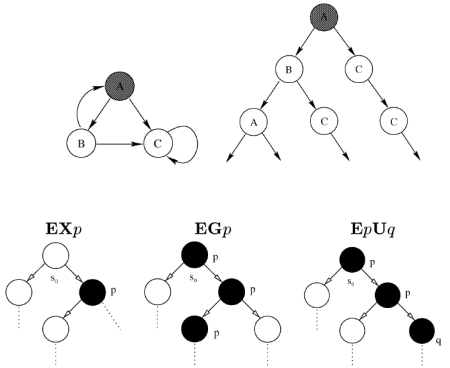
NATURAL LANGUAGE



SEMIFORMAL LANGUAGE

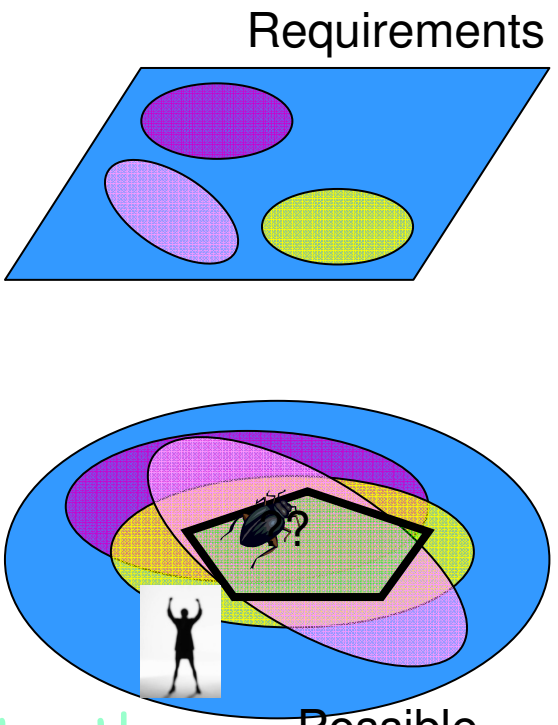


FORMAL LANGUAGE



Which flaws in requirements?

- A set of requirement is a set of constraints over possible evolutions of the entities in the domain
- Possible questions
  - Are my requirements too strict?
  - Are my requirements too weak?
- Possible checks
  - Consistency check (too strict?)
    - is there at least one admissible behaviour?
  - Possibility check (too strict?)
    - is a given desirable behaviour admissible?
  - Assertion check (too weak?)
    - is a given undesirable behaviour excluded?
- Warning: no way to formalize design intent!



The role of SMT
Modeling continuous evolution

- Requirements combine discrete variables with continuous, real-valued variables
  - e.g. speed, position, time elapse, timers
- One state is an assignment to discrete and continuous variables
- Evolution of the scenario
  - discrete transitions: no time elapse, change in discrete state
  - continuous transitions: time elapse, no change in discrete state, continuous state changed according to flow equations
  - example: $p(t+\Delta) = p(t) + v \cdot \Delta$
- The problem is hard
  - provably undecidable
  - apply incomplete techniques (bounded model checking, CEGAR)

Main project results

- A requirements analysis methodology
  - integrating informal and formal techniques
  - hiding formal techniques as much as possible
  - controlled natural language
- A support toolset
  - based on standard commercial tools: RSA, RequisitePro
  - integrating verification engine NuSMV and MathSAT
- Formalization of substantial fragments of ETCS specifications
- Training and ongoing work
  - two-days workshop
  - three weeks of training to ETCS experts
  - "next-steps" workshop in Februrary in Lille

Other applications of SMT

Applications to High-level Hardware Design

- Ongoing work with Intel Haifa
  - Application described in "high level" language
  - words and memories are not blasted into bits
- Custom decision procedure for Bit Vectors
- Applications
  - Register-transfer level circuits
  - Microcode
- Functionalities
  - more scalable verification
    - currently based on boolean SAT
  - tight integration with symbolic simulation
    - pipe of proof obligations
  - Automated Test Pattern Generation
    - enumerate many different randomized solutions

Analysis of Railways Control Software

- Control software for Interlocking
  - controls devices in train station
  - Application independent scheduler
  - Parameterized, object oriented
  - Instantiation with respect to station topology

- Model Checking to analyze single modules
  - SMT-based software model checking
  - checking termination, functional properties
- Compositional reasoning for global proofs
  - based on scheduler structure
- Reverse engineering from the code
  - inspection, what-if reasoning

- Other potential role of SMT solving
  - dealing with quantified formulae over lists of entities

Parametric Schedulability Analysis

- Schedulability analysis
  - given set of processes and scheduling policy
  - check whether deadlines can be met
- Key problem: sensitivity analysis
  - where do the numbers come from?
  - typically, these are estimates
  - traditional schedulability theory based on numerical raesoning, lifting results to practical cases may be nontrivial
- Goal: analyze sensitivity with respect to variations
- Analytical construction of schedulability region!
- The role of SMT
  - SMT allows for parametric representation
  - SMT-based bounded model checking to generate one fragment of unschedulability region
  - iterate to generate all fragments
  - CEGAR to terminate the iteration

Design Mutation in Avionics

- The problem: find "good" spatial position of aircraft components with respect to safety constaints
  - no electrical components "below" component that potential leakage
  - not all components implementing critical function on same impact trajectory
- Required functionalities
  - is a configuration satisfactory
  - reasons for violation
  - find acceptable solution
  - find optimal solution
- Encode problem into SMT
  - may require dedicated, custom theory
  - may require extension to "optimal constraints"

Conclusions

- SMT solvers increasingly effective
  - more expressive languages
  - more functionalities
  - faster solvers
- Better solutions to "traditional" problems
  - formal verification
  - ATPG
- Possible solutions to non-standard problems
  - requirements validation
  - design mutation
  - schedulability analysis