SMT, CALO PCE, and SAVH

John Rushby

Computer Science Laboratory
SRI International
Menlo Park CA USA

Overview

- SAT and SMT solvers, and their applications
- Building a faster SMT solver
- Working with inconsistent knowledge
  - MaxSAT and MaxSMT
  - Application to CALO PCE
- Maximal assignments
  - SMTmax and MaxSMTmax
  - Application to AI planning and diagnosis
  - Application to SAVH
- SMT as disruptive technology
  - Paradigm shift in verification: The Evidential Tool Bus
  - And opportunities in AI (and Biology and . . .)

Anything a SAT solver can do, an SMT solver can do better

SAT Solving

- Find satisfying assignment to a propositional logic formula
- Formula can be represented as a set of clauses
  - CNF: conjunction of disjunctions
  - Find an assignment of truth values to variable that makes at least one literal in each clause TRUE
- Example: given following 4 clauses
  - A, B
  - C, D
  - E
  - $\bar{A}, \bar{D}, \bar{E}$

  One solution is $A, C, E, \bar{D}$

  (A, D, E is not and cannot be extended to be one)
- Do this when there are 1,000,000 variables and clauses

SAT Solvers

- SAT solving is the quintessential NP-complete problem
- But now amazingly fast in practice (most of the time)
  - Breakthroughs (starting with Chaff) since 2001
  - Sustained improvements, honed by competition
- Has become commodity technology
  - MiniSAT is 700 SLOC
- Can think of it as massively efficient search
  - So use it when your problem can be formulated as SAT
- Used in bounded model checking and in AI planning
  - Routine to handle 10^{300} states

Satisfiability Modulo Theories (SMT)

- SAT can encode operations and relations on bounded integers (bitvector representation), and other finite data types and structures
- But not unbounded or infinite types (e.g., reals), or structures (e.g., queues, lists)
- And even bounded arithmetic can be slow
- There are fast decision procedures for these theories
- But they work only on conjunctions
- General propositional structure requires case analysis
    - Should use efficient search strategies of SAT solvers

That's what an SMT solver does

SMT Solving

- Individual decision procedures decide conjunctions of formulas in their decided theories
- Combinations of decision procedures (using, e.g., Nelson-Oppen or Shostak methods) decide conjunctions over the combined theories (e.g., arithmetic plus arrays)
- SMT allows general propositional structure
  - e.g., $(x \leq y \vee y = 5) \wedge (x < 0 \vee y \leq x) \wedge x \neq y$

    ...possibly continued for 1000s of terms
- Should exploit search strategies of modern SAT solvers
- So replace the terms by propositional variables
  - $(A \vee B) \wedge (C \vee D) \wedge E$
- Get a solution from a SAT solver (if none, we are done)
  - e.g., A, D, E

SMT Solving by "Lemmas On Demand"

- Restore the interpretation of variables and send the conjunction to the core decision procedure
  - e.g., $x \leq y \wedge y \leq x \wedge x \neq y$
- If satisfiable, we are done
- If not, ask SAT solver for a new assignment—but isn't it expensive to keep doing this?
- Yes, so first, do a little bit of work to find fragments that explain the unsatisfiability, and send these back to the SAT solver as additional constraints (i.e., lemmas)
  - $A \wedge D \supset \neg E$
- Iterate to termination (e.g., B, D, E: $y = 5, y < x$: $y = 5, x = 6$)
- This is called "lemmas on demand" (de Moura, Ruess, Sorea) or "DPLL(T)"; it yields effective SMT solvers

Bounded Model Checking (BMC)

- Given system specified by initiality predicate I and transition relation T on states S
- Is there a counterexample to property P in k steps or less?
- Find assignment to states $s_0, \ldots, s_k$ satisfying

  $$I(s_0) \wedge T(s_0, s_1) \wedge T(s_1, s_2) \wedge \cdots \wedge T(s_{k-1}, s_k) \wedge \neg(P(s_1) \wedge \cdots \wedge P(s_k))$$

- Given a Boolean encoding of I, T, and P (i.e., circuit), this is a propositional satisfiability (SAT) problem
- But if I, T and P use decidable but unbounded types, then it's an SMT problem: infinite bounded model checking
- (Infinite) BMC also generates test cases (and plans)
  - Counterexample to negation of property
- Extends from refutation to verification via k-induction

Example: Real Time

- Continuous time excludes automation by finite state methods
- Timed automata methods handle continuous time
  - But are defeated by the case explosion when (discrete) faults are considered as well
- SMT solvers can handle both dimensions
  - With discrete time, can have a clock module that advances time one tick at a time
    - ⋆ Each module sets a timeout, waits for the the clock to reach that value, then does its thing, and repeats
  - Better: move the timeout to the clock module and let it advance time all the way to the next timeout
    - ⋆ These are Timeout Automata (Dutertre and Sorea): and they work for continuous time
  - In addition, need k-induction, disjunctive invariants

Example: Biphase Mark Protocol

- Biphase Mark is a protocol for asynchronous communication
  - Clocks at either end may be skewed and have different rates, and jitter
  - So have to encode a clock in the data stream
  - Used in CDs, Ethernet
  - Verification identifies parameter values for which data is reliably transmitted
- Verified by human-guided proof in ACL2 by J Moore (1994)
- Three different verifications used PVS
  - One by Groote and Vaandrager used PVS + UPPAAL
  - Required 37 invariants, 4,000 proof steps, hours of prover time to check

Biphase Mark Protocol (ctd)

- Brown and Pike recently did it with sal-inf-bmc
  - Used timeout automata to model timed aspects
  - Statement of theorem discovered systematically using disjunctive invariants (7 disjuncts)
  - Three lemmas proved automatically with 1-induction,
  - Theorem proved automatically using 5-induction
  - Verification takes seconds to check
  - Demo:
    sal-inf-bmc -v 3 -d 5 -i -l l0 -l l1 -l l2 biphase t0
- Adapted verification to 8-N-1 protocol (used in UARTs)
  - Additional lemma proved with 13-induction
  - Theorem proved with 3-induction (7 disjuncts)
  - Revealed a bug in published application note

Fast SMT Solvers

- SMT solvers are being honed by competition
  - Initiated by Leonardo and Harald
  - Now institutionalized as part of CAV, FLoC
- Various divisions (depending on the theories considered)
  - Equality and uninterpreted functions
  - Difference logic ($x - y < c$)
  - Full linear arithmetic
    - ⋆ For integers as well as reals
  - Arrays . . . etc.
- ICS won in 2004
- Yices and Simplics (prototypes for next ICS) won the hard divisions in 2005, came second in all the others
- Next ICS should win in 2006

Building a Fast(er) SMT Solver

- Individual decision procedures need to be fast
  - Linear arithmetic procedure should be effective for difference logic (don't want a discrete switch)
- Need fast and effective interaction with the SAT solver
  - Good, but cheap explanations
  - Fast backtracking
- Congruence closure integrated with SAT for fast propagation
- Choices must be validated by extensive benchmarking
- A topic for a future talk by Bruno and Leonardo

Working With Inconsistent Knowledge

- In AI applications, often have inconsistent knowledge
  - E.g., from different sources, ignorance of true state
- Rather than UNSAT, we want a SAT assignment for some subset of constraints
- We can weight the knowledge according to "credibility," then want a SAT assignment of maximum weight: MaxSAT
- May also want to find the source of inconsistency: unsat core
- CALO needs these capabilities to draw conclusions from knowledge provided by different machine learners
  - Extension to reason about equality is attractive
- So we're building the Probabilistic Consistency Engine (PCE)
- A topic for a future talk by Tomas

MaxSAT via SMT

- This is not what we do, but gives the idea
- Description is simpler if we interpret weights as penalties for violating a constraint
- Then want assignment of minimum weight
- For a constraint C_i of weight W_i
- Assert $C_i \vee y_i = W_i$ to SMT solver, where y_i is a new arithmetic variable
  - Or, equivalently, $\neg C_i \supset y_i = W_i$
- In a satisfying assignment, $y_1 + y_2 + \cdots y_n$ is the total weight of violated constraints

Implementing MaxSAT via SMT (ctd.)

- So we can check whether a solution with weight at most m exists by asserting the constraint $y_1 + y_2 + \cdots y_n \leq m$ to SMT solver and asking whether the resulting set of clauses is satisfiable
- SMT solver can do this because it handles linear arithmetic
- We want a satisfying assignment of minimum weight
- But we know that all feasible m must lie between 0 and $M = W_1 + W_2 \cdots W_n$
- So do a binary search for the least m in $[0 \ldots M]$
- This requires $\log M$ invocations of SMT solver
- Can get anytime solutions (satisfiable but not necessarily minimal) by starting with a large value for m (e.g., M)

MaxSMT

- This is what we actually do (I think)
- CALO mostly needs MaxSAT (rather than MaxSMT)
- So start by making the SAT solver state of the art
  - Good cache utilization is vital
- Build the propagation over weights into the SAT core
  - Rather than delegate to arithmetic procedure of SMT
- Binary search destroys solver context
  - And repeatedly encounters phase transition region
  - So creep up to max from one side
  - Anytime solution is still possible
- Believed to be the fastest MaxSAT solver
  - And actually does MaxSMT
- A topic for a future talk by Tomas and Leonardo

Maximal Assignments

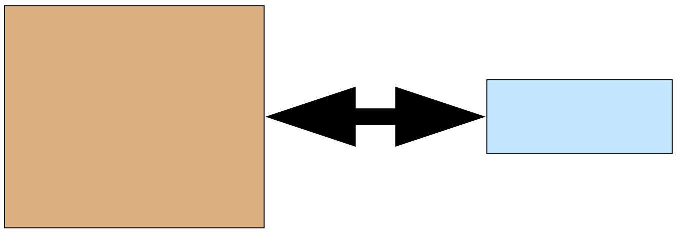
- The Simplex linear arithmetic solver decides whether a set of constraints is satisfiable
  - And can maximize any expression under those constraints
- Can solve an SMT problem, then maximize target expression under the satisfying assignment
- Then seek new assignments with larger maximum
  - Test the maximum periodically, and terminate branches that do not better current maximum
- Call this SMTmax, can probably extend to MaxSMTmax
- One use is test case generation
  - SMT covers the control structure
  - SMTmax allows boundary coverage

Spacecraft Autonomy for Vehicles and Habitats (SAVH)

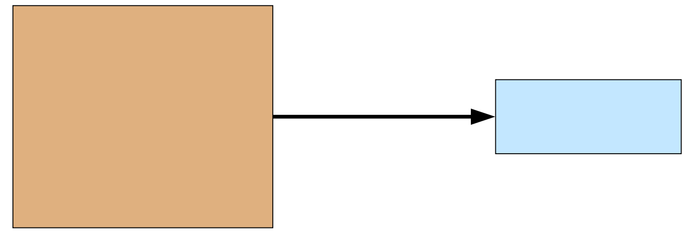
- Part of Return to the Moon
    - Looks like Apollo but much more automation
    - Though the astronauts can meddle
- Automation driven by planners (EUROPA2)
- And plan execution engines (PLEXIL)
- We're part of a V&V team
- Explore robustness of models, plans, executions
- I suspect MaxSMTmax will allow new approaches here (see later)
- A topic for a future talk by Shankar

SMT as Disruptive Technology: Verification

Backend verifiers

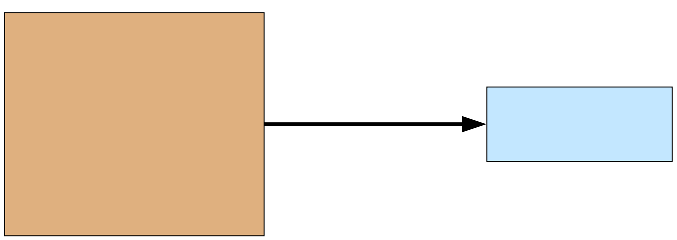


Integrated

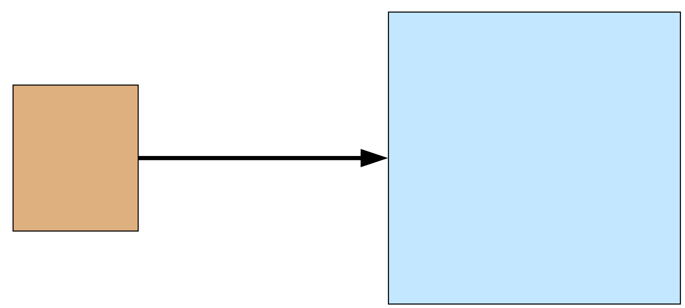


Endgame

Evolution of endgame verifiers



Decision Procedures



SMT solver

SMT as Disruptive Technology: Beyond Verification

- Modern formal methods tools do more than verification
- They also do refutation (bug finding)
- And test-case generation
- And controller synthesis
- And construction of abstractions and abstract interpretation
- And generation of invariants
- And . . .
- Observe that these tools can return objects other than verification outcomes
    - Counterexamples, test cases, abstractions, invariants

  Hence, heterogeneous integration

Integration of Heterogeneous Components

Effective tools are specialized often integrate many components

For example, software model checkers generally have:

- C front end with CFG analyzer
- Predicate abstractor
  - Which uses decision procedures
  - And possibly a model checker
- Model checker and counterexample generator
- Counterexample concretizer and refinement generator
  - Which uses Craig interpolation
  - Or unsat cores

And a control loop around the whole lot

Another Example: LAST

- LAST (Xia, DiVito, Muñoz) generates MC/DC tests for avionics code involving nonlinear arithmetic (with floating point numbers, trigonometric functions etc.)
- Applied it to Boeing autopilot simulator
  - Modules with upto 1,000 lines of C
  - 220 decisions
- Generated tests to (almost) full MC/DC coverage in minutes

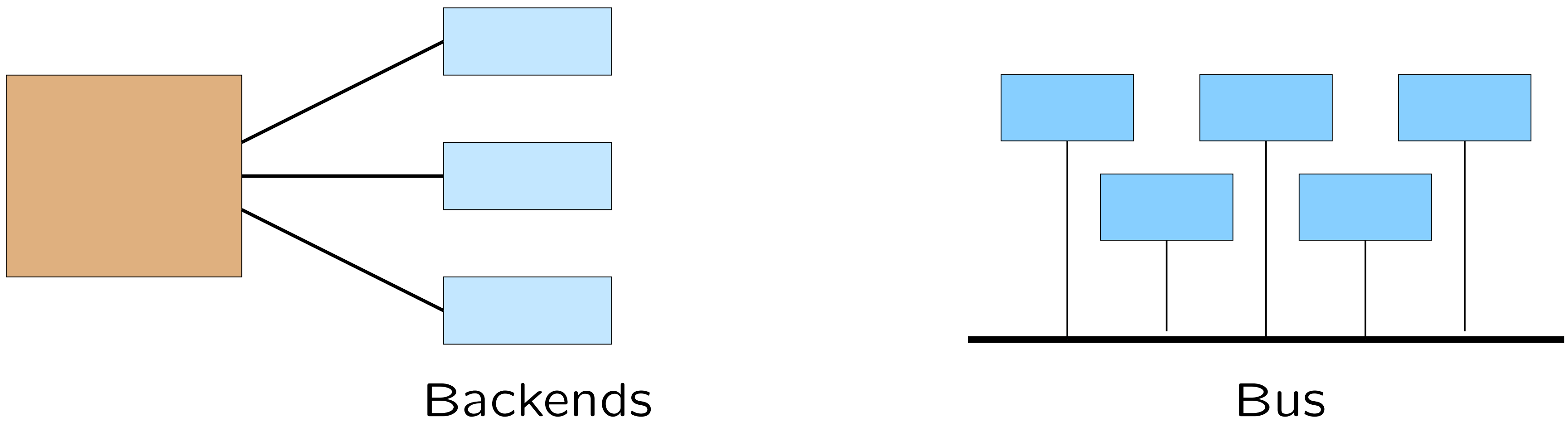
Structure of LAST

- It's built on Blast (Henzinger et al)
  - A software model checker, itself built of components
  - Including CIL and CVC-Lite
- But extends it to handle nonlinear arithmetic using RealPaver (a numerical nonlinear constraint unsatisfiability checker)
  - Added 1,000 lines to CIL front end for MC/DC
  - Added 2,000 lines to RealPaver to integrate with CVC-Lite (Nelson-Oppen style)
  - Changed 2,000 lines in Blast to tie it all together
- Aside: note they chose CVC-Lite rather than ICS
  - CVC-Lite is a very poor SMT solver
  - But it's more open than ICS
  - Combination is unsound, but that's ok for refutation

A Tool Bus

- How can we construct these customized combinations and integrations easily and rapidly?
- The integrations are coarse-grained (hundreds, not millions of interactions per analysis), so they do not need to share state
- So we could take the outputs of one tool, massage it suitably and pass it to another and so on
- A combination of XML descriptions, translations, and a scripting language could probably do it
- Suitably engineered, we could call it a tool bus

From Backends to Bus



- Bus is a federation of equals
- Theorem prover is just another component

But . . .

- But we'd need to know the names and capabilities of the tools out there and explicitly to script the desired interactions
  - And we'd be vulnerable to change
- Whereas I would like to exploit whatever is out there
  - And in 15 years time there may be lots of things out there
- That is, I want the bus to operate declaratively
  - By implicit invocation
- And I want evidence that supports the overall analysis
  (i.e., the ingredients for a safety or assurance case)
- That is, I want a semantic integration

A Formal Tool Bus

- The data manipulated by tools on bus are formulas in logic
- In fact, they can be seen as formulas in a logic
  - The Formal Tool Bus Logic
  - Each tool operates on a sublogic
  - Syntactic differences masked with XML wrappers
- No point in limiting the expressiveness of the tool bus logic
  - Should be at least as expressive as PVS
    - ⋆ Higher order, with predicate, structural, and dependent subtypes, abstract data types, recursive and inductive definitions, parameterized theories, interpretations
  - With structured representations for important cases
    - ⋆ State machines (as in SAL), counterexamples, process algebras, temporal logics . . .
    - ⋆ Handled directly by some tools, can be expanded to underlying semantics for others

Tool Bus Judgments

The tools on the bus evaluate and construct predicates over expressions in the logic—we call these judgments

Parser: A is the AST for string S

Prettyprinter: S is the concrete syntax for A

Typechecker: A is a well-typed formula

Finiteness checker: A is a formula over finite types

Abstractor to PL: A is a propositional abstraction for B

Predicate abstractor: A is an abstraction for formula B wrt. predicates ϕ

GDP: A is satisfiable

GDP: C is a context (state) representing input G

SMT: ρ is a satisfying assignment for A

Tool Bus Queries

- Tools publish their capabilities and the bus uses these to organize answers to queries

  Query: well-typed?(A)

  Response: PVS-typechecker(...) $\vdash$ well-typed?(A)

  The response includes the exact invocation of the tool concerned

- Queries can include variables

  Query: predicate-abstraction?(a, B, ϕ)

  Response:

  SAL-abstractor(...) $\vdash$ predicate-abstraction?(A, B, ϕ)

  The tool invocation constructs the witness, and returns its handle A

Tool Bus Operation

- The tool bus operates like a distributed datalog framework, chaining on queries and responses
- Similar to AIC's Open Agent Architecture
  - And maybe similar to MyGrid, Linda, . . . ?
- Can have hints, preferences etc.
- Tools can be local or remote
- Tools can run in parallel, in competition
- The bus needs to integrate with version management

Scripting

Three levels of scripting

Tools:

- Tools should be scriptable
- Better functionality, performance than wrappers
- E.g., SAL model checkers are Scheme scripts over an API
- Test generator is another script over the same API

Wrappers:

- Some functionality can be achieved by a little programming and maybe some tool invocation

Tool Bus:

- Scripts are chains of judgments

Tool Bus Scripts

- Example
  - If A is a finite state machine and P a safety property, then a model checker can verify P for A
  - If B is a conservative abstraction of B, then verification of B verifies A
  - If A is a state machine, and B is predicate abstraction for A, then B is conservative for A
- How do we know this is sound?
- And that we can trust the computations performed by the components?

An Evidential Tool Bus

- Each tool should deliver evidence for its judgments
  - Could be proof objects (independently checkable trail of basic deductions): research topic 'cos raw objects too big
  - Could be reputation ("Proved by PVS")
  - Could be diversity ("using both ICS and CVC-Lite")
  - Could be declaration by user
    - ⋆ "Because I say so"
    - ⋆ "By operational experience"
    - ⋆ "By testing"
- And the tool bus assembles these (on demand)
- And the inferences of its own scripts and operations
- To deliver evidence for overall analysis that can be considered in a safety or assurance case—hence evidential tool bus

The Evidential Tool Bus

- There should be only one evidential tool bus
- Just like only one WWW
- How to do it?
  - Standards committee?
  - Competition and cooperation!
- Probably not difficult to integrate multiple buses
  - Need agreement on ontologies
  - Fairly minimal glue code to link them together
- I'd like to build one
  - Initially to integrate PVS and SAL
  - And to reconstruct Hybrid-SAL
- A topic for a future talk by Sam

SMT as Disruptive Technology: AI

- SMT solvers can do metric and temporal planning for AI
  - Rather like test generation with BMC
  - But a planning language and front end (e.g., STRIPS) generates better problems for the SMT solver
  - Demonstrated by Bart Peintner et al using ARIO
- And MaxSMT should be good for model based diagnosis
- Conjecture that SMT solvers have stronger foundation, higher performance than heuristic planning and constraint engines, and greater power than pure SAT solvers
  - Adopt their good ideas, if any
- Anything a SAT solver can do, an SMT solver can do better
- Want to investigate this, and opportunities in AI