SMT Solvers:
A Disruptive Technology

John Rushby

Computer Science Laboratory
SRI International
Menlo Park, California, USA

SMT Solvers

- SMT stands for Satisfiability Modulo Theories
- SMT solvers generalize SAT solving by adding the ability to handle arithmetic and other decidable theories
- SAT solvers are used for
  - Bounded model checking, and
  - AI planning,

  among other things
- Anything a SAT solver can do, an SMT solver can do better
- I'll describe these from the informed consumer's point of view

Overview

- SAT solving
- SMT solvers
- Application to verification
  - Via bounded model checking and k-induction
  - With a demo
- Application to AI planning and scheduling
  - With a demo
- Extensions to MaxSMT and OptSMT
- Conclusions

SAT Solving

- Find satisfying assignment to a propositional logic formula
- Formula can be represented as a set of clauses
  - In CNF: conjunction of disjunctions
  - Find an assignment of truth values to variable that makes at least one literal in each clause TRUE
  - Literal: an atomic proposition A or its negation $\bar{A}$
- 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 1,000,000s of 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 a commodity technology
  - MiniSAT is 700 SLOC
- Can think of it as massively effective 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

SAT Plus Theories

- SAT can encode operations and relations on bounded integers
  - Using bitvector representation
  - With adders etc. represented as Boolean circuits

  And other finite data types and structures
- But cannot do not unbounded types (e.g., reals), or infinite structures (e.g., queues, lists)
- And even bounded arithmetic can be slow when large
- 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

Decision Procedures

- Decision procedures are specific to a given theory
- Tell whether a formula is inconsistent, satisfiable, or valid
- Can decide conjunctions of formulas
- Or whether one formula is a consequence of others
  - E.g., does $4 \times x = 2$ follow from $x \leq y$, $x \leq 1 - y$, and $2 \times x \geq 1$ when the variables range over the reals?
- Decision procedures may use heuristics for speed, but must always give the correct answer, and terminate
  (i.e., must be sound and complete)

Decidable Theories

- Many useful theories are decidable
  (at least in their unquantified forms)
  - Equality with uninterpreted function symbols
    $x = y \wedge f(f(f(x))) = f(x) \supset f(f(f(f(f(y))))) = f(x)$
  - Function, record, and tuple updates
    $f \textbf{ with } [(x) := y](z) \stackrel{\text{def}}{=} \textbf{if } z = x \textbf{ then } y \textbf{ else } f(z)$
  - Linear arithmetic (over integers and rationals)
    $x \le y \wedge x \le 1 - y \wedge 2 \times x \ge 1 \supset 4 \times x = 2$
  - Special (fast) case: difference logic
    $x - y < c$
- Combinations of decidable theories are (usually) decidable

$e.g., 2 \times car(x) - 3 \times cdr(x) = f(cdr(x)) \supset$

$f(cons(4 \times car(x) - 2 \times f(cdr(x)), y)) = f(cons(6 \times cdr(x), y))$

Uses equality, uninterpreted functions, linear arithmetic, lists

SMT Solving

- Individual and combined decision procedures decide conjunctions of formulas in their decided theories
- 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
  - i.e., $(A \vee B) \wedge (C \vee D) \wedge E$
- Get a solution from a SAT solver (if none, we are done)
  - e.g., A, D, E
- Restore the interpretation of variables and send the conjunction to the core decision procedure
  - i.e., $x \leq y \wedge y \leq x \wedge x \neq y$

SMT Solving by "Lemmas On Demand"

- 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)
  - $A \wedge D \supset \bar{E}$ (equivalently, $\bar{A} \vee \bar{D} \vee \bar{E}$)
- Iterate to termination
  - e.g., $A, C, E, \bar{D}$
  - i.e., $x \leq y, x < 0, x \neq y, y \not\leq x$ (simplifies to $x < y, x < 0$)
  - A satisfying assignment is $x = -3, y = 1$
- This is called "lemmas on demand" (de Moura, Ruess, Sorea) or "DPLL(T)"; it yields effective SMT solvers

Fast SMT Solvers

- There are several effective SMT solvers
  - Ours are ICS (released 2002),
    Yices, Simplics (prototypes for next ICS)
  - European examples: Barcelogic, MathSAT
- SMT solvers are being honed by competition
  - Provoked by our benchmarking in 2004
  - Now institutionalized as part of CAV, FLoC

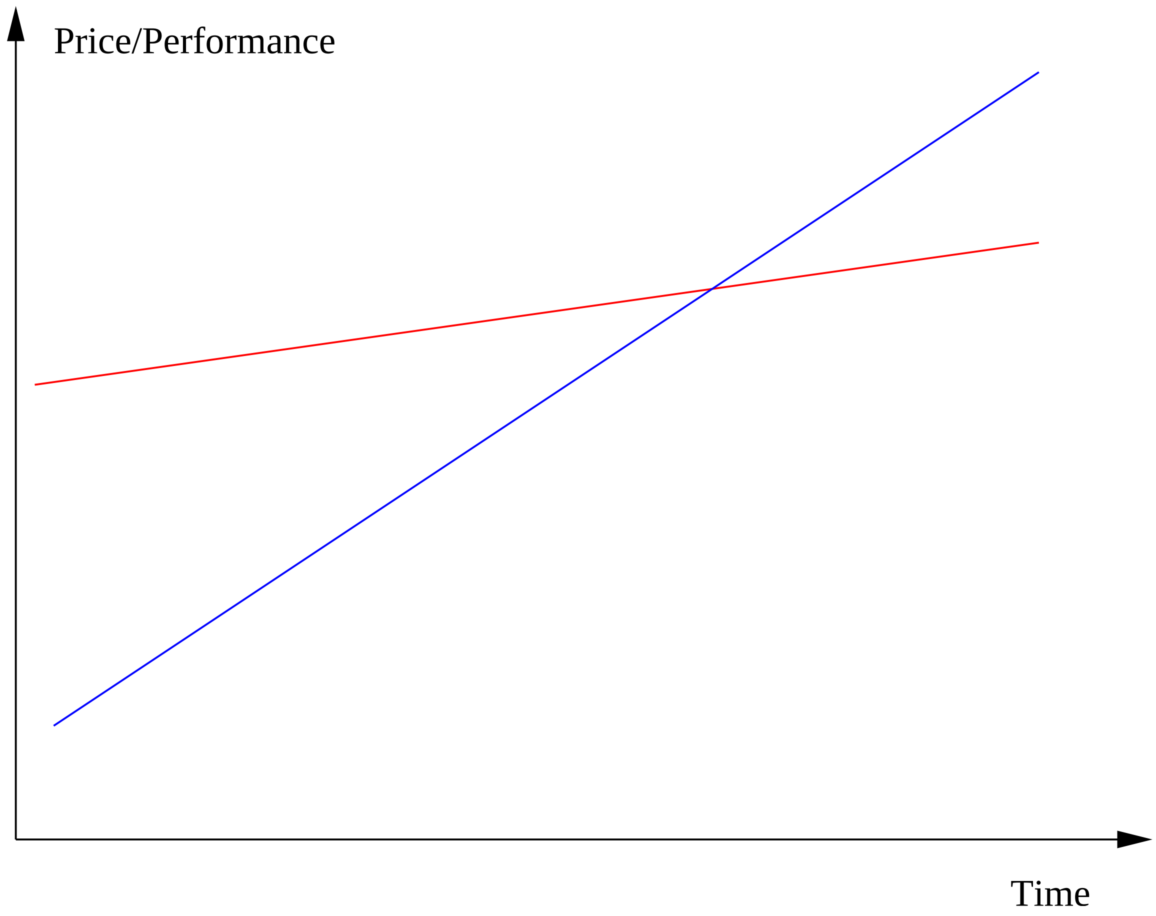
SMT Competition

- 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 to Barcelogic in all the others
  - Let's take a look

Building Fast(er) SMT Solvers

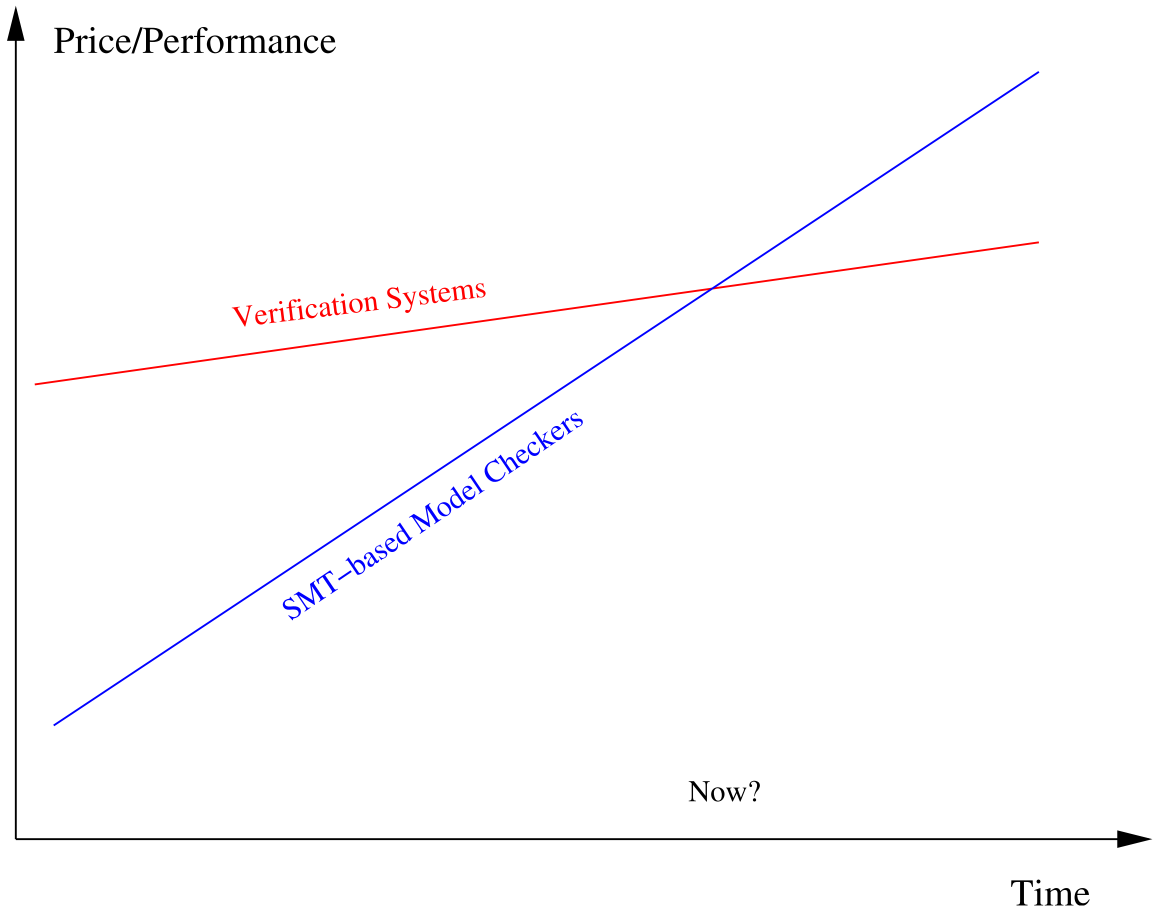
- Individual decision procedures need to be fast
  - Especially linear arithmetic (Simplex)
  - Linear arithmetic procedure should also be effective for difference logic (not a discrete switch to Bellman-Ford)
- Need fast and effective interaction with the SAT solver
  - Good, but cheap explanations
  - Fast backtracking
- SAT solver must be fast, good cache performance
- Equality integrated with SAT for fast propagation
- Choices must be validated by extensive benchmarking
- Look out for the 2006 competition

Disruptive Technology

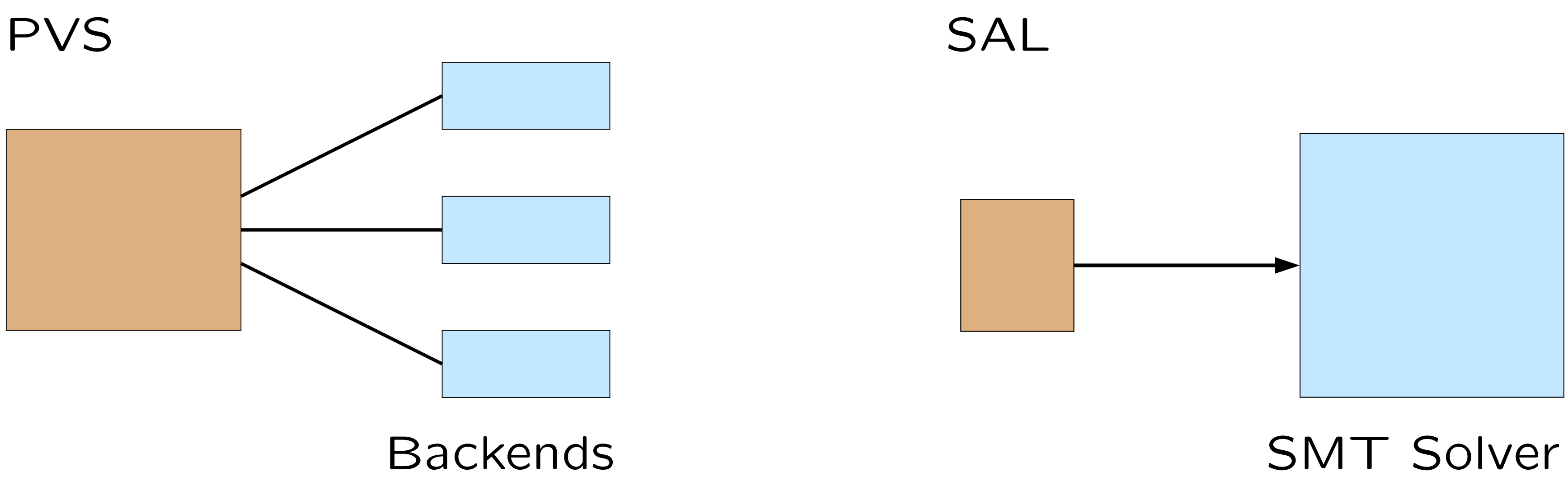


Disruption is when low-end technology overtakes the price performance of high-end

SMT Solvers as Disruptive Technology



Verification Systems vs. SMT-Based Model Checkers



Actually, both kinds will coexist as part of the evidential tool bus—the topic for a different talk

Evolution of SMT-Based Model Checkers

- Replace the backend decision procedures of a verification system with an SMT solver, and specialize and shrink the higher-level proof manager
- Example:
  - SAL language has a type system similar to PVS, but is specialized for specification of state machines
    (as transition relations)
  - The SAL infinite-state bounded model checker uses an SMT solver (ICS), so handles specifications over reals and integers, uninterpreted functions
  - Often used as a model checker (i.e., for refutation)
  - But can perform verification with a single higher level proof rule: k-induction (with lemmas)
  - Note that counterexamples help debug invariant

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
  - State the goal as negated property

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

k-Induction

- BMC extends from refutation to verification via k-induction
- Ordinary inductive invariance (for P):

  Basis: $I(s_0) \supset P(s_0)$

  Step: $P(r_0) \wedge T(r_0, r_1) \supset P(r_1)$
- Extend to induction of depth k:

  Basis: No counterexample of length k or less

  Step: $P(r_0) \wedge T(r_0, r_1) \wedge P(r_1) \wedge \cdots \wedge P(r_{k-1}) \wedge T(r_{k-1}, r_k) \supset P(r_k)$

  These are close relatives of the BMC formulas
- Induction for $k = 2, 3, 4 \ldots$ may succeed where $k = 1$ does not
- Is complete for some problems (e.g., timed automata)
  - Fast, too, e.g., Fischer's mutex with 83 processes

Application: Verification of Real Time Programs

- 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

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

Application: AI Planning and Scheduling

- This is speculative: I don't know much about AI planning
- SAT-based planning is essentially the same technology as BMC
  - Uses different languages in front (e.g., PDDL)
  - And may be able to break into independent subproblems
- SMT-based planning is similar, except we can have metric quantities like mass, power, and can do scheduling over real time
  - Because we can do arithmetic

Example: Simple Rover

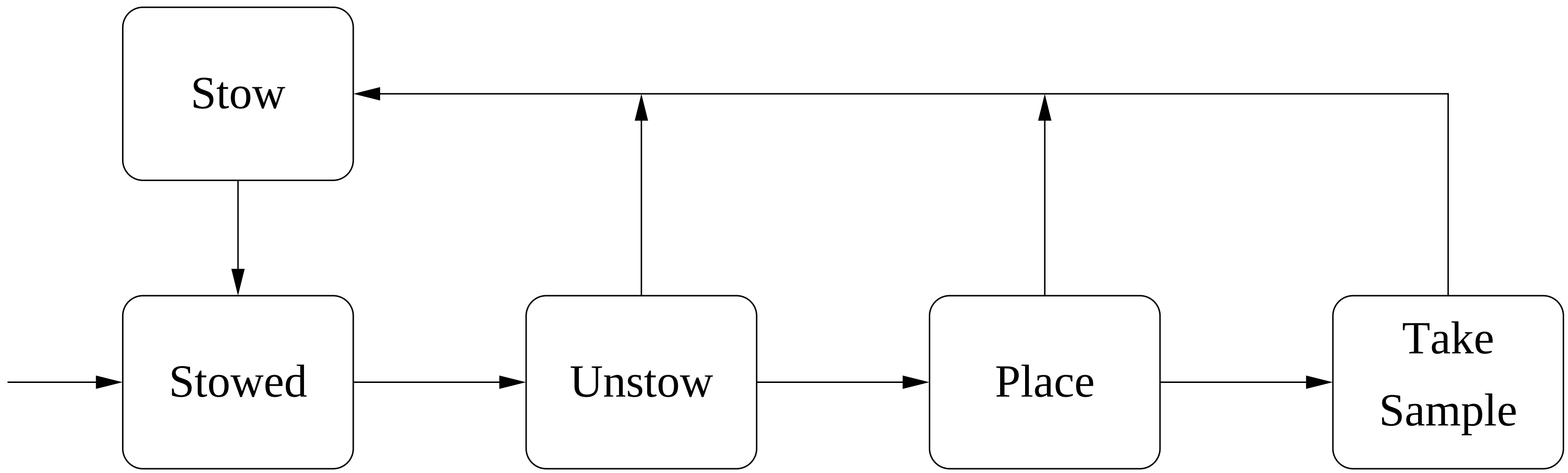
- Consider a simple planetary rover with three components
  - Navigator
  - Instrument
  - Radio

  Each consume power and take time to do their things
- We have flight rules
  - Must not move while the instrument is unstowed
- And a goal
  - Go to Rock4, take a sample, and radio it back
  - Without depleting the battery

Rover Navigator

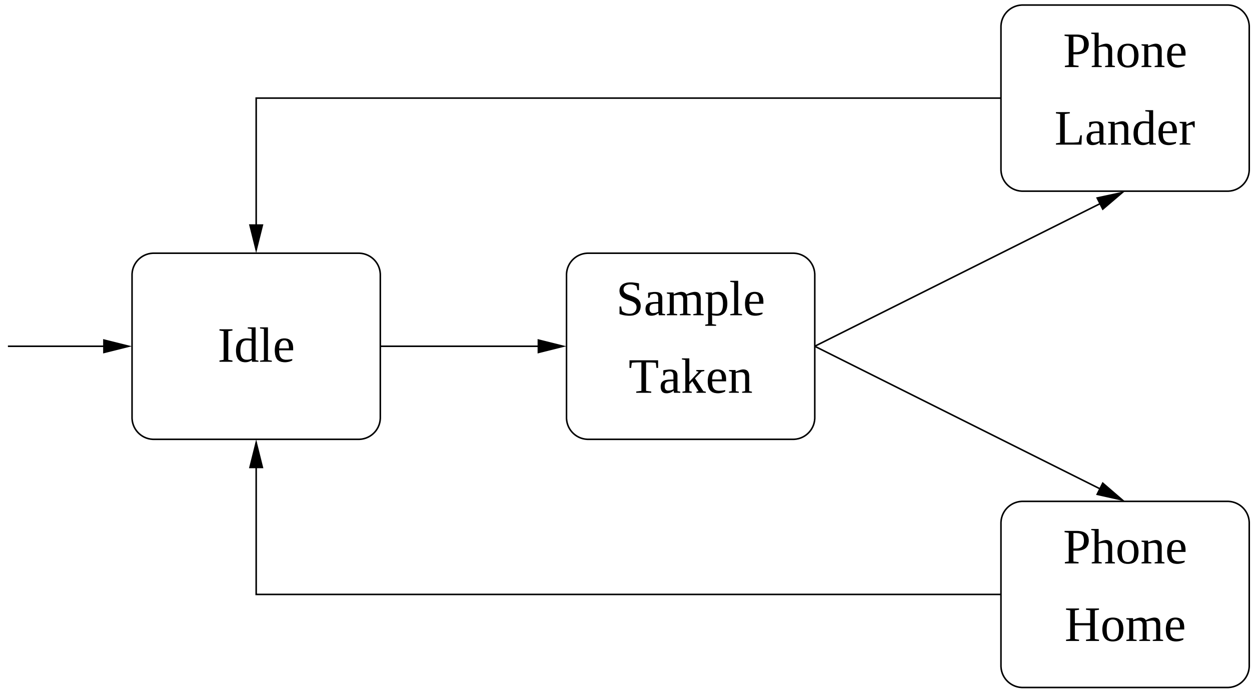
- Takes at least 10 mins to get anywhere
- Consumes 400 mwh of battery power

Rover Instrument



- Takes between 2 and 6 mins to stow/unstow, uses 20 mwh
- Takes between 3 and 12 mins to place
- Takes between 20 and 25 mins to sample, uses 120 mwh

Rover Radio



- Starts transmission within 20 to 25 mins of sample
- Chooses nondeterministically between lander and home
- But home uses 600 mwh, lander uses 20 mwh
- Both take between 2 and 5 mins

Rover Flight Rules

- Rover must not move while the instrument is unstowed
- Original spec wove this into the descriptions of Navigator and Instrument
- Instead, we encode it in a synchronous observer which says OK as long as flight rules are satisfied

Rover Goals

- Go to Rock4, take a sample, and radio it back
- Without depleting the battery (really a flight rule)
- Can state these in the goal property, or use another synchronous observer
  - We do both

Rover System and Plan Description

- System is asynchronous composition of the components
  - And the clock
- All synchronously composed with the flight rules and goal observers
- ```
  System: MODULE = (Nav [] Instr [] Radio [] Clock)
                         || flight_rules || goals;
  ```
- Plan requires satisfaction of properties observed by flight rules and goals, plus others stated directly
  - All negated inside an invariant
- ```
  sched_sys: THEOREM System |- AG(NOT(
     OK AND done
     AND measurement_done
     AND battery > 0));
  ```

Plan Output

demo: sal-inf-bmc -v 3 rover sched_sys -d 14

time = 0 nav_get_going

time = 50 nav_arrive

time = 50 instr_unstow

time = 56 instr_place

time = 68 instr_take_sample

time = 68 radio_note_samp

time = 91 inst_stow

time = 91 radio_ready_to_phone

time = 96 radio_phone_lander

- Martha Pollack et al have done similar with SMT solver Ario
- Need to benchmark performance against conventional planner
- I certainly prefer our specification

Optimization

- We have an automated test case generator `sal-atg`
- Takes specifications annotated with trap variables for structural coverage goals
- And incrementally finds long tests that visit many goals in sequence
- Works by greedily reaching any goal, then extending the test by restarting the bounded model checker from there
- Implemented as less than 100 lines of Scheme script (SAL is scriptable)
- Speculate that we can generate long plans for multiple goals in a similar way

Extensions to MaxSMT and OptSMT

- 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
- These can be implemented by SMT and extended to MaxSMT
- May also want not just a satisfying assignment to an SMT problem, but one that maximizes some specific constraint: OptSMT

MaxSAT via SMT

- This is not what we actually 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
- Can obviously find a solution with weight $M = W_1 + W_2 \cdots W_n$

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 closer what we actually do
- 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
- Actually does MaxSMT, MaxSAT as special case
- But believed to be the fastest MaxSAT solver

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 OptSMT, can probably extend to OptMaxSMT
- One use is test case generation
  - SMT covers the control structure
  - OptSMT allows boundary coverage

Conclusions

- SMT makes SAT much more useful
  - More expressive
  - More efficient
- Many problems can be cast as SAT, SMT, MaxSMT, OptSMT
- And can then use these powerful solvers
- - Off the shelf automation, so new areas can be automated
  - And combination problems can use a single solver
- Specialized solvers may be relegated to niches
  - This is disruption
  - Needs to be validated by benchmarking
- Planned extensions to SMT solvers: bitvectors, quantifier elimination, evidence

To Learn More

- Our systems, PVS, SAL, ICS and our papers are all available from http://fm.csl.sri.com
- Slides available at http://www.csl.sri.com/users/rushby/slides
- Thanks to Bruno Dutertre, Grégoire Hamon, Leonardo de Moura, Sam Owre, Harald Rueß, Hassen Saïdi, N. Shankar, and Maria Sorea