From Satisfiability to Verification Modulo Theories

http://www.vmt-lib.org/

Satisfiability Modulo Theories

- Fragments of first order logic
- Symbols interpreted with respect to background theory
  - Arithmetic
  - Uninterpreted functions
  - Arrays
- Satisfiability modulo the background theory
  - Is there a satisfying theory-interpretation for the the given formula?

SMT-LIB, SMT-COMP

- SMT solvers
  - YICES, CVC, OpenSMT, MathSAT, Z3, …
- SMT-LIB initiative
  - definition of a standard language
  - creation of a large collection of benchmarks
- SMT-COMP
  - Compare SMT solvers on common benchmarks
  - fostered tremendous progress in the performance
- Other benefits to the field
  - Higher quality perception from user base
  - Users suggest research directions

Why SMT?

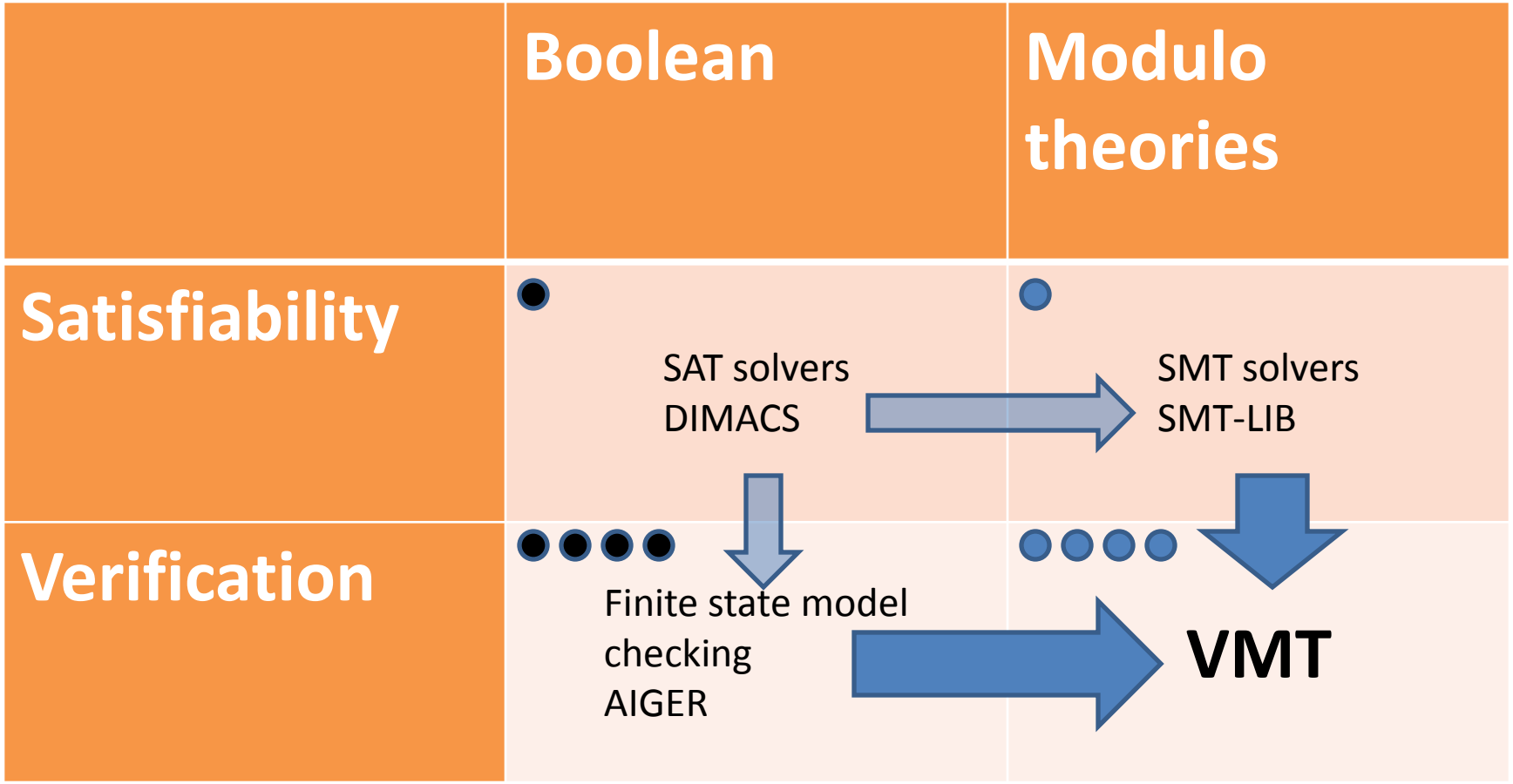
- Many practical problems in verification arise from the analysis of *transition systems* that can be naturally represented in symbolic form within the SMT framework
- “Modulo theory” as “beyond boolean”
- Some domains “beyond boolean”:
  - word-level circuits
  - timed systems
  - hybrid systems
  - microcode
  - software
- Industrial users: Microsoft, Intel, RockwellCollins, …

What is not in SMT

- Satisfiability
  - One formula, find satisfying assignment
  - *Combinatorial*
- Verification
  - One transition system
  - Find behaviour of interest
  - *Sequential*

- Key notion
  - dynamic aspect implicit in transition system
  - Nowhere in SMT

Satisfiability vs Verification



The Verification Modulo Theories initiative

- Focus
  - verification problems
  - for transition systems
  - symbolically described in SMT

- Aims:
  - Define a standardized language
  - Collect a library of benchmarks
  - Empower tools community
  - Set up a competition
  - Launch a workshop

Why SMT-LIB is not enough?

- SMT does not allow for direct modeling of dynamic aspects!
  - Reachability
  - Termination
  - Complex temporal properties

- Doesn't SMT-LIB contain some verification problems?
  - But they are engine-specific problems!
    - bounded reachability problems
    - proof obligations from inductive proofs
  - How about other techniques?
    - FRAIG-based analysis?
    - Abstraction-refinement at the level of the transition system?
    - Combination with finite-state model checking?
    - Multiple properties, over-approximated reachability

First-Order Transition systems: intuitions

- Basic ingredients
  - States
  - initial states
  - transition relation

- Induces a state-transition structure
  - Cfr Kripke structure in discrete case
    - Each state labeled with true propositions
    - An interpretation for boolean state variables
    - One transition corresponds to one edge
  - Add “theory” information in each world
    - Each state has an interpretation for state variables

Symbolic Transition System

- V as vector (current) state variables
  - State as assignment to V
  - Variables may have complex domains: arrays, maps, relations, …
- V’ as (next) state variables

- I, T as SMT formulae
- I(V) initial states
- T(V, V’) transition relation

- We “only” need a **`next(.)`** operator
  - **`next(x) = x + 3`**
- Remark: **`next(.)`** is not a function symbol!
  - a decoration to support the automatic generation of the primed variables, …
  - and the implicit mapping between current and primed variables
- Remark: we can not have **`cur_x`** and **`next_x`**

Rigid vs Flexible interpretation

- Rigid: interpretation retained over all states in trace
- Flexible: interpretation can change in different state

- Rigid symbols
  - E.g. functional block abstraction, ALU(V)
  - Parameters such as threshold to model delays in cyclic processes

- Flexible symbols
  - Time dependent In flow / out flow, unknown analytic form
  - Axioms might be used to limit the value of functions

Functional vs Relational

- Next state defined as *function* of current state:
  - $\mathbf{next(v_i) := F_i(v)}$
  - $\mathbf{next(v_i) := F_i(v, next(v_{j<i}))}$

- Next states in *relation* with current ones
  - $\mathbf{next(v_1) + next(v_2) <= v_1 + v_2}$

Modeling Style

- Per-variable modeling
  - For each variable, state under which conditions it changes value
- Per-transition modeling
  - For each transition, state preconditions and effects on all variables
  - Disjunctive
    - Precond & effects
  - Conjunctive
    - Precond -> effects
  - Equivalent only under specific conditions

Modeling constructs

- Which language constructs?
  - **`ASSIGN`**
  - **`INIT, TRANS, INVAR`**

- About deadlocks
  - Functional approach guarantees deadlock freedom if functions are total
  - Relational approach, invariants: all bets are off…

Components and composition

- Single component vs multiple components
  - May be useful at a high level
  - Hard to standardize
- Which forms of composition?
  - Synchronous
  - asynchronous
- Proposal: synchronous
  - Logic-based modeling inherently synchronous
  - Asynchronous composition requires suitable encoding
    - See HyDI language in NuSMV

Inertia

- Logic vs Law of inertia?
  - does a variable change if not stated otherwise?
  - Suppose **`next(x) := x +1`**
  - How about **`y`**, **`z`**, **`w`**, …?
- Syntactic sugar?
  - Need to identify “affected vars”

Parameterization

- Ground vs paramenterized?
- Some descriptions are parameterized over finite known domain
  - Classical planning
    - **`move-from-to(?b:block, ?f:loc, ?t:loc)`**
  - Security protocols
    - **`send(?m:msg, ?c:chan)`**
  - Grounding upfront may lead to blow ups

Temporal Properties

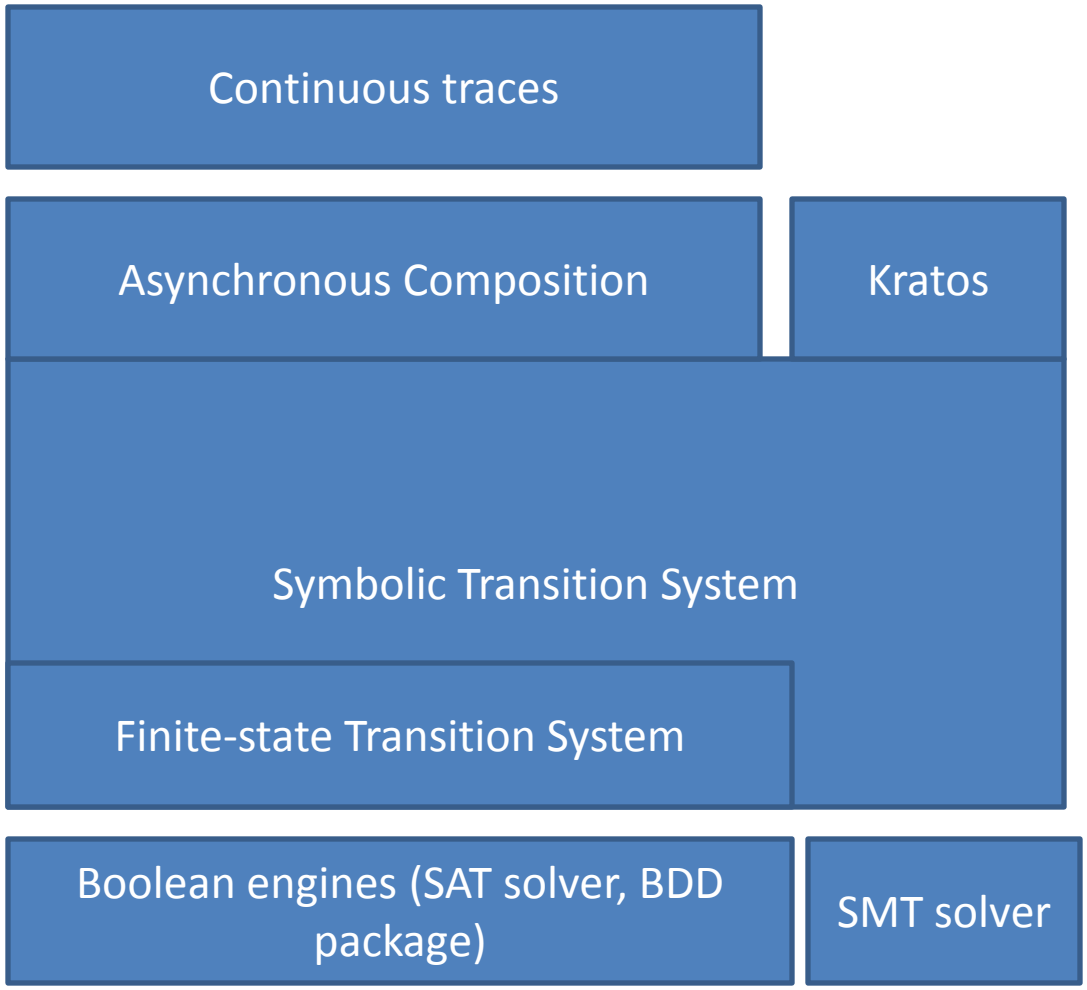
- Languages to express properties
  - Invariants
  - Temporal properties (CTL, LTL, RELTL, …)
  - Fairness conditions
  - Termination
- Single property vs multiple properties

Verification Modulo Theories: Tools

- UCLID
  - Mutable functions
  - Finite horizon
- VAPOR
  - Uninterpreted functions
- SAL
  - Parameterized timed/hybrid systems
- MCMT
  - Parameterized infinite state systems
- NuSMV
  - SMT-based extension, tight integration with MathSAT

The NuSMV layering

- L0: Finite state
- L1: STS
- L2: asynch
- L3: from discrete to continuous traces



What is likely out?

- Program-to-program properties
  - Equivalence checking
  - Refinement checking
- Sequential software requires built-in support for
  - Inertia
  - Control-flow graph
  - Recursion
  - Memory model
- Concurrent software
  - Various forms of synchronization, preemption, resource contention, …
- Concrete languages (e.g. MISRA C, AADL)
  - Cfr boogie

Links to other initiatives

- SMT-LIB
  - Leverage as much as possible
  - Ideally, VMT grammar reuses SMT grammar
- NTS competition
  - Need to sync
    - E.g. role of CFG, inertia
    - First step: benchmark conversion
  - NIA category in VMT-LIB?

Who is on board?

- We welcome on board everyone interested!
- Idea informally discussed with Leonardo De Moura, Bruno Dutertre, Viktor Kunchak, Armin Biere, Sanjit Seshia, and many people in Trento
- Who may be interested
  - SAL (Dutertre), UCLID (Sanjit Seshia), NTS (particular case), MCMT (Ranise, Ghilardi, Bruttomesso), VAPOR (Sakallah)
- Challenge for the Rich Models Toolkit action?

Next steps

- Public announcement
- Web site
  - http://www.vmt-lib.org
- Mailing list at
  - vmt-discussion@fbk.eu
- Proposal for concrete VMT language
- Benchmarks collection

Conclusions

- SMT
  - Impressive increase of expressiveness
  - Limited to combinational case
- VMT
  - Same expressiveness
  - Lift to natively deal with transition systems

- A new generation of verification engines!

Comments

- Why not joining SMT? [Natasha]
- Control flow graph [Alessandro, Cesar, Marque]
  - Reserved keyword/annotation
- Asynchronous composition/Scheduler? [Cesar]
- Fairness in model or in property? [Cesar]
  - Fair transition system
  - But, careful with fair states/
- Can we deal with quantifiers? [Barbara]
- Can we express games? With quantifiers? Directly? [Barbara]