Pluggable SAT-Solvers for SMT-Solvers

Bas Schaafsma

DISI, University of Trento
&
Fondazione Bruno Kessler

May 29, 2013

The SMT problem

The SAT problem

Given a Boolean formula $\mathcal{F}$, is there an assignment for which $\mathcal{F}$ evaluates to `true`?

The SMT problem

SAT extended with a set of theories $\mathcal{T}_1 \cup \mathcal{T}_2 \cdots \cup \mathcal{T}_n$

Example ($\mathcal{EUF} \cup \mathcal{LA}(\mathbb{Z})$):

$$(x + 2y = 6 \vee y = 9) \wedge \neg(f(x) = f(y)) \wedge x = 2$$

Some Useful Theories

Theory of Linear Arithmetic ($\mathcal{LA}$)

$$\mathcal{F}_{\mathcal{LA}} = L_{(x=2)} \wedge L_{(x<3)}$$

Theory of BitVectors ($\mathcal{BV}$)

$$\mathcal{F}_{\mathcal{BV}} = L_{(zext^{\langle 2 \rangle}(x^{\langle 2 \rangle})) >_u 15^{\langle 4 \rangle})}$$

Theory of Arrays ($\mathcal{ARR}$)

$$\mathcal{F}_{\mathcal{ARR} \cup \mathcal{LA}} = L_{(read(a,1)=2)} \wedge \neg L_{(read(write(a,3,1),i)=read(a,i))}$$

Applications of SMT-Solvers

- Bounded Model Checking
- Equivalence Testing [GPB01]
- Property Driven Reachability Testing [CNR12]
- Scheduling [ABP+11]
- Test Case Generation [GLM12]
- Software model checking through Predicate Abstraction [FQ02]
- Program Synthesis [SGCF11]
- ...

Pluggable SAT solvers: Motivation

- Developing a new (allround) SMT solver entails more than a new SAT solver. $\rightarrow$ MathSAT5 $\sim$ 150kloc vs MiniSAT $\sim$ 6kloc
- Success of SAT solvers highly dependent on heuristics.
- Tuning SAT solvers requires investment of time and money.
- SAT-Solver is a deciding factor for $\mathcal{BV}$ & $\mathcal{BV} \cup \mathcal{ARR}$ instances.
- We want to combine state-of-the-art SAT solvers & SMT solvers.
- This is NOT a straight forward bitblasting approach.

The DPLL algorithm

```
Preprocess(F)
while true do
    BCP(F)
    if not conflict then
        if all variables assigned then
            return SAT
        end if
        decide()
    else
        C_conflict ←analyze()
        if top level conflict found then
            return UNSAT
        end if
        backtrack(C_conflict)
    end if
end while
```

DPLL($\mathcal{T}$) = DPLL + ..

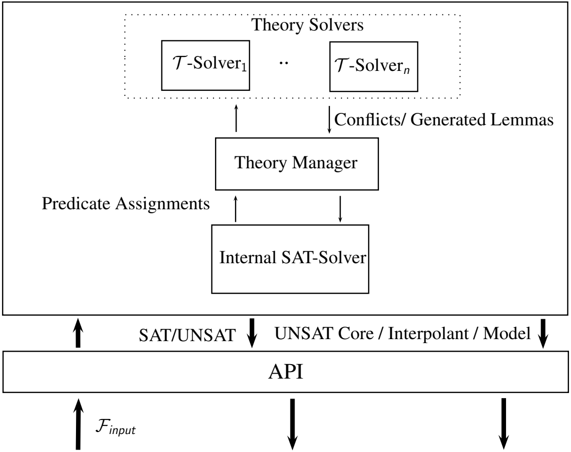
- For correctness:
  - Theory consistency checks.
  - Case splitting.
- For optimization:
  - Early pruning.
  - Theory deductions.
- (Incrementality)

For specific details check [ST09].

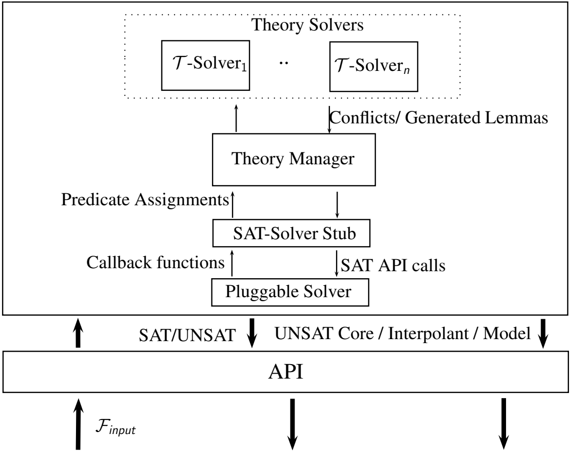
(Simplified) DPLL($\mathcal{T}$) algorithm

Preprocess($\mathcal{F}$)
while true **do**
    BCP($\mathcal{F}$)
    if not conflict and theories consistent **then**
        if all variables assigned and no case splitting needed.
        then
            return SAT
        end if
        decide()
    else
        $C_{\text{conflict}} \leftarrow$ *analyze()*
        if top level conflict found **then**
            return UNSAT
        end if
        backtrack(C_{conflict})
    end if
end while

DPLL($\mathcal{T}$) Architectural Overview



DPLL($\mathcal{T}$) + Pluggable Solver Architectural Overview



Pluggable SAT solvers: A quick overview

- 3rd Party SAT solvers can be plugged in MathSAT5 by:
  - Implementing a worker interface.
  - Invoking required callback functions during search.
- The worker interface allows MathSAT5 to:
  - Specify the problem for the SAT solver to solve.
  - Communicate deduced values.
- Callbacks allow the SAT solver to:
  - Communicate found (partial) models to MathSAT5
  - Invoke $\mathcal{T}$-consistency checks.

Pluggable SAT solvers: Requirements

- Must be able to act as an enumerator.
- Should support, solving under assumptions.
- Able to create new variables, add new clauses during search.
- Support variable freezing and reintroduction of eliminated variables.
- In order to support popping, must be able to delete all clauses containing certain variables.

Worker Interface Functions

```
void solve(std::vector<int>& assump,
           std::vector<int>& c_assump);

bool add_clause(std::vector<int>& clause,
                bool permanent,
                bool during_callback);

void set_frozen(int var, bool b);

int new_var(bool polarity, bool dvar);

void enqueue_assignment(int assignment);

void remove_clauses_containing(int v);
```

Callback functions

```
TCODE  no_conflict_ater_bcp(std::vector<int>& conf);

TCODE  model_found(std::vector<int>& conflict);

void   inform_hook_of_assignment(int assignment);

void   inform_hook_of_new_level();

void   inform_hook_of_backtrack(int level);

void  ask_hook_for_t_reason(int assignment,
                             std::vector<int>& r);
```

Pluggable SAT solvers: Two Case Studies

- Extending Minisat (& Cleaneling).
- Extending Fiver.

Case Study A: Extending Minisat

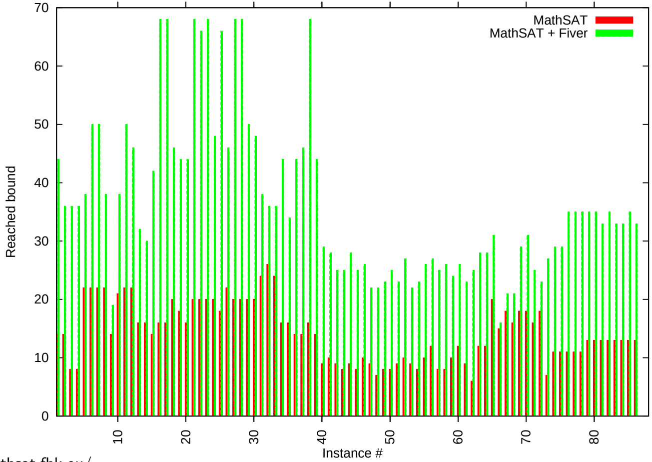
- The internal addClause method should be changed such that:
  - Clauses are added at the correct level.
  - For conflicts, jump back to the level, the conflict was introduced.
- The analyze method must take into account that assignments can be from deductions, asking the reason if necessary.
- After each round of BCP, in search which does not result in, theory propagation should be called -until fixpoint-.

- Once a complete model has been found a complete theory check should be called.
- Changing cleaneling is pretty similar!
- Example implementations for pluggable versions of Minisat & Cleaneling are available @ http://mathsat.fbk.eu.
- The changes required for each solver are around 180 lines of code.
- Does not support variable elimination.

Case Study B: Extending Fiver

- Done completely at Intel. -minus some help in debugging-
- Supports preprocessing!

Analysis of pluggable solver performance on Intel $\mathcal{BV}$ instances



Analysis of pluggable solver performance on $\mathcal{BV} \cup \mathcal{ARR}$ instances

Benchmark Family	Size	MathSAT5$_{\text{MiniSat}}$			
		#Solved	RT (sec)	#TO	#MO
brummayerbiere2	22	**15**	**1831**	**5**	**2**
brummayerbiere	293	184	17044	97	12
calc2	36	**36**	**4183**	**0**	**0**
stp	40	**29**	**1765**	**3**	**8**

Benchmark Family	Size	MathSAT5			
		#Solved	RT (sec)	#TO	#MO
brummayerbiere2	22	15	2218	5	2
brummayerbiere	293	**229**	**25698**	**64**	**0**
calc2	36	30	7855	6	0
stp	40	26	2659	6	8

DEMO

Conclusion

We have presented a framework with which SAT-Solvers can be plugged in MathSAT5 and used transparently. Next we have demonstrated the utility of such an approach on different instances.

Future Work

- Provide support for proof logging, needed for other MathSAT functionalities such as Interpolation.
- Experiment with different type enumerators such as look-ahead Solvers.
- Experiment with pluggable Theory Solvers.

Questions?

http://mathsat.fbk.eu

References I

[ABP+11] Carlos Ansótegui, Miquel Bofill, Miquel Palahí, Josep Suy, and Mateu Villaret.
Satisfiability modulo theories: An efficient approach for the resource-constrained project scheduling problem.
In *SARA*, 2011.

[CNR12] Alessandro Cimatti, Iman Narasamdya, and Marco Roveri.
Verification of Parametric System Designs.
In *Proc. FMCAD*. FMCAD, 2012.

References II

[FQ02] Cormac Flanagan and Shaz Qadeer.
Predicate abstraction for software verification.
In *Proceedings of the 29th ACM SIGPLAN-SIGACT symposium on Principles of programming languages*, POPL '02, pages 191–202, New York, NY, USA, 2002. ACM.

[GLM12] Patrice Godefroid, Michael Y. Levin, and David Molnar.
Sage: Whitebox fuzzing for security testing.
Queue, 10(1):20:20–20:27, January 2012.

[GPB01] Evgueni Goldberg, Mukul R. Prasad, and Robert K. Brayton.
Using sat for combinational equivalence checking, 2001.

References III

[SGCF11] Saurabh Srivastava, Sumit Gulwani, Swarat Chaudhuri, and Jeffrey S. Foster.
Path-based inductive synthesis for program inversion.
In *PLDI*, pages 492–503, 2011.

[ST09] Roberto Sebastiani and Armando Tacchella.
SAT Techniques for Modal and Description Logics.
In *Handbook of Satisfiability*, chapter 25, pages 781–824. IOS Press, 2009.