Combining SMT with Theorem Proving for AMS Verification

The best of both worlds

Yan Peng & Mark Greenstreet

University of British Columbia
Vancouver, BC

NASA Formal Methods Symposium, April 29, 2015
Outline

- AMS verification
  - AMS designs are ubiquitous
  - Motivation
  - Contributions

- Integrating SMT with theorem proving
- Proving global convergence for a digital PLL
- Conclusion
AMS designs are ubiquitous
Motivation

Different design models and methods, thus more complex to model.

Time-scales vary widely from sub-picosecond to milliseconds or seconds, makes it harder to simulate.

Abstractions on time-scales can hide bugs in the implementation.

Simulation?

Formal methods

Analytical approach

- Circuits are intended to be correct
- Verify the intuitive argument
Motivation - the best of both worlds

AMS design verification requires huge amounts of arithmetic reasoning and reasoning about sequences which requires induction.

SMT and theorem proving are complimentary to each other:

- SMT - Excellent performance in linear and non-linear arithmetic reasoning.
- Theorem proving - Strong support for induction and systematic model & proof management.

We are using ACL2 and Z3 as our prototyping tools.
Contributions

- We demonstrate the value of combining SMT with theorem proving for cyber-physical system verification with a focus on utilizing the non-linear arithmetic capabilities.
- The first integration of an SMT solver into the ACL2 theorem prover.
- A software architecture for integrating a SMT solver with a theorem prover that addresses many technical challenges.
- A reusable recurrence model for a state-of-the-art digital PLL.
- A proof of global convergence for the digital PLL.
Outline

- Characterize AMS verification problems
- Integrating SMT with theorem proving
  - Architecture
  - Technical issues
  - What’s trusted?
- Proving global convergence for a digital PLL
- Conclusion
Clause processors in ACL2

A clause processor takes a goal and decomposes it into a conjunction of subgoals. Each subgoal is called a clause.

ACL2 supports two kinds of clause processors: verified and trusted.

- verified - the correctness of the clause processor is proven within ACL2.
- trusted - the results of the clause processor are accepted without proof.

We integrate Z3 into ACL2 as a trusted clause processor.
Architecture of **Smtlink**

![Diagram showing the architecture of Smtlink]

- **lisp (ACL2)**
- **python (z3)**

### Translation Steps

1. **Original Clause**: ACL2 (lisp)
   - Expand & Simplify
   - Original Clause

2. **Expanded Clause**: ACL2 (lisp)
   - Expand
   - Expanded Clause
   - Original Expanded
   - (implies expanded original)
   - Expand & original
   - G
   - A₁, A₂, ..., Aₘ
   - G' \( \land \) Aₙ
   - (proven)
   - Generate Return Clause

3. **SMT Clause**: (python)
   - acl2SMT
   - \( \neg G_{Z3} \)
   - Not(clause)
   - Satisfiable?
   - Z3
   - sat, unsat, or unknown

### Equations

\[
(\land_{i=1}^{m} A_i) \quad ; \text{each } A_i \text{ verified by ACL2}
\]

\[
((\land_{i=1}^{m} A_i) \land G') \Rightarrow G \quad ; \text{verified by ACL2}
\]

\[
G_{Z3} \Rightarrow G' \quad ; \text{we trust translation step 2}
\]

\[
G_{Z3} \quad ; \text{verified by Z3}
\]

\[
G \quad ; \text{verified by ACL2}
\]
All methods of the underlying SMT solver are invoked through methods of an object called acl2SMT.

This architecture is generic enough to be combined with other SMT solvers by extending this class.
Technical issues: reals vs. rationals

\[ \forall x, y, z. \ c(x, y, z) \]

clause from ACL2

\[ \text{clause processor} \rightarrow c_1 \land c_2 \land \cdots \land c_k \]

clauses returned by clause processor:

\[ c_1 \land c_2 \land \cdots \land c_k \Rightarrow c(x, y, z) \]

■ Challenge: ACL2 has rationals and Z3 has reals.
  ▶ In ACL2, \( \neg \exists x. \ x^2 = 2 \) is a theorem.
  ▶ In Z3, \( \exists x. \ x^2 = 2 \) is a theorem.

■ Solution: only use Z3 to prove propositions where all variables are universally quantified.
  ▶ E.g. we don’t support defun-sk, exists, forall, etc.
  ▶ This is enforced syntactically in our clause processor.
Technical issues: user defined functions

- **Challenge:**
  - ACL2 supports arbitrary lisp functions.
  - Z3 functions are more like macros (no recursion).

- **Solution:**
  - Set up translation for a basic set of functions.
  - Expand non-recursive functions.
  - Expand recursive functions to bounded depth.
  - Deeper calls are declared to return an arbitrary value of the appropriate type.
  - Expansion done on ACL2’s representation: can verify correctness.
Other issues:

- Claims can contain non-polynomial terms.
  - Replace offensive subexpression with a variable.
  - User adds constraints about these variables.
  - These constraints are returned as clauses for ACL2 to prove.

- ACL2 may need hints to discharge clauses returned from the clause processor.
  - Solution: nested hints.
  - These hints tell the clause processor what hints to attach to returned clauses.

- These features provide a very flexible back-and-forth between induction proofs in ACL2 and handling the details of the algebra with Z3.
What’s trusted?

<table>
<thead>
<tr>
<th>translation</th>
<th>others</th>
<th>expansion &amp; simplification</th>
</tr>
</thead>
<tbody>
<tr>
<td>LOC(fraction)</td>
<td>656(39%)</td>
<td>453(27%)</td>
</tr>
</tbody>
</table>

- **Translation** code is straightforward and easy to check.
- **Others** are mostly boilerplate code for integrating general clause processors.
Characterize AMS verification problems

Integrating SMT with theorem proving

Proving global convergence for a digital PLL
  - The digital phase-locked loop
  - Modeling the digital PLL
  - Prove global convergence

Conclusion
A state-of-the-art Digital PLL [CNA10]

- A PLL outputs a signal with a frequency that's N times of the input signal. The output should also aligns the input in phase.

- Three state variables:
  - capacitance setting (digital)
  - supply voltage (linear),
  - phase correction (time-difference of digital transitions).
From Spectre simulation, \( f_{dco}(c, v) \approx \frac{1+\alpha v}{1+\beta c} f_0 \).

We use recurrence function to model the circuit behaviour:
\[
[c(i + 1), v(i + 1), \phi(i + 1)] = \text{next}(c(i), v(i), \phi(i)).
\]
The proof - the high level description

\[ V_{\text{max}} \leq V \leq V_{\text{min}} \]

\[ V_{\text{hi}} \leq V \leq V_{\text{lo}} \]

\[ C_{\text{min}} \leq C \leq C_{\text{max}} \]

\[ f_{\text{dco}} = f_{\text{ref}} \]

- Initial to wall, Z3
- Climb the wall, Z3
The proof - the high level description

- Initial to wall, Z3
- Climb the wall, Z3
- Leave the wall, bounded model checking, Z3
  - Unwind the recurrence for bounded depth. Z3 shows that all points have left the wall.
The proof - the high level description

- Initial to wall, Z3
- Climb the wall, Z3
- Leave the wall, unbounded model checking, Z3
- Spirals along the $f_{dco} = f_{ref}$ line to the middle yellow region, the most technical theorem, ACL2
The proof - the main theorem

These two distances are the same.

Show that $\phi < 0$ at this point and conclude $c_2$ is closer to $c_{eq}$ than $c_1$.

When we encounter heavy non-linear arithmetic reasoning, we use Smtlink.

Smtlink solves the key polynomial inequality that sets the foundation for further inequalities to hold.
Some statistics

- 13 page long hand-written proof.
- 75 lemmas, 10 of which were discharged using the SMT solver.
- Of those ten, one was the key, polynomial inequality from the manual proof.
- ACL2 completes the proof in a few minutes running on a laptop computer.
- We found one error in the process of transcribing the hand-written proof to ACL2.
Conclusion

- We built a sound and extensible integration of an SMT solver into a theorem prover.
- We demonstrated the effectiveness of the approach by proving global convergence for a state-of-the-art AMS design.
- Benefits we can get from analytical approach:
  - Ranges for initial states, parameters and etc.
  - Proofs are easy to extend. (E.g. insert uncertainty into the model, or minor revision on the model)
Future work:

- Extend our integration and contribute to the ACL2 community.
  - Integrate all code into ACL2.
  - Fully exploit Smtlink to shorten my proof.
  - Automatically generated hints.
  - Checked counterexample reports.

- Automate AMS proofs.

- Example problems from other physical domains: medical control systems, machine learning problems and etc.
References


Formally characterize AMS verification problem

∀s(0) ∈ Q₀ ∀i ≥ 0. s(i) ∈ Q

∀x(0) ∈ Q₀ ∀t ≥ 0. x(t) ∈ Q

<table>
<thead>
<tr>
<th>Digital</th>
<th>Analog</th>
<th>AMS</th>
</tr>
</thead>
<tbody>
<tr>
<td>s(i + 1) = next(s(i), in(i))</td>
<td>$\frac{dx}{dt} = f(x, in, u)$</td>
<td>$\frac{dx}{dt} = f_q(x)$</td>
</tr>
<tr>
<td>q(i + 1) = d(q(i), th(x))</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Two features in formal model of AMS designs:
- Large non-linear arithmetic formulas
- Properties for sequences of states
## SMT & Theorem proving

<table>
<thead>
<tr>
<th>What</th>
<th>SMT</th>
<th>Theorem proving</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Satisfiability Modulo Theory</strong></td>
<td>Powerful (non)linear arithmetic solver and others</td>
<td>Computer aided theorem proving</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Strength</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1. Systematic proof management</td>
<td>2. Induction proofs</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Weakness</th>
<th>Tool</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Lack of induction</td>
<td>Z3[DMB08]</td>
<td></td>
</tr>
<tr>
<td>2. Lemmas don’t connect</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Manual and tedious proofs</td>
<td>ACL2[KM97]</td>
<td></td>
</tr>
</tbody>
</table>
Theorem (Geometric Sum)

Suppose $r \in \mathbb{R}$, $n \in \mathbb{N}$, $r > 0$ and $r \neq 1$. Then,

$$\sum_{i=0}^{n} r^i = \frac{1 - r^{n+1}}{1 - r}$$

<table>
<thead>
<tr>
<th>Setup</th>
<th>LOC</th>
<th># of theorems</th>
<th>runtime(s)</th>
<th>code time</th>
</tr>
</thead>
<tbody>
<tr>
<td>raw Z3(can’t)</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>raw ACL2(proved)</td>
<td>169</td>
<td>19</td>
<td>0.14</td>
<td>2 days</td>
</tr>
<tr>
<td>arithmetic-5(proved)</td>
<td>29</td>
<td>1</td>
<td>0.15</td>
<td>10 min</td>
</tr>
<tr>
<td>ACL2 &amp; Z3(proved)</td>
<td>72</td>
<td>2</td>
<td>0.06</td>
<td>20 min</td>
</tr>
</tbody>
</table>
Polynomial inequalities

**Theorem (Polynomial inequality)**

Suppose $x \in \mathbb{R}$ and $y \in \mathbb{R}$, then

\[
1.125x^2 + y^2 \leq 1 \\
x^2 - y^2 \leq 1 \\
3(x - 2.125)^2 - 3 \leq y
\]

does not have a solution.
Theorem (Polynomial inequality)

Suppose \( x \in \mathbb{R} \) and \( y \in \mathbb{R} \), then

\[
1.125x^2 + y^2 \leq 1
\]

\[
x^2 - y^2 \leq 1
\]

\[
3(x - 2.125)^2 - 3 \leq y
\]

does not have a solution.

<table>
<thead>
<tr>
<th>Setup</th>
<th>LOC</th>
<th># of theorems</th>
<th>runtime(s)</th>
<th>code time</th>
</tr>
</thead>
<tbody>
<tr>
<td>raw Z3(proved)</td>
<td>27</td>
<td>1</td>
<td>0.0004</td>
<td>10 min</td>
</tr>
<tr>
<td>raw ACL2(failed)</td>
<td>40</td>
<td>-</td>
<td>-</td>
<td>10 min</td>
</tr>
<tr>
<td>arithmetic-5(failed)</td>
<td>41</td>
<td>-</td>
<td>-</td>
<td>10 min</td>
</tr>
<tr>
<td>ACL2 &amp; Z3(proved)</td>
<td>59</td>
<td>1</td>
<td>0.02</td>
<td>10 min</td>
</tr>
</tbody>
</table>
Technical issues: typed vs. untyped

- This is not a theorem in ACL2:

```
(defthm not-really-a-theorem
  (iff (equal x y) (zerop (- x y))) )
```

- Here is a counter-example:

<table>
<thead>
<tr>
<th>Expression</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>(- 'dog (list &quot;hello&quot;, 2, 'world))</td>
<td>0</td>
</tr>
<tr>
<td>(zerop (- 'dog (list &quot;hello&quot;, 2, 'world)))</td>
<td>t</td>
</tr>
<tr>
<td>(equal 'dog (list &quot;hello&quot;, 2, 'world))</td>
<td>nil</td>
</tr>
<tr>
<td>(iff (equal 'dog (list &quot;hello&quot;, 2, 'world))</td>
<td>nil</td>
</tr>
<tr>
<td>(zerop (- 'dog (list &quot;hello&quot;, 2, 'world)))</td>
<td></td>
</tr>
</tbody>
</table>

- But this is a theorem:

```
(defthm this-is-a-theorem
  (implies (and (rationalp x) (rationalp y))
           (iff (equal x y) (zerop (- x y))) ))
```
Technical issues: typed vs. untyped

Solution: user adds type assertions to antecedent.
- These are almost always needed anyways.
- This requirement is not a significant burden for the user.
Technical issues: principle for ensuring soundness

- \( T_i \)s are the type predicates; \( h_j \)s are the “other” hypothesis; \( C \) is the conclusion of the theorem. The ACL2 theorem is defined as:

\[
\forall x_1, x_2, \ldots, x_m \in U. \quad \left( \bigwedge_{i=1}^{m} T_i(x_i) \land \bigwedge_{j=1}^{n} h_j(x) \right) \Rightarrow C(x) \quad (1)
\]

- \( S_1, S_2, \ldots, S_m \) are the SMT sorts corresponding to the type recognizers \( T_1, T_2, \ldots, T_m \); \( \tilde{h}_j(x) \) is the translation of \( h(x) \); and \( \tilde{C}(x) \) is the translation of \( C(x) \). The corresponding SMT theorem is defined as:

\[
\forall x_1 \in S_1, x_2 \in S_2, \ldots, x_m \in S_m. \quad \left( \bigwedge_{j=1}^{n} \tilde{h}_j(x) \right) \Rightarrow \tilde{C}(x) \quad (2)
\]
Soundness is ensured if:

- $\forall x_i \in U. \ T_i(x_i) \implies x_i \in S_i$
- $\forall x_1, x_2, ..., x_m \in U. \ (\bigwedge_{i=1}^{m} T_i(x_i)) \implies (h_j(x) \implies \tilde{h}_j(x))$
- $\forall x_1, x_2, ..., x_m \in U. \ (\bigwedge_{i=1}^{m} T_i(x_i)) \implies (\tilde{C}(x) \implies C(x))$

For Smtlink construction, that means:

- Types as translated by Smtlink must be no stronger than those of the ACL2 theorem.
- Hypotheses must be no stronger than those of the ACL2 theorem.
- The conclusion must be at least as strong.
Modeling the digital PLL

\[ c(i + 1) = \text{saturate}(c(i) + g_c \text{sgn}(\phi), c_{\text{min}}, c_{\text{max}}) \]

\[ v(i + 1) = \text{saturate}(v(i) + g_v(c_{\text{center}} - c(i)), v_{\text{min}}, v_{\text{max}}) \]

\[ \phi(i + 1) = \text{wrap}(\phi(i) + (f_{\text{dco}}(c(i), v(i)) - f_{\text{ref}}) - g_{\phi} \phi(i)) \]

\[ f_{\text{dco}}(c, v) = \frac{1 + \alpha v}{1 + \beta c} f_0 \]

\[ \text{saturate}(x, lo, hi) = \min(\max(x, lo), hi) \]

\[ \text{wrap}(\phi) = \begin{cases} 
\text{wrap}(\phi + 1), & \text{if } \phi \leq -1 \\
\phi, & \text{if } -1 < \phi < 1 \\
\text{wrap}(\phi - 1), & \text{if } 1 \leq \phi 
\end{cases} \]

- By simulation we get the model for \( f_{\text{DCO}} \).
- This approach is similar to the one proposed in [ASZT07].