Automated Specification Analysis Using an Interactive Theorem Prover

Harsh Raju Chamarthi and Pete Manolios

Northeastern University

October 31, 2011
Motivation

- Teaching freshmen how to reason about programs using ACL2s
Motivation

- Teaching freshmen how to reason about programs using ACL2s
- Success of QuickCheck
Motivation

- Teaching freshmen how to reason about programs using ACL2s
- Success of QuickCheck
- Combining testing and theorem-proving (ACL2 2011 Workshop)
Motivation

- Teaching freshmen how to reason about programs using ACL2s
- Success of QuickCheck
- Combining testing and theorem-proving (ACL2 2011 Workshop)
- Can we do even better?
Motivation

- Teaching freshmen how to reason about programs using ACL2s
- Success of QuickCheck
- Combining testing and theorem-proving (ACL2 2011 Workshop)
- Can we do even better?
- Apply technology behind ACL2 to help the regular programmer

Counterexamples!!
Motivation

- Teaching freshmen how to reason about programs using ACL2s
- Success of QuickCheck
- Combining testing and theorem-proving (ACL2 2011 Workshop)
- Can we do even better?
- Apply technology behind ACL2 to help the regular programmer
Overview

Goal

Analyse specifications - Find counterexamples!
Overview

Goal
Analyse specifications - Find counterexamples!

The problem
What to do when the *Search Procedure*
doesnt return an answer?
Overview

Goal
Analyse specifications - Find counterexamples!

The problem
What to do when the *Search Procedure* doesn't return an answer?
Overview

Goal
Analyse specifications - Find counterexamples!

The problem
What to do when the *Search Procedure* doesn't return an answer?

- QuickCheck
- Decision Procedure
- Constraint Solver
Overview

Goal
Analyse specifications - Find counterexamples!

The problem
What to do when the Search Procedure doesn't return an answer?

The main idea
Reduce the search space and guide the procedure towards a counterexample
The Main Idea

Property P

Is P false?

Search Proc
The Main Idea

Property P \rightarrow \text{Is } P \text{ false?} \rightarrow \text{Yes}

Search Proc
The Main Idea

Property P \rightarrow Is P false? \rightarrow Yes
The Main Idea

Property P \rightarrow Is P false? \rightarrow \text{Dont Know}

Search Proc
The Main Idea

Property P → Search Proc → Is P false? → Don't Know
The Main Idea

Property $P$ → Is $P$ false?

Search Procedure
The Main Idea

Property P → Guide
The Main Idea

Property $P$ for $x_1, x_2, \ldots, x_n$
The Main Idea

Property $P$ → Guide

Select var

$x_1, x_2, \ldots, x_n$
The Main Idea

Property $P$ → Guide

Select var
Assign value

$x_1, 3, \ldots, x_n$
The Main Idea

Property $P$ → Guide

Select var
Assign value
Simplify using ITP
The Main Idea

- Property $P$
  - $P'$
  - $x_1, \ldots, x_n$

Guide
- Select var
- Assign value
- Simplify using ITP

Inconsistent?
The Main Idea

Property $P$  

$x_1, x_2, \ldots, x_n$

Guide

Is $P'$ false?

Search

Backtrack!!

Inconsistent?

Select var

Assign value

Simplify using ITP
The Main Idea

Property $P$  

$P'$  

$x_1, \ldots, x_n$  

Guide  

Search  

Is $P'$ false?
The Main Idea

- Is $P$ true?
- $\text{ITP}$
- $P$

- Is $P$ false?

- $p_1$
- $p_2$
- ...
- $p_n$
The Main Idea

Is P true?

ITP

Yes

Is P true?

P
The Main Idea

Is P true?

ITP

Yes

P

Is P false?

Search
The Main Idea

Is P true?

ITP

Dont Know

P

Is P false?

Dont Know
The Main Idea

Is $P$ true?

ITP

Dont Know

$p_1, p_2, ..., p_n$
The Main Idea

Is P true?

\[ p_1, p_2, \ldots, p_n \]

Is P false?
The Main Idea

Search

Is P false?

ITP

Is P true?

\[ p_1 \]

\[ p_2 \]

\[ \vdots \]

\[ p_n \]
Assumptions

- Specification language $L$
  - Multi-sorted first-order logic
  - Extensible
  - Executable
Assumptions

- Specification language $L$
  - Multi-sorted first-order logic
  - Extensible
  - Executable
Assumptions

- Specification language $L$
  - Multi-sorted first-order logic
  - Extensible -- can introduce new function and predicate symbols using well-founded recursive definitions
  - Executable
Assumptions

➤ Specification language $L$
  ➤ Multi-sorted first-order logic
  ➤ Extensible
  ➤ Executable
Assumptions

- Specification language $L$
  - Multi-sorted first-order logic
  - Extensible
  - Executable

- Properties of form $hyp_1 \land \cdots \land hyp_n \Rightarrow concl$
  - No nested quantifiers
  - Implicitly universally quantified
Assumptions

- Specification language $L$
  - Multi-sorted first-order logic
  - Extensible
  - Executable

- Properties of form $hyp_1 \land \cdots \land hyp_n \Rightarrow concl$
  - No nested quantifiers
  - Implicitly universally quantified
Assumptions

- Specification language $L$
  - Multi-sorted first-order logic
  - Extensible
  - Executable

- Properties of form $hyp_1 \land \cdots \land hyp_n \Rightarrow concl$
  - No nested quantifiers
  - Implicitly universally quantified

- An Interactive Theorem Prover (ITP) that can reason about specifications in $L$.
  - Smash

- Simplify
Assumptions

- Specification language $L$
  - Multi-sorted first-order logic
  - Extensible
  - Executable

- Properties of form $\text{hyp}_1 \land \cdots \land \text{hyp}_n \Rightarrow \text{concl}$
  - No nested quantifiers
  - Implicitly universally quantified

- An Interactive Theorem Prover (ITP) that can reason about specifications in $L$.
  - Smash takes as input a goal, a formula written in $L$, and returns a list of equi-valid subgoals.
  - Simplify
Assumptions

- Specification language $L$
  - Multi-sorted first-order logic
  - Extensible
  - Executable
- Properties of form $\text{hyp}_1 \land \cdots \land \text{hyp}_n \Rightarrow \text{concl}$
  - No nested quantifiers
  - Implicitly universally quantified
- An Interactive Theorem Prover (ITP) that can reason about specifications in $L$.
  - **Smash** takes as input a goal, a formula written in $L$, and returns a list of equi-valid subgoals.
  - **Simplify** takes as input an $L$-formula, $c$, and a list of formulas, $H$, and returns a simplified formula that is equivalent to $c$ assuming $H$ are true.
Example

Property

\[ x = \text{hash}(y) \]
\[ y = \text{hash}(z) \]
\[ x + y \neq v^2 \]
\[ z > 10 \]
\[ w < \min(x, y) \]
\[ \Rightarrow w < z \]
Example

Property

\[ x = \text{hash}(y) \]
\[ y = \text{hash}(z) \]
\[ x + y \neq v^2 \]
\[ z > 10 \]
\[ w < \min(x, y) \]
\[ \Rightarrow w < z \]
Example

Property

\[ x = \text{hash}(y) \]
\[ y = \text{hash}(z) \]
\[ x + y \neq v^2 \]
\[ z > 10 \]
\[ w < \min(x, y) \]

\[ \Rightarrow w < z \]
Example

Property

\[ x = \text{hash}(y) \]
\[ y = \text{hash}(z) \]
\[ x + y \neq v^2 \]
\[ z > 10 \]
\[ w < \min(x, y) \]
\[ \Rightarrow w < z \]

Equality dependency Graph

Rest dependency G

Select z
Example

Property

\[
\begin{align*}
  x &= \text{hash}(y) \\
  y &= \text{hash}(z) \\
  x + y &\neq v^2 \\
  z &> 10 \\
  w &< \text{min}(x, y) \\
  \Rightarrow w &< z
\end{align*}
\]

Equality dependency Graph

Rest dependency G
Example

Assign $z = 34$ (decision)

Property

\[
\begin{align*}
x & = \text{hash}(y) \\
y & = \text{hash}(34) \\
x + y & \neq v^2 \\
w & < \min(x, y) \\
\Rightarrow w & < 34
\end{align*}
\]
Example

Property

\[ x = \text{hash}(y) \]
\[ y = \text{hash}(34) \]
\[ x + y \neq v^2 \]
\[ w < \min(x, y) \]
\[ \Rightarrow w < 34 \]
Example

Propagate with Simplify

Property

\[ x = 3623878690 \]
\[ y = 268959709 \]
\[ x + 268959709 \neq v^2 \]
\[ if(x < 268959709) \]
\[ w < x \]
\[ w < 268959709 \]
\[ \Rightarrow w < 34 \]
Example

Property

\[ x = 3623878690 \]
\[ y = 268959709 \]
\[ x + 268959709 \neq \nu^2 \]
\[ \text{if}(x < 268959709) \]
\[ w < x \]
\[ w < 268959709 \]
\[ \Rightarrow w < 34 \]
Example

Property

\[ x = 3623878690 \]
\[ y = 268959709 \]
\[ x + 268959709 \neq \nu^2 \]
\[ \text{if}(x < 268959709) \]
\[ w < x \]
\[ w < 268959709 \]
\[ \Rightarrow w < 34 \]
Top-level **Analyze** algorithm

Input: Property $P$, Summary $summary$

Output: Status, Summary of the analysis of $P$

if $P$ is closed then return $\text{AnalyzeConst}(P)$

initialize

Search till ...

if $\text{StopCond}(summary)$ then return (done, $summary$)

Decompose $P$ into subgoals using ITP

If progress then Recurse on each subgoal

return (not-done, $summary$)
Top-level **Analyze** algorithm

Input: Property $P$, Summary $summary$
Output: Status, Summary of the analysis of $P$

if $P$ is closed then return $\text{AnalyzeConst}(P)$

**initialize**
Search till ...
if $\text{StopCond}(summary)$ then return (done, $summary$)

Decompose $P$ into subgoals using ITP
If progress then Recurse on each subgoal
return (not-done, $summary$)
Top-level **Analyze** algorithm

Input: Property $P$, Summary $summary$
Output: Status, Summary of the analysis of $P$

if $P$ is closed then return $\text{AnalyzeConst}(P)$

$n, status := 0, \text{not-done}$

Search till ...

if $\text{StopCond}(summary)$ then return (done, $summary$)

Decompose $P$ into subgoals using ITP
If progress then Recurse on each subgoal
return (not-done, $summary$)
Top-level **Analyze** algorithm

**Input:** Property $P$, Summary $summary$

**Output:** Status, Summary of the analysis of $P$

- if $P$ is closed then return $AnalyzeConst(P)$
- initialize
- **Search till** ...
- if $StopCond(summary)$ then return (done, $summary$)

Decompose $P$ into subgoals using ITP

If progress then Recurse on each subgoal

return (not-done, $summary$)
Top-level **Analyze** algorithm

**Input:** Property $P$, Summary $summary$

**Output:** Status, Summary of the analysis of $P$

if $P$ is closed then return $AnalyzeConst(P)$

initialize

while $\neg StopCond(summary) \land n \leq slimit$ do

   $A, n := \text{Search}(P), n + 1$

   $summary := \text{updateA}(summary, P, A)$

if $StopCond(summary)$ then return $(\text{done}, summary)$

Decompose $P$ into subgoals using ITP

If progress then Recurse on each subgoal

return $(\text{not-done}, summary)$
Top-level Analyze algorithm

Input: Property $P$, Summary $summary$
Output: Status, Summary of the analysis of $P$
  if $P$ is closed then return $\text{AnalyzeConst}(P)$
  initialize
  Search till ...
  if $\text{StopCond}(summary)$ then return (done, $summary$)

Decompose $P$ into subgoals using ITP
If progress then Recurse on each subgoal
return (not-done, $summary$)
Top-level **Analyze** algorithm

**Input:** Property $P$, Summary $summary$

**Output:** Status, Summary of the analysis of $P$

if $P$ is closed then return $\text{AnalyzeConst}(P)$

initialize

Search till ...

if $\text{StopCond}(summary)$ then return (done, $summary$)

**Decompose $P$ into subgoals using ITP**

If progress then Recurse on each subgoal

return (not-done, $summary$)
**Top-level Analyze algorithm**

Input: Property $P$, Summary $summary$

Output: Status, Summary of the analysis of $P$

if $P$ is closed then return $\text{AnalyzeConst}(P)$

initialize

Search till ...

if $\text{StopCond}(summary)$ then return (done, $summary$)

$S := \text{Smash}(P)$

$summary := \text{updateS}(summary, P, S)$

If progress then Recurse on each subgoal

return (not-done, $summary$)
Top-level **Analyze** algorithm

Input: Property $P$, Summary $summary$
Output: Status, Summary of the analysis of $P$

if $P$ is closed then return $AnalyzeConst(P)$

initialize
Search till ...
if $StopCond(summary)$ then return (done, $summary$)

Decompose $P$ into subgoals ...
If progress then Recurse on each subgoal
return (not-done, $summary$)
Top-level **Analyze** algorithm

Input: Property $P$, Summary $summary$

Output: Status, Summary of the analysis of $P$

if $P$ is closed then return $\text{AnalyzeConst}(P)$

initialize

Search till ...

if $\text{StopCond}(summary)$ then return (done, $summary$)

Decompose $P$ into subgoals ...

if $S \neq \{P\}$ then

    for all $p \in S$ do

        $status, summary := \text{Analyze}(p, summary)$

        if $status = \text{done}$ then return (done, $summary$)

    return (not-done, $summary$)
Top-level **Analyze** algorithm

Input: Property $P$, Summary $summary$
Output: Status, Summary of the analysis of $P$

if $P$ is closed then return $\text{AnalyzeConst}(P)$

initialize
Search till ...
if $\text{StopCond}(summary)$ then return (done, $summary$)

Decompose $P$ into subgoals ...
If progress then Recurse on each subgoal ...
return (not-done, $summary$)
Search Algorithm

Input: Property $P$ with at least one free variable
Output: A counterexample (assignment) or fail

Local Stack $A$ of (var, val, # assigns, type, property)
Initialize and Select first variable
Iteratively construct counterexample or fail
**Search Algorithm**

Input: Property $P$ with at least one free variable
Output: A counterexample (assignment) or $fail$

local Stack $A$ of (var, val, # assigns, type, property)

Initialize and Select first variable
Iteratively construct counterexample or $fail$
Search Algorithm

Input: Property $P$ with at least one free variable
Output: A counterexample (assignment) or fail

local Stack $A$ of (var, val, # assigns, type, property)
$A, i, x := [], 0, Select(P)$
Iteratively construct counterexample or fail
Search Algorithm

Input: Property $P$ with at least one free variable
Output: A counterexample (assignment) or fail

local Stack $A$ of (var, val, # assigns, type, property)

$A, i, x := [], 0, \textbf{Select}(P)$

Iteratively construct counterexample or fail
Search Algorithm

Input: Property $P$ with at least one free variable
Output: A counterexample (assignment) or $fail$
local Stack $A$ of (var, val, # assigns, type, property)
$A, i, x := [], 0, Select(P)$
while true do
  $v, t := Assign(x, P)$
  $P' := Propagate(x, v, P)$
  if $\neg$inconsistent($P'$) then
    Extend $A$, continue search if not done
  else if $A \neq []$ then
    $backtrack$
    $fail$ if ...
Search Algorithm

Input: Property $P$ with at least one free variable
Output: A counterexample (assignment) or \textit{fail}

local Stack $A$ of (var, val, # assigns, type, property)

$A, i, x := [], 0, \text{Select}(P)$

while true do

$v, t := \text{Assign}(x, P)$

$P' := \text{Propagate}(x, v, P)$

if $\neg \text{inconsistent}(P')$ then

Extend $A$, continue search if not done

else if $A \neq []$ then

\textit{backtrack}

\textit{fail} if ...
Search Algorithm

Input: Property $P$ with at least one free variable
Output: A counterexample (assignment) or fail

local Stack $A$ of (var, val, # assigns, type, property)

$A, i, x := [], 0, \text{Select}(P)$

while true do

$v, t := \text{Assign}(x, P)$

$P' := \text{Propagate}(x, v, P)$

if $\neg \text{inconsistent}(P')$ then

Extend $A$, continue search if not done

else if $A \neq []$ then

\text{backtrack}

\text{fail} if ...

Search Algorithm

Input: Property $P$ with at least one free variable
Output: A counterexample (assignment) or \textit{fail}

local Stack $A$ of (var, val, \# assigns, type, property)

$A, i, x := [], 0, \text{Select}(P)$

while true do
    $v, t := \text{Assign}(x, P)$
    $P' := \text{Propagate}(x, v, P)$
    if $\neg\text{inconsistent}(P')$ then
        Extend $A$, continue search if not done
    else if $A \neq []$ then
        \textit{backtrack}
    fail if ...


Search Algorithm

Input: Property $P$ with at least one free variable
Output: A counterexample (assignment) or fail

local Stack $A$ of (var, val, # assigns, type, property)

$A, i, x := [], 0, \text{Select}(P)$

while true do

$v, t := \text{Assign}(x, P)$

$P' := \text{Propagate}(x, v, P)$

if $\neg \text{inconsistent}(P')$ then

if $t = \text{``decision''}$ then $i := i + 1$

$A := \text{push}((x, v, i, t, P), A)$

if $A$ is complete then return $A$

else

$i, P, x := 0, P', \text{Select}(P')$

else if $A \neq []$ then

backtrack

fail if ...
Search Algorithm

Input: Property $P$ with at least one free variable
Output: A counterexample (assignment) or $fail$

local Stack $A$ of (var, val, # assigns, type, property)

$A, i, x := [], 0, Select(P)$

while true do

$v, t := Assign(x, P)$

$P' := Propagate(x, v, P)$

if $\neg$inconsistent($P'$) then

 Extend $A$, continue search if not done

else if $A \neq []$ then

   $backtrack$

fail if ...
Search Algorithm

Input: Property \( P \) with at least one free variable
Output: A counterexample (assignment) or fail

local Stack \( A \) of (var, val, # assigns, type, property)

\[ A, i, x := [], 0, \text{Select}(P) \]

while true do

\[ v, t := \text{Assign}(x, P) \]
\[ P' := \text{Propagate}(x, v, P) \]

if \( \neg \text{inconsistent}(P') \) then

Extend \( A \), continue search if not done

else if \( A \neq [] \) then

repeat

\[ (x, \_ , i, t, P) := \text{head}(A) \]
\[ A := \text{pop}(A) \]

until \( t = \text{"decision"} \land i \leq \text{blimit} \lor A = [] \)

fail if ...
**Search Algorithm**

Input: Property $P$ with at least one free variable  
Output: A counterexample (assignment) or *fail*  
local Stack $A$ of (var, val, # assigns, type, property)  
$A, i, x := [], 0, \text{Select}(P)$  
while true do  
    $v, t := \text{Assign}(x, P)$  
    $P' := \text{Propagate}(x, v, P)$  
    if $\neg\text{inconsistent}(P')$ then  
        Extend $A$, continue search if not done  
    else if $A \neq []$ then  
        backtrack  
    *fail* if ...
Search Algorithm

Input: Property $P$ with at least one free variable
Output: A counterexample (assignment) or fail
local Stack $A$ of (var, val, # assigns, type, property)
$A, i, x := [], 0, \text{Select}(P)$
while true do
    $v, t := \text{Assign}(x, P)$
    $P' := \text{Propagate}(x, v, P)$
    if $\neg \text{inconsistent}(P')$ then
        Extend $A$, continue search if not done
    else if $A \neq []$ then
        \textit{backtrack}
        if $A = [] \land (t = "\text{implied}" \lor i > \text{blimit})$ then
            return \textit{fail}
Select Algorithm

Input: Property $P$ with at least one free variable
Output: A free variable in $P$

if $\exists h \in hyps(P)$ of form $x = c$ then return $x$
**Select Algorithm**

Input: Property $P$ with at least one free variable  
Output: A free variable in $P$  

if $\exists h \in \text{hyps}(P)$ of form $x = c$ then return $x$
Select Algorithm

Input: Property $P$ with at least one free variable
Output: A free variable in $P$

1. if $\exists h \in hyps(P)$ of form $x = c$ then return $x$

2. $G_\equiv := \text{EqualityDependencyGraph}(P, \text{vars}(P))$

\begin{itemize}
  \item 1. Case: $x = y$. Add $x \leftrightarrow y$.
  \item 2. Case: $x = \text{fterm}$
    \begin{itemize}
      \item $y \in \text{freeVars(fterm)}$ and
      \item $x \notin \text{freeVars(fterm)}$
    \end{itemize}
    add $x \rightarrow y$
\end{itemize}
Select Algorithm

Input: Property $P$ with at least one free variable
Output: A free variable in $P$

if $\exists h \in \text{hyps}(P)$ of form $x = c$ then return $x$

$G_{=} := \text{EqualityDependencyGraph}(P, \text{vars}(P))$

Do SCC on $G_{=}$, collect the leaf components in $L$

$\text{leaves}_{=} :=$ pick $x$ from each $l \in L$
Select Algorithm

Input: Property $P$ with at least one free variable
Output: A free variable in $P$

1. if $\exists h \in \text{hyps}(P)$ of form $x = c$ then return $x$
2. $G \equiv := \text{EqualityDependencyGraph}(P, \text{vars}(P))$
3. $G_{\cong} := \text{RestDependencyGraph}(P, \text{leaves}_{\equiv})$

1. $x \ni y$ where $\ni \in \{<, \leq, >, \geq\}$: No edge
2. $x \ni \text{fterm}$ such that $\ni$ is a binary relation,
   $y \in \text{freeVars(fterm)}$ and $x \notin \text{freeVars(fterm)}$:
   Add $x \to y$
3. $R(\text{term}_1, \text{term}_2, \ldots, \text{term}_n)$, such that
   $x \in \text{freeVars(}_i, y \in \text{freeVars(}_j$, $i \neq j$,
   $n \geq 2$ and $R$ is an arbitrary $n$-ary relation:
   Add $x \leftrightarrow y$. 
Select Algorithm

Input: Property $P$ with at least one free variable
Output: A free variable in $P$

if $\exists h \in hyps(P)$ of form $x = c$ then return $x$

$G_\subseteq := \text{EqualityDependencyGraph}(P, vars(P))$

SCC on $G_\subseteq$

$G_\bowtie := \text{RestDependencyGraph}(P, \text{leaves}_\subseteq)$

Do SCC on $G_\bowtie$ to get dag $D_\bowtie$
Select Algorithm

Input: Property $P$ with at least one free variable
Output: A free variable in $P$

if $\exists h \in \text{hyps}(P)$ of form $x = c$ then return $x$

$G_{\equiv} := \text{EqualityDependencyGraph}(P, \text{vars}(P))$

SCC on $G_{\equiv}$

$G_{\bowtie} := \text{RestDependencyGraph}(P, \text{leaves}_{\equiv})$

Do SCC on $G_{\bowtie}$ to get dag $D_{\bowtie}$

$X := \text{the leaf in } D_{\bowtie} \text{ with maximum } i_{\equiv} \text{ value}$

return $X$

\[ i_{\equiv}(x) \text{ denotes number of nodes that can reach it in } G_{\equiv} \]
\[ i_{\equiv}(X) \text{ denotes max value of } i_{\equiv} \text{ among nodes } \in X \]
Hardware: Finding hazards in a pipeline

Analysing a 3-stage Pipeline

1. Fetch
2. Read
3. Execute/Write-back
Hardware: Finding hazards in a pipeline

Analysing a 3-stage Pipeline

1. Fetch
2. Read
3. Execute/Write-back

Primary Concern
Avoid resource conflicts (Data/Control hazards)
Hardware: Finding hazards in a pipeline

Analysing a 3-stage Pipeline

1. Fetch
2. Read
3. Execute/Write-back

Primary Concern
Avoid resource conflicts (Data/Control hazards)

Correctness
Show all behaviors of MA are observationally equivalent to behaviors of ISA
Hardware: Finding hazards in a pipeline

Primary Concern
Avoid resource conflicts (Data/Control hazards)

Correctness
Show all behaviors of MA are observationally equivalent to behaviors of ISA

Can we find design errors that lead to hazards?

1. Assuming designer has modelled both ISA and MA
2. Formalize above correctness condition
3. Analyze it using our method (demo)
Hardware: Finding hazards in a pipeline

**Primary Concern**
Avoid resource conflicts (Data/Control hazards)

**Correctness**
Show all behaviors of MA are observationally equivalent to behaviors of ISA

**Can we find design errors that lead to hazards?**

**Observations**

1. No assertions were written
2. No lemmas were specified
3. No manual tests or test driver given.
Software: Comparison with Alloy

Alloy

- Alloy is a declarative modeling language based on sets and relations (relational logic with transitive closure).
- Used for describing and analyzing high-level specifications and designs.

Automatic Analysis

Given a bound on # model elements, called scope, Alloy models (and its specifications) translated into Boolean formulas and shipped to off-the-shelf SAT solvers.
## Software: Comparison with Alloy

<table>
<thead>
<tr>
<th>Property</th>
<th>Alloy Analyzer</th>
<th>Our method</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Scope</td>
<td>Time</td>
</tr>
<tr>
<td>delUndoesAdd</td>
<td>25</td>
<td>26.41</td>
</tr>
<tr>
<td>addIdempotent</td>
<td>25</td>
<td>37.76</td>
</tr>
<tr>
<td>addLocal</td>
<td>3</td>
<td>0.08</td>
</tr>
<tr>
<td>lookupYields</td>
<td>3</td>
<td>0.05</td>
</tr>
<tr>
<td>writeRead</td>
<td>34</td>
<td>99.69</td>
</tr>
<tr>
<td>writeIdempotent</td>
<td>33</td>
<td>44.13</td>
</tr>
<tr>
<td>hidePreservesInv</td>
<td>61</td>
<td>24.91</td>
</tr>
<tr>
<td>cutPaste</td>
<td>3</td>
<td>0.20</td>
</tr>
<tr>
<td>pasteCut</td>
<td>3</td>
<td>0.20</td>
</tr>
<tr>
<td>pasteAffectsHidden</td>
<td>27</td>
<td>117.63</td>
</tr>
<tr>
<td>markSweepSound</td>
<td>8</td>
<td>47.34</td>
</tr>
<tr>
<td>markSweepComplete</td>
<td>7</td>
<td>58.12</td>
</tr>
</tbody>
</table>

**Table:** Comparison with Alloy Analyzer (AA)
## Software: Comparison with Alloy

<table>
<thead>
<tr>
<th>Property</th>
<th>Alloy Analyzer</th>
<th>Our method</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Scope</td>
<td>Time</td>
</tr>
<tr>
<td>delUndoesAdd</td>
<td>25</td>
<td>26.41</td>
</tr>
<tr>
<td>addIdempotent</td>
<td>25</td>
<td>37.76</td>
</tr>
<tr>
<td>addLocal</td>
<td>3</td>
<td>0.08</td>
</tr>
<tr>
<td>lookupYields</td>
<td>3</td>
<td>0.05</td>
</tr>
<tr>
<td>writeRead</td>
<td>34</td>
<td>99.69</td>
</tr>
<tr>
<td>writeIdempotent</td>
<td>33</td>
<td>44.13</td>
</tr>
<tr>
<td>hidePreservesInv</td>
<td>61</td>
<td>24.91</td>
</tr>
<tr>
<td>cutPaste</td>
<td>3</td>
<td>0.20</td>
</tr>
<tr>
<td>pasteCut</td>
<td>3</td>
<td>0.20</td>
</tr>
<tr>
<td>pasteAffectsHidden</td>
<td>27</td>
<td>117.63</td>
</tr>
<tr>
<td>markSweepSound</td>
<td>8</td>
<td>47.34</td>
</tr>
<tr>
<td>markSweepComplete</td>
<td>7</td>
<td>58.12</td>
</tr>
</tbody>
</table>

**Table:** Comparison with Alloy Analyzer (AA)
Software: Comparison with Alloy

Methodology

Modeled above examples in ACL2, mimicking original formulation in Alloy.

Used set types and map types i.e., binary relations, provided by our data definition framework.
Software: Comparison with Alloy

Observations

1. The ordered sets and records library in ACL2 distribution, powerful enough to prove all the properties that Alloy posits are true

2. No intermediate lemmas provided, no hint or guidance offered to the theorem prover

3. Highlights effectiveness of powerful libraries by the tool-writer put to use by the choice of right abstractions by the programmer
Discussion on the advantages of using ITP

- Prune away huge subspaces
- Extensible
- Domain-specific lemma libraries → powerful domain-specific reasoning
- User can also help formalize facts/insight
Discussion on the advantages of using ITP

- Prune away huge subspaces
- Extensible
- Domain-specific lemma libraries → powerful domain-specific reasoning
- User can also help formalize facts/insight
Discussion on the advantages of using ITP

- Prune away huge subspaces
- Extensible
- Domain-specific lemma libraries → powerful domain-specific reasoning
- User can also help formalize facts/insight
Discussion on the advantages of using ITP

- Prune away huge subspaces
- Extensible
- Domain-specific lemma libraries → powerful domain-specific reasoning
- User can also help formalize facts/insight
Discussion on the advantages of using ITP

- Prune away huge subspaces
- Extensible
- Domain-specific lemma libraries → powerful domain-specific reasoning
- User can also help formalize facts/insight
Conclusions

- Automatically analyze properties, interleaving ITP and testing in a fine-grained fashion
- **Search** algorithm guides testing when it is stuck (Decision Procedures can also benefit)
- **Select** algorithm can be used as a starting point by concolic testing
- Combining automated methods with ITP technology results in a more powerful, yet automated method.
- Better interactive theorem proving experience
The End

Thank you