BRICS Qualifying Exam

Automation for Formal Verification: Exploiting Structure

M. Oliver Möller

Supervision: Michael I. Schwartzbach / Kim Guldstrand Larsen
Roadmap

Part I  Automation in theorem proving: congruence closure

Part II  Automation in model checking: hierarchical decomposition

Part III  Automation in concurrency: identifying togetherness
Roadmap

Part I  Automation in theorem proving: congruence closure
  ★ congruence closure framework
  ★ bit-vector theories
  ★ computational complexity of solving

Part II  Automation in model checking: hierarchical decomposition

Part III  Automation in concurrency: identifying togetherness
Automated Theorem Proving

\[ \Gamma \vdash \varphi \]

\[ \Gamma : \text{assumptions} \]

\[ \varphi : \text{conclusion} \]

Inference system \( \Rightarrow \) Theorem prover

We want *reasonable* tool support:

- high-level strategies (like *induct* & *simplify*)
- heuristics to search for proofs
- full automation in simple cases
Automated Theorem Proving

\( \Gamma \vdash \varphi \)

\( \Gamma \) : assumptions
\( \varphi \) : conclusion

Inference system \( \rightsquigarrow \) Theorem prover
sequent calculus \( \rightsquigarrow \) PVS

We want reasonable tool support:

- high-level strategies (like induct \& simplify) ✓
- heuristics to search for proofs ✓
- full automation in simple cases congruence closure
Encoding Equations in a DAG

Given: \( fff x = f x, \quad x = y \)

Conclude: \( fffff y = f x \)

as a graph
Encoding Equations in a DAG

Given: \( f f f x = f x, \quad x = y \)

Conclude: \( f f f f f y = f x \)

as a graph

\[
\begin{align*}
ffffy &= fffx \\
ffffx &= ffx \\
fffx &= fx
\end{align*}
\]
Shostak Style Framework

Theorem

\[ T \vdash a = b \iff \text{node}(a) \hat{R}_T \text{node}(b) \]

Shostak ’84: It is possible to combine several theories in this framework, if they are algebraically solvable.

For every theory we have

**canonizer:** term \( t \) \( \longrightarrow \) \( \sigma(t) \)

unique representation: \( \models t = u \iff \sigma(t) \equiv \sigma(u) \)

**solver:** equation \( t = u \) \( \longrightarrow \) \( \bigwedge_i x_i = s_i \)

explicit description of all solutions
Bit-Vector Theories

Core

- $x[n] : \text{bvec}_n$
  number of bits $n$ a constant
- Constants $c[n], c \in \mathbb{N}$
- Concatenation:
- Extraction:
  $$x[3][0 : 1] \rightarrow x[3][0 : 1]$$
Bit-Vector Theories

Core

- $x[n] : bvec_n$
  number of bits $n$ a constant
- Constants $c[n]$, $c \in \mathbb{N}$
- Concatenation:
- Extraction:
  $x[3][0 : 1] \rightarrow x[3][0 : 1]$

Extensions

- Boolean operations
  $x[3] \land y[3]$
- Arithmetic (modulo $2^n$)
- Variable length
  $x[n] : bvec_n$
  $n : \mathbb{N}$
- Variable Extraction
  $x[3][i:j] : bvec_{j-i+1}$
  $i, j : \mathbb{N}$
Canonizing Bit-Vector Terms

**Phase α:** flatten extractions

\[
\begin{align*}
(x_3 \otimes y_3)[0:3] & \rightsquigarrow x_3 \otimes y_3[0:0]
\end{align*}
\]

**Phase β:** glue together

\[
\begin{align*}
x_3[0:1] \otimes x_3[2:2] & \rightsquigarrow x_3
\end{align*}
\]
Bit-Vector Solver: Chop

(1) \( p[n] \otimes t = q[n] \otimes u \)

\[
\begin{cases}
  p[n] = q[n] \\
  t = u 
\end{cases}
\]

(1') \( p[n] \otimes t = q[m] \otimes u, \ n < m \)

\[
\begin{cases}
  p[n] = \sigma(q[m][0 : n - 1]) \\
  \sigma(q[m][n : m - 1]) \otimes u = t \\
  q[m] = \sigma(q[m][0 : n - 1]) \otimes \sigma(q[m][n : m - 1])
\end{cases}
\]

(1'') \( p[n] \otimes t = q[m] \)

\[
\begin{cases}
  p[n] = \sigma(q[m][0 : n - 1]) \\
  \sigma(q[m][n : m - 1]) = t \\
  q[m] = \sigma(q[m][0 : n - 1]) \otimes \sigma(q[m][n : m - 1])
\end{cases}
\]
Bit-Vector Solver: Propagate

(2) \[ c = d, c \neq d \] \hspace{1cm} \text{FAIL}

(3) \[ t = t \]

(4) \[ \begin{aligned} p &= t \\ q &= u \end{aligned} \hspace{1cm} q \preceq t \hspace{1cm} \begin{aligned} p &= t[q/u] \\ q &= u \end{aligned} \]

(5) \[ \begin{aligned} p &= q \\ q &= r \end{aligned} \hspace{1cm} \begin{aligned} p &= r \\ q &= r \end{aligned} \]

(6) \[ \begin{aligned} p &= q \\ q &= p \end{aligned} \hspace{1cm} \{ p = q \} \]

(7) \[ \begin{aligned} p &= t \\ p &= u \end{aligned} \hspace{1cm} \begin{aligned} p &= t \\ u &= t \end{aligned} \]

(8) \[ c = t, t \notin C \hspace{1cm} \{ t = c \} \]

(9) \[ p = q \otimes t, p \neq \sigma(q \otimes t) \hspace{1cm} \{ q \otimes t = p \} \]
Complexity of the Satisfiability Problem

Core Theory [in P]
Complexity of the Satisfiability Problem

[Diagram showing the complexity of the satisfiability problem with various combinations of operations and their complexity status (in P or NP-complete)].
Complexity of the Satisfiability Problem

```
Complexity
Core Theory
+ variable Extraction
+ Arithmetic
+ Boolean Operations
+ variable Width

[?]  

[NP-complete]
```

```
Core Theory
+ variable Extraction
+ Arithmetic
+ Boolean Operations

[NP-complete]
```

```
Core Theory
+ variable Extraction
+ Arithmetic
+ Boolean Operations

[NP-complete]
```

```
Core Theory
+ variable Extraction
+ Arithmetic

[NP-complete]
```

```
Core Theory
+ variable Extraction
+ Boolean Operations

[NP-complete]
```

```
Core Theory
+ Arithmetic
+ Boolean Operations

[NP-complete]
```

```
Core Theory
+ Boolean Operations

[NP-complete]
```

```
Core Theory

[in P]
```

Complexity of the Satisfiability Problem

Extended Counting Theory

only composition variable Width

[undecidable]

[in PSPACE]

Core Theory

+ variable Extraction

+ Arithmetic

+ Boolean Operations

+ variable Width

[?]
Expressiveness of Solving

∀x.∃y.∃z.(x ∨ y) ↔ ¬z

\[ \text{solve}((x ∨ y) ↔ ¬z) = \begin{cases} 
  x = a \\
  y = b \\
  z = \neg a \land \neg b 
\end{cases} \]

∀x.∃y.∃z.(x ∨ y) ↔ ¬z : true
Expressiveness of Solving

\forall x. \exists y. \exists z. (x \lor y) \leftrightarrow \neg z

\text{solve}( (x \lor y) \leftrightarrow \neg z ) = \begin{cases} 
  x & = a \\
  y & = b \\
  z & = \neg a \land \neg b 
\end{cases}

\forall x. \forall y. \forall z. (x \lor y) \leftrightarrow \neg z : \text{false} \quad \exists x. \forall y. \forall z. (x \lor y) \leftrightarrow \neg z : \text{false}

\forall x. \forall y. \exists z. (x \lor y) \leftrightarrow \neg z : \text{true} \quad \exists x. \forall y. \exists z. (x \lor y) \leftrightarrow \neg z : \text{true}

\forall x. \exists y. \forall z. (x \lor y) \leftrightarrow \neg z : \text{false} \quad \exists x. \exists y. \forall z. (x \lor y) \leftrightarrow \neg z : \text{false}

\forall x. \exists y. \exists z. (x \lor y) \leftrightarrow \neg z : \text{true} \quad \exists x. \exists y. \exists z. (x \lor y) \leftrightarrow \neg z : \text{true}
Quantification Lemma

General Method

• start with a quantified equation
  \[ Q_1 x_1 \ldots Q_n x_n \cdot t = u \]

• let \( S = \text{solve}(t = u) \);
  derive \( \bigwedge_i x_i = s_i \) where
  - all \( x_i \) appear in the right order
  - terms \( s_i \) do not refer to \( x_j \) with \( j > i \)

• check for \( \forall \)-quantified \( x_i \), whether \( s_i \) is \textit{unrestricted} \n
\[ \Rightarrow \quad \text{Solving essentially decides all quantified versions} \]
Complexity Revisited

Extended Counting Theory

[undecidable]

only composition
variable Width

Core Theory
+ variable Extraction
+ Arithmetic
+ Boolean Operations
variable Width

[no complete solver exits]

Core Theory
+ variable Extraction
+ Arithmetic
+ Boolean Operations

PSPACE-hard

Core Theory
+ variable Extraction
+ Arithmetic
+ Boolean Operations

Core Theory
+ variable Extraction
+ Arithmetic

Core Theory
+ variable Extraction

Core Theory
+ Arithmetic
+ Boolean Operations

Core Theory

Core Theory

only composition

only composition
variable Width
but with upper bound

only composition

[in P]

[in P]

+ Arithmetic
Summary on Part I

We established

- canonizer and solver for the core theory
- two extensions:
  - boolean operations (canonizer + solver)
  - variable width (solver)
- quantification lemma

Further development?

- high computational complexity
- interesting fragments?
  \[ \Rightarrow \text{algebraically solvability a serious restriction} \]
- extension of the framework?
  \[ \Rightarrow \text{seems to boil down to a case-split} \]
Roadmap

Part I Automation in theorem proving: congruence closure
   ★ congruence closure framework
   ★ bit-vector theories
   ★ computational complexity of solving

Part II Automation in model checking: hierarchical decomposition
   ★ model checking and temporal logic
   ★ a technique for temporal scaling
   ★ hierarchical structuring

Part III Automation in concurrency: identifying sequential parts
Model Checking

\[ M \models \varphi \]

- \( M \): description of the system
- \( \varphi \): desired property

- easier than proving a general theorem
- completely automatic (’yes’ or counterexample)
- efficient algorithms tailored for classes of problems
Temporal Logics

- TCTL
  - $\mu$ - calculus
  - bisimulation
- ATL interaction with environment
- $L_\nu$
- CTL lifeness properties
- HM safety properties
Temporal Logics

- TCTL
- μ-calculus
- bisimulation

EXPTIME

- ATL
  - interaction with environment

L_\nu

- CTL
  - lifeness properties

PSPACE

- HM
  - safety properties

computational complexity
Problems with Model Checking

- exponential growth of the state space  
  ⇒ state explosion problem

- restricted expressiveness of the model  
  ⇒ experience required to make apt abstractions

- restricted expressiveness of the logics  
  ⇒ properties often have to be encoded

- excessive time/space consumption  
  ⇒ heuristics to make state space exploration more efficient
Temporal Scaling

\[
P1 \parallel P2 \parallel P3 \parallel P4 \parallel P5
\]
Temporal Scaling

Instead of:

\[ P1 \parallel P2 \parallel P3 \parallel P4 \parallel P5 \]

Use:

hide busy in

\[ P1 \parallel P2 \parallel P3 \parallel P4 \parallel P5 \]
Model Checking with MOCHA

- parallel execution of components
- communication via shared variables (1 write/multi read)
- ATL model checker
- invariant check:
  allows temporal scaling via “next” $\Theta$ for $P$
  *(Rajeev Alur and Bow-Yaw Wang, CONCUR’99)*
- heuristic to preprocess a system for “next”
Incremental Clustering

**input:** hypergraph $\mathcal{H} = (\mathcal{C}, \mathcal{E})$

**output:** forest with leaves $\mathcal{C}$

*PriorityQueue* $Q$

$forest := \{ \}$

For all considered candidates $\mathcal{A} \subseteq \mathcal{C}$

$\text{insert}(\mathcal{A}, Q)$

While notempty($Q$)

$\mathcal{A} := \text{top}(Q)$

fresh node $\mathcal{A}$

$forest := forest + (\mathcal{A} \mapsto \mathcal{A})$

redirect hyperedges: $A_i \in \mathcal{A}$ becomes $\mathcal{A}$

For all $\mathcal{B} \in Q$ with $\mathcal{B} \cap \mathcal{A} \neq \emptyset$

$\text{remove}(\mathcal{B}, Q)$

For all new candidates $\mathcal{D}$ containing $\mathcal{A}$

$\text{insert}(\mathcal{D}, Q)$
Asynchronous Parity Computer

Clients : issue value true or false

Joins : compute xor

Root : acknowledges
Good Heuristic Function

\[ \text{Cover-Number} \quad \text{Size} \quad \text{Edges} \quad \text{Depth} \]
Good Heuristic Function

\[ r_{pref}^+(\mathcal{A}) := \frac{|\{e \in \mathcal{E} \mid e \subseteq \mathcal{A}\}|}{|\mathcal{A}|^2} + \frac{\varepsilon_1}{|\{e \in \mathcal{E} \mid e \cap \mathcal{A} \neq \emptyset\}|} + \frac{\varepsilon_2}{\text{depth}(\mathcal{A})} \]

Cover-Number  Size  Egdes  Depth
Good Heuristic Function

\[ r_{pref}^+(\mathcal{A}) := \frac{|\{e \in \mathcal{E} \mid e \subseteq \mathcal{A}\}|}{|\mathcal{A}|^2} + \frac{\varepsilon_1}{|\{e \in \mathcal{E} \mid e \cap \mathcal{A} \neq \emptyset\}|} + \frac{\varepsilon_2}{\text{depth}(\mathcal{A})} \]

Cover-Number  Size  Egdes  Depth
Summary on Part II

We established

• incremental method for hierarchical decomposition
  (has direct application with temporal scaling)

• heuristic function based on 4 criteria:
  Cover-Number, Size, Edges, Depth

• sample problem, where this heuristic is well-behaved

Left to do

• validate method/function with other sample topologies

• investigate applicability beyond invariant properties

• (possibly) integration in the next MOCHA release
Guiding Idea

System:
parallel components

Connections:
influence behavior

Structure:
often has *asymmetries*

\{A,B,C\} belong *strongly* together
\{D, E\} are not as “together” as \{E, F\}

General Observation:
If we can adapt our analysis to asymmetries, it becomes either

- more expressive
- or more efficient

We aim to describe, detect and exploit “Togetherness”
Describe Togetherness

Desired Characterization:

\{A, B\} are together [to extend \(\alpha\)] \iff \ ?

Likely: the links influence this

Unlikely: link information suffices

Methodologies:

- Identification by categorical models
- \textit{CCS}- or \textit{CSP}-like frameworks
- Properties derived from automata interaction

\[\Rightarrow\text{ If goal too ambitious, restrict to special cases}\]
Detect Togetherness

**To be expected:** Good characterizations are computationally expensive

⇒ identify typical patterns

If infeasible, investigate *why*

⇒ lower bounds / hardness results

Reasonable hope for *partial* solutions

- develop heuristics (guided search)
- describe special cases
Exploit Togetherness

Given System

Abstraction

Galois Connection

\[ \forall P, P^A : \alpha(P) \subseteq P^A \iff P \subseteq \gamma(P^A) \]

Property holds in abstraction \(\rightarrow\) Property holds in actual system
Starting Point: Brick Sorter

Verify controller, running on LEGO® RCX
- Specified as timed automaton (UPPAAL)
- Proven Property: *always will kick off black*

Problem

model checking slow: many delay steps without real progress

Possible solutions

- temporal scaling: $\approx \text{'next } \Theta \text{ for } P'$
- abbreviations: $1000a \rightarrow 512a + 256a + 128a + 64a + 32a + 8a$
- building abstraction and prove it correct
Summary

Previous work

• Bit-Vectors: 2 canonizers, 3 solvers, 2 new complexity results
• Model-Checking: heuristic to partition a system hierarchically

Future plans

• Develop a variation of ’next’, tailored for timed automata
• Explore the phenomenon of Togetherness in concurrent systems in greater detail