Advances in Automated Theorem Proving

Thomas Ball, MSR, representing work of many
Logic and Complexity

- Undecidable (FOL + LIA)
- Semi Decidable (FOL)
- NEXPTIME (EPR)
- PSPACE (QBF)
- NP (SAT)
Times Have Changed
Then

NP-Completeness
Reduction from
SAT

All NP problems
reduce

SAT
reduce

your problem
Now
Now

Reduce your problem to SAT and solve with SAT solver
Logic and Complexity

Logic is “The Calculus of Computer Science” Zohar Manna

Practical problems often have **structure** that can be exploited.

Diagram:
- Undecidable (FOL + LIA)
- Semi Decidable (FOL)
- NEXPTIME (EPR)
- PSPACE (QBF)
- NP (SAT)
Satisfiability Modulo Theories (SMT)

- machine integers
- integers
- floating point
- reals
- bit vectors
- strings
- arrays
- lists
- sets
- maps
Satisfiability

\[ x^2 + y^2 < 1 \text{ and } xy > 0.1 \quad \text{sat, } x = \frac{1}{8}, y = \frac{7}{8} \]

\[ x^2 + y^2 < 1 \text{ and } xy > 1 \quad \text{unsat, Proof} \]
Pex, Automated White box Testing for .NET

Pex and Moles are Visual Studio 2010 Power Tools that help Unit Testing .NET applications.

- **Pex automatically generates test suites with high code coverage.** Right from the Visual Studio code editor, Pex finds interesting input-output values of your methods, which you can save as a small test suite with high code coverage. Microsoft Pex is a Visual Studio add-in for testing .NET Framework applications.

- **Moles allows to replace any .NET method with a delegate.** The Fakes Framework in Visual Studio 2012 is the next generation of Moles & Stubs, and will eventually replace it. Moles supports unit testing by providing isolation by way of detours and stubs. The Moles framework is provided with Pex, or can be installed by itself as a Microsoft Visual Studio add-in.
WWW.pex4fun.com
Dynamic Test Generation

```c
void top(char input[4])
{
    int cnt = 0;
    if (input[0] == 'b') cnt++;
    if (input[1] == 'a') cnt++;
    if (input[2] == 'd') cnt++;
    if (input[3] == '!') cnt++;
    if (cnt >= 4) crash();
}
```

Negate each constraint in path constraint
Solve new constraint → new input

Key technology: satisfiability solving
Z3 is a collection of Symbolic Reasoning Engines

- Automated Theorem Prover
- Leonardo de Moura and Nikolaj Bjørner

- SAT
- Simplex
- Rewriting
- Superposition
- Congruence Closure
- Groebner Basis
- Existential Elimination
- Euclidean Solver
Some (MSR) Applications

• Functional verification: Dafny, VCC
• Defect detection: Corral, HAVOC, Poirot, SLAM
• Test generation: Pex, SAGE
• Design-space exploration: Formula, SpecExplorer
• New programming languages: F*
Learn about Z3 and get the source code!

• Start here
• Strategies
• Source code
  – http://z3.codeplex.com/
Part 1

Revisiting Foundations in Light of SMT
Symbolic Automata and Transducers

Margus Veanes, Nikolaj Bjørner
(POPL 2011)
Core Question

Can classical automata theory and algorithms be extended to work modulo large (infinite) alphabets $\sum$?
Symbolic Automata: Relativized Formal Language Theory

Symbolic Word Transducers
≡
Classical Word Transducers \(\text{modulo } Th(\Sigma)\)

Classical Word Transducers
(e.g. decoding automata, rational transductions)

Classical I/O Automata
(e.g. Mealy machine)

Symbolic Word Acceptors
≡
Classical Word Acceptors \(\text{modulo } Th(\Sigma)\)
(NFA, DFA)

regex matching

string transformation
Symbolic Finite Automaton (SFA)

- Alphabet is an **effective Boolean Algebra** \( A \)
- Labels are **predicates over** \( A \)

**one symbolic transition:**

\[
\lambda x. \ 'a' \leq x \leq \ 'd'
\]

**denotes many concrete transitions:**

\[
[ 'a' \leq x \leq \ 'd']
\]
Boolean operations over SFAs

- Intersection (product of transitions)

\[ A_1 \times A_2 : \]

\[ p_1 \xrightarrow{\varphi_1} q_1 \]

\[ p_2 \xrightarrow{\varphi_2} q_2 \]

\[ p_1 \xrightarrow{\varphi_1 \land \varphi_2} q_1 \]

\[ p_2 \xrightarrow{\varphi_1 \land \varphi_2} q_2 \]

Delete when \( \varphi_1 \land \varphi_2 \) unsat
Intersection example

let $\varphi_k(x) \iff ((x \mod k) = 0)$
Symbolic Finite Transducer (SFT)

- Classical transducer *modulo* a rich *label theory*
- Core Idea: represent labels with guarded transformers
  - Separation of concerns: finite graph / theory of labels

Concrete transitions:

\[ p \xrightarrow{\text{\textbackslash x80}}q \]
\[ p \xrightarrow{\text{\textbackslash xC2\x80}}q \]
\[ p \xrightarrow{\text{\textbackslash x7FF}}q \]

1920 transitions

Symbolic transition:

\[ \lambda x. \ 80_{16} \leq x \leq 7FF_{16}/\]
\[ [C0_{16} | x_{(10,6)}, 80_{16} | x_{(5,0)}] \]

Guard

Bitvector operations
Algorithms

• **New** algorithms for SFAs and SFTs

• Extensions of classical algorithms *modulo* $\text{Th}(\Sigma)$

• Big-O complexity matches that of classical algorithms, with factor for decision procedure
Analysis

Example 1: $\exists x (\text{utf8encode}(x) \notin R_{\text{utf8}})$ ?

1. $E = SFT(\text{utf8encode})$
2. $A = \text{Complement}(SFA(R_{\text{utf8}}))$
3. $B = \lambda x. A(E(x))$
4. $B \neq \emptyset$ ?

Example 2: $\lambda x. \text{utf8decode}(\text{utf8encode}(x)) \equiv Id$ ?

Does there exist an input $x$ that causes a bad output?
For More Information

- Bek
  http://rise4fun.com/Bek/tutorial

- Bex
  http://rise4fun.com/Bex/tutorial

- Rex
  http://rise4fun.com/rex/

- Fast
  http://rise4fun.com/Fast/tutorial
A Recipe for Success?

Reduce your problem to SAT and solve with SAT solver
Part 2. The Secret Sauce

Harnessing the power of symbolic abstractions in your domain
Idea:
Represent exponentially many concrete objects efficiently (with useful operations)
Syntactic String Transformations (from Examples)

Flash Fill feature in Excel 2013

Demo!

Reference:
Automating String Processing in Spreadsheets using Input-Output Examples, POPL 2011, Gulwani
Syntactic String Transformations: Language

Trace Expr e := $\text{Concatenate}(f_1, \ldots, f_n)$

Base Expr f := $s$

| $\text{SubStr}(v_i, p_1, p_2)$

Position Expr p := $k$

| $\text{Pos}(r_1, r_2, k)$
Let \( w = \text{SubString}(s, p, p') \)
where \( p = \text{Pos}(r_1, r_2, k) \) and \( p' = \text{Pos}(r_1', r_2', k') \)

Two special cases:
- \( r_1 = r_2' = \epsilon \): This describes the substring
- \( r_2 = r_1' = \epsilon \): This describes boundaries around the substring
Too many choices for a Trace Expression

Input \((425) - 706 - 7709\)

Output \(425 - 706 - 7709\)

Constans
Number of all possible trace expressions is \textit{exponential} in size of output string.

To represent/learn all trace expressions, it suffices to represent/learn all base expressions for each substring of the output.

- \# of substrings is just \textit{quadratic} in size of output string!

- A \textit{DAG} based data-structure represents all trace expressions transforming input to output, and supports efficient intersection operation
Too many choices for a SubStr Expression

Various ways to extract “706” from “425-706-7709”:

• Chars after 1\textsuperscript{st} hyphen and before 2\textsuperscript{nd} hyphen.
  \[
  \text{Substr}(v_1, \text{Pos}(\text{HyphenTok}, \varepsilon, 1), \text{Pos}(\varepsilon, \text{HyphenTok}, 2))
  \]

• Chars from 2\textsuperscript{nd} number and up to 2\textsuperscript{nd} number.
  \[
  \text{Substr}(v_1, \text{Pos}(\varepsilon, \text{NumTok}, 2), \text{Pos}(\text{NumTok}, \varepsilon, 2))
  \]

• Chars from 2\textsuperscript{nd} number and before 2\textsuperscript{nd} hyphen.
  \[
  \text{Substr}(v_1, \text{Pos}(\varepsilon, \text{NumTok}, 2), \text{Pos}(\varepsilon, \text{HyphenTok}, 2))
  \]

• Chars from 1\textsuperscript{st} hyphen and up to 2\textsuperscript{nd} number.
  \[
  \text{Substr}(v_1, \text{Pos}(\text{HyphenTok}, \varepsilon, 1), \text{Pos}(\varepsilon, \text{HyphenTok}, 2))
  \]
Synthesizing SubStr Expressions

The number of $\text{SubStr}(v,p_1,p_2)$ expressions that can extract a given substring $w$ from $v$ can be large!

To represent/learn all SubStr expressions, we can independently represent/learn all choices for each of the two index expressions.

This allows for representing and computing $O(n_1 \times n_2)$ choices for SubStr using size/time $O(n_1 + n_2)$. 
Ranking

General Principles

- Prefer shorter programs, shorter string expression, regular expressions.
- Prefer programs with less number of constants.
Conclusions

- Logic as our calculus
- Automation
- Logical thinking