http://lara.epfl.ch

Laboratory for Automated Reasoning and Analysis

Viktor Kuncak
Assistant Professor, IC

a project: http://JavaVerification.org
ongoing class: http://RichModels.org/LAT
Spring, will be like: http://lara.epfl.ch/sav09
Automated Reasoning

General Problem Solver (Newell, Simon 1959)
  – would take any problem description theorems, chess games, ...
  – output a solution

GPS was too ambitious to be useful

Trend since then: look at specific **domains**

An important domain:
  – reasoning about models of computer systems (software, hardware, embedded systems)
  – math, algorithms, software tools for this
Software Verification

Java source code

proc remove(x : Node) {
    Node p=x.prev; n=x.next;
    if (p!=null) p.next = n;
    else root = n;
    if (n!=null) n.prev = p;
}

no errors, crashes
x.next.prev = x
tree is sorted

(error in program)

Jahob Verifier
javaVerification.org

(automatically generated mathematical proof that)
program satisfies the properties

desired properties

no errors, crashes
x.next.prev = x
tree is sorted

(error in program)
(or property)
A Desired Property: No Crashes  
(from a BBC article)

Cryosat, a satellite worth 135m euro  
October 2007
Desired Properties of Data Structures

unbounded number of objects, dynamically allocated

shape not given by types, may change over time

class Node {
    Node f1, f2;
}

Declaration alone admits both trees & lists – need “invariants”
More Examples of Desired Properties

dynamically allocated arrays

node is stored in the bucket given by the hash of node’s key

instances do not share array

numerical quantities

value of size field is number of stored objects

\[ \text{size} = |\{x. \text{next}*(\text{first},x)\}| \]
Specification in Jahob
specs as verified comments
public interface is simple

class List {
    private List next;
    private Object data;

    private static List root;
    private static int size;

    private static ghost specvar nodes :: Object;
    public static ghost specvar content :: Object;

    invariant nodesDef: "nodes = {n. n \neq null \land (root,n) \in \{up, down\}}";
    invariant contentDef: "content = {x. \exists n. x = List.data n}";

    invariant sizeInv: "size = cardinality content";
    invariant treeInv: "tree [List.next]";
    invariant rootInv: "root \neq null \rightarrow (\forall n. List.next n \neq root)";
    invariant nodesAlloc: "nodes \subseteq Object.alloc";
    invariant contentAlloc: "content \subseteq Object.alloc";

    public static void addNew(Object x)
    /*: requires "comment ''xFresh'' (x \notin content)"
    modifies content
    ensures "content = old content \cup \{x\}" */
    {
        List n1 = new List();
        n1.next = root;
        n1.data = x;
    }
Verifying the addNew method

Verification steps

- generate verification condition (VC) in logic, stating “The program satisfies its specification”
- split VC into a conjunction of smaller formulas $F_i$
- prove each $F_i$ conjunct using a number of specialized theorem provers
Jahob Verifier

SPASS -> E -> CVC3 -> Z3 -> MONA

Coq -> Coq interface

Isabelle -> Isabelle interface

splitter, dispatcher, syntactic prover

verification conditions (VCs)

interactively proven lemmas

implementation, specification, proof hints

vcgen -> desugar

lexer, parser, resolver

field constraint analysis

BAPA
Nature of Research in LARA

Two kinds of activities (closely related):
  – Algorithms, Decidability, and Complexity (understand the problem we are solving)
  – Making algorithms work in practice

We work with two kinds of objects:
  – programs (syntax trees, as in compilers)
  – logical formulas (for properties and programs)

\[ \forall C. \exists p \in C. ( A(p) \rightarrow (\forall x \in C. A(x)) ) \]
One aspect of our work:

Algorithms for checking validity of logical formulas that describe correctness
Algorithmic Difficulty for Arithmetic

Formula in arithmetic (with +, *)

\(-\text{next0}^*(\text{root0,n1}) \land \\ x \notin \{\text{data0}(n) \mid \text{next0}^*(\text{root0,n})\} \land \\ \text{next=next0[n1:=\text{root0}] \land} \\ \text{data=data0[n1:=x]} \rightarrow \\ |\{\text{data}(n) \mid \text{next}^*(n1,n)\}| = \\ |\{\text{data0}(n) \mid \text{next0}^*(\text{root0,n})\}| + 1\)
Algorithmic Difficulty for full FOL

Formula in first-order logic

\[ \neg \text{next}^0(\text{root}0, n1) \land \\
x \notin \{\text{data}0(n) \mid \text{next}^0(\text{root}0, n)\} \land \\
\text{next} = \text{next}[n1 := \text{root}0] \land \\
\text{data} = \text{data}0[n1 := x] \\rightarrow \\
|\{\text{data}(n) \mid \text{next}^*(n1, n)\}| = \\
|\{\text{data}0(n) \mid \text{next}^0(\text{root}0, n)\}| + 1 \]

- formula is valid
- can loop if there are infinite counterexamples!
- formula has finite counterexample

first-order logic theorem prover
Decision Procedures

formula in decidable logic

\[ \neg \text{next}^0(root0,n1) \land \\
x \notin \{\text{data}(n) \mid \text{next}^0(root0,n)\} \land \\
\text{next} = \text{next}^{0[n1:=\text{root}0]} \land \\
\text{data} = \text{data}^{0[n1:=x]} \Rightarrow \\
|\{\text{data}(n) \mid \text{next}^*(n1,n)\}| = \\
|\{\text{data}^0(n) \mid \text{next}^0(root0,n)\}| + 1 \]

never loops!
always works

formula is valid

formula has a counterexample
Example of Decidable Logics

- Integer arithmetic with only addition
- Integer arithmetic with only multiplication
- Real arithmetic with both addition and multiplication
- Set algebra (without nested sets)
- First-order logic with only two variables
- Logic of sets and elements interpreted over trees

Our Correctness Condition Formula

\[\neg \text{next}0^*(\text{root}0, n1) \land x \notin \{\text{data}0(n) \mid \text{next}0^*(\text{root}0, n)\} \land \text{next} = \text{next}0[n1 := \text{root}0] \land \text{data} = \text{data}0[n1 := x] \rightarrow \]

\[|\{\text{data}(n) \cdot \text{next}^*(n1, n)\}| =
|\{\text{data}0(n) \cdot \text{next}0^*(\text{root}0, n)\}| + 1\]

“The number of stored objects has increased by one.”

Expressing this VC requires a rich logic
– transitive closure * (in lists and also in trees)
– unconstraint functions (data, data0)
– cardinality operator on sets | ... |

We have a decidable logic that can express this!
One component of this logic:

**Boolean Algebra with Presburger Arithmetic**

\[
S ::= V \mid S_1 \cup S_2 \mid S_1 \cap S_2 \mid S_1 \setminus S_2 \\
T ::= k \mid C \mid T_1 + T_2 \mid T_1 - T_2 \mid C \cdot T \mid \text{card}(S) \\
A ::= S_1 = S_2 \mid S_1 \subseteq S_2 \mid T_1 = T_2 \mid T_1 < T_2 \\
F ::= A \mid F_1 \land F_2 \mid F_1 \lor F_2 \mid \neg F \mid \exists S. F \mid \exists k. F
\]

Not widely known: Feferman, Vaught: 1959

Our results

- first implementation for BAPA (CADE’05)
- first, exact, complexity for full BAPA (JAR’06)
- polynomial-time fragments of QFBAPA (FOSSACS’07)
- first, exact, complexity for QFBAPA (CADE’07)
- generalizations to bags (VMCAI’08, CAV’08, CSL’08)
Ruzica Piskac

3rd year PhD student
- MSc at the Max-Planck Institute
- Microsoft Research internship (Summer 2008)
- working on algorithms for proving formulas about sets, multisets, function images, cardinality

Combining Theories with Shared Set Operations. Symposium on frontiers of combining systems (FroCoS 2009)
Fractional Collections with Cardinality Bounds. Computer Science Logic (CSL 2008)
Linear Arithmetic with Stars. Computer Aided Verification (CAV 2008)
Decision Procedures for Multisets with Cardinality Constraints. Verification Model Checking, Abstract Interpretation (VMCAI 2008)

\[ \forall e. u(e) = 1 \land \forall e. u \geq A(e) \geq 1 \land \forall e. u \geq B(e) \geq 1 \land \forall e. u \geq U(e) \geq 1 \]

We next apply the definition of the cardinality operator, \(|C| = \sum_{e \in E} C(e)|:

\[ n_1 + n_2 < n_3 + n_4 \land n_1 = \sum_{e \in E} A(e) \land n_2 = \sum_{e \in E} U(e) \land n_3 = \sum_{e \in E} (A \cap B)(e) \land n_4 = \sum_{e \in E} (A \cup B)(e) \land \forall e. U(e) = 1 \land \forall e. 0 \leq A(e) \leq 1 \land \forall e. 0 \leq B(e) \leq 1 \land \forall e. 0 \leq U(e) \leq 1 \]
On Decision Procedures for Algebraic Data Types with Abstractions. EPFL Technical report, 2009

Hossein Hojjat

2nd year PhD student
- MSc from Eindhoven, Netherlands

Current work:
• verifying (Scala) programs
• using formulas for automated reasoning
• building automated reasoning
Giuliano Losa

1\textsuperscript{st} year PhD student - MSc at EPFL

-Current work: verifying distributed algorithms

Co-supervised w/ Prof. Rachid Guerraoui

Can we \textbf{prove} that “the penguins will indeed survive”, (even in presence of evil penguins) and can automated reasoning help in this process?
Some Further Directions

Java or Scala source code

```scala
proc remove(x : Node) {
  Node p=x.prev; n=x.next;
  if (p!=null) p.next = n;
  else root = n;
  if (n!=null) n.prev = p;
}
```

no errors, crashes

x.next.prev = x

tree is sorted

(automatically generated mathematical proof that) program satisfies the properties

verification +
test generation +
Documentation also @ run-time, for embedded software

no errors, crashes, tree is sorted

failing test case
Laboratory for
Automated Reasoning and Analysis

Viktor Kuncak
Assistant Professor, IC

a project: http://JavaVerification.org

ongoing class: http://RichModels.org/LAT

Spring, will be like: http://lara.epfl.ch/sav09