NATURAL PROOFS FOR STRUCTURE, DATA, AND SEPARATION

XIAOKANG QIU        PRANAV GARG
ANDREI STEFANESCU    P. MADHUSUDAN
UNIVERSITY OF ILLINOIS AT URBANA-CHAMPAIGN

POPL 2013, STUDENT SHORT TALK
ROME, ITALY
MOTIVATION

Automatic Software Verification

- Modular annotations + VC-Generation + SMT solvers
- Success stories (Boogie, VCC, Verifast, etc.)

How about dynamically modified heap?

- Structure (e.g., “p points to the root of a tree”)
- Data (e.g., “the integers stored in a list is sorted”)
- Separation (e.g., “the procedure modifies only elements reachable using the next field”)

Key Challenge: Expressive Power vs. Automated Reasoning
EXPRESSIVENESS VS. AUTOMATICITY

Expressive Logics:
Separation logics, HOL, Matching logic, etc.

Decidable Logics:
STRAND, LISBQ, CSL, etc.

Natural Proof?
NATURAL PROOFS: IN A NUTSHELL

• Handle a logic that is very **expressive**
  (inevitably undecidable)

• Retain automaticity at the same level as decidable logics

• Identify a class of **simple and natural proofs** \( N \)
  such that
  • Many correct programs can be proved using a proof in class \( N \)
  • The class \( N \) is effectively searchable
    (searching thoroughly for a proof in \( N \) is efficiently decidable)
  • “Unfold recursive defs + formula abstraction”
CONTRIBUTION

• **DRYAD\textsubscript{tree} [POPL'12]:** Natural proofs for trees
  • only for trees (can’t say “x points to a tree and y points into the tree”)
  • only functional recursion (no while-loops, too weak for loop invariants)
  • only classical logic (no frame reasoning)

• **Aim**
  • To provide a single logical framework that supports natural proofs for general properties of structure, data, and separation

• **Contribution**
  • **DRYAD:** A dialect of separation logic
    • no explicit quantification, but supports recursive definitions
    • admits a deterministic translation to classical logic
  • Develop natural proofs for this logic using decision procedures (powered by SMT solvers)
  • A much wider variety of programs verified automatically and efficiently
**BINARY SEARCH TREE: AN EXAMPLE OF DRYAD**

\[ \text{bst}^*(x) = (x = \text{nil} \land \text{emp}) \lor \]
\[ (x \mapsto xl, xr, xk)^* \left( \text{bst}^*(xl) \land \text{keys}^*(xl) < \{xk\}\right)^* \left( \text{bst}^*(xr) \land \{xk\} < \text{keys}^*(xr) \right) \]

\[ \text{keys}^*(x) = (x = \text{nil}; \]
\[ (x \mapsto xl, xr, xk)^* \text{true} : \{xk\} \cup \text{keys}^*(xl) \cup \text{keys}^*(xr) ; \]
\[ \text{default} : \emptyset ; \]

\[ \psi \equiv \text{bst}^*(\text{root}) \land (\text{bst}^*(\text{curr})\ )^* \text{true} \]
\[ \land (k \in \text{keys}^*(\text{root}) \leftrightarrow k \in \text{keys}^*(\text{curr})\ )^* \text{true} \]

(loop invariant for “bst-search”:

- \textit{root} points to a BST and \textit{curr} points into the BST;
- \textit{k} is stored in the BST \textbf{iff} \textit{k} is stored under \textit{curr}.)
DRYAD VS. SEPARATION LOGIC

\[ U^*(x) \overset{\text{def}}{=} (x = \text{nil} \land \text{emp}) \lor (x \mapsto x_l, x_r \ast U^*(x_l)) \lor (x \mapsto x_l, x_r \ast U^*(x_r)) \]

Separation Logic: any tree and any path

DRYAD: the heaplet is exactly the reachable locations from \( x \)
TRANSLATE DRYAD TO A CLASSICAL LOGIC

The scope (heaplet required) of a formula can be syntactically determined

- singleton heap $x \mapsto y : \{x\}$
- recursive definitions $U^*(x) : \text{reach}(x)$
- Connective $t \sim t' : \text{scope}(t) \cup \text{scope}(t)$

the domain of a heaplet can be modeled as a set of locations, and the heaplet semantics can be expressed using free set variables

Example: $\text{tree}^*(x) \ast \text{tree}^*(y)$

can be translated to

$$\begin{align*}
\text{tree}(x) \land \text{tree}(y) \land & \\
\text{reach}(x) \cap \text{reach}(y) = \emptyset \land & \\
\text{reach}(x) \cup \text{reach}(y) = G
\end{align*}$$

(still quantifier-free)
NATURAL PROOFS FOR DRYAD

We consider a toy programming language that explicitly manages the heap.

We consider programs with modular annotations (pre/post, loop invariant in DRYAD).

Natural Proofs in 4 steps

1. Translate DRYAD to classical logic
2. Generate the verification condition (compute strongest-post)
3. Unfold recursive definitions across the footprint (still precise)
4. Formula Abstraction and solve in SMT (recursive definitions uninterpreted, becomes sound but incomplete)
A prototype verifier with Z3 as the backend solver
(more details at http://web.engr.illinois.edu/~qiu2/dryad/)

- 10+ data structures
  - singly-linked list, sorted list, doubly-linked list, cyclic list, max-heap, BST, Treap, AVL tree, red-black tree, binomial heap...
- 100+ DRYAD-annotated programs
  - textbook algorithms, Glib library, OpenBSD library, Linux kernel, an ongoing OS-Browser verification project...

- All these VCs were proved by Z3!
- Few routines exceed 1 sec, even fewer exceed 100 secs
  - “RBT-delete_iter” spent 225 secs, “binomial-heap-merge_rec” spent 153 secs

(To the best of our knowledge)

First terminating automatic mechanism that can prove such a wide variety of programs full-functionally correct
DRYAD is an expressive, tractable dialect of separation logic

A deterministic translation to classical logic

Extended the natural proof scheme to let the user
• specify structural properties of the heap
• reason with multiple structures
• reason with separation (frame reasoning)

A tool that handles Hoare-triples automatically using SMT solvers
(more details at http://web.engr.illinois.edu/~qiu2/dryad/)

Ongoing: Natural Proofs for C
(encode the natural proof scheme in ghost code for VCC)
THANK YOU!