Towards Verified Distributed Software Through Refinement of Formal Archetypes

Mani Chandy\textsuperscript{1}, Brian Go\textsuperscript{1}, Sayan Mitra\textsuperscript{2}, Jerome White\textsuperscript{1}

\textsuperscript{1}California Institute of Technology
\textsuperscript{2}University of Illinois at Urbana Champaign
Overview

- Organize library of theorems for distributed systems
  - Both discrete and continuous

- Stepwise refinement and program transformation using a theorem prover

- Transform theorem-proven code to actual code
  - Java, Erlang, C#
  - mobile robots

- Use organization/abstraction to verify cross-domain problems

- Teaching
Motivating Example: Mobile Agents

- Handwaving doesn’t always work!
- Robots communication via message passing
- Each robot moves to midpoint of neighbors
- Will the robots form an equidistance straight line?
Handwaving: It Doesn’t Work
Handwaving: It Does Work

Max error

Max error

Verified Software Workshop - Experiments • 9 October 2008 – p. 5
The Challenge

- Not all results demonstrated by handwaving are true
- Not all results demonstrated by handwaving are false

Testing is difficult
- Multiple agents
- Continuous and asynchronous movement

Model checking is difficult (state space problem)

Proofs checked by humans is difficult
- Expertise in problem domain
- Expertise in predicate calculus

Proofs checked by computer require a great deal of time
Our Approach: Reuse

- Reuse/refinement have demonstrated value in object oriented programming
  - Generic solutions refined to obtain specific ones
  - Generic objects can be reused
- We use the same (old) idea for theorems
  - Refine generic algorithms and checked proofs
  - Obtain programs and proofs for specific problems
Example: Consensus

- Given: Distributed system with \( n \) agents
- Final state of each agent is the
  - Minimum
  - Maximum
  - Greatest common divisor
  - Least common multiple
  - Convex hull
of the initial values of agents
Refinement Overview

Consensus

Operator: Assc, Comm, Idem
Shared State

Program transformation

Operator: Assc, Comm, Idem
Message Passing

Operator: min, max, lcm, ... in PVS

Java Agents

Erlang Agents

Hand transformation
Specification of Consensus

- Given: Distributed system with \( n \) agents indexed \( 0, \ldots, n - 1 \)
- System state is an array \( S \) where \( S[j] \) is the state of agent \( j \)
- Initial system state: \( S^0 \)
- Desired final state: \( S^* \) where \( \forall j : S^*[j] = f(S^0) \)
- Interested in properties of \( f \) …
Operator Refinement

- $f$ is the aggregation ($fold$) of a composable operator $o$

\[
f(S) = S[0] \circ S[1] \circ \ldots \circ S[n-1]
\]

\[
= fold(S, J, o) = \begin{cases} 
S[0] & \text{if } |J| = 1, \\
fold(S, \{J_0, \ldots, J_{n-2}\}, o) \circ S[J_{n-1}] & \text{otherwise.}
\end{cases}
\]

where $J$ is a set of $n$ agents in $S$

- $o$ is commutative, associative, idempotent
An Abstract Shared-State Algorithm

- State transition defined for all \( j, k : S'[j] = S[j] \circ S[k] \)
- As a predicate, for all \( S \)

\[
\text{transition?}(S, S') \equiv \forall j : \exists J : j \in J \land S'[j] = \text{fold}(S, J, o)
\]

- Predicates allow us to talk about action sets
- Easier to reason about in PVS
Prove Abstract Algorithms in PVS

Prove consensus abstractions using generic operator $o$

- Invariant (conservation law of $f$)

\[ \forall S : fold(S, J, o) = fold(S^0, J, o) \]

Proof

\[ \forall S, S' : transition?(S, S') \implies fold(S, J, o) = fold(S', J, o) \]

where $J$ is the fullset of agents in $S$

- Progress (see paper)
Message Passing Transformation

For algorithms with specification: \( R \leadsto Q \), if there is a shared state algorithm with a proof of the following form:

\[ \exists P_0, \ldots, P_k \text{ where} \]
\[ P_0 = R, P_k = Q \]
\[ \forall j : \text{stable?}(P_j) \text{ i.e. } \forall a : \{P_j\}a\{P_j\} \]
\[ \forall j < k : \exists \text{fair?}(a) : \{P_j\}a\{P_{j+1}\} \]

and \( P_j \) is in “conjunctive” form

\[ \forall j : P_j = \rho_0(S[0]) \land \ldots \land \rho_{n-1}(S[n-1]) \]

Then the program can be transformed mechanically into a message passing program (and vice-versa)\(^a\)

\(^a\)Chandy, Mitra, Pilotto: FORMATS ’08
PVS Instantiation

- Instantiate theorem with concrete operator
- Prove operators have desired properties
  - Enforced through PVS ASSUMPTION
  - This is the only proof obligation!
- Examples: min, max, lcm, gcd, convex hull
  - Already exist in PVS
Implementation of \texttt{min}

- Implement Java programs with abstract operator $o$
- All agent objects reach a consensus of initial values
- Involves
  - Refining the $o$ operator to the \texttt{min} operator
  - Proving that the Java/PVS \texttt{min} are equivalent
  - Ensuring agent transitions are valid
- Repeat for other object-oriented languages like C#
Java/PVS Operator Mapping

\[ o: T \rightarrow T \]

\[ \text{fold: (S, J, o) } \rightarrow T \]

- Min
- Max
- GCD
- LCM

- Min
- Max
- GCD
- LCM

Operator

FoldableOperator
Refinement: Another Example

Dynamic Game Theory

Linear Dynamic Game Theory

Refinement in PVS

Mobile Agents
Shared State

Asynchronous Solutions
to Systems of Equations

Refinement in PVS

Mobile Agents
Message Passing

Mobile Robots

Transformation in PVS

Hand transformation from PVS to robot code
Contributions

- Apply old idea: reuse abstract algorithms and proofs
- Algorithm abstractions checked by theorem prover
  - High level: consensus
  - Low level: \( \text{min, max, lcm, gcd} \)
- Transform shared-state to message passing
- Mapped PVS to Java, Erlang, \ldots
  - Not checked mechanically
- Developing distributed systems course based on this idea
Questions and Request

- Developing a distributed systems course based on reuse
- Looking for collaborators interested in developing the course

See our website:
http://www.infospheres.caltech.edu/vstte08