cXprop
Postpass optimization for TinyOS applications
Nathan Cooprider  John Regehr
School of Computing, University of Utah
{coop, regehr}@cs.utah.edu

Summary
Creating TinyOS applications from generic components often leads to applications containing dead code and constant variables. These unnecessarily consume SRAM and Flash memory. We address the problem with cXprop, a static-analysis-based post-processing tool for the C code produced by the nesC compiler. cXprop performs “conditional X propagation,” a generalization of the well-known conditional constant propagation algorithm where X is an abstract value propagation domain supplied by the user. cXprop is interprocedural and achieves reasonable precision on pointer-rich codes. We use novel concurrency abstractions to analyze global state inside of TinyOS’ restrictive concurrency model. The dataflow information produced by cXprop supports reduction in code size through interprocedural dead code elimination, and reduction in data size by finding global variables that do not use their full bitwidth. For example, state variables and boolean flags often use an entire byte to store a single bit. We have validated the code produced by cXprop through random testing and by dynamically checking its results against actual executions. cXprop reduces application code size on average by 10% for TinyOS applications and predicts savings of 50-100 bytes of SRAM. It is available as open source software.

Features
- Three modes of execution
- CP and DCE
- Program state assertion
- Dynamic state printing
- Pluggable abstract domains
- Handles concurrent code
- Three concurrency abstractions
- Models pointers and structs
- limited array support at this time

Pluggable abstract domains
- Constant
- Value-set
- Parity
- Interval
- Bitwise

Current results
- Graph 1: The percentage of information collected by cXprop during its analysis with the different domains.
- Graph 2: Comparison of cXprop processed code size code size from normal nesC compilation.
- Graph 3: Estimated savings in RAM through global bit compression.

Background Work


Tool components
The abstract interpreter is made up of several parts. However, there is a clean interface for the abstract domains. This makes implementing new domains straightforward.

cXprop provides a novel concurrency model for analyzing global variables in the presence of interrupts and atomic sections. Atomic sections are connected in the control flow graph.

Future work
We are continuing to improve and enhance cXprop. We plan to continue to improve the speed, precision, and expressiveness of the analysis while looking for new ways to apply the information gained by cXprop.

Acknowledgments
This work is supported by National Science Foundation CAREER Award CNS-0448047

http://www.cs.utah.edu/~coop/research/cxprop