1. Introduction
We designed and built a prototype optimization infrastructure, called Varia. Varia represents optimizations as inference rules and executes using forward chaining. Varia makes adding and combining optimizations easy—every rule is eligible to fire at each step, so optimizations are automatically combined.

We have coded five optimizations in our system. Most of the difficulty in developing Varia has been related to the execution model—we plan to encapsulate our solutions in a new rule engine and domain-specific language for optimizations. We hope Varia will be useful to explore the space of optimizations and to drive optimization in our research compilers.

2. Project goals
• Tool for exploring space of optimizations
• Prototype for future compiler infrastructure
• Easy to describe new optimizations
• Easy to combine optimizations

3. Current implementation
• Jess expert system for rule execution
• Analyses and transformations implemented:
  • Constant propagation, Constant folding
  • Branch elimination, Reachability
  • Available expressions, Reaching definitions

4. Optimization model
Every artifact in Varia is either an inference rule or a fact. Facts are stored in a database and include both data-flow facts and an encoding of the CFG to analyze and transform. Facts look as follows:

- CFG representation facts
  - Instructions
  - Blocks
- Data-flow facts
  - Properties on edges
  - Points
  - Properties
  - Where properties could be
  - var x = S
  - var x is dead

Rules can create new data-flow facts or transform program facts. Optimization stops when no rule can fire. Every data-flow fact asserted by Varia is safe to use for program transformation as soon as it is asserted—analysis can safely stop at any time.

5. Finding fixpoints
Several standard data-flow analyses use intersection to combine facts at join points. To propagate facts through join points, a fact must be found on each edge. In Varia, we represent this by asserting a hypothetical fact. On the graph below we know x=4 on edge b1→b2. If we knew x=4 on edge b3→b2, we could deduce x=4 at the top of block b2.

6. Example: sequence of optimizations vs. combined optimizations
Run in sequence: constant prop; branch elim

No change after branch elimination

Combining with branch elimination yields more precise results

7. Challenges
• Efficient execution: trimming the search space
• Proof of termination
• Soundness of rules

8. Future work
• Build a new rules engine
• Integrate with real compilers: Quick C, Phoenix
• Phase ordering and transformation profitability

http://www.eecs.harvard.edu/triforce/varia

Kelly Heffner kheffner@eecs.harvard.edu
Kevin Redwine redwine@eecs.harvard.edu

Harvard DEAS
PLDI 2005, Chicago, IL