ILOC VALIDATION

William Bricken
June 2002

An overview of the verification techniques used in ILOC implementations.

Test-vectors

The ~200 MCNC benchmarks come with a complete set of test-vectors, exhaustive in the case of combinational circuits, and as good as can be expected for the sequential circuits. All ILOC transformations are correct wrt the test vectors. In the case of parametric circuits that we generate (adders, multipliers, comparators, etc), we also generate test-vectors from RTL behavioral equations.

Logic Test Problems

In the case of core ILOC algorithms, we have a test-suite of over 1200 logic problems of all degrees of difficulty and complexity, most of them constructed to test difficult and pathological cases. This suite includes:

- tautologies constructed for code testing
  - arithmetic
  - literal pervasion
  - form pervasion
  - algebraic
  - pattern-matching
  - query reduction
  - challenge
  - circuit transformations
- minimization constructed for code testing
  - literal pervasion
  - form pervasion
  - algebraic
  - query reduction
  - challenge
- collections of challenge problems
  - Newell-Simon human problem solving
  - Pelletier theorem prover failures
  - exponential SAT
- entire textbooks
  - Manna-Waldinger
  - Klenk logic
  - Klenk minimization
  - Gersting two-level
  - Gersting multiple minima
  - Hachtel factoring

Validation Techniques

For the ***core ILOC algorithms***:

- They are implemented by two completely different approaches (fully recursive and stack-based), which are used for algebraic cross-checking.

- Each transformation is tested against the ~1200 logic tautology and minimization test sets

- Each transformation is verified by applying a tautology checker (equivalence checking) to before and after forms.

- Each transformation is also verified using case analysis (dynamic test-vector generation), similar to BDD approaches.

For the ***circuit-based ILOC algorithms***:

- Each transformation is tested against industry furnished test-vectors for the ~200 MCNC benchmarks

- Each transformation is verified using case analysis on input variables (BDD-like)

- Each transformation is verified algebraically using two techniques:
  equational transformation, i.e. A ==> B
  conversion to logical standard form, i.e. A=B ==> (A B)((A)(B))

Implementation

There's a distinction to be made between correct mathematical algorithms and correct implementations of algorithms.

For the ***mathematical algorithms***: we have all the standard proofs of consistency, completeness, convergence, etc.

For the ***implementation***: we have the above verification tests, and as well:

- The functions which comprise the recursive version of the IL core implementation have been proved correct algebraically

- The code structure is highly modular, with various redundant implementation and verification techniques such as
  object-oriented models for cell and circuit properties
  recursive functions for process tracing
  guarded transformations for functional i/o consistency
  logical variable names for all constants

- All code is extensively cross-verified and error-message interrupted at every transformation step, for e.g.

    - type-checking each cell (A cell is a compound logic gate.)
    - structural type verification
    - semantic labels for each cell id
    - output and function traces
    - dynamic random and boundary condition test-vector generation
    - partial functional evaluation
    - cell pattern-matching, abstraction, and expansion
    - separate validation of circuit structure after each transform, to check for
        - duplicate cells
        - duplicate functionality with different ids
        - cell id consistency after transforms
        - upper and lower network connectivity for each cell
        - structurally isolated cells

- All sets of circuit transformations are under version control, including

    - circuit type checking
    - debug processing modes, with conditional error messages and process interrupts
    - renumbering and circuit id control
    - statistical verification of circuit composition
    - file comparison and equivalence

A typical analysis run (in debug mode) on a single circuit might generate 20 to 30 transformation files, each cross-verified for correctness.

Hardware Technology Mapping

The circuit-based algorithms have been mapped to several hardware architectures, and verified independently for each hardware model. These include:

- logic networks with arbitrary node logic
- flattened forms with all logic in one node
- two-input standard gate libraries
- circuit library expansion/abstraction
- Comesh diagonal array-based form (old version)
- Comesh block-array (in process of testing)