Block-Abstraction Memoization with CEGAR
(In-Place vs. Copy-On-Write Refinement)

Karlheinz Friedberger

**Overview**
Block-abstraction memoization (BAM) [5] is a technique for software verification that aims towards a modular scalable analysis for large programs. It is based on common concepts like
- configurable program analysis (CPA) [1] and caching and information reuse.

BAM is independent of the underlying analysis and can be used in combination with
- predicate abstraction [2]
- explicit-state model checking [3]
- BDD-based software verification [4]

**Control Flow of BAM**
BAM computes new states for the state space based on blocks, the cache, and the underlying analysis.

- If the block is removed, there is no further exploration.
- If the block is entered, compute successors with the underlying analysis.
- If the block is entered and is in the cache, compute the block abstraction with the nested CPA-algorithm.

**State-Space Exploration**
Basic steps of BAM:
- The program is divided into blocks (functions or loops).
- The nested CPA algorithm explores and analyzes the state space of each block.
- Block abstractions are cached for reuse.

**Optimization and Heuristics**
- **Reducer**: Hide unnecessary information in states to increase cache hit rate.
- **Aggressive caching**: Over-approximate entries when accessing the cache.
- **Refinement strategies**: Refine one, some, or all states along a counterexample trace.

**In-Place Refinement**
- Change existing block abstractions.
- Remove several parts of the reached state space.
- Recomputation can lead to repeated counterexamples due to information loss.

**Copy-on-Write Refinement**
- Invalidate only some reached states.
- Use copies of existing block abstractions.

**Evaluation**
For simple tasks with only zero or one refinements both approaches behave identical. For difficult tasks that need more refinements (and thus more runtime) the copy-on-write approach shows its benefit over the in-place approach.

**References**