L05e. Systems from Components

Scan me

Introduction:

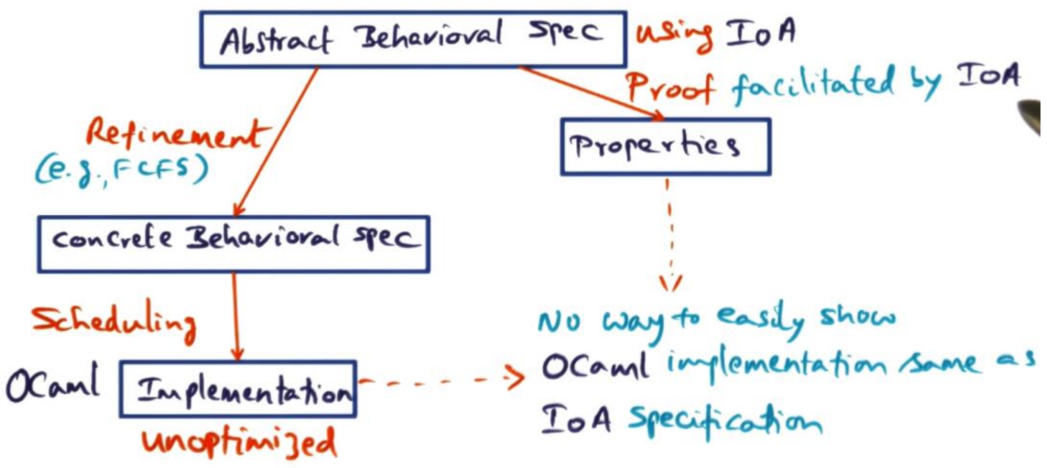
- Hardware design uses a component-based approach to build large and complex hardware systems.
- Using the same approach to build software, reusing software components, will facilitate:
  - Easy testing and optimization.
  - Easy system extension through the addition and deletion of components.
- Possible issues:
  - Might affect the efficiency and performance due to the additional function calls.
  - Could lead to loss of locality.
  - Could lead to unnecessary redundancies.

Design Cycle:

- Specification: IOA is used to express abstract specifications of the system at the level of individual components.
  - C-like syntax.
  - Includes a composition operator.
- Code: OCMAL programming language is used to convert the specification to code that can be executed.
  - Object oriented.
  - Efficient code similar to C.
  - Good complement to IOA.
- Optimization: NuPrl framework is used to optimize OCAML code to prepare it for deployment.

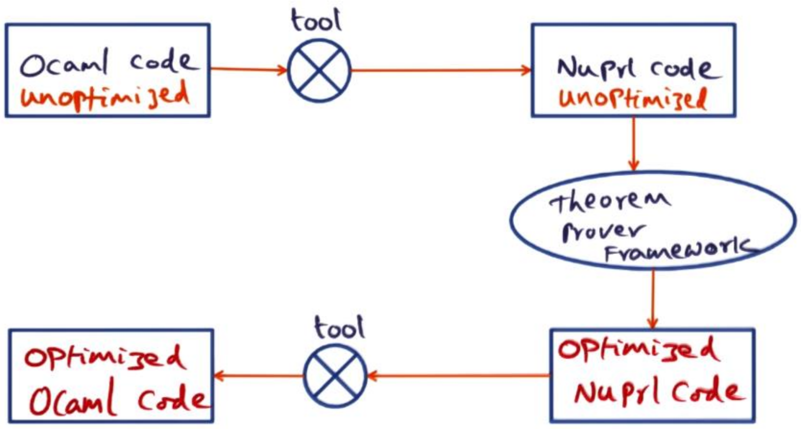
From Specifications to Implementation:

- Abstract Behavioral Specifications:
  - Describes the functionality of the sub-systems in terms of requirements using IOA.
  - IOA facilitate proofing of the Abstract Behavioral Specifications against the required properties.
- Concrete Behavioral Specifications: Produced by refining the Abstract Behavioral Specifications.
- Implementation: Translating the Concrete Behavioral Specifications into code using OCAML.



From Implementation to Optimization:

- The unoptimized OCAML code will be converted to unoptimized NuPrl code.
- The unoptimized NuPrl code will be converted to an optimized NuPrl code using the Theorem Prover Framework.
- The optimized NuPrl code will be converted to an optimized OCAML code.



Synthesizing a TCP-IP Stack:

- Specify the Abstract Behavioral Specifications and produce the Concrete Behavioral Specifications.
- Using an ensemble suite of micro-protocols to produce the unoptimized OCAML code.
- Sources of possible optimization:
  - Explicit memory management instead of implicit garbage collection.
  - Avoid marshaling/unmarshaling across layers.
  - Buffering in parallel with transmission.
  - Header compression.
  - Locality enhancement for common code sequences.

- Using NuPrl:
  - Static optimization: A NuPrl expert and an OCAML expert goes through the protocol stack layer by layer and identify optimization possibilities.
  - Dynamic optimization: Collapsing multiple layers, if possible, to avoid latency.
  - This is achieved by deriving common case predicates from the state of the protocol using conditional statements.
  - If the CCP is satisfied, the protocol layers will be bypassed by a generated code for this CCP.