PCES kickoff meeting; 1-2 August 2000, Colorado Springs

CARA: Composition, Abstraction and Refinement for Aspects

John Rushby (for N. Shankar)

Computer Science Laboratory
SRI International
Menlo Park, CA

Preamble

- The Aspect-Oriented Programming (AOP) analysis of why embedded systems are hard to construct has a lot of merit
  - Nonfunctional aspects cut across the functional decomposition of the system
  - And their code pervades its implementation
- These are seen as programming issues because the functional and nonfunctional aspects first come together at this time
  - Would do better to confront them earlier, during specification and refinement
- Would then have options in addition to code weaving
  - Architecture and middleware support for selected aspects (e.g., partitioning, fault tolerance, scheduling)
  - Inline reference monitors for certain safety aspects
  - Event-triggered services for others

Goal

- Analyze and construct embedded systems by manipulating their specifications rather than their code
    - Perform those manipulations through mechanically-supported formal calculations
- Reduce the time and cost to develop embedded systems
- And the time and cost to modify them
- Increase the assurance provided

Formal Mechanisms

- Specification of functional and nonfunctional properties, and of environment models
- Composition of properties (aspects), not just components
- Refinement that maintains global invariants and conforms to environment models of other components
- Abstraction to generate small models from large by projecting out specific aspects or behaviors

CARA: Composition, Abstraction and Refinement For Aspects

- Will build on PVS and SAL

  PVS: Widely used verification system (specification language integrated with powerful theorem proving and model checking support)

  SAL (under construction, jointly with Stanford, and in collaboration with Berkeley): combination of PVS and other systems interacting through a common intermediate language to provide highly automated analysis for concurrent and reactive systems

- SAL focuses on abstraction

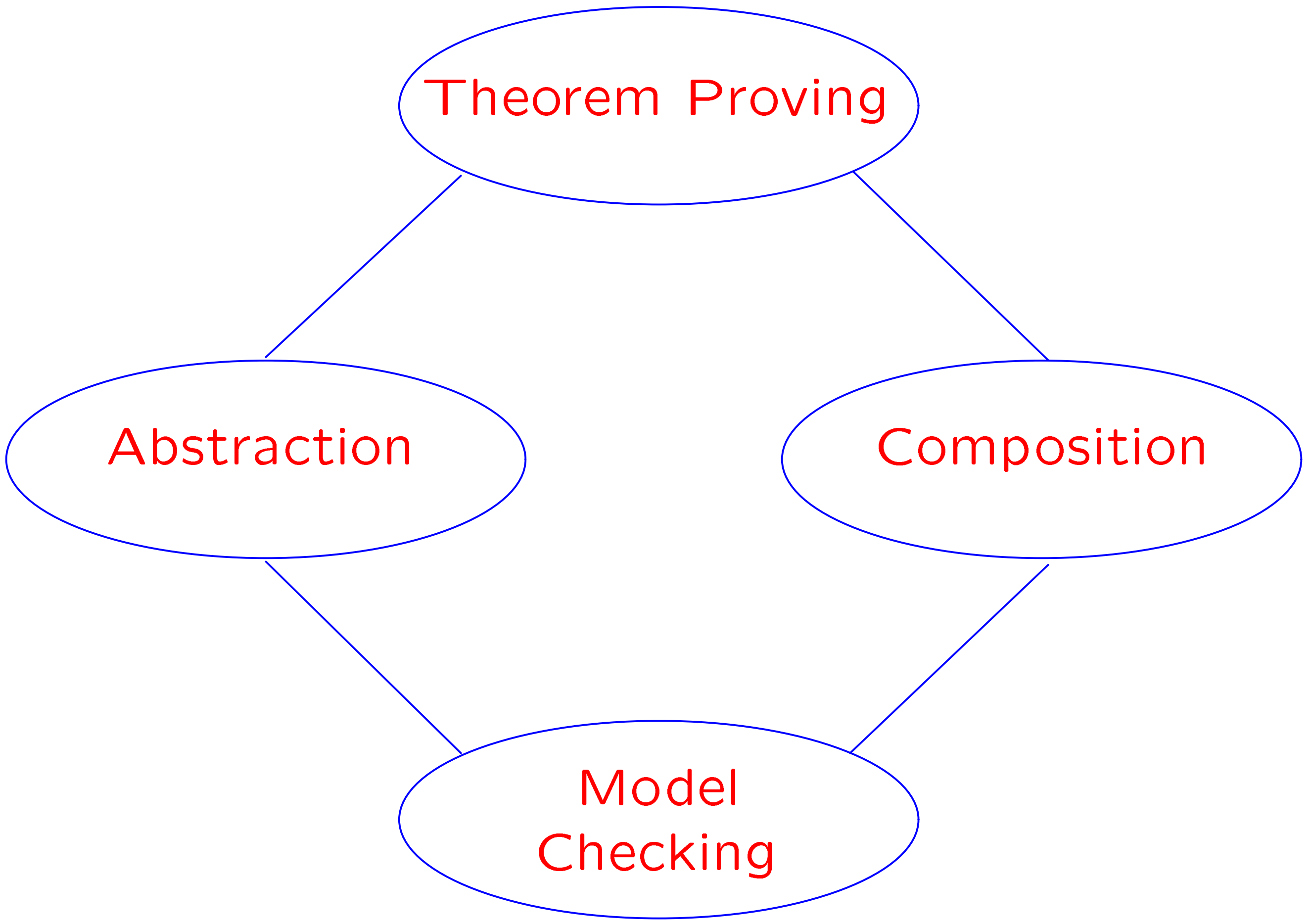
  CARA will focus on composition and refinement

Aside: Practical Application of our Tools

The bibliography on the PVS web site `pvs.csl.sri.com` lists over 300 papers describing PVS and its applications; most are by external users of PVS:

- PVS was used in Ontario Hydro's certification of the digital shutdown system at the Darlington Nuclear Power Station (the only digital shutdown system in North America)
- PVS was used in requirements analysis for new Space Shuttle software, and for Cassini
- PVS is used in digital design and protocol verification by several major manufacturers
- PVS is being used in verification of a next-generation FADEC (engine controller) for passenger aircraft

Composition Joins Abstraction as a Bridge Between Algorithmic and Deductive Methods in Analysis



Composition and Refinement

- Abstraction and refinement are not inverses
  - E.g., a predicate abstraction of the Bakery algorithm for mutual exclusion is not suited for refinement to a distributed implementation
- Similarly, composition in support of development and refinement is different than composition in support of verification
  - For refinement, we must not require prescience: developer of one component cannot perfectly anticipate properties that must be guaranteed in order to satisfy assumptions of components it will be composed with
    - ⋆ E.g., Deallocated cells must be colored red

Lazy Composition [Shankar 1997, LNCS 1536]

- Traditional assume-guarantee compositional reasoning requires the proof obligations to be discharged at the point of composition
  - I.e., guarantees of A must entail assumptions of B, and vice-versa
- Lazy composition allows proof obligations to be carried along and discharged during refinement
- Analogous to code weaving in that refinement cuts across component boundaries, but the “weaving” is applied to specifications

Architecture and Middleware Mapping

- Certain aspects such as partitioning and fault tolerance in real-time systems are best provided by targeting a specific architecture or middleware that provides those aspects
  - E.g., the Time-Triggered Architecture (TTA) of Kopetz
- Need to capture formally the aspects provided by these architectures and middleware services
  - Some of these are elusive (not trace properties): e.g., partitioning, security
- Must also capture assumptions under which they work
  - Some of these also are elusive: e.g., the transparent fault tolerance of TTA requires that replicas are allocated to different components
- Refinement must be tailored to target the chosen architecture and middleware services

Later: Weaving Specifications and Code

- Systems such as SASI and Naccio enforce certain aspects by modifying system code to invoke Inline Reference Monitors (IRM) or Edge Triggered Services (ETS), respectively
- Extend this to specification, then generate and weave the glue code using enhanced form of the PVS evaluator
  - Given a specification of the aspect desired (security policy or service, respectively) compile a suitable IRM or ETS and weave in its calls
- For mobile or mutually suspicious environment, extend to a kind of PCC
  - Most properties of interest are decidable
  - Clients generate decision procedures and validate supplied code themselves, rather than check a proof provided by the supplier

Technology Transfer and Adoption

- The project will be driven by experiments and by interactions with our user community
- Will validate the utility of CARA through experiments and (in the third year) a challenging case study
  - Derived from our work with aerospace companies (e.g., next-generation FADEC)
- The CARA tools will be freely distributed and will be our main path for technology transfer