Experience & Impact

Dewayne E Perry
ENS 623A
Office Hours: T/Th 11:00-12:00
perry @ ece.utexas.edu
www.ece.utexas.edu/~perry/education/382v-s06/

Prototype Syntax

- system :: <modspec> <impl>
- modspec :: modname <preddef> <typename> <opspec>
- preddef :: predname <predpar> [wff + primitive]
- predpar :: type predparname
- opspec :: opname <oppar> <pre: wff> <post: wff> <obl: wff>
- oppar :: modetype opparname
- impl :: opname <oppar> <localdef> <opcall> <proppre> <proppost> <propobl>
- localdef :: type varname
- opcall :: op <oparg> <pre> <post> <obl> <known> <floor <ceiling> <labelname>
- Linkedwff :: wff <label>

Views

Implemented 7 views for the prototype:

- system overview
- specification overview
- detailed specification view
- implementation overview
- basic completeness view
- full completeness view
- interconnection view

Specification Semantics

- Symbol Table
  - <name, nametype> must be unique
  - delete definition, uses remain -> warning
  - delete all uses, def remains -> warning
- Type Checking of Arguments
  - argument undefined -> warning
  - parameter undefined -> warning
  - arg type different from par type -> error
- Specification Consistency
  - wff inconsistent wrt list -> error
- Specification Completeness
  - check for metas
  - IN par has at least on precondition
  - OUT par has at least one postcondition

Implementation Semantics

- Symbol Table
  - Op name must be defined - ie, the operation must have a specification in order to be called
- Type Checking of Arguments
  - Arg type must match par type
- Propagated Specification Consistency
  - Guaranteed by the construction semantics
- Propagated Specification Completeness
  - The same as for specs

Propagation Preliminaries

- Preconditions
  - satisfied
  - reaches ceiling
  - propagated
- Postconditions
  - unknown until known
  - known until contradicted
  - while known, satisfies pre/obl
  - known after last call is propagated
- Obligations
  - satisfied
  - reaches floor
  - propagated

Comments on Implementation

- Syntax got much bigger than originally thought because of slight differences in semantics.
- Unparsing not too much of a problem; but could not do formatting as would have liked.
- Symbol Table structure is not quite right.
  - local tables for parameter def and usage
  - better access for larger symbol tables
- Prototype is SLOW.
  - template build and arg fill
  - especially propagation
  - clip and attendant processing
  - still not sure what underlying processing is entailed by the system

Comments on Attributes

- Need more boolean attributes in order to remember the current state of affairs and reduce the amount of reprocessing.
- Would like to be able to display an attribute grammar in a separate window.
  - could then treat the interconnection and propagation information as an attribute, not as part of the implementation language.
  - would get a clean separation of the specification, implementation, and program construction languages.

Comments on ARL

- No Enumerations
  - many cases where boolean is not sufficient
  - use of integers too archaic
- No User Defined Data Structures
  - only trees
  - excess baggage (eg, a stack as a grammar)
  - problems with non-connected subtrees and references
- Daemons: No Collapsing of Cases
  - eg, create and insert often identical
- No Debugging Facilities
  - trace (pause for each arl routine)
  - selectable trace
  - inspection (ie, be able to use debug)

Needs for "Real" Implementation

- Specialized Clip facility
- Appropriate Symbol Table organization
- Interconnection information as attributes
- Multiple windowing and multiple fonts
- Module Specifications in separate files
- Module Implementations in separate files
- General data base for browsing (for predicates, types and operations)

Software Evolution and 'Light' Semantics

ICSE 1999 Most Influential Paper Award from ICSE 1989
The Inscape Environment

Motivation for Inscape

- Came from building systems
- Pieces often did not fit together
  - informal interface descriptions
  - incomplete interface descriptions
  - often dependent on folklore
- Changes resulted in surprises/faults
  - complexity
  - inability to foresee consequences
- Three intertwined and essential problems
  - complexity
    - *'light' semantics*
  - composition
    - *interface specifications - designer intent*
  - evolution
    - *establish semantic dependencies - user intent*
    - *implications of interface and implementation changes*

Complexity

- Complexity
  - Intricacy of detail
    - analogies: long logical proofs, 4-voice fugues
    - difficult to understand
    - change is difficult and error prone
    - difficult for creator, compounded for others
- Wealth of detail
  - analogies: toccata, Strauss symphonic poem
    - details obscure 'real' patterns
    - individual detail of minor importance
  - sw: often incomprehensible by single person
    - complexity and scale interact
    - growth by multiple distinct components, not replication of a small set
  - intricacy may be buried in the midst of wealth

Complexity

- Invisibility
  - semantic intricacy and wealth mostly implicit
  - code is the desiccated product of a long intellectual process
  - cope by attaching meaning to syntactic details
    - comments
    - suggestive names
    - abstraction and encapsulation
- Claim
  - *'Wealth' is dominant in almost all software systems we build*
    - support for this in fault studies
      - ~80% faults fixed quickly
      - prevention of problems via knowledge
  - 'Wealth' requires different strategy
    - manage small details vs automate deep insights
    - many small theorems vs few large theorems

'Light' Semantics

- Beyond type checking
- Short of full-scale theorem proving
- Possible forms
  - partial semantic information
  - partial/simple use of semantic information
  - approximations
- Inscape explored aspects of the first two

Inscape - Contributions

- Specifications
  - obligations
  - multiple results
- Construction and evolution
  - constructive use of specifications
  - structured exception handling
  - propagation logic
  - 'light' semantic dependencies

Inscape - Construction

- *Basic rule: preconditions and obligations must be satisfied or propagated to the interface*
- Logical barriers: precondition ceilings and obligations floors
- Propagation rules for language structures: assignment, sequence, selection, iteration, and encapsulation
- Completeness of implementation:
  - empty precondition ceilings
  - empty obligation floors
  - no iteration errors
- Correctness of implementation
  - propagated interface 'matches' the specified interface

Inscape - Exceptions

- *Precluded* - precondition satisfied
- *Pruned* - validated -> assumed
- *Reported* - propagated to the interface
- *Recovered* - retry (possibly repair)
- *Repaired* - fixed and merged
- *Ignored* - results are satisfactory, merged
- *Coalesced* - merged and propagated
- *Introduced* - some result propagated as exception

Inscape - Evolution

- Interface evolution
  - kinds of changes: predicates, interfaces, exception conditions
- Effect on implementations
  - dependency re-analysis: no effect, code no longer needed, new code needed
  - exception analysis: removal, handling, preclusion, new handlings
- Implementation evolution
  - add/remove satisfaction/dependence
  - add/remove logical barriers
  - add/remove propagated predicates
  - add/remove/alter exception handling
  - effect interface

Related Work

- Some influence on industrial software engineering
  - Lockheed
  - Anderson Consulting
- Some related research
  - Daniel Jackson's Aspect
  - Don Batory's GenVoca
  - Borgida/Devanbu ICSE99 paper
    - Descriptive Logics (DL) and
    - Intermediate Definition Languages (IDL)

Related - Anderson

- ➔ Software Interface Specification and Analysis
  - language to specify behavior of modules
  - prototype to support plug&play analysis/testing
- ➔ Component based SW Engineering
  - visually specify component based distributed systems
  - semantically analyzed
  - transformed to executable code (Corba/OLE)
- ➔ Eagle
  - internal component framework
  - deployed through Anderson and client companies
- ➔ David Curtis, one of the authors for the Corba Component Model

Related - Aspect

- ➔ 'Aspect is an attempt to find some middle ground between program verification and type checking'
  - annotations which assert dependencies between procedure inputs and outputs
  - dependency analyses to check code against the annotations
- ➔ Aspect is necessary but not sufficient
  - if an error determined, there is an error
- ➔ Important characteristics
  - specification is partial
  - checking is straightforward
  - analysis is non-local (will miss bugs)
  - more akin to liveness than safety assertions, ie good at catching errors of omission

Related - GenVoca

- ➔ Problem space: generate system compositions
  - not all syntactically correct compositions are semantically correct
  - use design rules (domain specific constraints) to check automatically
- ➔ Model state of the design, not execution state
  - primitive predicates
  - pre, post, obl in terms of primitives
  - use pattern matching and simple deduction
- ➔ Analysis
  - constraints satisfied at a distance (non-adjacent)
  - propagation rules for checking
- ➔ Rationale
  - shallow consistency checking goes a long way
  - granularity: Inscape - function; GenVoca - subsystem (fewer of them and fewer predicates)
  - leverage of standardization - limits problems space

Related - DL and IDL

- ➔ Rationale in IDL context
  - stronger guarantees about *intended semantics*
  - benefits even from partially characterization
- ➔ Descriptive logic features
  - effective reasoning
  - middle ground between signature and specification matching
  - designed for modeling real-world domains
- ➔ Illustrates relevance for
  - data invariants
  - pre/post needed for methods
  - conditions leading to exceptions
  - aspects of event dynamics.
- ➔ With these, you can do
  - compatibility testing of specifications
  - local consistency checking
  - more thorough treatment of exceptions
  - variability in services provided

Summary

- ➔ **Inscape**
  - ↳ managing the connection between design and implementation
  - ↳ rich project - rich set of problems
  - ↳ frustrating -
    - ➢ found little interest internally in doing specifications
    - ➢ hence, external, not internal, influence
- ➔ **'Light' semantics a rich area for research**
  - ↳ sufficient for a large part of system evolution
  - ↳ especially at design/architecture level
    - ➢ domain specific leverage
    - ➢ benefits of larger granularity
- ➔ **Special thanks to those who worked with me for their significant contributions**
- ➔ **Many thanks to the ICSE99 PC!**