Descriptive Specifications

- Focuses on Properties
  - Describes the desired properties of a system rather than its desired behavior
- Formalisms
  - Axiomatic (Logic)
  - Algebraic

Formalisms Provide Preciseness

- Use of Mathematical Formalisms
  - Properties are specified precisely by building on top of the precise mathematical syntax and semantics of the underlying formalisms
- Mathematical Foundations
  - Predicate logic, set theory, abstract algebra

Today's Lecture

- Introduce Descriptive Specifications
  - E-R Diagrams (Semi-Formal)
  - Axiomatic
  - Algebraic
  - Tour of the RAISE system
    - Developed in Denmark
    - Sold to European Manufacturing companies
    - Using RAISE to create these types of specifications
      - Has a full tool suite
Entity-Relationship Diagrams

- A semi-formal notation for describing the structure and relationships of data
  - Akin to how Data Flow Diagrams are a semi-formal notation for describing the operations that access and manipulate data

Problems
- Syntax and Semantics are not precisely defined
- Lack of Expressive power
  - Requires the use of natural language annotations

Example ER Diagram

(taken from textbook page 200)

---

ER Diagrams and UML

- ER Diagrams can be seen as precursors to UML's Class Diagrams

Differences
- Operations and inheritance are added

Advantages
- ER notation was never standardized, UML's class diagrams provide a standard notation
  - However, remember that they are both semi-formal

Logic Specifications

- Vocabulary of Logical Expressions
  - Variables, constants, predicates, functions
  - Connectives: and (\(\land\)), or (\(\lor\)), not (\(\neg\)), implies (\(\Rightarrow\)), equivalent (\(\equiv\))
  - Quantifiers: exists (\(\exists\)), for all (\(\forall\))

- Combined with Vocabulary of Application
  - Example: set operators (\(\in\), \(\cup\), \(\cap\), )
  - Example: ADT operators (Push, IsFull, )
Logic Specifications

¥ Examples
—x > y and y > z implies x > z
—for all x (exists y (y = x + z))

¥ Additional Notes
—Variables are either free or bound
  ¥ A formula with all variables bound is called closed; closed formulas are always either true or false
—Expressions are theories in the logic
—V&V amounts to theorem proving

Creating Logic Specifications

¥ Helper Predicates and Functions
—Define the base properties of interest
  ¥ Used as a domain-specific vocabulary
—Modularize the specification
  ¥ e.g., defined in one spec; used in another

¥ Examples
—height(bob) = 72; tall(bob)
—for p: person (height(p)>60 implies tall(p))

Logic Specification Techniques

¥ Preconditions and Postconditions
—Textbook gives lots of examples on 204-205
Assume <i1, i2, i3, > are input values
Assume <o1, o2, o3, > are output values

¥ A property is defined
  {Pre(i1, i2, i3, )}
  P
  {Post(o1, o2, o3, , i1, i2, i3, >}

¥ Example
  {exists z (i1 = z * i2)}
  P
  {o1 = i1/i2}

Logic Specification Techniques

¥ Invariants and Assertions
—Logic specs are used to assert properties of portions of code as well
—For instance, to assert something that is always true of a routine or to record the assumptions about variables passed to a procedure
  {n > 0}
procedure reverse (a: in out int_array; n: in int)
  {for all i (1<=i<=n) implies (a(i) = old_a(n-i+1))}
Algebraic Specifications

- Make use of heterogeneous algebra
  - a collection of different sets on which several operations are defined
  - Traditional algebras are homogeneous, one set and a several operations; e.g. integers
  - Heterogeneous algebras contain multiple sets
    - e.g. length(ken) = 3
    - Here we have the set of strings and integers with one operation length defined

RAISE

- A Method and a Language
- Specification Language: RSL
- Specifications Refined in Levels
  - Associated consistency proof obligations
- Proofs of Properties Aided by Tools

Rigorous Approach to Industrial Software Engineering

Background Information

- In RAISE, they make use of a funny notion of the domain and range of a function
- Each function consists of a set of tuples. The domain is the set of elements that make up the first element of each tuple; the range is the set of elements that make up the second set of each tuple

Example

<table>
<thead>
<tr>
<th>S</th>
<th>Empty Set</th>
</tr>
</thead>
<tbody>
<tr>
<td>{}</td>
<td>S = {}</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>S</th>
<th>Domain</th>
<th>Range</th>
</tr>
</thead>
<tbody>
<tr>
<td>(1,2)</td>
<td>S = {1}</td>
<td>S = {2}</td>
</tr>
<tr>
<td>(1,2), (3,4)</td>
<td>S = {1, 3}</td>
<td>S = {2, 4}</td>
</tr>
<tr>
<td>(3, 4)</td>
<td>S = {3, 4}</td>
<td></td>
</tr>
</tbody>
</table>
RAISE Specification of POTS

* Plain Old Telephone Service

scheme POTS =
  class
    type
    value
  variable
RAISE Specification of POTS

scheme POTS =
  class
type Line,
  Status = Line \vec{m}\{On_Hook, Off_Hook\},
  Calls = Line \vec{m}\ Line

RAISE Specification of POTS

scheme POTS =
  class
type Line,
  Status = Line \vec{m}\{On_Hook, Off_Hook\},
  Calls = Line \vec{m}\ Line

RAISE Specification of POTS

scheme POTS =
  class
type Line,
  Status = Line \vec{m}\{On_Hook, Off_Hook\},
  Calls = Line \vec{m}\ Line

RAISE Specification of POTS

scheme POTS =
  class
type Line,
  Status = Line \vec{m}\{On_Hook, Off_Hook\},
  Calls = Line \vec{m}\ Line

RAISE Specification of POTS

scheme POTS =
  class
type Line,
  Status = Line \vec{m}\{On_Hook, Off_Hook\},
  Calls = Line \vec{m}\ Line
RAISE Specification of POTS

scheme POTS =
    class
        type Line,
            Status = Line \rightarrow \{On_Hook, Off_Hook\},
            Calls = Line \rightarrow Line
        value go_off_hook : Line \rightarrow Unit,

place_call : Line \times Line \rightarrow Bool,
scheme POTS =
  class
    type Line,
    Status = Line \rightarrow \{On_Hook, Off_Hook\},
    Calls = Line \rightarrow Line
    value go_off_hook : Line \rightarrow Unit,
    go_on_hook : Line \rightarrow Unit,
    place_call : Line \times Line \rightarrow Bool,
    end_call : Line \rightarrow Unit
  variable line_status : Status = [ L |-> On_Hook | L : Line ],
  active_calls : Calls = [ ]
RAISE Specification of POTS

axiom

forall L, L1, L2 : Line

\[ \text{go\_off\_hook}(L) \]

\[ \text{go\_on\_hook}(L) \]

\[ \text{place\_call}(L_1, L_2) \]

\[ \text{end\_call}(L) \]
RAISE Specification of POTS

axiom forall L, L_1, L_2 : Line
  \[ \text{go\_off\_hook}(L) \text{ post } \text{line\_status} = \text{line\_status}' \quad [L \rightarrow \text{Off\_Hook}], \]

\[ \text{go\_on\_hook}(L) \text{ post } \text{line\_status} = \text{line\_status}' \quad [L \rightarrow \text{On\_Hook}], \]

axiom forall L, L_1, L_2 : Line
  \[ \text{go\_off\_hook}(L) \text{ post } \text{line\_status} = \text{line\_status}' \quad [L \rightarrow \text{Off\_Hook}], \]

\[ \text{go\_on\_hook}(L) \]

place\_call(L_1, L_2) as S
RAISE Specification of POTS

axiom forall L, L1, L2 : Line ¥
go_off_hook(L) post line_status = line_status\[ L \rightarrow Off_Hook \],

go_on_hook(L) post line_status = line_status\[ L \rightarrow On_Hook \],

place_call(L1, L2) as S
post S \Rightarrow L_1 \neq L_2

∧ L_2 \notin dom active_calls\[ L_1 \rightarrow L_2 \]
RAISE Specification of POTS

axiom forall L, L1, L2 : Line ¥
   go_off_hook(L) post  line_status = line_status'  [ L |->
   Off_Hook ],

   go_on_hook(L) post  line_status = line_status'  [ L |->
   On_Hook ],

   place_call(L1, L2) as S
   post S ⇒ L1 ≠ L2 ∧ active_calls = active_calls'  [ L1 |-> L2 ]
   ∧ L2 ∉ dom active_calls' ∧ L2 ∉ rng active_calls'
   pre  line_status(L1) = Off_Hook
   ∧ L1 ∉ dom active_calls

February 17, 2000  ' Kenn eth M. Anderson, 2000 45
RAISE Specification of POTS

end_call(L)

post if \( L \in \text{dom active_calls} \)

then

else

end
RAISE Specification of POTS

end_call(L)
post if L ∈ dom active_calls
then active_calls = active_calls \ { L }
else

end

RAISE Specification of POTS

end_call(L)
post if L ∈ dom active_calls
then active_calls = active_calls \ { L }
else ∃ L₃ : Line ¥
active_calls (L₃) = L
active_calls = active_calls \ { L₃ }
end
end_call(L)
  post  if  L ∈ dom active_calls' 
    then active_calls = active_calls' \ { L }
    else ∃ L3 : Line ¥
      active_calls' (L3) = L ∧
      active_calls = active_calls' \ { L3 }
  end
  pre  L ∈ dom active_calls ∨ L ∈ rng active_calls