Lecture 19: Descriptive Specifications

Kenneth M. Anderson

Foundations of Software Engineering

CSCI 5828 - Spring Semester, 1999

Today's Lecture

- Introduce Descriptive Specifications
  - Axiomatic
  - Algebraic
  - Tour of the RAISE system
    - Developed in Denmark
    - Sold to European Manufacturing companies
    - Using RAISE to create these types of specifciations
      - Has a full tool suite

Descriptive Specifications

- Focuses on Properties, Not Behaviors
  - A declaration of a result, rather than a sequence of actions that produce the result
- Formalisms
  - Logical
  - Algebraic
- Mathematical Foundations
  - Predicate logic, set theory, abstract algebra

Logic Specifications

- Vocabulary of Logical Expressions
  - Variables, constants, predicates, functions
  - Connectives: and ($\wedge$), or ($\vee$), 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

- Additional Notes
  - Variables are either *free* or *bound*
  - 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
  - Modularize the specification
- Property Assertions
  - Preconditions and postconditions
  - Invariants

RAISE

Rigorous Approach to Industrial Software Engineering

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

RAISE Specification of POTS*

*Plain Old Telephone Service

RAISE Specification of POTS*

scheme POTS =

RAISE Specification of POTS*

scheme POTS =
  class
    type

    value

    variable

RAISE Specification of POTS*

scheme POTS =
  class
    type

RAISE Specification of POTS*

scheme POTS =
  class
    type Line,

RAISE Specification of POTS*

```
scheme POTS =
    class
        type    Line,
                Status = Line m→ {On_Hook, Off_Hook},
```


RAISE Specification of POTS*

```
scheme POTS =
    class
        type    Line,
                Status = Line m→ {On_Hook, Off_Hook},
                Calls = Line m→ Line
```


RAISE Specification of POTS*

```
scheme POTS =
    class
        type    Line,
                Status = Line m→ {On_Hook, Off_Hook},
                Calls = Line m→ Line
        value
```


RAISE Specification of POTS*

```
scheme POTS =
    class
        type    Line,
                Status = Line m→ {On_Hook, Off_Hook},
                Calls = Line m→ Line
        value   go_off_hook : Line → Unit,
```

RAISE Specification of POTS*

```
scheme POTS =
  class
    type    Line,
            Status = Line m→ {On_Hook, Off_Hook},
            Calls = Line m→ Line
    value   go_off_hook : Line → Unit,
            go_on_hook : Line → Unit,
```


RAISE Specification of POTS*

```
scheme POTS =
  class
    type    Line,
            Status = Line m→ {On_Hook, Off_Hook},
            Calls = Line m→ Line
    value   go_off_hook : Line → Unit,
            go_on_hook : Line → Unit,
            place_call : Line × Line → Bool,
```


RAISE Specification of POTS*

```
scheme POTS =
  class
    type    Line,
            Status = Line m→ {On_Hook, Off_Hook},
            Calls = Line m→ Line
    value   go_off_hook : Line → Unit,
            go_on_hook : Line → Unit,
            place_call : Line × Line → Bool,
            end_call : Line → Unit
```


RAISE Specification of POTS*

```
scheme POTS =
  class
    type    Line,
            Status = Line m→ {On_Hook, Off_Hook},
            Calls = Line m→ Line
    value   go_off_hook : Line → Unit,
            go_on_hook : Line → Unit,
            place_call : Line × Line → Bool,
            end_call : Line → Unit
    variable
```

RAISE Specification of POTS*

```
scheme POTS =
  class
    type      Line,
              Status = Line -m-> {On_Hook, Off_Hook},
              Calls = Line -m-> Line
    value     go_off_hook : Line → Unit,
              go_on_hook : Line → Unit,
              place_call : Line × Line → Bool,
              end_call : Line → Unit
    variable  line_status : Status = [ L ↦ On_Hook | L : Line ],
```


RAISE Specification of POTS*

```
scheme POTS =
  class
    type      Line,
              Status = Line -m-> {On_Hook, Off_Hook},
              Calls = Line -m-> Line
    value     go_off_hook : Line → Unit,
              go_on_hook : Line → Unit,
              place_call : Line × Line → Bool,
              end_call : Line → Unit
    variable  line_status : Status = [ L ↦ On_Hook | L : Line ],
              active_calls : Calls = [ ]
```


RAISE Specification of POTS

RAISE Specification of POTS

```
axiom
```

RAISE Specification of POTS

axiom forall L, L_1, L_2 : Line •

RAISE Specification of POTS

axiom forall L, L_1, L_2 : Line •
    go_off_hook(L)

    go_on_hook(L)

    place_call(L_1, L_2)

    end_call(L)

RAISE Specification of POTS

axiom forall L, L_1, L_2 : Line •
    go_off_hook(L)

RAISE Specification of POTS

axiom forall L, L_1, L_2 : Line •
    go_off_hook(L)` post line_status = line_status` † [L ↦ Off_Hook],

RAISE Specification of POTS

axiom forall L, L_1, L_2 : Line •
go_off_hook(L) post line_status = line_status` † [L ↦ Off_Hook],

go_on_hook(L)

RAISE Specification of POTS

axiom forall L, L_1, L_2 : 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],

RAISE Specification of POTS

axiom forall L, L_1, L_2 : 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(L_1, L_2) as S

RAISE Specification of POTS

axiom forall L, L_1, L_2 : 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(L_1, L_2) as S
post S ⇒ $L_1 \neq L_2$

RAISE Specification of POTS

axiom forall L, L_1, L_2 : 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(L_1, L_2) as S
post S ⇒ $L_1 \neq L_2$ ∧ active_calls = active_calls` † [L_1 ↦ L_2]

RAISE Specification of POTS

axiom forall L, L_1, L_2 : 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(L_1, L_2) as S
post S ⇒ $L_1 \neq L_2$ ∧ active_calls = active_calls` † [L_1 ↦ L_2]
∧ $L_2 \notin$ dom active_calls`

RAISE Specification of POTS

axiom forall L, L_1, L_2 : 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(L_1, L_2) as S
post S ⇒ $L_1 \neq L_2$ ∧ active_calls = active_calls` † [L_1 ↦ L_2]
∧ $L_2 \notin$ dom active_calls` ∧ $L_2 \notin$ rng active_calls`

RAISE Specification of POTS

axiom forall L, L_1, L_2 : 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(L_1, L_2) as S
post S ⇒ $L_1 \neq L_2$ ∧ active_calls = active_calls` † [L_1 ↦ L_2]
∧ $L_2 \notin$ dom active_calls` ∧ $L_2 \notin$ rng active_calls`
pre

RAISE Specification of POTS

axiom forall L, L_1, L_2 : 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(L_1, L_2) as S
post S ⇒ $L_1 \neq L_2$ ∧ active_calls = active_calls` † [L_1 ↦ L_2]
∧ $L_2 \notin$ dom active_calls` ∧ $L_2 \notin$ rng active_calls`
pre line_status(L_1) = Off_Hook

RAISE Specification of POTS

axiom forall L, L_1, L_2 : 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(L_1, L_2) as S
post S ⇒ $L_1 \neq L_2$ ∧ active_calls = active_calls` † [L_1 ↦ L_2]
∧ $L_2 \notin$ dom active_calls` ∧ $L_2 \notin$ rng active_calls`
pre line_status(L_1) = Off_Hook
∧ $L_1 \notin$ dom active_calls

RAISE Specification of POTS

axiom forall L, L_1, L_2 : 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(L_1, L_2) as S
post S ⇒ $L_1 \neq L_2$ ∧ active_calls = active_calls` † [L_1 ↦ L_2]
∧ $L_2 \notin$ dom active_calls` ∧ $L_2 \notin$ rng active_calls`
pre line_status(L_1) = Off_Hook
∧ $L_1 \notin$ dom active_calls ∧ $L_1 \notin$ rng active_calls,

RAISE Specification of POTS

RAISE Specification of POTS

end_call(L)

RAISE Specification of POTS

end_call(L)
    post

RAISE Specification of POTS

end_call(L)
    post if L ∈ 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 •

         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
         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
```


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
    pre
```

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
    pre   L ∈ dom active_calls
```


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
    pre   L ∈ dom active_calls  ∨  L ∈ rng active_calls
```


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
    pre   L ∈ dom active_calls  ∨  L ∈ rng active_calls
end
```