Lecture 11: Descriptive Specifications (Continued)

Kenneth M. Anderson

Foundations of Software Engineering

CSCI 5828 - Spring Semester, 2000

Today's Lecture

- Finish RAISE example
- Examine APP Language
- Examine Inscape Interface Language

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

Are These Theorems of POTS?

❶ $\neg \exists L_1, L_2, L_3 : \text{Line} \bullet$

$$\text{active\_calls}(L_1) = L_3 \wedge \text{active\_calls}(L_2) = L_3 \wedge L_1 \neq L_2$$

❷ $\forall L_1, L_2 : \text{Line} \bullet$

$$\text{place\_call}(L_1, L_2)$$

$$\text{post line\_status}(L_2) = \text{Off\_Hook}$$

Practical Logic Specifications

- Most Software Faults Occur at Interfaces
  - Typically, function boundaries
- Assertions Can Specify Interface Properties
  - Preconditions to calling a function
  - Postconditions of returning from a function
- Guaranteeing Logical Properties
  - Dynamic (run-time) assertion checking
  - Static (compile-time) theorem proving

Self-Checking Programs

An alternative to program proving

- Dynamic Analysis of Logic-Based Specifications
- Examples
  - Anna (ANNotated Ada)
  - APP (Annotation PreProcessor for C)
- Requires Sample Inputs for Analysis

APP Assertion Language

- Annotations as Structured Comments

  /*@ ... @*/
- Basic Constructs

  assume (precondition)

  promise (postcondition)

APP Specification

```
void print_warning(code, line, file)
int code;
int line;
char* file;

/*@
    assume warnings_on;
    promise warnings_on;
@*/

{ ... }
```

APP Assertion Language

- Quantification

  some (existential quantifier)

  all (universal quantifier)

APP Specification

```
#define BUFFSIZE 80
char buffer[BUFFSIZE];

void fill_and_truncate()

/*@
    promise some (int i = 0; i < BUFFSIZE; i++) buffer[i] == '\0';
@*/

{ ... }
```

APP Assertion Language

- Additional Constraints

  return (constraint on return value)

  assert (constraint on intermediate state)

APP Specification

```
int square_root(x)
int x;

/*@
    assume x >= 0;
    return y where y >= 0;
    return y where y*y <= x && x < (y+1)*(y+1);
@*/

{ ... }
```

APP Specification

```
int swap(x,y)
int *x,*y;

/*@
    assume x && y && x != y;
    promise *x == in *y && *y == in *x;
@*/

{   *x = *x + *y;
    *y = *x - *y;
         /*@ assert *y == in *x; @*/
    *x = *x - *y;  }
```

APP Assertion Language

- Additional Constraints
  - return (constraint on return value)
  - assert (constraint on intermediate state)

APP Specification of place_call()

APP Specification of place_call()

```
#define LINE_MAX 128
typedef long line;
line active_calls[LINE_MAX];

int place_call(L1, L2)
L1, L2 : line;

/*@ ... @*/     /* precondition and postcondition assertions */

{ ... }         /* C implementation code and state assertions */
```

APP Specification of place_call()

```
/*@
    2: assume line_status(L1) == off_hook
          &&  active_calls[L1] == 0
          &&  !some(line L3 = 1; L3 < LINE_MAX; L3++)
                       active_calls[L3] == L1;
    1: return S where !S
                 || (L1 != L2 && active_calls[L1] == L2
                      &&  !in some(line L3 = 1; L3 < LINE_MAX; L3++)
                                 active_calls[L3] == L2);
@*/
```

Running with Self-Checking

- Sample Diagnostic Output

```
promise violated: file pc.c, line 11, function place_call
```

- Checking-level Is Set at Runtime
  - No need for recompilation
  - ❶ Run at level 1
  - ❷ If violations, re-run at level 2 for more info
- Violation Actions Can Be Used to Customize and Enhance Diagnostics

Logic-based Construction

- Logic Used to “Build” Rather Than “Prove”
- Functions Specified Independently
- Condition Satisfaction Connects Functions
  - Preconditions and postconditions treated like hardware input and output pins
  - Postconditions *satisfy* some preconditions
  - Preconditions *depend on* some postconditions

Inscape Interface Language

- Base Terms
  - Function and parameter names
  - Predicates
- Preconditions and Postconditions
- Special Postcondition: *Obligation*
  - Postcondition that must *eventually* be satisfied

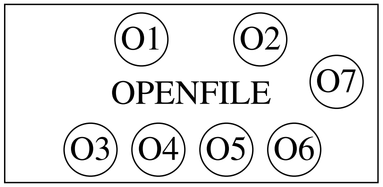
Graphical Notation

- Routines are presented as rectangles
  - each condition is numbered
    - preconditions are placed at the top
    - postconditions are placed at the bottom
    - obligations are placed on the right
  - conditions which “match” are connected
    - unmatched conditions are propagated to higher level entities

Inscape Specification

```
OpenFile (F, &FP)
   preconditions:
      LegalFileName(F)               O1
      FileExists(F)                  O2
   postconditions:
      LegalFileName(F)               O3
      FileExists(F)                  O4
      ValidFilePointer(FP)           O5
      FileOpen(FP)                   O6
   obligations:
      FileClosed(FP)                 O7
```

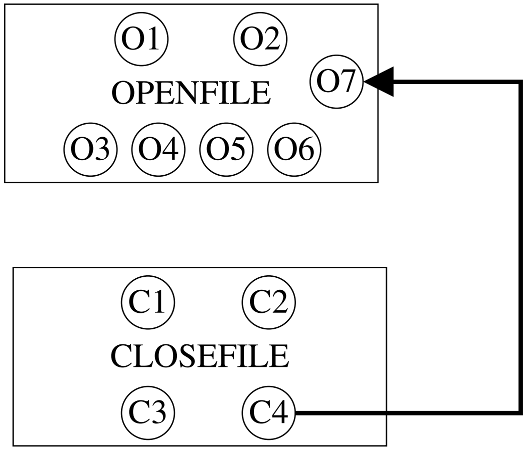
Example of Graphical Notation



Inscape Specification

```
CloseFile (&FP)
   preconditions:
      ValidFilePointer(FP)                 C1
      FileOpen(FP)                         C2
   postconditions:
      not(ValidFilePointer(out(FP)))       C3
      FileClosed(in(FP))                   C4
   obligations:
```

Example of Graphical Notation



Inscape Specification

ReadRecord (FP, R, &L,&Buffer)

preconditions:

ValidFilePointer(FP)	R1
FileOpen(FP)	R2
LegalRecordNumber(R)	R3
RecordExists(R)	R4
RecordReadable(R)	R5
RecordConsistent(R)	R6

Inscape Specification

ReadRecord (FP, R, &L,&Buffer)

postconditions:

ValidFilePointer(FP)	R7
FileOpen(FP)	R8
LegalRecordNumber(R)	R9
RecordExists(R)	R10
RecordReadable(R)	R11
RecordConsistent(R)	R12
Allocated(*Buffer)	R13
0 <= L <= Allocated(*Buffer)	R14
RecordIn(*Buffer)	R15

Inscape Specification

ReadRecord (FP, R, &L,&Buffer)

obligations:

Deallocated(*Buffer)	R16

Inscape Specification

ObtainRecord (FP, R, &L,&Buffer)

preconditions:

LegalFileName(F)	O1
FileExists(F)	O2
LegalRecordNumber(R)	R3
RecordExists(R)	R4
RecordReadable(R)	R5
RecordConsistent(R)	R6

Inscape Specification

ObtainRecord (FP, R, &L,&Buffer)

postconditions:

LegalFileName(F)	O3
FileExists(F)	O4
LegalRecordNumber(R)	R9
RecordExists(R)	R10
RecordReadable(R)	R11
RecordConsistent(R)	R12
Allocated(*Buffer)	R13
0 <= L <= Allocated(*Buffer)	R14
RecordIn(*Buffer)	R15

Inscape Specification

ObtainRecord (FP, R, &L,&Buffer)

obligations:

Deallocated(*Buffer)	R16