LECTURE 12: Z SPECIFICATIONS & THE SCHEMA CALCULUS

Software Engineering

Mike Wooldridge

1 The Truth About Schema Inclusion

- We saw last week how, a schema could be included by just listing its name in the declarations part of a schema. We now look at what this actually *means*.
- Suppose we had the following definition:

S_1
$v_1 : T_1$
$v_2 : T_2$

P_1
P_2

and later on

S_2
S_1 (* schema inclusion *)
$v_3 : T_3$

P_3

- Then this would have been equivalent to:

S_2
$v_1 : T_1$
$v_2 : T_2$
$v_3 : T_3$

P_1
P_2
P_3

- We now need to introduce *schema decoration*.
- Suppose we had the following declaration:

S_3
S_1'

P_4

then this declaration would have been equivalent to

S_3
$v_1' : T_1$
$v_2' : T_2$

$\left.\begin{array}{r} P_1 \\ P_2 \end{array}\right\}$ with all references to v_1, v_2 changed to v_1', v_2'.
P_4

- *Remember that the decorated form of a variable means "the variable* after *the operation has been performed"; the undecorated version means "the variable* before *the operation has been performed".*

- Let's now consider the Δ notation.
- Suppose we had:

S_4
ΔS_1

P_5

- This would have been equivalent to

S_4
S_1 (* include S_1 *)
S'_1 (* include S'_1 *)

P_5

- The Ξ notation means something similar. Suppose we had the schema:

S_5
ΞS_1

P_5

then this would expand to

S_5
S_1
S'_1

P_5
$v'_1 = v_1$
$v'_2 = v_2$

- So when we use the Ξ notation before a schema, it means "include the decorated and undecorated version of this schema, with the postcondition that all the variables remain unchanged."

2 The Schema Calculus

- One of the nice things about Z is that it allows us some sort of modular construction; we can build things in little pieces and put them together to make big pieces.
- The way we do this is by using the *schema calculus*.
- First we need to introduce *horizontal form schemas* (as opposed to the vertical form schemas we have been looking at so far).

- **Definition:** The following vertical-form schema

S
Declarations

P_1
P_2
$\ldots$
P_n

may be defined in the following horizontal form

$$S \mathrel{\widehat=} [Declarations \mid P_1;\ P_2;\ \cdots P_n]$$

- The symbol $\mathrel{\widehat=}$ is for schema definition; it may be read 'is defined to be'.
- Using $\mathrel{\widehat=}$, we can make one schema an alias for another:

$$NewPhoneBook \mathrel{\widehat=} PhoneBooks$$

- On the RHS of the $\mathrel{\widehat=}$ symbol can be any valid *schema calculus expression*.

- Such an expression may be a schema definition (as above); but we can also make new schemas using the propositional connectives $\wedge, \vee, \neg, \Rightarrow, \ldots$. Although these symbols are the same as in propositional logic, they have a different (but related) meaning.
- **Definition:** Two schemas are said to be type compatible if every variable common to both has the same type in both.
- We can use the connectives to make new schemas out of old ones only if they are type compatible. Let α be an arbitrary unary connective, β be an arbitrary binary connective, and S and T be the two schemas

  $$S \mathrel{\widehat{=}} [D_1;\ \cdots;\ D_m \mid P_1;\ \cdots;\ P_n]$$
  $$T \mathrel{\widehat{=}} [D_{m+1};\ \cdots;\ D_{m+p} \mid P_{n+1};\ \cdots;\ P_{n+q}]$$

  α S is the following schema

  $$[D_1;\ \cdots;\ D_m \mid \alpha(P_1 \wedge \cdots \wedge P_n)]$$

  If S and T are type compatible, then S β T is the following schema

  $$[D_1;\ \cdots;\ D_{m+p} \mid (P_1 \wedge \cdots \wedge P_n)\beta(P_{n+1} \wedge \cdots \wedge P_{n+q})]$$

- EXAMPLE: Specification of a robust 'Find' operation (i.e. one whose behaviour is defined even when the input name is not known).
- First define a schema which assigns the string 'okay' to a variable. This schema will be used to signify that an operation has been successful.

Success

$rep! : REPORT$

$rep! = \text{`}okay\text{'}$

- Then define a schema to capture the situation where a phone number is not in the database. Note that the schema causes an error message to be assigned to the report variable *rep!*.

NotKnown

$\Xi PhoneBook$
$name? : NAME$
$rep! : REPORT$

$name? \notin known$
$rep! = \text{`}name\ not\ known\text{'}$

- The robust 'Find' operation is

$$\begin{array}{l} \mathit{DoFindOp} \\ \mathrel{\widehat=} (\mathit{Find} \land \mathit{Success}) \lor \mathit{NotKnown} \end{array}$$

the full expansion of which is:

$$\begin{array}{|l} \hline \mathit{DoFindOp} \\ \mathit{known} : \mathbb{P}\,\mathit{NAME} \\ \mathit{known}' : \mathbb{P}\,\mathit{NAME} \\ \mathit{tel} : \mathit{NAME} \pfun \mathit{PHONE} \\ \mathit{tel}' : \mathit{NAME} \pfun \mathit{PHONE} \\ \mathit{name}? : \mathit{PHONE} \\ \mathit{phone}! : \mathit{PHONE} \\ \mathit{rep}! : \mathit{REPORT} \\ \hline ((\mathit{dom}\ \mathit{tel} = \mathit{known} \land \mathit{dom}\ \mathit{tel}' = \mathit{known} \\ \land\ \mathit{known}' = \mathit{known} \land \mathit{tel}' = \mathit{tel} \\ \land\ \mathit{name}? \in \mathit{known} \\ \land\ \mathit{phone}! = \mathit{tel}(\mathit{name}?)) \\ \land\ \mathit{rep}! = \text{`}\mathit{okay}\text{'}) \\ \lor \\ (\ \mathit{dom}\ \mathit{tel} = \mathit{known} \land \mathit{dom}\ \mathit{tel}' = \mathit{known} \\ \land\ \mathit{known}' = \mathit{known} \land \mathit{tel}' = \mathit{tel} \\ \land\ \mathit{name}? \notin \mathit{known} \\ \land\ \mathit{rep}! = \text{`}\mathit{name\ not\ known}\text{'}) \\ \hline \end{array}$$

- After logical simplification, the expanded schema becomes:

$$\begin{array}{|l} \hline \mathit{DoFindOp} \\ \mathit{known} : \mathbb{P}\,\mathit{NAME} \\ \mathit{known}' : \mathbb{P}\,\mathit{NAME} \\ \mathit{tel} : \mathit{NAME} \pfun \mathit{PHONE} \\ \mathit{tel}' : \mathit{NAME} \pfun \mathit{PHONE} \\ \mathit{name}? : \mathit{PHONE} \\ \mathit{phone}! : \mathit{PHONE} \\ \mathit{rep}! : \mathit{REPORT} \\ \hline \mathit{dom}\ \mathit{tel} = \mathit{known} \\ \land\ \mathit{known}' = \mathit{known} \land \mathit{tel}' = \mathit{tel} \\ \land\ ((\mathit{name}? \in \mathit{known} \\ \quad \land\ \mathit{phone}! = \mathit{tel}(\mathit{name}?) \\ \quad \land\ \mathit{rep}! = \text{`}\mathit{okay}\text{'}) \\ \quad \lor \\ \quad (\ \mathit{name}? \notin \mathit{known} \\ \quad \land\ \mathit{rep}! = \text{`}\mathit{name\ not\ known}\text{'})) \\ \hline \end{array}$$

Things to Note

- The use of abstraction: The derived version of *DoFindOp* is easier to read and understand than the expanded version!
- The behaviour of the system is now rigorously specified. For instance, we could prove that, when the precondition of the find operation is satisfied, then a phone number is found.
- Notice that the value of the variable *phone*! is undefined when the operation fails.