Values, Types, Variables, Operators

Hugh Darwen
hugh@dcs.warwick.ac.uk
www.dcs.warwick.ac.uk/~hugh

Anatomy of an Imperative

Example: Y := X + 1 ;

Denotes a variable

Denotes current value of a variable

A read-only operator

An update operator

X and 1 are arguments to the invocation of +

Y and X+1 are arguments to the invocation of :=

Important Distinctions

The following very important distinctions emerge from all this and should be firmly taken on board:

• Value versus variable
• Variable versus variable reference
• Update operator versus read-only operator
• Operator versus invocation
• Parameter versus argument
• Parameter subject to update versus parameter not subject to update

A Closer Look at an Operator (+)

Look – it’s a relation!

The predicate: a + b = c

Attributes a and b can be considered as the parameters of +.

It’s also a function: no two tuples with the same a value also have the same b value, so, given an a and a b, we know the c.

An Operator Definition

In Tutorial D:

OPERATOR HIGHER_OF ( A INTEGER, B INTEGER )
RETURNS INTEGER ;
IF A > B THEN RETURN A ;
ELSE RETURN B ;
END IF ;
END OPERATOR ;

So the invocation HIGHER_OF(2,3) = 3

What Is a Type?

A type (= “domain”) is a named set of values.

Examples:

WEEKDAY:
{ Monday, Tuesday, Wednesday, Thursday, Friday,
Saturday, Sunday }

INTEGER:
{ ..., -7, -6, -5, -4, -3, -2, -1, 0, 1, 2, 3, 4, 5, 6, 7, ... }
What is a Type For?

It is for constraining the values permitted for some purpose.

For example, constraining:
- the values that can be assigned to a variable
- the values that can be substituted for a parameter
- the values that an operator can yield when invoked
- the values that can appear for a given attribute of a relation

The declared type (of the variable, parameter, operator or attribute) constrains its possible values to be of that type.

What is the Type of This?

<table>
<thead>
<tr>
<th>StudentId</th>
<th>Name</th>
<th>CourseId</th>
</tr>
</thead>
<tbody>
<tr>
<td>S1</td>
<td>Anne</td>
<td>C1</td>
</tr>
<tr>
<td>S1</td>
<td>Anne</td>
<td>C2</td>
</tr>
<tr>
<td>S2</td>
<td>Boris</td>
<td>C1</td>
</tr>
<tr>
<td>S3</td>
<td>Cindy</td>
<td>C3</td>
</tr>
<tr>
<td>S4</td>
<td>Devinder</td>
<td>C1</td>
</tr>
</tbody>
</table>

Perhaps RELATION { StudentId SID, Name NAME, CourseId CID } where SID is the declared type of StudentId, NAME that of Name, and CID that of CourseId.

How to Write This as a Literal?

<table>
<thead>
<tr>
<th>StudentId</th>
<th>Name</th>
<th>CourseId</th>
</tr>
</thead>
<tbody>
<tr>
<td>S1</td>
<td>Anne</td>
<td>C1</td>
</tr>
<tr>
<td>S1</td>
<td>Anne</td>
<td>C2</td>
</tr>
<tr>
<td>S2</td>
<td>Boris</td>
<td>C1</td>
</tr>
<tr>
<td>S3</td>
<td>Cindy</td>
<td>C3</td>
</tr>
<tr>
<td>S4</td>
<td>Devinder</td>
<td>C1</td>
</tr>
</tbody>
</table>

See next slide …

A Relation Literal in Tutorial D

Try:

RELATION {
  TUPLE { StudentId S1, CourseId C1, Name Anne },
  TUPLE { StudentId S1, CourseId C2, Name Anne },
  TUPLE { StudentId S2, CourseId C1, Name Boris },
  TUPLE { StudentId S3, CourseId C3, Name Cindy },
  TUPLE { StudentId S4, CourseId C1, Name Devinder }
}

But this assumes that S1, C1, and Boris are themselves valid literals. They aren’t in most languages, including Tutorial D.

A Tuple Literal

From our first try:

TUPLE { StudentId S1, CourseId C1, Name Anne }

We can now write this as:

TUPLE {
  StudentId SID (‘S1’),
  CourseId CID (‘C1’),
  Name NAME (‘Anne’)
}

...and that’s just fine. (See the lecture Notes for the relation literal.)

Literals for Student Ids, etc

Recall the declared types of the attributes: SID, NAME, CID.
Suppose that values of type SID are represented by character strings (values of type CHAR).
Suppose that character strings are denoted by text in quotes, like this: ‘S1’.
Then a literal for the student identifier S1 might be: SID (‘S1’)
SID (‘S1’) is an invocation of an operator called SID whose single parameter is of type CHAR.
We call SID a selector, because it can be used to “select” any value of type SID.
Types and Representations

Consider the invocation SID (‘S1’), a literal of type SID.

SID is an operator that, when invoked with a suitable character string, returns a value of type SID.

Also, every value of type SID can be denoted by an invocation of operator SID.

We call such an operator a selector (for values of the type in question).

And the parameters (“signature”) of a selector we call a possible representation (possrep for short).

A Type Definition for SID

In Tutorial D:

```
TYPE SID POSSREP SID { C CHARCONSTRAINT constraint to specify exactly which strings are suitable } ;
```

The definition implies operator definitions:

```
OPERATOR SID ( C CHAR ) RETURNS SID ;
OPERATOR THE_C ( S SID ) RETURNS CHAR ;
```

whereby, e.g., THE_C ( SID (‘S1’) ) = ‘S1’

Type Constraint for SID

So, we can write the required type constraint something like this:

```
CONSTRAINT LENGTH ( C ) <= 5 AND
SUBSTRING ( C, 0, 1) = ‘S’ AND
IS_DIGITS (SUBSTRING ( C, 1 ))
```

Where:

- `SUBSTRING ( s, pos, l )` returns the substring specified by start position `pos` (base 0) and length `l` (or the remainder of `s` if `l` omitted), and
- `IS_DIGITS(s)` returns TRUE if every character of `s` is a digit, otherwise FALSE.

Defining a Subtype

An alternative method for defining a type is available when the required values are a subset of an existing type. For example, positive integers:

```
TYPE POSINT IS { INTEGER CONSTRAINT INTEGER > 0 } ;
```

Now:

- Attributes of type POSINT admit positive values only
- All operators on integers are available
- In fact, an expression of type POSINT is permitted wherever one of type INTEGER is permitted (substitutability)

But “type inheritance” is a big subject, beyond the scope of CS252.

What Is a Variable?

Here is a variable declaration in Tutorial D:

```
VAR SN SID INIT SID (‘S1’ ) ;
```

So a variable has a name, a declared type, and a value.

The value can change from time to time. The name and type cannot.

Updating a Variable

A value is assigned to a variable by invoking an update operator.

E.g., assignment (available on variables of all types):

```
SN := SID (‘S2’) ;
SN := SID ( LEFT ( THE_C (SN), 1 ) || ‘5’ ) ;
```

Additional update operators might be defined, invoked via CALL:

```
CALL SET_DIGITS ( SN , 23 ) ;
```

Pseudovariable assignment might be supported:

```
THE_C ( SN ) := ‘S2’ ;
SUBSTR (THE_C ( SN ), 2 ) := ‘23’ ;
```
You should now be able to distinguish clearly between:

- values and variables
- values and representations of values
- types and representations
- read-only operators and update operators
- operators and invocations
- parameters and arguments

EXERCISES
(see Notes)