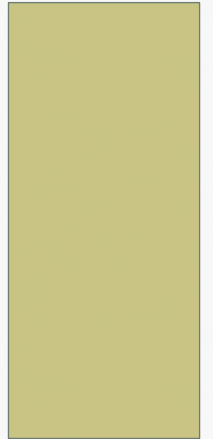


# IMPLEMENTING COMPONENTS IN RESOLVE

STEPHEN SCHAUB



# REALIZING CONCEPTS

- A Realization is an implementation of a Concept

```
Realization LightBulb_Bool_Realiz  
  for LightBulb_Template;  
  ...  
end;
```

- Realization must implement ...
  - Types
  - Operations... specified by its Concept
- Example: LightBulb\_Bool\_Realiz

# IMPLEMENTING TYPES

- Concept defines a type in terms of a math type

```
Concept LightBulb_Template;
```

```
Type Family LightBulb is modeled by B;
```

- Realization implements the type in terms of a Resolve type

```
Realization LightBulb_Bool_Realiz
```

```
for LightBulb_Template;
```

```
Type LightBulb is represented by Boolean;
```

# TYPE CONSTRAINTS

- Concept uses **constraint** clause

```
Concept LightBulb_N_Template;
```

```
    Type Family LightBulb is modeled by N;
```

```
    exemplar some_bulb;
```

```
    constraint some_bulb = 0 or some_bulb = 1;
```

- Realization uses **convention** clause

```
Realization LightBulb_N_Realiz
```

```
    for LightBulb_N_Template;
```

```
    Type LightBulb is represented by Integer;
```

```
    convention some_bulb = 0 or some_bulb = 1;
```

# TYPE INITIALIZATION

- Concept uses **initialization ensures** clause

```
Concept LightBulb_Template;  
  Type Family LightBulb is modeled by B;  
    exemplar some_bulb;  
    initialization ensures some_bulb = false;
```

- Realization uses **initialization** block

```
Realization LightBulb_Bool_Realiz  
  for LightBulb_Template;  
    Type LightBulb is represented by Boolean;  
    initialization  
      some_bulb := False();  
    end;
```

# NAME REFERENCES

- Realization refers to names defined in Concept

```
Concept Simple_Stack_Template(  
    type Entry; evaluates Max_Depth: Integer);  
  
Type Family Stack is modeled by Str(Entry);  
exemplar S;  
  
Realization Simple_Stack_Array_Realiz  
for Simple_Stack_Template;  
  
Type Stack is represented by Record  
    Contents: Array 1..Max_Depth of Entry;  
    Top: Integer;  
end;  
convention 0 <= S.Top <= Max_Depth;
```

The diagram illustrates name references in the provided code. Red circles highlight the following identifiers: `Entry` in the concept signature, `Stack` in the type family definition, `Entry` in the realization signature, and `Entry` in the record definition. A red arrow points from the first `Entry` to the second. A green arrow points from `Max_Depth` in the concept signature to `Max_Depth` in the record definition. A yellow arrow points from `Stack` in the type family definition to `Stack` in the realization signature. A grey arrow points from `S` in the exemplar declaration to `S` in the convention line.

# IMPLEMENTING OPERATIONS

- Concept specifies Operations

```
Concept Simple_Stack_Template(...);
```

```
Operation Push(alters E: Entry; updates S: Stack);
```

- Realization implements the operations

```
Realization Simple_Stack_Array_Realiz  
for Simple_Stack_Template;
```

```
Procedure Push(alters E: Entry; updates S: Stack);  
    S.Top := S.Top + 1;  
    S.Contents[S.Top] :=: E;  
end Push;
```

# IMPLEMENTATION FLEXIBILITY

- The mathematical model specified in the Concept can be implemented using any sufficiently capable Resolve type
  - Does not have to “match” the math type
- Example: LightBulb\_Int\_Realiz
  - Realizes a boolean-modeled light bulb using an Integer



# SOMETHING IS MISSING

- Our specification system is nearly complete
- Need a way to specify how Concept Type values map to Realization Type values
  - Consider LightBulb\_Template
    - Which value maps to on? off? How do you know?
  - Consider LightBulb\_Int\_Realiz
    - Which value maps to on? off? How do you know?
- This specification is needed by
  - Developers writing the Realization
  - Resolve's Verification System

# BASIC CORRESPONDENCE

When Realization values match Concept values, correspondence is simple equivalence

```
Concept LightBulb_Template;
```

```
  Type Family LightBulb is modeled by B;
```

```
    exemplar some_bulb;
```

```
Realization LightBulb_Bool_Realiz
```

```
    for LightBulb_Template;
```

```
  Type LightBulb is represented by Boolean;
```

```
    correspondence
```

```
      Conc.some_bulb = some_bulb;
```

Concept value



Realization value



# MAPPING CORRESPONDENCE

When Realization values do not match Concept values, write a formal math expression to map a Realization value to a Concept value.

```
Concept LightBulb_Template;
```

```
  Type Family LightBulb is modeled by B;
```

```
    exemplar some_bulb;
```

```
Realization LightBulb_Int_Realiz
```

```
  for LightBulb_Template;
```

```
  Type LightBulb is represented by Integer;
```

```
  correspondence
```

```
    Conc.some_bulb = (some_bulb = 1);
```

Concept value

Realization value

Concept	Realiz
false	0
true	1

# STACK EXAMPLE

- Stack Concept

- Represents stack using a math string

Type Family Stack is modeled by `Str(Entry);`

- Stack Realization

- Represents stack using an array and an index

Type Stack is represented by Record

Contents: Array 1..Max\_Depth of Entry;

Top: Integer;

end;

- Need to map array contents to Math string

# CONCATENATION

- Concatenation in Formal Math Notation

$$\prod_{i=start\_index}^{end\_index} < arrayname(i) >$$

- Concatenation in Resolve Math Notation

Concatenation *i*: Integer

where *start\_index* ≤ *i* ≤ *end\_index*,  
< *arrayname*(*i*) >

# STACK CORRESPONDENCE

Type Stack is represented by Record

Contents: Array 1..Max\_Depth of Entry;

Top: Integer;

end;

convention

$0 \leq S.Top \leq Max\_Depth;$

correspondence

Conc.S = Concatenation  $i: Integer$

where  $\underline{\hspace{2cm}} \leq i \leq \underline{\hspace{2cm}},$

$< \underline{\hspace{2cm}}(i) >;$

# VALID CORRESPONDENCE?

- Must check that
  - Correspondence holds for initial value
  - Operation implementations preserve correspondence
- Let's do some examples

```
Operation Push(alters E: Entry; updates S: Stack);  
  requires |S| + 1 <= Max_Depth;  
  ensures  S = #S o <#E>;
```

```
Operation Pop(replaces R: Entry; updates S: Stack);  
  requires |S| /= 0;  
  ensures #S = S o <R>;
```

# EXAMPLES

Statement	Conceptual Stack	Realized Stack
	< >	S.Top = 0 S.Contents = [ 0   0   0   0   0 ]
I := 5; Push(I, S);	< 5 >	S.Top = 1 S.Contents = [ 5   0   0   0   0 ]
I := 10; Push(I, S);	< 5, 10 >	S.Top = 2 S.Contents = [ 5   10   0   0   0 ]
I := 15; Push(I, S);	< 5, 10, 15 >	S.Top = 3 S.Contents = [ 5   10   15   0   0 ]
Pop(I, S);	< 5, 10 >	S.Top = 2 S.Contents = [ 5   10   ?   0   0 ]



# ALTERNATE SPEC

Consider this Stack spec:

```
Operation Push(alters E: Entry; updates S: Stack);  
  requires |S| + 1 <= Max_Depth;  
  ensures  S = <E> o #S;
```

```
Operation Pop(replaces R: Entry; updates S: Stack);  
  requires |S| /= 0;  
  ensures #S = <R> o S;
```

What needs to change in the Realization?

- Correspondence?
- Procedure implementations?

# EXAMPLES

Statement	Conceptual Stack	Realized Stack
	< >	S.Top = 0 S.Contents = [ 0   0   0   0   0 ]
I := 5; Push(I, S);	< 5 >	S.Top = 1 S.Contents = [ 5   0   0   0   0 ]
I := 10; Push(I, S);	< 10, 5 >	S.Top = 2 S.Contents = [ 5   10   0   0   0 ]
I := 15; Push(I, S);	< 15, 10, 5 >	S.Top = 3 S.Contents = [ 5   10   15   0   0 ]
Pop(I, S);	< 10, 5 >	S.Top = 2 S.Contents = [ 5   10   ?   0   0 ]

# ALTERNATE CORRESPONDENCE

Type Stack is represented by Record

Contents: Array 1..Max\_Depth of Entry;

Top: Integer;

end;

convention

$0 \leq S.Top \leq Max\_Depth;$

correspondence

$Conc.S = \mathbf{Reverse}(\text{Concatenation } i: \text{Integer}$

where  $1 \leq i \leq S.Top,$

$\langle S.Contents(i) \rangle);$

# EXERCISE

Modify `Stack_Array_Realiz` to use an array with indices from `0..MAX_DEPTH-1`