SFV Talk:
Part II: Annotation Command Programming in Isabelle

Frédéric Tuong

Trinity College Dublin

March 19, 2020
theory Example
imports "~/src/HOL/Multivariate_Analysis/ex/Approximations"
begin

lemma assumes "0 ≤ a ∧ a ≤ b ∧ b ≤ 4"
assumes "sin (a / 6) ≤ 1 / 2 ∧ 1 / 2 ≤ sin (b / 6)"
shows "a ≤ pi ∧ pi ≤ b"
using assms sin_pi6_straddle
by blast

end
theory Example
imports "~/src/HOL/Multivariate_Analysis/ex/Approximations"
begin

lemma assumes "0 ≤ a ∧ a ≤ b ∧ b ≤ 4"
assumes "sin (a / 6) ≤ 1 / 2 ∧ 1 / 2 ≤ sin (b / 6)"
shows "a ≤ pi ∧ pi ≤ b"
using assms sin_pi6_straddle
by blast

end
theory Example
imports "~/src/HOL/Multivariate_Analysis/ex/Approximations"

begin

lemma assumes "0 ≤ a ∧ a ≤ b ∧ b ≤ 4"
assumes "sin (a / 6) ≤ 1/2 ∧ 1/2 ≤ sin (b / 6)"
shows "a ≤ π ∧ π ≤ b"
using assms sin_pi6_straddle
by blast

end
Isabelle/C supporting CPP-like languages

```isar
(* Regular Isar (annotation) commands *)

东路

(* Regular Isar (annotation) commands *)

东路

(* Regular Isar (annotation) commands *)

东路

(* Regular Isar (annotation) commands *)

东路

(* Regular Isar (annotation) commands *)

东路

(* Regular Isar (annotation) commands *)

东路

(* Regular Isar (annotation) commands *)

东路

(* Regular Isar (annotation) commands *)

东路

(* Regular Isar (annotation) commands *)

东路

(* Regular Isar (annotation) commands *)

东路

(* Regular Isar (annotation) commands *)

东路
```

Any CPP-like language:
- C
- C++
- ACSL
- USE/OCL
- Java
- Promela
- printed output of spin trail file
- ...
Isabelle/C: the art of environment propagation

Any CPP-like language:
- C
- C++
- ACSL
- USE/OCL
- Java
- Promela
- printed output of spin trail file
- ...

```isabelle
example theory imports "~/src/HOL/Multivariate_Analysis/ex/Approximations"

begin

lemma assumes
0 ≤ a ∧ a ≤ b ∧ b ≤ 4
assumes sin (a/6) ≤ 1/2  1/2 ≤ sin (b/6)
shows a ≤ π ∧ π ≤ b
using assms sin_pi6_straddle
by blast

term arc_cos
definition arc_cos :: complex ⇒ complex
where" arc_cos λ ≡ λ ∨ i* Ln(z + i* csqrt(1 - z^2))

term "arc_cos 1 = 0"
find_theorems

end
```
Annotations in CPP-like Languages: A solved Problems?

- Semantics of an Annotation Command, Problems:

  - Resolving navigation ambiguity in annotations
    ```c
    int i = 123;               /*@ annotation i??? */
    for (int i = 0; i < n; i++) a+= a*i /*@ annotation i??? */
    ```
    How to prove that (the outer) i was equal to 123 while being in the loop?
    How to prove that (the inner) i will be set to 0 before entering the loop?
    How to prove that the outer i has a value different than the inner i?

  - Referring to C variables while in Isabelle propositions
    ```c
   /*@ assert(a >= i)*/
    ```

  - Dealing with the evaluation order of annotations
    ```c
   /*@ annotation_begin */ ... /*@ annotation_end */
    vs.
    /*@ annotation_end */ ... /*@ annotation_begin */
    (why not, like Haskell toplevel where definitions can be permuted)

- Propagating the Isabelle logical context during annotation evaluations
  (scheduling with directives, which must be all evaluated before parsing)
HOOKING UP BACK-ENDS

- General Mechanism to register a PIDE "command":

  \[ \text{Outer\_Syntax.command'}: K_{cmd} \to (\sigma \to \sigma) \text{ parser} \to \sigma \to \sigma \]

- Isabelle/Isar: "setup \langle\text{some SML function of type} : \sigma \to \sigma \rangle "
a shorthand for:
  "ML \langle\text{Theory.setu}p \text{ (some SML function of type} : \sigma \to \sigma \rangle \rangle "

```plaintext
definition "[...]"
datatype LIST = NIL | CONS nat LIST
fun height :: "LIST \Rightarrow nat"
  where "height NIL = 0"
        "height (CONS _ t) = Suc (height t)"
```

HOOKING UP BACK-ENDS

• General Mechanism to register a PIDE “command”:

\[ \text{Outer_Syntax.command'}: K_{\text{cmd}} \rightarrow (\sigma \rightarrow \sigma) \text{ parser } \rightarrow \sigma \rightarrow \sigma \]

• Isabelle/Isar: “setup \{ some SML function of type : \sigma \rightarrow \sigma \} “

 a shorthand for:

\[ \text{ML} \{ \text{Theory.setup (some SML function of type : \sigma \rightarrow \sigma)} \} \]

\[ \text{definition} \]

\[ \text{datatype} \]

\[ \text{fun} \]

\[ \text{ML} \{ \text{Theory.setup (} \text{(*definition ...*) } \} \}

\[ \text{ML} \{ \text{Theory.setup (} \text{(*datatype ...*) } \} \}

\[ \text{ML} \{ \text{Theory.setup (} \text{(*fun ...*) } \} \]
HOOKING UP BACK-ENDS

- General Mechanism to register a PIDE “command”:
  
  Outer_Syntax.command’: $K_{cmd} \rightarrow (\sigma \rightarrow \sigma) \text{ parser} \rightarrow \sigma \rightarrow \sigma$

- Isabelle/Isar: “setup ‹ some SML function of type : } \sigma \rightarrow \sigma ‹”

- Analogously, Isabelle/C provides an infrastructure to define “Annotation Commands”

  C_Annotation.command : $K_{cmd} \rightarrow (<n\text{-expr}> \rightarrow (\sigma \rightarrow \sigma) c\text{_parser}) \rightarrow \text{ unit}$
  
  C_Annotation.command’: $K_{cmd} \rightarrow (<n\text{-expr}> \rightarrow (\sigma \rightarrow \sigma) c\text{_parser}) \rightarrow \sigma \rightarrow \sigma$

- ... $\sigma$ is the logical context of the Isabelle system

- ... comprising in Isabelle/C an environment env and a stack of current parsed Shift-Reduce- AST’s