

SFV Talk:

Part II: Annotation Command Programming in Isabelle

Frédéric Tuong

Trinity College Dublin

March 19, 2020

Isabelle – The System

File Edit Search Markers Folding View Utilities Macros Plugins Help

Example.thy (~/2016/)

☒ Continuous checking
Prover: ready

Example.thy (~/2016/)

```
theory Example
imports "~~/src/HOL/Multivariate_Analysis/ex/Approximations"
      Language1 Language2 Language3
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

Language 1

term arc_cos

definition arc_cos :: "complex ⇒ complex" where
  "arc_cos ≡ λz. -i * Ln(z + i * csqrt(1 - z2))"

Language 2

term "arc_cos 1 = 0"

Language 3

find_theorems

end
```

Documentation Sidekick Theories

Default (HOL)

- ☐ Old_Datatype
- ☐ Phantom_Type
- ☐ Cardinality
- ☐ Numeral_Type
- ☐ Product_plus
- ☐ Product_Vector
- ☐ Convex
- ☐ Set_Algebras
- ☐ L2_Norm
- ☐ Euclidean_Space
- ☐ Linear_Algebra
- ☐ Finite_Cartesian_Product
- ☐ Countable
- ☐ Countable_Set
- ☐ Topology_Euclidean_Space
- ☐ Convex_Euclidean_Space
- ☐ Brouwer_Fixpoint
- ☐ Derivative
- ☐ Integration
- ☐ Cartesian_Euclidean_Space
- ☐ Complex_Analysis_Basics
- ☐ Complex_Transcendental
- ☐ Approximations
- ☐ Example

Output Query Sledgehammer Symbols

Isabelle/C: annotations as comments

```
/*@
theory Example
imports "~~/src/HOL/Multivariate_Analysis/ex/Approximations"
      Language1 Language2 Language3
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

*/
Language 1

term arc_cos

definition arc_cos :: "complex ⇒ complex" where
  "arc_cos ≡ λz. -i * Ln(z + i * csqrt(1 - z2))"

*/
Language 2

term "arc_cos 1 = 0"

*/
Language 3

find_theorems

end */
```

Isabelle/C: annotations as comments

```
/*@  
theory Example  
imports "~~/src/HOL/Multivariate_Analysis/ex/Approximations"  
        Language1 Language2 Language3  
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
```

```
*/
```

Language 1

```
/*@
```

```
term arc_cos
```

```
definition arc_cos :: "complex ⇒ complex" where  
  "arc_cos ≡ λz. -i * Ln(z + i * csqrt(1 - z2))"
```

```
*/
```

Language 2

```
/*@
```

```
term "arc_cos 1 = 0"
```

```
*/
```

Language 3

```
/*@
```

```
find_theorems
```

```
end */
```

Isabelle/C supporting CPP-like languages

`/*@`

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

`*/`

Language 1

`/*@`

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

`*/`

Language 2

`/*@`

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

`*/`

Language 3

`/*@`

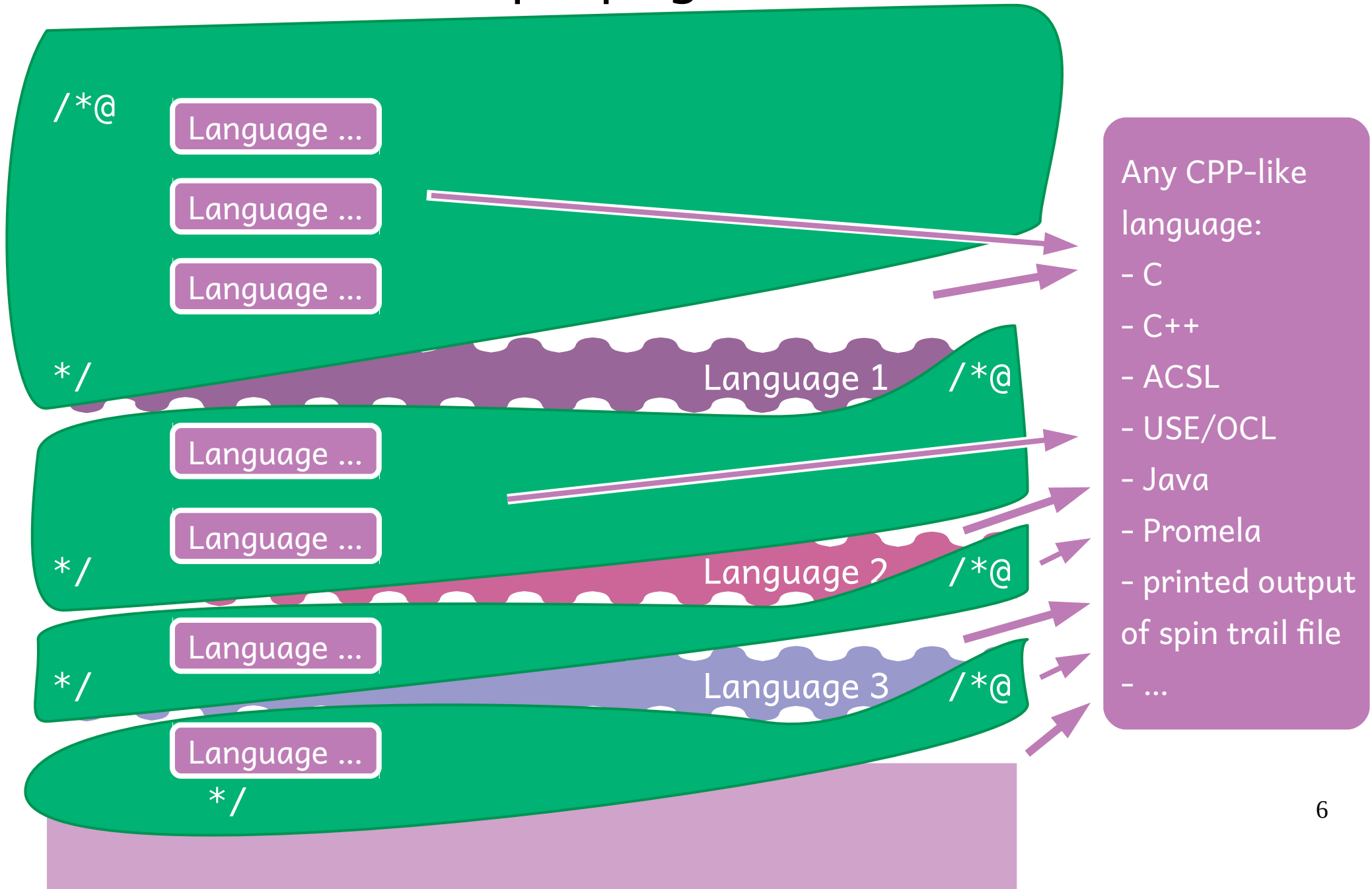
`(* Regular Isar (annotation) commands *)`

`*/`

Any CPP-like language:

- C
- C++
- ACSL
- USE/OCL
- Java
- Promela
- printed output of spin trail file
- ...

Isabelle/C: the art of environment propagation



Annotations in CPP-like Languages: A solved Problems?

- Semantics of an Annotation Command, Problems:

- Resolving navigation ambiguity in annotations

```
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

```
/*@ assert(a > i) */
```

- Dealing with the evaluation order of annotations

```
/*@ 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”:

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

- Isabelle/Isar : “setup < some SML function of type : $\sigma \rightarrow \sigma$ > “
a shorthand for:
“ML < Theory.setup (some SML function of type : $\sigma \rightarrow \sigma$) > “

definition “[...]”

datatype LIST = NIL | CONS nat LIST

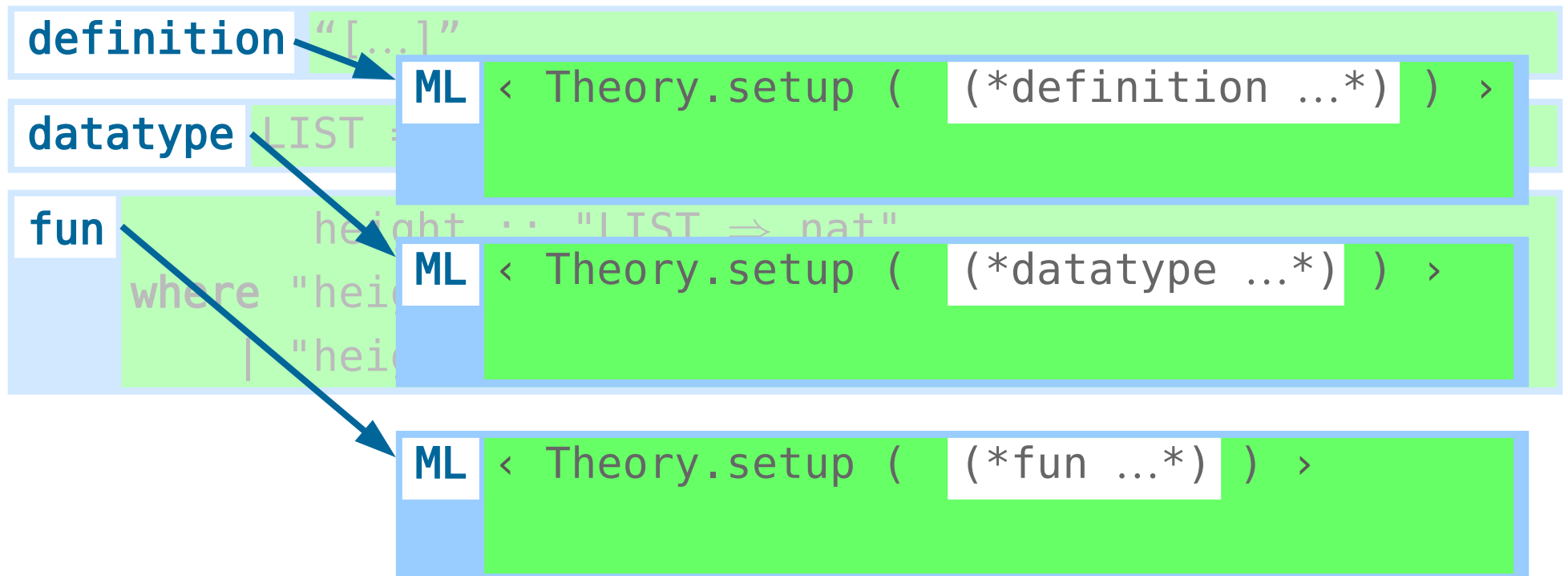
fun height :: “LIST $\Rightarrow$ nat”
where “height NIL = 0”
| “height (CONS _ t) = Suc (height t)”

HOOKING UP BACK-ENDS

- General Mechanism to register a PIDE “command”:

Outer_Syntax.command': $K_{cmd} \rightarrow (\sigma \rightarrow \sigma)$ parser $\rightarrow \sigma \rightarrow \sigma$

- Isabelle/Isar : “setup < some SML function of type : $\sigma \rightarrow \sigma$ > “
a shorthand for:
“ML < Theory.setup (some SML function of type : $\sigma \rightarrow \sigma$) > “



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 : $K_{cmd} \rightarrow (<n\text{-expr}> \rightarrow (\sigma \rightarrow \sigma) \text{c_parser}) \rightarrow \text{unit}$

C_Annotation.command': $K_{cmd} \rightarrow (<n\text{-expr}> \rightarrow (\sigma \rightarrow \sigma) \text{c_parser}) \rightarrow \sigma \rightarrow \sigma$

- ... σ 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

DEMO