What is a Proof in Isabelle/HOL ?

Burkhart Wolff
LMF, Université Paris-Saclay

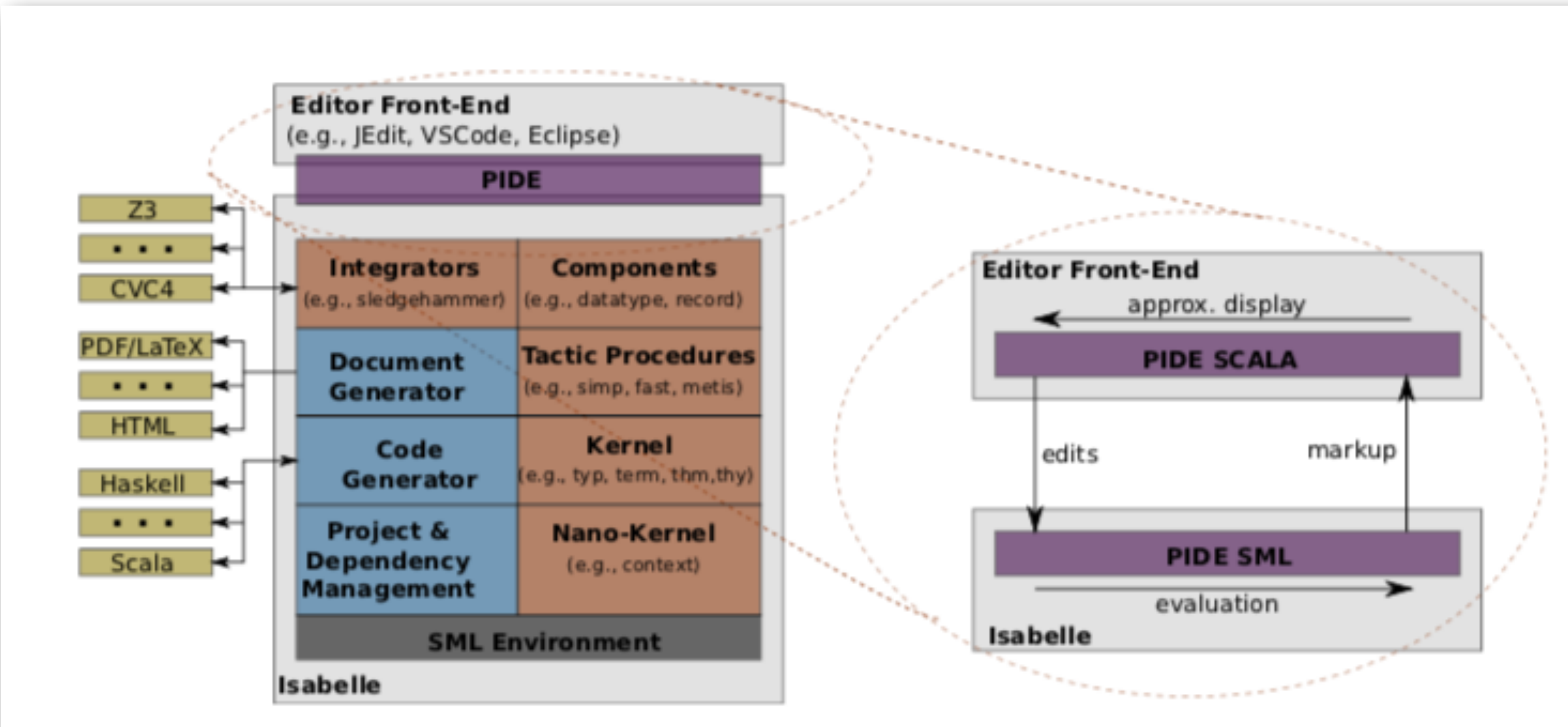
LMF Seminary @ ENS Ulm, 24 Nov 2021

Abstract

I give a System-oriented talk for mathematicians and computer-scientist on system architecture, its links to theoretical foundations and the basic pragmatics of the Isabelle interactive proof system.

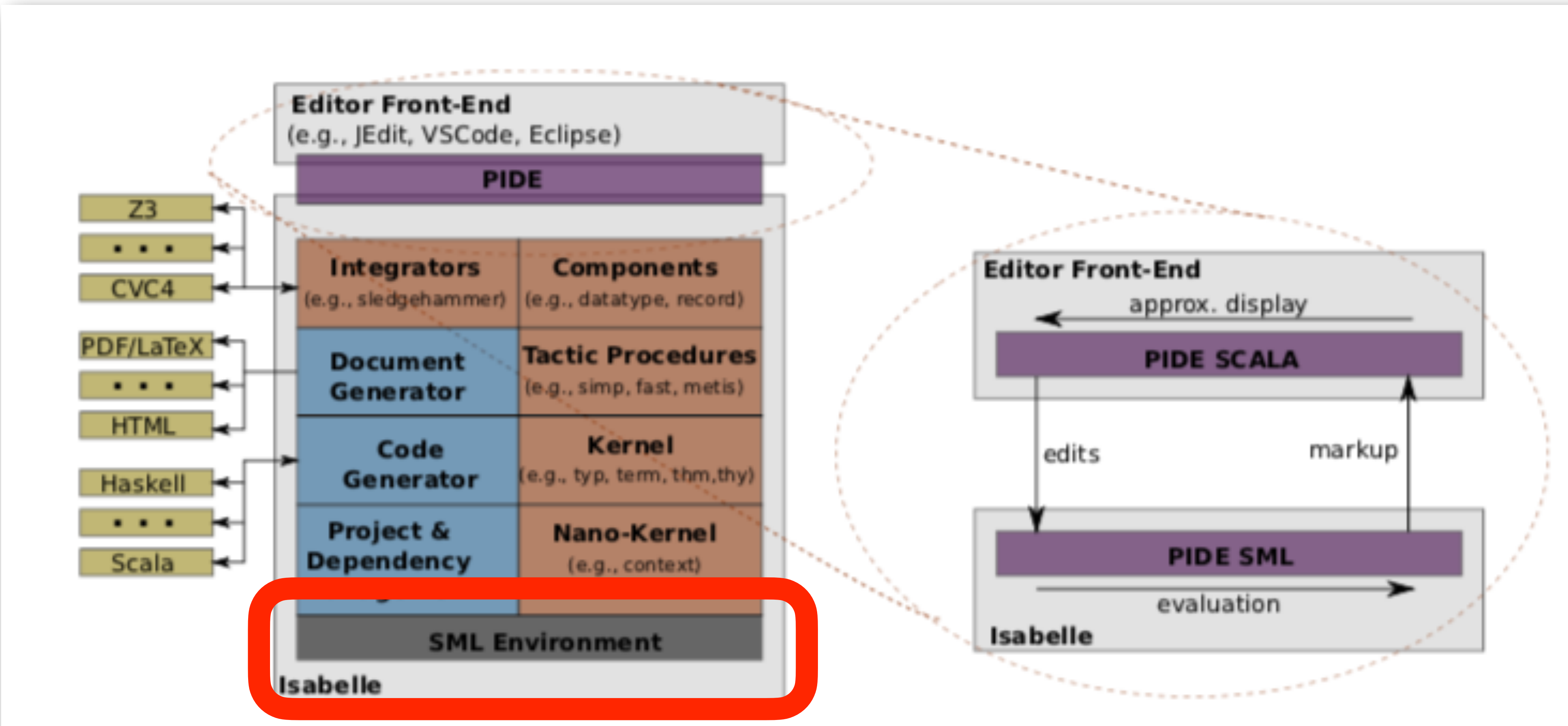
Isabelle - The SYSTEM

- Isabelle is
  - an extensible programming system (component framework)
  - based on a parallel functional programming language SML
  - geared towards ITP, but strong ATP support



Part I : SML

- Conceived in the early 80ies for Interactive Theorem Provers, the
d

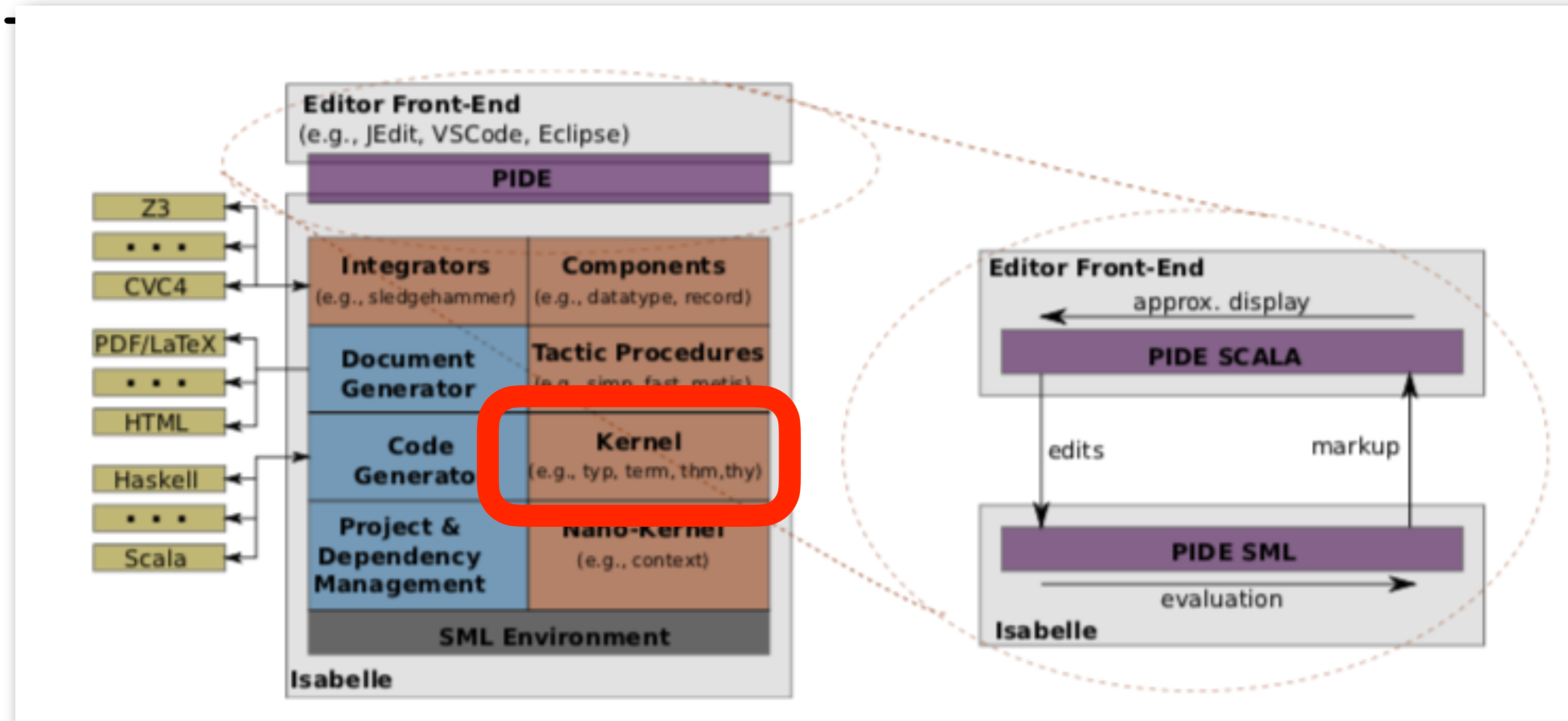


- less known than, say, OCaml and F#, but ...
- less known that Haskell, but significantly simpler and easier to learn+use

Part I : SML

DEMO

Part II : Kernel



Metalogic "Pure" providing Proof-terms

The typed λ-calculus

- Type Inferences:

$$\frac{}{\Sigma, \Gamma \vdash c_i :: \theta\ (\Sigma\ c_i)} \qquad \frac{}{\Sigma, \Gamma \vdash x_i :: \Gamma\ x_i}$$

$$\frac{\Sigma, \Gamma \vdash E :: \tau \Rightarrow \tau' \quad \Sigma, \Gamma \vdash E' :: \tau}{\Sigma, \Gamma \vdash E\ E' :: \tau'}$$

$$\frac{\Sigma, \{x_i \mapsto \tau\} \uplus \Gamma \vdash E :: \tau'}{\Sigma, \Gamma \vdash \lambda x_i . E :: \tau \Rightarrow \tau'}$$

The typed λ-calculus

Theoretical Properties (without proof)

- typed λ-calculi come with three congruences:
  - α congruence
    (terms are congruent modulo renaming of bound variables)
  - β congruence
    (applications of abstractions to terms reduce to substitutions
  - η congruence: unused bindings in abstractions cancel.
- for typed terms, the congruence $t \leftrightarrow_{\alpha\beta\eta} t'$
  is decidable (reduce to β-normalform, expand to
  η-longform, rename vars via α in some canonical order)

The typed λ-calculus

Theoretical Properties (without proof)

- Systems like Coq, Isabelle, HOL4 can use (some form of) typed λ-calculi as universal term-representation with binding operators such as ∀, ∃, sums, integrals, ...

- The type inference problem is decidable, i.e. for

$$\Sigma, ? \vdash t :: ??$$

there is an algorithm that finds solutions for ? and ?? if existing.

Pure in Typed λ-calculus

- Isabelle Kernel provides a minimal logic:

Σ_{Pure} = { $\_ \Longrightarrow \_ \mapsto prop \Rightarrow prop \Rightarrow prop$,

$\_ \equiv \_ \mapsto \alpha \Rightarrow \alpha \Rightarrow prop$,

$\bigwedge \_.\_ \mapsto (\alpha \Rightarrow prop) \Rightarrow prop$ }

where we will equivalently write

$\bigwedge x. P$ for $\bigwedge \_.\_ (\lambda x. P)$. (Quantifier notation)

Pure in Typed λ-calculus

- By the way: HOL is encoded in Pure:

$\Sigma_{HOL} = \Sigma_{Pure} \uplus$

$\{$ Trueprop $\mapsto$ bool $\Rightarrow$ prop,

True $\mapsto$ bool, False $\mapsto$ bool,

_∧_ $\mapsto$ bool⇒bool⇒bool, _∨_ $\mapsto$ bool⇒bool⇒bool,

_ ⟶ _ $\mapsto$ bool ⇒ bool ⇒ bool, ¬ _ $\mapsto$ bool ⇒ bool,

_ = _ $\mapsto$ α⇒α⇒ bool,

∀_._ $\mapsto$ (α⇒bool)⇒ bool,

∃_._ $\mapsto$ (α⇒bool)⇒ bool $\}$

- … + 8 axioms…

Pure in Typed λ-calculus

- Minimal logic (Pure) has the rules:

Proofs as λ-terms

p, q	$=$	h	Hypothesis
	$\mid$	$c_{\{\overline{\alpha} \mapsto \overline{\tau}\}}$	Proof constant (reference to axiom / theorem)
	$\mid$	$p \cdot t$	$\bigwedge$ -elimination
	$\mid$	$p \bullet q$	$\Longrightarrow$ -elimination
	$\mid$	$\boldsymbol{\lambda} x :: \tau.\ p$	$\bigwedge$ -introduction
	$\mid$	$\boldsymbol{\lambda} h : \varphi.\ p$	$\Longrightarrow$ -introduction

Proof checking

$$\frac{}{\Gamma, h : t, \Gamma' \vdash h : t} \qquad \frac{\Theta(c) = \varphi}{\Gamma \vdash c_{\{\overline{\alpha} \mapsto \overline{\tau}\}} : \varphi\{\overline{\alpha} \mapsto \overline{\tau}\}}$$

$$\frac{\Gamma \vdash p : \bigwedge x :: \tau.\ \varphi \quad \Gamma \vdash t :: \tau}{\Gamma \vdash p \cdot t : P\{x \mapsto t\}} \qquad \frac{\Gamma, x :: \tau \vdash p : \varphi}{\Gamma \vdash \boldsymbol{\lambda} x :: \tau.\ p : \bigwedge x :: \tau.\ \varphi}$$

$$\frac{\Gamma \vdash p : \varphi \Longrightarrow \psi \quad \Gamma \vdash q : \varphi}{\Gamma \vdash p \bullet q : \psi} \qquad \frac{\Gamma, h : \varphi \vdash p : \psi \quad \Gamma \vdash \varphi :: \mathsf{prop}}{\Gamma \vdash \boldsymbol{\lambda} h : \varphi.\ p : \varphi \Longrightarrow \psi}$$

- … and an axiomatisation of ≡.

Pure in Typed λ-calculus

- Isabelle CAN produce proof terms, in default mode, however, the Pure logic inferences do not.

- Rather, the essence of core inferences is captured in an abstract data-type

$$\text{thm} = \text{“}\Gamma \vdash_{\Theta} \varphi\text{”}$$

- … in order to save memory and time.

Pure in Typed λ-calculus

- Simplified, Pure has the rules:

$$\frac{A \in \Theta}{\vdash A}\ (axiom) \qquad \frac{}{A \vdash A}\ (assume)$$

$$\frac{\Gamma \vdash B[x] \quad x \notin \Gamma}{\Gamma \vdash \bigwedge x.\ B[x]}\ (\bigwedge\text{-}intro) \qquad \frac{\Gamma \vdash \bigwedge x.\ B[x]}{\Gamma \vdash B[a]}\ (\bigwedge\text{-}elim)$$

$$\frac{\Gamma \vdash B}{\Gamma - A \vdash A \Longrightarrow B}\ (\Longrightarrow\text{-}intro) \qquad \frac{\Gamma_1 \vdash A \Longrightarrow B \quad \Gamma_2 \vdash A}{\Gamma_1 \cup \Gamma_2 \vdash B}\ (\Longrightarrow\text{-}elim)$$

- … and an axiomatisation of ≡.

Part II : Kernel

DEMO

So: What is a proof in Isabelle ?

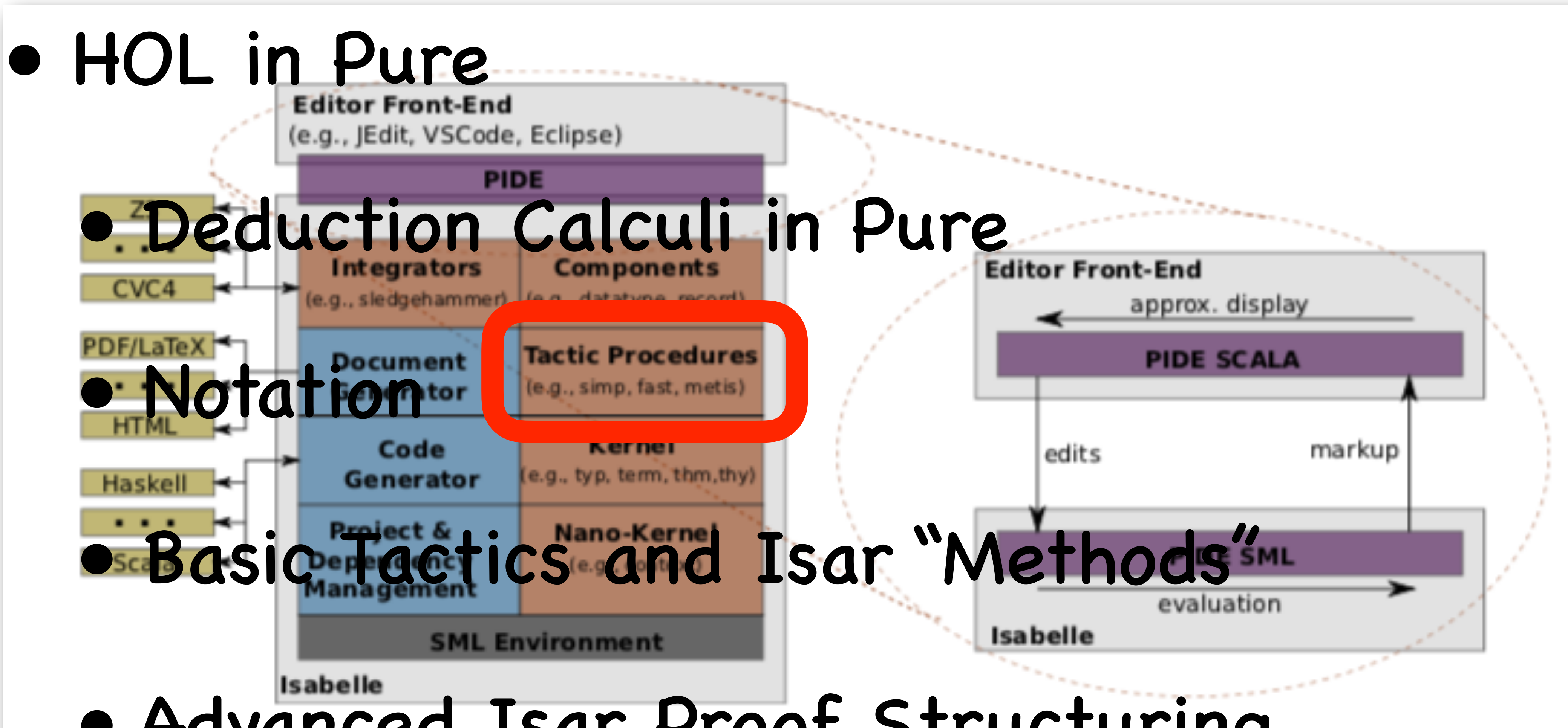
- First answer: a thm with or without a proof term.
- Remark in recent journal paper "From LCF to Isabelle/HOL"[NW21]:

> We see incidentally two meanings of the word proof :
> 1. formal deductions of theorems from axioms using the inference rules of a logical calculus;
> 2. executable code written using tactics or other primitives, expressing the search for such deductions.

- Since the the first answer is nice, but remarkably far from mathematics or Formal Methods engineering we turn to the latter.

Part III :

Tactic Procedures and Isar



- HOL in Pure
- Deduction Calculi in Pure
- Notation
- Basic Tactics and Isar “Methods”
- Advanced Isar Proof Structuring ….

„Pure": A (Meta)-Language for Deductive Systems

- Pure is a language to write logical rules (a "meta-logic")
- Higher-Order Logic (HOL) is our working logic.
- Equivalent notations for natural deduction rules (Textbook and Isabelle/HOL:)

$$\frac{A_1 \quad \ldots \quad A_n}{A_{n+1}}$$

theorem
assumes A_1
and ...
and A_n
shows A_{n+1}

$A_1 \Longrightarrow (\ldots \Longrightarrow (A_n \Longrightarrow A_{n+1})\ldots),$

$[\![A_1; \ldots; A_n]\!] \Longrightarrow A_{n+1},$

„Pure“: A (Meta)-Language for Deductive Systems

- Pure allows also to represent and reason over more complex rules involving the concept of “Discharge” of (hypothetical) assumptions:*

$(P \Longrightarrow Q) \Longrightarrow R$:

$$\frac{\begin{array}{c}[P] \\ \vdots \\ Q\end{array}}{R}$$

theorem
  assumes "$P \Longrightarrow Q$”
  shows "R"

* We follow the notation of van Dahlen’s Book: “Logic and Structure”. Available online.

„Pure": A (Meta)-Language for Deductive Systems

- Pure allows even more complex rules involving "local fresh variables" in sub-proofs:

$\bigwedge x.\ (P\ x \Longrightarrow Q\ x) \Longrightarrow R$:

$$\frac{\begin{array}{c}[P]_x\\ \vdots\\ Q\end{array}}{R}$$

theorem
  fix x
  assumes "P x ⟹ Q x"
  shows "R"

Key Concepts: Rule-Instances

- A Rule-Instance is a rule where the free variables in its judgements were substituted
  by a common substitution σ:

$$\frac{A \quad B}{A \wedge B}\,\text{conjI} \quad \xrightarrow{\ \sigma\ } \quad \frac{3 < x \quad x \leq y}{3 < x \wedge x \leq y}$$

where σ is {A ↦ 3<x, B ↦ x≤y} and
equality on terms is $\leftrightarrow_{\alpha\beta\eta}$.

Key Concepts: Resolution

- A Rule-Instance can be constructed by unification up to $\leftrightarrow_{\alpha\beta\eta}$ by a step called resolution:

$$\frac{A \quad B}{A \wedge B}$$

$$3 < x \wedge x \leq y$$

Key Concepts: Formal Proofs

- A series of inference rule instances is usually displayed as a Proof Tree (or : Derivation or: Formal Proof)

$$\text{subst}\;\dfrac{\dfrac{\text{sym}\;\dfrac{f(a,b) = a}{a = f(a,b)} \qquad \dfrac{f(a,b) = a \qquad f(f(a,b),b) = c}{f(a,b) = c}\;\text{subst}}{a = c}\;\text{trans} \qquad \dfrac{}{g(a) = g(a)}\;\text{refl}}{g(a) = g(c)}$$

- The hypothetical facts at the leaves are called the assumptions of the proof (here $f(a,b) = a$ and $f(f(a,b),b) = c$).

Part II : Kernel

DEMO

Key Concepts: Discharge

- A key requisite of ND is the concept of discharge of assumptions allowed by some rules (like impI)

$$
\dfrac{
  \dfrac{
    \text{sym}\,\dfrac{[f(a,b) = a]}{a = f(a,b)}
    \qquad
    \dfrac{[f(a,b) = a] \quad f(f(a,b),b) = c}{f(a,b) = c}\,\text{subst}
  }{a = c}\,\text{trans}
  \qquad
  \dfrac{}{g(a) = g(a)}\,\text{refl}
}{
  \dfrac{g(a) = g(c)}{f(a,b) = a \rightarrow g(a) = g(c)}
}\;\text{subst}
$$

- The set of assumptions is diminished by the discharged hypothetical facts of the proof (remaining: $f(f(a,b),b) = c$).

Sequent-style vs. ND calculus

- Both styles are linked by two transformations called "lifting over assumptions":

$$\frac{A_1 \quad \ldots \quad A_n}{A_{n+1}}$$

$$\Updownarrow$$

$$\frac{X_1..X_n \Longrightarrow A_1 \qquad\qquad X_1..X_n \Longrightarrow A_n}{X_1..X_n \Longrightarrow A_{n+1}}$$

Key Concepts: Discharge

- We can now explain the discharge mechanism by meta-implications carrying the local assumptions around:

$$\frac{}{\Gamma \Longrightarrow f(a,b) = a}\ \text{assumption}$$

$$\frac{\Gamma \Longrightarrow f(a,b) = a}{\Gamma \Longrightarrow a = f(a,b)}\ \text{sym} \qquad \frac{f(a,b) = a \quad f(f(a,b),b) = c}{\Gamma \Longrightarrow f(a,b) = c}\ \text{subst}$$

$$\frac{\Gamma \Longrightarrow a = f(a,b) \qquad \Gamma \Longrightarrow f(a,b) = c}{\Gamma \Longrightarrow a = c}\ \text{trans} \qquad \frac{}{\Gamma \Longrightarrow g(a) = g(a)}\ \text{refl}$$

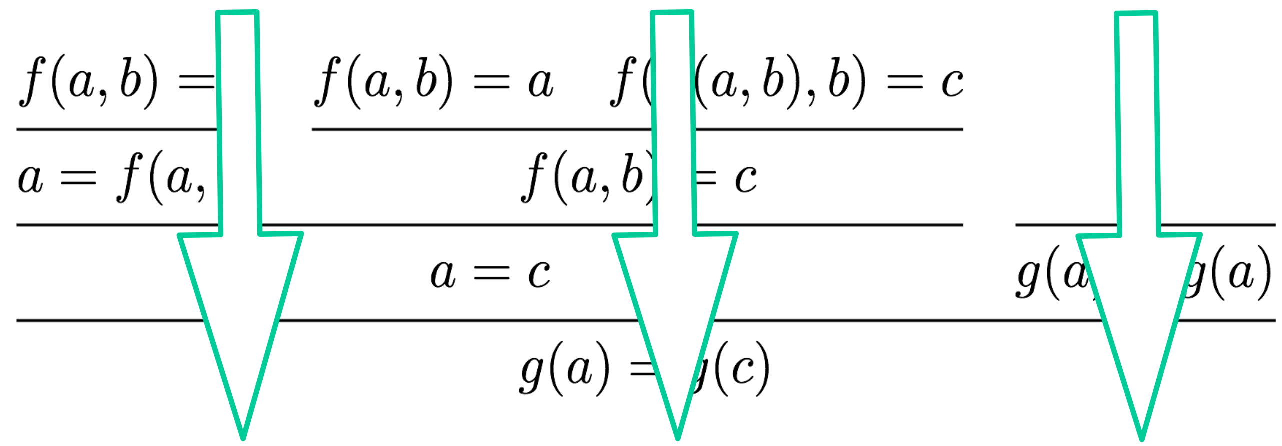
$$\frac{\Gamma \Longrightarrow a = c \qquad \Gamma \Longrightarrow g(a) = g(a)}{\Gamma \Longrightarrow g(a) = g(c)}\ \text{subst}$$

$$\frac{\Gamma \Longrightarrow g(a) = g(c)}{\{\} \Longrightarrow f(a,b) = a \to g(a) = g(c)}$$

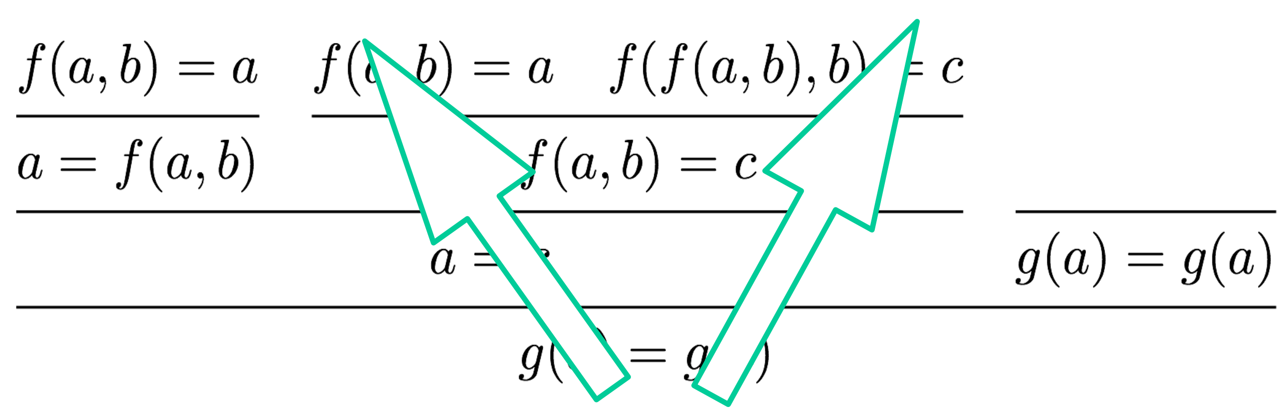
- where Γ is just $f(a,b) = c$

Proof Construction

- Proofs can be constructed in two ways
  - Top down,
    from assumptions
    to conclusions
    (Forward chaining)



  - Bottom up,
    decomposing conclusions
    to necessary assumptions
    (Backward Chaining)



Forward Proofs

- Isabelle/Isar supports a proof "commands" for step-wise forward proofs:
- General format:

```
lemmas <name> : <thm> [attribute]
```

Forward Proofs

- Local forward proof constructions by attributes

  - <thm>[THEN <thm>] (unifies conclusion vs. premise)
  - <thm>[OF <thm>] (unifies premise vs. conclusion)
  - <thm>[symmetric] (flips an equation)
  - <thm>[of (<term> | _)*] (instantiates variables)
  - <thm>[simp] (simplifies a thm)
  - <thm>[simp only: <thm>] (simplifies a thm)

Apply-Style Proofs

- Isabelle supports a proof language for step-wise backwards proofs: "apply style" proofs
- General format:

```
lemma <name> : "<formula>"
    apply(<method>)
    ...
    apply(<method>)
    done
```

- Abbreviation:

by(<method>)   is   apply(<method>) done

Apply-Style Proofs

- core - methods at a glance

```
assumption        — discharge conclusion
rule <thm>        — introduction rules
erule <thm>       — elimination rules
drule <thm>       — destruction rule
```

- Variants avec substitution

```
rule_tac <substitution> in <thm>
erule_tac <substitution> in <thm>
drule_tac <substitution> in <thm>
```

Part III : Tactics

DEMO

Advanced Isar Features

- The Isar (Intelligible Structured Automated Reasoning) is
  a proof generation language which is
  - block-structured
  - provides an abstraction from a goal-state via
    - reordering
    - abstraction of assumptions and fixes, patterns, …
    - named and un-named management of local assumptions
    - support of common proof formats such as proof by contradiction, induction, cases distinctions ,…
  - document generation.   Some samples:

A Structured „Classical“ Proof

- Example: (Nested) Proof by Contradiction

```
theorem "((A ⟶ B) ⟶ A) ⟶ A"
proof
  assume "(A ⟶ B) ⟶ A"
  show A
  proof (rule classical)
    assume "¬ A"
    have "A ⟶ B"
    proof
      assume A
      with ‹¬ A› show B by contradiction
    qed
    with ‹(A ⟶ B) ⟶ A› show A ..
  qed
qed
```

Nameless selection from local context

A Structured „Classical" Proof

- Example: A Calculational Proof

```
lemma (in group) group_right_inverse: "x * inverse x = 1"
proof -
  have "x * inverse x = 1 * (x * inverse x)"
     by (simp only: group_left_one)
  also have "... = 1 * x * inverse x"
     by (simp only: group_assoc)
  also have "... = inverse (inverse x) * inverse x * x * inverse x"
     by (simp only: group_left_inverse)
  also have "... = inverse (inverse x) * (inverse x * x) * inverse x"
     by (simp only: group_assoc)
  also have "... = inverse (inverse x) * 1 * inverse x"
     by (simp only: group_left_inverse)
  also have "... = inverse (inverse x) * (1 * inverse x)"
     by (simp only: group_assoc)
  also have "... = inverse (inverse x) * inverse x"
     by (simp only: group_left_one)
  also have "... = 1"
     by (simp only: group_left_inverse)
  finally show ?thesis .
qed
```

A Structured „Classical" Proof

- Example: Induction, Calculation, Patterns … towards a comprehensive human-readable proof presentation format

```
theorem sum_of_odds:
  "(∑i::nat=0..<n. 2 * i + 1) = n^Suc (Suc 0)"
  (is "?P n" is "?S n = _")
proof (induct n)
  show "?P 0" by simp
next
  fix n
  let ?two="Suc(Suc(0))"
  have "?S (n + 1) = ?S n + 2 * n + 1"
    by simp
  also assume "?S n = n^?two"
  also have "... + 2 * n + 1 = (n + 1)^?two"
    by simp
  finally show "?P (Suc n)"
    by simp
qed
```

introducing abbreviations by pattern-matching

proof structuring method

local abbreviation

intermediate goal

catching intermediate lemma

induction hyp

presenting main goal in terms of abbrevs

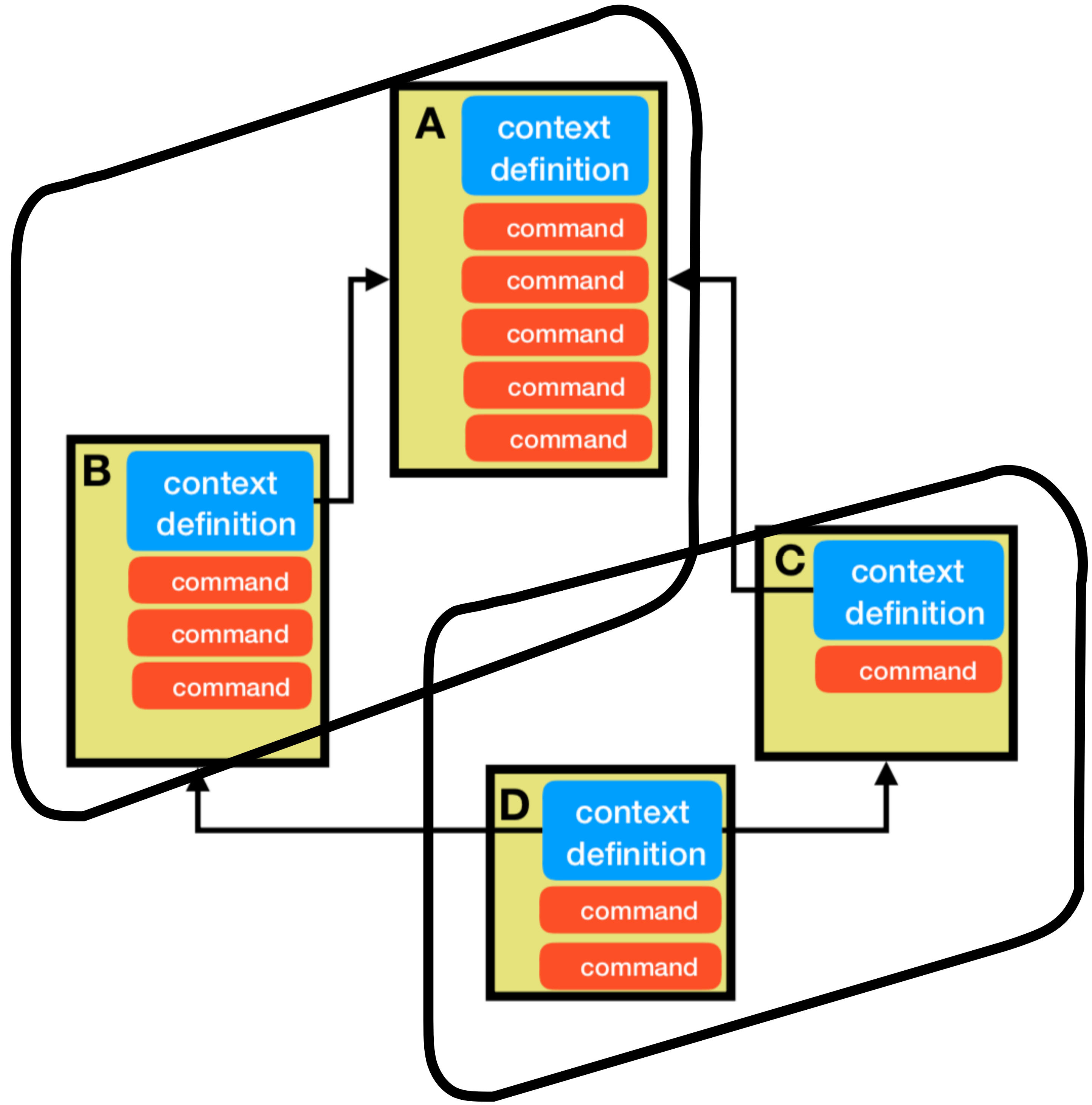
Conclusion

- The Isabelle environment provides a modern interactive proof assistant
- … in the LCF prover tradition, based on a meta-logic and a kernel architecture
- … based on many very advanced technologies from parallel SML over HO-Unification to PIDE
- … offers structured proofs and a proof archive (google Isabelle AFP)
- … includes many leading edge AUTOMATED proof techniques such as Paramodulation, Tableaux-Proving, SMT-Solvers, Arithmetic decision procs …

Thank You !

Seminary on Isabelle @ ENS Saclay:
[1]Interactive Theorem Proving and Applications.
Material and Videos :
https://www.lri.fr/~wolff/teach-material/2020-2021/M2-CSMR/index.html
[2] The Isabelle Club @ VALS / LMF
https://vals.lri.fr/isabelleclub/IsabelleClub/

We have funding for an Internship (Stage) up to 6 months for a serious Isabelle/HOL-CSP project in the domain of Autonomous Cars !!! Contact me !



PIDE is implemented in 50 kloc Scala and has connectors for Coq and Isabelle

OUR SOLUTION

ISABELLE/C - A FRAMEWORK FOR C-TOOLS

- A new set of commands, most notably the new C-command inside PIDE:
  - Fully editable, IDE support
  - navigation
  - C11 syntax
  - Generic, programmable Annotations

C2.thy (~/codebox/isabelle_c/C11-FrontEnd/examples/)

```
section‹Some realistic Selection sort with Input and Output›

C‹
#include <stdio.h>

int main()
{
  int array[100], n, c, d, position, swap;

  printf("Enter number of elements\n");
  scanf("%d", &n);

  printf("Enter %d integers\n", n);

  for (c = 0; c < n; c++) scanf("

  for (c = 0; c < (n - 1); c++)
  {
    position = c;
```

```
:: int
C local variable "n"
bound variable
```

DEMO

Isabelle/C and Isabelle/C/AutoCorres

DEMO

Isabelle/C and Isabelle/C/AutoCorres

CONCLUSION

- Isabelle/C : a generic Front-End for C providing
general IDE support
- Follows idea of "Isabelle as Eclipse",
enabling Integrated Documents with Ontology Support
- Instantiatable with various semantic interpretations of C,
and developed libraries in HOL
- Platform for verification "back-ends" in Test and Proof
- Strong mechanism for plugin-separation
as well as plugin-collaboration