

# Capturing & Inferring the Proof Process

Andrius Velykis

School of Computing Science  
Newcastle University, UK

*with contributions from AI4FM members*



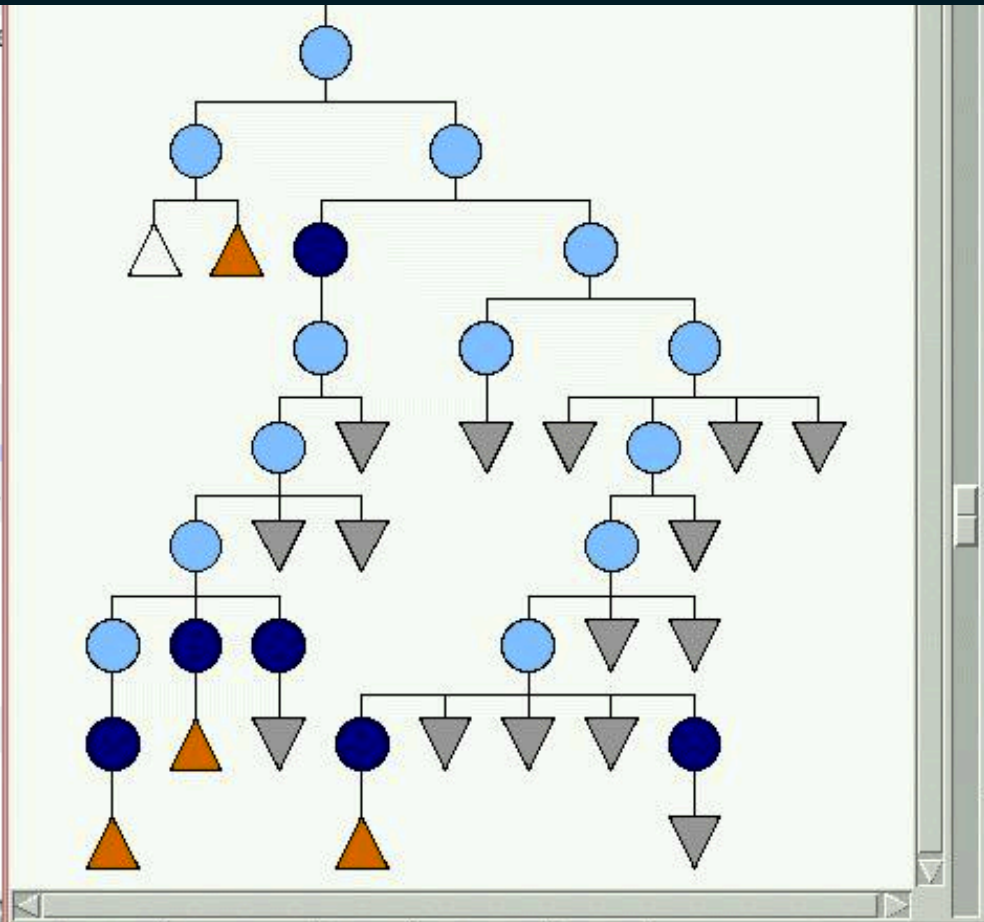
# Motivation

- ▶ **Extract** & reuse high-level proof strategy (cf. Gudmund's talk)
- ▶ **Capture** enough information to facilitate understanding of **high-level reasoning**
- ▶ Can we infer the *proof process* automatically?

```

apply (unfold equateK_def equate_def retrPart_def)
apply (rule set_eqI)
apply (simp del: if_image_distrib)
apply (rename_tac xs)
apply (intro iffI)
apply (elim exE)
proof -
case goal1
then have xsA: "xs = {e. (m e = m e2 → m e1 =
and kRan: "k ∈ range (λe. if m e = m e2 then
from kRan have "k = m e1 ∨ k ∈ range m - {m e2}
by (rule equateRange)
with xsA show ?case
proof (cases "k = m e1")
case goal1
hence "xs = ⋃{s. (∃k. s = {e. m e = k} ∧ k
by auto
thus ?case by simp
next
case goal2 then have xsR: "xs = {e. m e = k

```



L4  
L6  
L5  
L17  
L18  
L19  
L20  
L8  
L9  
L12  
L11  
L13  
L21  
L22  
L23  
L14  
L15  
L24  
L16  
L10

Previous sequent  
a0 : A  
H : nth\_error (a :: l) 0 = Some a0  
=====  
a :: l = firstn 0 (a :: l) ++ (a0 :: nil) ++ skipn (0 + 1) (a :: l)

Dismiss 2 open goals (no new)

Sequent 1 of nth\_error\_split

# What describes a proof process?

Things we need to collect (and infer) to capture expert's ideas

# Proof **granularity**

Layers of abstraction: from high-level plan to tactic steps

# Proof **structure**

Parallel proof branches instead of a sequence of steps

# Multiple **attempts**

Full development, including failed attempts

```

1987 1999
1988 2000 text{*computing div by shifting *}
1989 2001
1990 2002 lemma pos_zdiv_mult_2: "(0::int) \<le> a ==> (1 + 2*b) div (2*a) = b div a"
1991 - proof cases
1992 -   assume "a=0"
1993 -     thus ?thesis by simp
1994 - next
1995 -   assume "a\<noteq>0" and le_a: "0\<le>a"
1996 -   hence a_pos: "1 \<le> a" by arith
1997 -   hence one_less_a2: "1 < 2 * a" by arith
1998 -   hence le_2a: "2 * (1 + b mod a) \<le> 2 * a"
1999 -     unfolding mult_le_cancel_left
2000 -     by (simp add: add1_zle_eq add_commute [of 1])
2001 -   with a_pos have "0 \<le> b mod a" by simp
2002 -   hence le_addm: "0 \<le> 1 mod (2*a) + 2*(b mod a)"
2003 -     by (simp add: mod_pos_pos_trivial one_less_a2)
2004 -   with le_2a
2005 -   have "(1 mod (2*a) + 2*(b mod a)) div (2*a) = 0"
2006 -     by (simp add: div_pos_pos_trivial le_addm mod_pos_pos_trivial one_less_a2
2007 -       right_distrib)
2008 -   thus ?thesis
2009 -     by (subst zdiv_zadd1_eq,
2010 -       simp add: mod_mult_mult1 one_less_a2
2011 -       div_pos_pos_trivial)
2012 - qed
2003 +   using pos_divmod_int_rel_mult_2 [OF _ divmod_int_rel_div_mod]
2004 +   by (rule div_int_unique)
2013 2005
2014 2006 lemma neg_zdiv_mult_2:
2015 2007   assumes A: "a \<le> (0::int)" shows "(1 + 2*b) div (2*a) = (b+1) div a"

```

# Proof cleanup

One-liners lose ideas how the proof was reached

# Intent

Tags with high-level explanation (coupled with *features*)

## Proof features

What triggered the proof step?

- ▶ Shape of assumption / goal
- ▶ Available lemma
- ▶ Certain datatypes or records

*Infer*—or—*ask* the expert (à la Clippy in MS Office)?

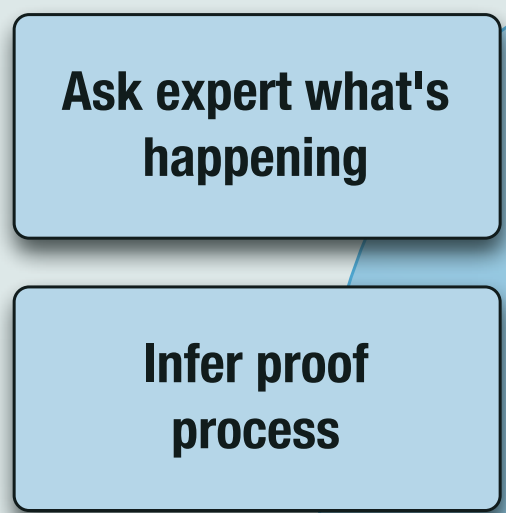
# ProofProcess framework

- ▶ “Wire-tap” theorem prover
  - Prototypes for Isabelle/HOL and Z/EVES
- ▶ Generic core + prover-specific extensions
- ▶ Matchers for features/structure
- ▶ Proof history
- ▶ Accommodate any proof process?

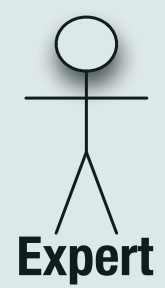
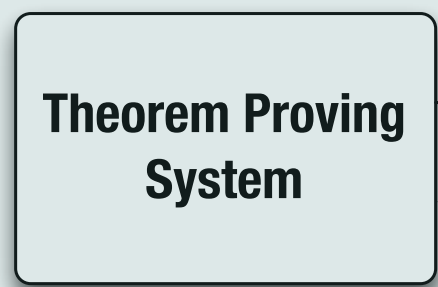
## ProofProcess model



## Create ProofProcess Model



## ProofProcess Integration



# Architecture



# Capture the role of lemmas

- ▶ Capture the need for lemma and its features
  - Implicit search (cf. Alan's talk)
- ▶ Trace lemma usage in automatic proof tactics (e.g. `apply auto`)

# Find affected goal terms

- ▶ Identify expert's path of reasoning by narrowing the scope
  - Impact analysis?
- ▶ Discover “unimportant” terms by inverting the scope
- ▶ Extract a toy problem

# Fuzzy match with previous proofs

- ▶ Consult captured proof process to recognise a similar one
  - Partial match proof features and intents
  - Match any part of previous proof process
- ▶ Similar to strategy extraction/reuse
  - Better results “in retrospect”?

# Custom proof features

- ▶ Abstract, domain specific, difficult to express formally
- ▶ ... vs predefined features (e.g. *shape*)
- ▶ Can we automate the reuse?

# Proof Process

**extract and  
reuse  
proof  
strategies**

**improve proof  
maintenance**

**train  
interactive  
theorem  
proving**

# Understand interactive proof

Focus on industrial models with *families* of similar proof obligations

# Capture expert's proof process

Collect information to extract reusable proof strategies

