

Improving the Usability of HOL through Controlled Automation Tactics

Eunsuk Kang and Mark Aagaard
Waterloo Formal Methods Group



University of Waterloo, Canada

Motivations

- Despite a rich set of available tactics & functions, theorem proving in HOL is not straightforward
- Many user interactions are tedious and repetitive
 - Lots of effort spent on frequent low-level operations (i.e. “doing the grunt work”)
- How can we improve HOL?
 - **Our focus:** Make **common cases** easy & fast

Outline

- ➔ ■ **Challenge #1: Choosing Tactics to Use**
 - Solution: Controlled Automation Tactics
- Challenge #2: Working with Assumptions
 - Solution: Extend **ELIM_TAC** with Assumption Handling
- Challenge #3: Complexity of Rewriting
 - Solution: Controlled Rewriting Tactics
- Case Studies

Challenge #1: Choosing Tactics to Use

- Main challenge: Too many of them
- “What’s the next step? Which tactic do I apply?”
 - Stiff learning curve for a beginner
 - Non-trivial even for an expert user
- e.g. HOL reference manual – 900+ pages
 - A significant amount of time is spent on browsing through the manual

High-level vs. Low-level Tactics

- **High-level tactics** – “Do as much as you can”
 - e.g. **PROVE_TAC**, **DECIDE_TAC**, **RES_TAC**, etc.
 - Model elimination, decision procedures, resolution, etc.
- Advantage: Completes all or parts of the proof for the user – **if it works**
- Disadvantage: May do more than what user expects (e.g. “things changed magically”)

High-level vs. Low-level Tactics

- **Low-level tactics** – “Do one thing, but do it right”
 - **CONJ_TAC**, **DISCH_TAC**, **EXIST_TAC**, etc.
- Advantage: **No surprises** for the user
- Disadvantage:
 - Tedious to remember names & syntax
 - High-level automated provers (e.g. **PROVE_TAC**) are preferred if a goal is easily provable (but often hard to tell this just by inspection)

Outline

- Challenge #1: Choosing Tactics to Use
- ■ **Solution: Controlled Automation Tactics**
- Challenge #2: Working with Assumptions
 - Solution: Extend **ELIM_TAC** with Assumption Handling
- Challenge #3: Complexity of Rewriting
 - Solution: Controlled Rewriting Tactics
- Case Studies

Our Approach: Controlled Automation

- A **balanced medium** between high-level & low-level tactics
- Controlled automation tactics should:
 - Perform operations **exactly** as the user intends
 - **Give no surprises** - if an operation can't be fulfilled, terminate immediately
 - **Provide improved automation**: Replace existing low-level tactics & remove user's burden of remembering them

Controlled Automation Tactics: Overview

- **GOAL:** Replace most common low-level functions
- Three ideas:
 1. Perform most common logical operations by **guessing “the common thing to do next”**
 2. When a guess requires external guidance, user passes in extra info. through a simple, intuitive mechanism called **hints**
 3. Using **assumption labeling**, let user operate on both the conclusion of the goal and assumptions

Predicting Decomposition Operation

- Observation: Often, we can make a good guess of what user wants to do next by looking at the **outermost logical operator**
- Examples:
 - If the conclusion of a goal is in form $\text{P} \wedge \text{Q}$, apply **CONJ_TAC**
 - If the conclusion is in form $\exists x. \text{P } x$, provide a witness to eliminate the existential quantifier

Common Structural Tactics in HOL

- Identify most commonly used operations for goal decomposition
- Replace them with a new tactic called **ELIM_TAC**

Operator	HOL Tactics
\wedge	CONJ_TAC
\vee	DISJ_TAC
\implies	DISCH_TAC
\exists	EXISTS_TAC
\forall	GEN_TAC

ELIM_TAC: Syntax

- Syntax:

```
ELIM_TAC target_label [hint]
```

- **target_label**: a string label for an assumption
(empty string "" to refer to the conclusion of a goal)
- **hint**: ML data structure for passing in “extra” information

```
datatype hint =  
  ASM of string  
| WITNESS of term  
| INSTANCE of term  
| MATCH of term  
| SPLIT of term  
| SKOLEM;
```

Examples: `ELIM_TAC`

```
Initial goal:  
  P /\ Q  
By: ELIM_TAC "" [];  
2 subgoals:  
  Q  
  
  P
```

```
Initial goal:  
  P ==> Q  
By: ELIM_TAC "" [];  
1 subgoal:  
  Q  
  -----  
  asm. P
```

<- conclusion of goal
<- assumption list

Existential Quantifier Elimination

Initial goal:
 $\exists x. x > 0 \wedge x < 2$

By: **ELIM_TAC?**

- Next step: Eliminate the existential quantifier using a witness `1`
- **Problem:** Dangerous to make a guess for witness without user's input
- **ELIM_TAC** must somehow receive this extra bit of information

Quantifier Elimination using a Hint

Initial goal:

```
 $\exists x. x > 0 \wedge x < 2$ 
```

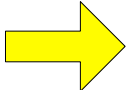
```
By: ELIM_TAC "" [WITNESS `1`];
```

1 subgoal:

```
 $1 > 0 \wedge 1 < 2$ 
```

- **WITNESS** (ML type: `term -> hint`) is a constructor of `hint` that specifies the witness for \exists elimination

Outline

- Challenge #1: Choosing Tactics to Use
 - Solution: Controlled Automation Tactics
-  ■ **Challenge #2: Working with Assumptions**
 - Solution: Extend `ELIM_TAC` with Assumption Handling
- Challenge #3: Complexity of Rewriting
 - Solution: Controlled Rewriting Tactics
- Case Studies

Challenge #2: Working with Assumptions

- Assumption handling is controversial; we think that it is a matter of personal preference
- Depending on one's style of proof, it is useful (and often necessary) to operate on current assumptions

Initial goal:

Q

1. **P** **==>** **Q**

2. P

- In this example: apply modus ponens between two assumptions

Assumption Handling in HOL

- Only a few built-in HOL functions to manipulate assumptions
 - By default, can't point directly to an assumption using a numeric index in HOL
 - Users often end up “hacking” HOL and create their own ML functions to do this
 - e.g. `get_assumption: int -> term`
- For every structural tactic, user must remember corresponding **forward inference rules**
 - e.g. `DISCH_TAC` vs. `MP`

Outline

- Challenge #1: Choosing Tactics to Use
 - Solution: Controlled Automation Tactics
- Challenge #2: Working with Assumptions
 - ➔ ■ **Solution: Extend `ELIM_TAC` with Assumption Handling**
- Challenge #3: Complexity of Rewriting
 - Solution: Controlled Rewriting Tactics
- Case Studies

Extending `ELIM_TAC` with Assumption Handling

- Extend `ELIM_TAC` to operate on assumptions
 - Same concept as before: **Guess “what to do”** by looking at the structure of an assumption
 - **Uniform** interface for working with **both** the conclusion and the assumptions of a goal
- Instead of numbers, use **user-provided names** to label assumptions
 - + Robust to changes in proof
 - A few more keystrokes for the user (**short-term vs. long-term** gains)

Example: `ELIM_TAC`

- Previous example on modus ponens

```
Initial goal:
  Q
  -----
  p_imp_q. P ==> Q
          p. P
By: ELIM_TAC "p_imp_q" [ASM "p"];
1 subgoal:
  Q
  -----
  p_imp_q. Q
          p. P
```

- Equivalent to: `Eliminate` the implication in ``P ==> Q`` using assumption ``P``

Another Example: `ELIM_TAC`

- Extract the subterm `t2` from `t1 /\ t2 /\ t3`:

```
Initial goal:
P
-----
a. t1 /\ t2 /\ t3
By: ELIM_TAC "a" [];
1 subgoal:
P
-----
a. t1 /\ t2
a_t3. t3
```

```
Initial goal:
P
-----
a. t1 /\ t2 /\ t3
By: ELIM_TAC "a" [MATCH `t2`];
1 subgoal:
P
-----
a. t1 /\ t3
a_t2. t2
```

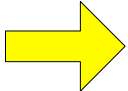
- Users can use hints to specify their **intention**

Making Common Operations Fast

- Some operations on assumptions take **multiple steps**; **ELIM_TAC** performs them in a **single step**

Operator	Applied to:	
	Conclusion	Assumption
\wedge	CONJ_TAC	CONJUNCT1, CONJUNCT2
\vee	DISJ_TAC	DISJ_CASES_TAC, ASM_CASES_TAC
\implies	DISCH_TAC	MP
\exists	EXISTS_TAC	(UNDISCH_TAC + CONV_TAC + GEN_TAC + DISCH_TAC)
\forall	GEN_TAC	SPEC

Outline

- Challenge #1: Choosing Tactics to Use
 - Solution: Controlled Automation Tactics
- Challenge #2: Working with Assumptions
 - Solution: Extend **ELIM_TAC** with Assumption Handling
-  ■ **Challenge #3: Complexity of Rewriting**
 - Solution: Controlled Rewriting Tactics
- Case Studies

Challenge #3: Complexity of Rewriting

- Challenge: Difficult to control where to rewrite

```
Theorem: ADD_SYM
[] |-  $\forall m n. m + n = n + m$ 
```

```
Initial goal:
  x - (z + y) = x - (y + z)
By: REWRITE_TAC [ADD_SYM] ;
```

Q. What happens?

A. It will loop forever. By default, **REWRITE_TAC** is recursive (i.e. “do as much as you can”)

Rewriting Once in HOL

- Try **ONCE_REWRITE_TAC**, which does not do recursive rewriting

```
Theorem: ADD_SYM
          [] |-  $\forall m n. m + n = n + m$ 
```

Initial goal:

$$x - (z + y) = x - (y + z)$$

By: **ONCE_REWRITE_TAC [ADD_SYM]** ;

1 subgoal:

$$x - (y + z) = x - (z + y)$$

- This is still not what we wanted. We want to rewrite only one of the two sides

Specifying Rewrite Location in HOL

- Try **GEN_REWRITE_TAC**, which accepts “conversions” as search strategies:

Initial goal:

$$x - (z + y) = x - (y + z)$$

By: **GEN_REWRITE_TAC** (**RATOR_CONV** o **ONCE_DEPTH_CONV**)
[ADD_SYM] ;

1 subgoal:

$$x - (y + z) = x - (y + z)$$

- RATOR_CONV**: Applies rewrite only to operator “**f**” of a function application in form, “**f a**”
- ONCE_DEPTH_CONV**: Applies rewrite only once, using depth-first search

Rewriting Tactics and Conversions

- Lots of built-in rewriting tactics, inference rules and conversions – difficult to remember all these

```
REWRITE_TAC  
ONCE_REWRITE_TAC  
PURE_ONCE_REWRITE_TAC  
GEN_REWRITE_TAC  
ASM_REWRITE_TAC  
PURE_ONCE_ASM_REWRITE_TAC  
ONCE_ASM_REWRITE_TAC  
FILTER_ASM_REWRITE_TAC  
REWRITE_RULE  
... and more
```

```
LAND_CONV  
RAND_CONV  
RATOR_CONV  
ABS_CONV  
COMB_CONV  
DEPTH_CONV  
ONCE_DEPTH_CONV  
REDEPTH_CONV  
TOP_DEPTH_CONV  
... and more
```

- **How can we reduce user's effort in remembering them?**

Outline

- Challenge #1: Choosing Tactics to Use
 - Solution: Controlled Automation Tactics
- Challenge #2: Working with Assumptions
 - Solution: Extend **ELIM_TAC** with Assumption Handling
- Challenge #3: Complexity of Rewriting
 - ➔ ■ **Solution: Controlled Rewriting Tactics**
- Case Studies

Controlled Automation for Rewriting

- **GOAL:** Replace many of the existing rewriting tactics, rules, and conversions
- Two underlying support mechanisms:
 1. Give user a **precise control** over a rewriting location using **hints** - “surgeon’s tool for rewriting”
 2. Using **assumption labeling**, allow user to apply a rewrite rule directly to an assumption or rewrite the goal using assumptions

Controlling Rewriting Location

- Use **higher-order matching** to specify where to rewrite
- Define our own rewrite tactic called **EQUATE_TAC**:

```
EQUATE_TAC [hint]
```

- On previous example:

```
Initial goal:
```

$$x - (z + y) = x - (y + z)$$

```
By: EQUATE_TAC [MATCH `z + y`, THM ADD_SYM];
```

```
1 subgoal:
```

$$x - (y + z) = x - (y + z)$$

- **MATCH** specifies a subterm to rewrite & **THM** specifies a rewrite rule

Rewriting and Assumptions

- Extend **EQUATE_TAC** to operate on assumptions

```
EQUATE_TAC target_label [hint]
```

- New types of hints & extended roles for existing ones

Hint	Provided to:	
	ELIM_TAC	EQUATE_TAC
ASM	=> elimination (modus ponens)	Rewrite rule
ASMS	-	Rewrite rules
MATCH	\wedge , \vee elimination	Rewrite match
THM, THMS	-	Rewrite rule(s)

Controlled Rewriting Tactics

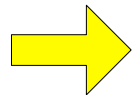
- **Given:** Supporting mechanisms for specifying a subterm & rewriting with assumptions
- **Classify common rewriting tasks** in HOL into three categories:
 - **Definitional expansion (`EXPAND_TAC`)**
 - **Term simplification (`REDUC_TAC`)**
 - **Other equational rewriting (`EQUATE_TAC`)**
- All three tactics share the same syntax as **`ELIM_TAC`**

Comparison of Controlled Rewriting Tactics

	Purpose	Rewriting Behaviour
EXPAND_TAC	Automatically expands the definition of a constant stored in the current HOL theory.	Rewrites once.
REDUC_TAC	Simplifies a target expression as much as possible using built-in theorems	Repeatedly applies rewrite rules.
EQUATE_TAC	Performs general rewriting operations.	Rewrites once.

Outline

- Challenge #1: Choosing Tactics to Use
 - Solution: Controlled Automation Tactics
- Challenge #2: Working with Assumptions
 - Solution: Extend **ELIM_TAC** with Assumption Handling
- Challenge #3: Complexity of Rewriting
 - Solution: Controlled Rewriting Tactics

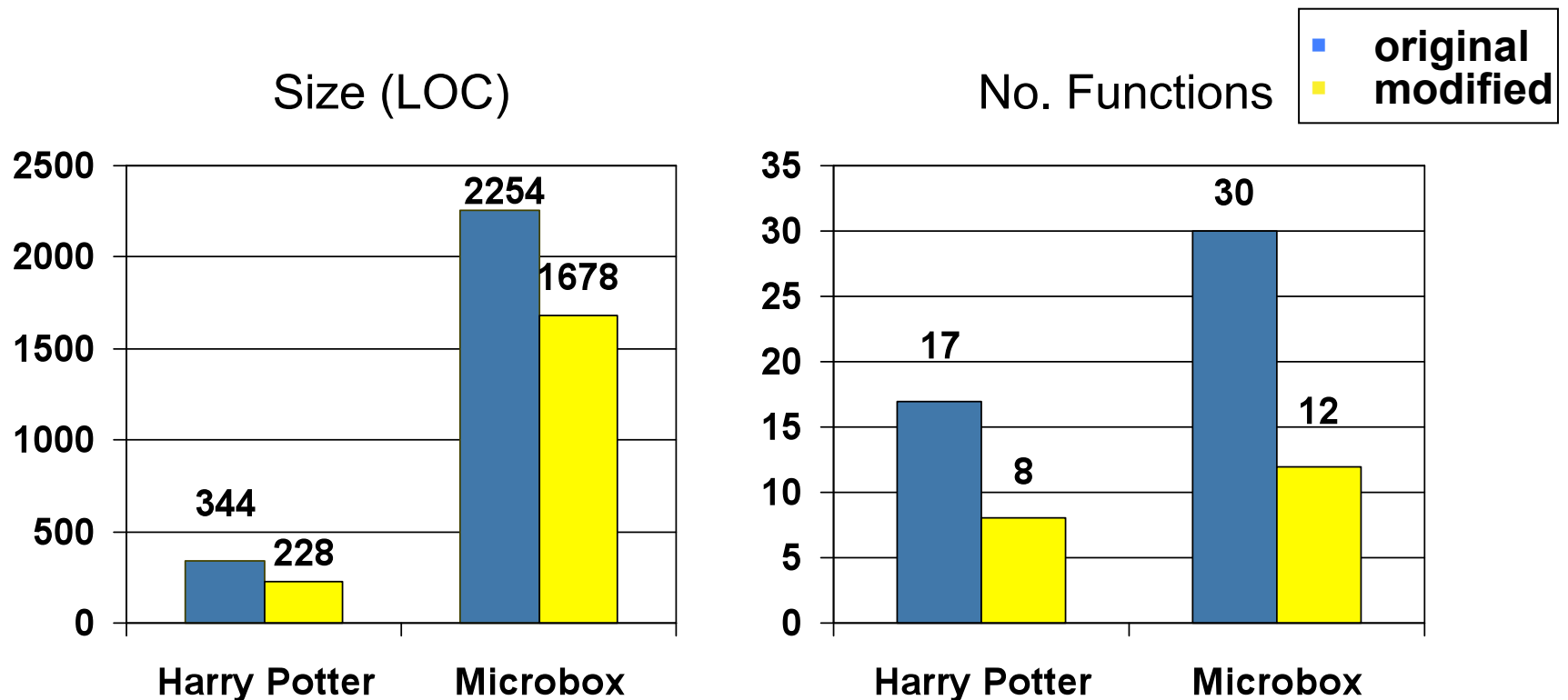


- **Case Studies**

Case Studies

- Used our tactics to re-do two sample proofs
 - A logic puzzle from Harry Potter
 - Proofs about superscalar microprocessor correctness from Microbox project
- Comparison with original proofs:
 - Size of proof script (LOC)
 - Number of unique tactics, inference rules, and conversions used

Case Studies: Results



- Around **25~33%** reduction in the size of proof scripts, and **55~60%** in the number of HOL functions

Case Studies: Remaining Functions

- HOL functions that remained in the modified proof scripts (excluding our controlled automation tactics):
 - **High-level automated provers**: `PROVE_TAC`, `METIS_TAC`, etc.
 - **Special proof techniques**: `Induct`
 - **Tacticals for combining tactics**: `REPEAT`, `THEN`
 - **Auxiliary functions**: used to add, remove, or rename assumptions
- Most of the low-level tactics/rules have been replaced by the controlled automation tactics!

Summary

- Challenges in improving the usability of HOL:
 - **Tactics, assumption handling, and term rewriting**
- Our solutions = **Controlled automation tactics**
 - Primitive logical operations: **ELIM_TAC**
 - Rewriting operations: **EXPAND_TAC, EQUATE_TAC, REDUC_TAC**
- **Hints**: a simple, intuitive mechanism for user to:
 - Provide our tactics with information required in **precisely** carrying out desired operations
- **Results**: A significant reduction in the number of HOL functions that users need to remember