Quantifier Heuristics in HOL4

Thomas Tuerk

1st February 2011

Unwind Library

- eliminating quantifiers is often beneficial
- HOL4's simplifier uses the *unwind* library
- the unwind library can handle simple examples

Unwind Library Example

$$\forall x\ y.\ P(x,y) \wedge (x = c) \Longrightarrow Q(x) \quad \longleftrightarrow \quad \forall y.\ P(c,y) \Longrightarrow Q(c)$$

- the unwind library is fast and often useful
- however, it is restricted to equality and certain patterns

Unwind Library Failing Example

$$\forall x.\ (\exists y.\ P(x,y) \wedge (x = c)) \Longrightarrow Q(x)$$

Quantifier Heuristics Library

- this talk presents the *Quantifier Heuristics Library*
- it can handle more complicated terms
- automatically uses matching as well as equality
- uses information about datatypes
- allows partial instantiations
- is user extendable
- allows guessing without proof

Quantifier Heuristics Library Examples

Quantifier Heuristics Library Examples

$$\forall x.\ (\exists y.\ P(x,y) \wedge (x = c)) \Longrightarrow Q(x) \quad \leftrightarrow \quad (\exists y.\ P(c,y)) \Longrightarrow Q(c)$$

$$\exists x.\ P(f(x)) \ \wedge\ (f(x) = f(c)) \quad \leftrightarrow \quad P(f(c))$$

$$\exists x.\ P(x) \wedge ((c_1, x) = c_2) \quad \leftrightarrow \quad P(\mathit{SND}\ c_2) \wedge (c_1 = \mathit{FST}\ c_2)$$

$$\forall x.\ \mathit{IS\_SOME}(x) \Longrightarrow P(x) \quad \leftrightarrow \quad \forall x\_x.\ P(\mathit{SOME}(x\_x))$$

$$\forall x.\ x \neq [] \Longrightarrow P(x) \quad \leftrightarrow \quad \forall x\_t\ x\_h.\ P(x\_h :: x\_t)$$

General Idea

- Given a term $\exists x.\ P(x)$ one is interested in finding an instantiation i such that $\exists x.\ P(x) \Leftrightarrow \exists fv.\ P(i(fv))$ holds.
- Similarly, given $\forall x.\ P(x)$ one is interested in finding i such that $\forall x.\ P(x) \Leftrightarrow \forall fv.\ P(i(fv))$ holds.
- This leads to the following definitions of *guesses*:

$$\mathit{GUESS\_EXISTS}\ (\lambda fv.\ i(fv))\ (\lambda x.\ P(x)) \stackrel{\text{def}}{=} \exists x.\ P(x) \Leftrightarrow \exists fv.\ P(i(fv))$$

$$\mathit{GUESS\_FORALL}\ (\lambda fv.\ i(fv))\ (\lambda x.\ P(x)) \stackrel{\text{def}}{=} \forall x.\ P(x) \Leftrightarrow \forall fv.\ P(i(fv))$$

- Idea: construct guesses bottom up

Stronger Guesses I

- Problem: *GUESS_EXISTS* and *GUESS_FORALL* do not behave well for bottom up analysis
- they don't carry enough information / they are too weak
- let's introduce stronger guesses for existential quantification
- i is chosen, because it satisfies P:

$$\mathit{GUESS\_TRUE}\ (\lambda fv.\ i(fv))\ (\lambda x.\ P(x)) \stackrel{\text{def}}{=} \forall fv.\ P(i(fv))$$

- i is chosen, because all other instantiations do not satisfy P:

$$\mathit{GUESS\_EXISTS\_STRONG}\ (\lambda fv.\ i(fv))\ (\lambda x.\ P(x)) \stackrel{\text{def}}{=} \forall x.\ P(x) \Longrightarrow \exists fv.\ x = i(fv)$$

Stronger Guesses II

- *GUESS_TRUE* and *GUESS_EXISTS_STRONG* behave nicely
- *GUESS_TRUE* behaves nicely with disjunctions

$$\mathit{GUESS\_TRUE}\ i_{fv}\ (\lambda x.\ P(x)) \Longrightarrow \mathit{GUESS\_TRUE}\ i_{fv}\ (\lambda x.\ P(x) \vee Q(x))$$

- *GUESS_EXISTS_STRONG* behaves nicely with conjunctions

$$\mathit{GUESS\_EXISTS\_STRONG}\ i_{fv}\ (\lambda x.\ P(x)) \Longrightarrow \mathit{GUESS\_EXISTS\_STRONG}\ i_{fv}\ (\lambda x.\ P(x) \wedge Q(x))$$

- other, more complicated rules exists as well

Stronger Guesses III

- Guesses dual to *GUESS_TRUE* and *GUESS_EXISTS_STRONG* are introduced for universal quantification
- i is chosen, because it does not satisfy P:

$$\mathit{GUESS\_FALSE}\ (\lambda fv.\ i(fv))\ (\lambda x.\ P(x)) \stackrel{\text{def}}{=} \forall fv.\ \neg(P(i(fv)))$$

- i is chosen, because all other instantiations satisfy P:

$$\mathit{GUESS\_FORALL\_STRONG}\ (\lambda fv.\ i(fv))\ (\lambda x.\ P(x)) \stackrel{\text{def}}{=} \forall x.\ \neg P(x) \Longrightarrow \exists fv.\ x = i(fv)$$

Hierarchy of Guesses

$$\begin{array}{ccc}
\mathit{GUESS_TRUE} & \overset{dual}{\longleftrightarrow} & \mathit{GUESS_FALSE} \\
\Downarrow & & \Downarrow \\
\mathit{GUESS_EXISTS} & \overset{dual}{\longleftrightarrow} & \mathit{GUESS_FORALL} \\
\Uparrow & & \Uparrow \\
\mathit{GUESS_EXISTS_STRONG} & \overset{dual}{\longleftrightarrow} & \mathit{GUESS_FORALL_STRONG}
\end{array}$$

Selected Inference Rules

$$\begin{array}{ll}
\mathit{GUESS_EXISTS}\ i_{fv}\ (\lambda x.\ P(x)) & \Longleftrightarrow \\
\mathit{GUESS_FORALL}\ i_{fv}\ (\lambda x.\ \neg P(x)) & \\[1ex]
\mathit{GUESS_FALSE}\ i_{fv}\ (\lambda x.\ P(x)) & \Longrightarrow \\
\mathit{GUESS_TRUE}\ i_{fv}\ (\lambda x.\ P(x) \Rightarrow Q(x)) & \\[1ex]
\mathit{GUESS_FORALL}\ i_{fv}\ (\lambda x.\ P(x)) & \Longrightarrow \\
\mathit{GUESS_FORALL}\ i_{fv}\ (\lambda x.\ P(x) \vee q) & \\[1ex]
\mathit{GUESS_FALSE}\ i_{fv}\ (\lambda x.\ P(x)) \wedge & \\
\mathit{GUESS_FALSE}\ i_{fv}\ (\lambda x.\ Q(x)) & \Longrightarrow \\
\mathit{GUESS_FALSE}\ i_{fv}\ (\lambda x.\ P(x) \vee Q(x)) & \\[1ex]
\forall y.\ \mathit{GUESS_FORALL}\ (\lambda fv.\ i(fv, y))\ (\lambda x.\ P(x, y)) & \Longrightarrow \\
\mathit{GUESS_FORALL}\ (\lambda (fv, y).\ i(fv, y))\ (\lambda x.\ \forall y.\ P(x, y)) &
\end{array}$$

Base Case: Equation

- equations allow the following guesses

$$\begin{array}{l}
\mathit{GUESS_TRUE}\ (\lambda fv.\ i)\ (\lambda x.\ x = i) \\
\mathit{GUESS_EXISTS_STRONG}\ (\lambda fv.\ i)\ (\lambda x.\ x = i)
\end{array}$$

- using matching, one also gets

$$\begin{array}{l}
P(i) = Q(i) \implies \\
\mathit{GUESS_TRUE}\ (\lambda fv.\ i)\ (\lambda x.\ P(x) = Q(x))
\end{array}$$

- one can also use disequations

$$\begin{array}{l}
\forall fv.\ P(i(fv)) \neq Q(i(fv)) \implies \\
\mathit{GUESS_FALSE}\ (\lambda fv.\ i(fv))\ (\lambda x.\ P(x) = Q(x))
\end{array}$$

Base Case: Datatype Cases

- Many datatypes like lists or options consist of exactly two cases. Such case theorems can be used as follows:

$$\begin{array}{l}
(\forall x.\ x = c_1 \vee \exists fv.\ x = c_2(fv)) \implies \\
\mathit{GUESS_FORALL_STRONG}\ (\lambda fv.\ c_2(fv))\ (\lambda x.\ x = c_1)
\end{array}$$

- Types like pairs only allow a single form:

$$\begin{array}{l}
(\forall x.\ \exists fv.\ x = c(fv)) \implies \\
\mathit{GUESS_FORALL_STRONG}\ (\lambda fv.\ c(fv))\ (\lambda x.\ P(x))
\end{array}$$

$$\begin{array}{l}
(\forall x.\ \exists fv.\ x = c(fv)) \implies \\
\mathit{GUESS_EXISTS_STRONG}\ (\lambda fv.\ c(fv))\ (\lambda x.\ P(x))
\end{array}$$

Overview

- the ideas described so far are implemented by *quantHeuristicsLib*
- the core of this framework is a bottom-up search for guesses
- quantifier reordering and minimising of variable occurrences aid the search
- guesses are used to instantiate existential, universal and unique existential quantification
- the main tools of quantHeuristicsLib are
  - `QUANT_INSTANTIATE_CONV`
  - `QUANT_INSTANTIATE_TAC`
  - `ASM_QUANT_INSTANTIATE_TAC`
  - `QUANT_INST_ss`
- standard Boolean operations and equations are built in
- a list of *quantifier heuristics parameters* (qp) can be used for extensions

Quantifier Heuristic Parameters

- quantifier heuristic parameters extend the search for guesses
- usually they contain information about special predicates or datatypes

```
type quant_param =
  {distinct_thms      : thm list,
   cases_thms         : thm list,
   inference_thms     : thm list,
   rewrite_thms       : thm list,
   convs              : conv list,
   filter             : (term -> term -> bool) list,
   heuristics         : ... list,
   top_heuristics     : ... list,
   final_rewrite_thms : thm list};
```

Predefined QPs

- `quantHeuristicsLib` defines the following QPs
  - `TypeBase_qp`
  - `stateful_qp`
  - `get_qp___for_types`
- `quantHeuristicsArgsLib` defines QPs for common datatypes
  - `option_qp`
  - `list_qp`
  - `num_qp`
  - `pair_default_qp`
  - `record_default_qp`
- all these are combined in `std_qp`

Unjustified Guesses

- so far, all guesses are justified by theorems
- unjustified guesses are supported as well
- unjustified guesses result in implications

$$\forall x.\ P(x) \Longrightarrow \forall fv.\ P(i(fv))$$

$$\exists fv.\ P(i(fv)) \Longrightarrow \exists x.\ P(x)$$

- sometimes these implications are sufficient
- consequence conversions are used to apply these implications at subpositions
- `QUANT_INST_CONV` and `QUANT_INST_TAC` allow to instantiate quantifiers at subpositions with user-provided values

Conclusion

- the quantifier heuristics library is powerful
- it is easy to use
- easily extendable by adding theorems
- details can be customised using user-defined heuristics written in ML
- however, slower than unwind library

Examples

$\exists x.\ \mathit{if}\ \ b(x)\ \mathit{then}\ ((x = 2) \wedge Q_1(x))\ \mathit{else}\ (Q_2(x) \wedge (x = 2)) \leftrightarrow$
$\mathit{if}\ \ b(2)\ \mathit{then}\ Q_1(2)\ \mathit{else}\ Q_2(2)$

$$\begin{array}{lcl}
\exists! x.\ (x = 2) \wedge Q(x) & \leftrightarrow & Q(2) \\
\exists x.\ P(f(x))\ \wedge\ (f(x) = f(c)) & \leftrightarrow & P(f(c)) \\
\exists x.\ P(x) \wedge ((c_1, x) = c_2) & \leftrightarrow & P(\mathit{SND}\ c_2) \wedge (c_1 = \mathit{FST}\ c_2) \\
\exists p.\ (x = \mathit{FST}(p)) \wedge Q(p) & \leftrightarrow & \exists p_2.\ Q((x, p_2)) \\
\forall x.\ (x \neq 0) \Longrightarrow P(x) & \leftrightarrow & \forall x_n.\ P(\mathit{SUC}(x_n)) \\
\forall x.\ (x = 7) \wedge P(x) & \leftrightarrow & F \\
\exists x.\ (f(x) = f(c)) \vee P(x) & \leftrightarrow & T
\end{array}$$