Case-Analysis for Rippling and Inductive Proof

Moa Johansson[1]
Joint work with Lucas Dixon[2] and Alan Bundy[2]

Dipartimento di Informatica, Università degli Studi di Verona, Italy.[1]

School of Informatics, University of Edinburgh, UK.[2]

Interactive Theorem Proving Conference
12 July 2010

Introduction: Automating Inductive Proofs

- Induction: reasoning about recursion.
  - Programs, data-structures...
- Automation difficult. Guidance needed:
  - e.g. case-splitting for inductively defined datatypes.

Aim: Automation of theorems involving conditional definitions.

- Half of the functions in Isabelle's list library involve conditional statements.
- e.g. *member*, *delete*, *subtraction*.

Proof Planning and Rippling

- **Proof Planning:** Families of proofs with similar structure.
  - e.g. inductive proofs.
- **Rippling:** Heuristic for guiding rewriting in step cases.
  - Annotate differences between induction hypothesis and conclusion.
  - Rewrites must reduce differences.
    - Top of term tree: Get instance of IH.
    - Position of $\forall$ quantified variable in IH.
  - Fully automatic and guarantees termination.
  - Does not require rewrite rules to be oriented.

Case Analysis for Rippling

- **IsaPlanner:** Higher-order proof-planner for Isabelle.
- Extend rippling in IsaPlanner to cover case analysis for:
  - If-statements.
  - Case-statements over datatypes.
- **Retain termination:**
  - Ripple-step becomes: rewriting + case-splitting.

A Simple Example

Definition of *max:*

$$\begin{aligned} max\ 0\ y = {}& y && (1)\\ max\ (Suc\ x)\ y = {}& case\ y\ of\ 0 \Rightarrow Suc\ x && (2)\\ & \mid Suc\ z \Rightarrow Suc(max\ x\ z) \end{aligned}$$

Commutativity of *max*:

Inductive hypothesis (IH): $\forall b.\ max\ a\ b = max\ b\ a$

Step-case goal: $max\ \boxed{Suc\ \underline{a}}\ \lfloor b' \rfloor = max\ \lfloor b' \rfloor\ \boxed{Suc\ \underline{a}}$

A Simple Example

Definition of *max:*

$$
\begin{aligned}
max\ 0\ y &= \ y && (1)\\
max\ (Suc\ x)\ y &= case\ y\ of\ 0 \Rightarrow Suc\ x && (2)\\
&\quad \mid Suc\ z \Rightarrow Suc(max\ x\ z)
\end{aligned}
$$

Commutativity of *max*:

Inductive hypothesis (IH): $\forall b.\ max\ a\ b = max\ b\ a$

Step-case goal: $max\ \boxed{Suc\ \underline{a}}\ \lfloor b' \rfloor = max\ \lfloor b' \rfloor\ \boxed{Suc\ \underline{a}}$

Apply (2):

$$\boxed{case\ b'\ of\ 0 \Rightarrow Suc\ a \mid Suc\ z \Rightarrow Suc(\underline{max\ a\ \lfloor z \rfloor})}\ = max\ \lfloor b' \rfloor\ \boxed{Suc\ \underline{a}}$$

Applying the Split

Isabelle automatically derives splitting rules for case-statements for each datatype:

$$\llbracket ?n = 0 \Longrightarrow ?P(?f_1);\ \forall x.\ (?n = Suc\ x) \Longrightarrow ?P(?f_2\ x) \rrbracket \Longrightarrow$$
$$?P(case\ ?n\ of\ 0 \Rightarrow ?f_1 \mid (Suc\ x) \Rightarrow (?f_2\ x))$$

$?P$ matches context of case-statement.

- Case-split as resolution step.
- Interactive setting: User gives instantiation of $?P$.
- Automatic proof: $?P$ in head position - match any term, many trivial unifiers, large search space.

Restricted Unification

$$?P(\underbrace{case\ ?n\ of\ 0\ \Rightarrow ?f_1 \mid (Suc\ x)\ \Rightarrow\ (?f_2\ x)}_{\text{meta-variable argument}})$$

$$\underbrace{case\ b'\ of\ 0 \Rightarrow Suc\ a \mid Suc\ z \Rightarrow Suc(max\ a\ z)}_{\text{subterm of interest}} = max\ b'\ Suc\ a$$

- Only apply to terms containing argument of meta-variable $?P$.
- Find instantiation for $?P$, then safe to apply regular resolution.
- Algorithm using *Zippers*.
- Traverse term, find matching subterm. Zipper keeps track of its context.
- Use context of subterm to provide instantiation of $?P$: $\lambda v.\ v = max\ b'\ Suc\ a$

Example cont.

A ripple step with case-analysis:

$$\left\{\begin{array}{l} max\ \boxed{Suc\ \underline{a}}\ \ \lfloor b' \rfloor = max\ \lfloor b' \rfloor\ \boxed{Suc\ \underline{a}} \\ \qquad\qquad\qquad \downarrow \text{Rewrite using: max def.} \\ case\ b'\ of\ 0 \Rightarrow Suc\ a \mid Suc\ z \Rightarrow Suc(max\ a\ z) = max\ b'\ Suc\ a \\ \qquad\qquad\qquad \downarrow \text{Apply case-split.} \end{array}\right.$$

$$b' = 0 \implies Suc\ a = max\ 0\ (Suc\ a) \tag{3}$$

$$b' = Suc\ z \implies \boxed{Suc(\underline{max\ a\ \lfloor z \rfloor})} = max\ \lfloor Suc\ z \rfloor\ \boxed{Suc\ \underline{a}} \tag{4}$$

Goal 3 is solved by simplification (no rippling embedding).

Rippling continues on goal 4.

Evaluation

- Case-statements common, not previously covered by rippling.
- Implementation in IsaPlanner:
  Rippling + Case Analysis + Lemma Calculation
- Corpus of 87 inductive theorems. If- and case statements.
  - Lists, natural numbers, binary trees.
  - Isabelle library, inductive TP literature, dependently typed programming.
- Prover given only function definitions. No extra lemmas supplied.

Evaluation

- 47/87 new theorems proved automatically.
- Remaining theorems:
  - More sophisticated reasoning about side-conditions.
  - Conjecturing of conditional lemmas.
- Example failed proof: *sorted(insertionSort(l))*
- Need lemma with assumption:
  $sorted\ m \Longrightarrow sorted(insert\ x\ m)$.

Comparison to Simplification-Based Technique

- Isabelle's simplifier allows splitting on if-statements.
- Case-statements not attempted: may cause non-termination.

Coverage:

- Proved by Induction+Simp technique: 37
- Proved by Rippling: 47

Termination:

- Rippling terminates on all examples.
- Induction+Simp often fails to terminate:
  - Proofs it cannot solve.
  - When asked for alternative proofs.
  - Stuck trying to conjecture increasingly complex lemmas.

Other Approaches

- **Recursion Analysis:** Choose induction scheme avoiding need for case-splits.
  - **ACL2**, **VeriFun:** No datatypes, destructor style, recursion on several arguments instead of case statements.
- **Isabelle:** HO, datatypes, constructor style.
  Want to work directly with these.
- Further Work: Automatic construction/selection of induction schemes in IsaPlanner.
- May still need case-analysis even with more elaborate induction schemes.
  - Case-statement introduced by auxiliary lemma, rewrite from other conditional function definition.

Summary

- Case-analysis needed for inductive proofs about conditional functions.
- Fully automatic technique implemented in IsaPlanner.
- Incorporated with rippling.
- Ensures termination also in presence of case-statements over datatypes.
- Proves 47/87 theorems in evaluation corpus.
- Further work:
  - More sophisticated reasoning with assumptions
  - Conditional lemma discovery.