After today

- Week 9
  - Tu: Pat Rondon
  - Th: Ravi/Nathan
- Week 10
  - Tu: Nathan/Ravi
  - Th: Class canceled
- Finals week
  - Th: Zach, John

Techniques on last time!

Induction

- Is just another way of proving universal quantifiers
- Important technique for proving properties of inductively defined structures, such as recursive data structures, recursive programs, and derivations

Principle of induction

- Induction over natural numbers

\[
P(0) \quad \forall \ i: \text{Nat} \ P(i) \implies P(i+1)
\]

- More generally: well-founded induction

\[
\forall \ i \ (\forall j < i . P(j)) \implies P(i) \quad (c, N) \text{ is a well-founded order}
\]

- Structural induction: well-founded induction where the induction is done on the (usually syntactic) structure of elements of N

- Well founded order: for every non-empty subset, there is a minimal element (an element such that no other element is smaller than it)
- Equivalently: there are no infinite descending chains
Automating induction

- Automating induction is hard
  - not many theorem provers do it automatically
- Difficulties
  - determine when to apply induction
  - determine what variables to apply induction on
  - derive a strong enough hypothesis

Induction in ACL2

- Similarity between recursive definitions and induction
- A recursive function calls itself with arguments that are “smaller” (by some measure) than the original arguments
- A proof of $P(x)$ by induction consists of proving $P(x)$ by assuming that $P$ holds for terms “smaller” (by some measure) than $x$

Induction in ACL2

- ACL2 proves theorems about recursive functions, and so the above similarity leads to a scheme for applying induction
- Suppose that a formula $P$ to be proven contains a call to a function $f$ that is defined recursively, and suppose each recursive call drives down the measure
- Then we can apply induction on the structure of the recursive definition of $f$

Simple example

```lisp
(defun sum (n)
  (if (= n 0)
      0
      (+ n (sum (- n 1))))
)
```

```lisp
(editthm (implies (>= n 0)
  (= (sum n)
      (/ (* n (+ n 1)) 2))))
```

Two parts

- First part: identify possible measures
  - done when a recursive function is defined
- Second part: apply induction during a proof
  - use the analysis done during the first part to guide the application

Identifying measures

- Suppose we have $(f \ x_1 \ldots \ x_n) = \text{body}$, where the body contains recursive calls to $f$
- We are looking for a measure function $m$ that satisfies the following:
  - for every sub-term of the form $(f \ y_1 \ldots y_k)$, we must have $(r \ (m \ y_1 \ldots y_k) \ (m \ x_1 \ldots x_n))$
  - where $r$ is a well-founded relation
Example

\((\text{funny-fact } n \ i) =\)
\((\text{IF } (\leq n 0) 1\)
\((\text{IF } (= i 0) (* n (\text{funny-fact } (- n 1) i))\)
\((\text{IF } (> i 0) (* n (\text{funny-fact } (- n i) i))\)
\((* n (\text{funny-fact } (+ n i) i))))\)

Refine: identifying measures

- Suppose we have \((f x_1 \ldots x_n) = \text{body}\), where the body contains recursive calls to \(f\).
- We are looking for a measure function \(m\) that satisfies the following:
  - for every sub-term of the form \((f y_1 \ldots y_n)\), we must have:
    
    \((\text{IMPLIES } (\text{AND } t_1 \ldots t_k) (r (m y_1 \ldots y_n) (m x_1 \ldots x_n)))\)
  - where \(r\) is a well-founded relation

Build the “machine” for a function

\((\text{funny-fact } n \ i) =\)
\((\text{IF } (\leq n 0) 1\)
\((\text{IF } (= i 0) (* n (\text{funny-fact } (- n 1) i))\)
\((\text{IF } (> i 0) (* n (\text{funny-fact } (- n i) i))\)
\((* n (\text{funny-fact } (+ n i) i))))\)

Use machine to test measure

- For example, for case 3:

  \((\text{IMPLIES } (\text{AND } (\text{NOT } (= n 0)) \text{ (NOT } (= i 0)) \text{ (NOT } (> i 0)) \text{ (< } (+ n i) n))\)

How do you guess measure?

- Rely on lemmas that have been labeled with the “induction” hint
- These so-called induction lemmas state under what conditions certain measures decrease
- For example:

  \((\text{IMPLIES } (\text{is-cons-cell } X) \\text{ (NOT } (> (\text{length } (\text{cdr } X)) (\text{length } X))))\)
How do you guess measure?

• Suppose we have a function append:

\[
\text{append} \ (L_1 \ L_2) = \\
\begin{cases} 
\text{cons (car } L_1\text{) (append (cdr } L_1\text{) } L_2) & \text{if } \text{is-cons-cell } L_1 \\
L_2 & \text{otherwise}
\end{cases}
\]

(\text{IMPLIES (is-cons-cell } X\text{)} \\
\ (< \ (\text{length } (\text{cdr } X) \ (\text{length } X))))

<table>
<thead>
<tr>
<th>case</th>
<th>test</th>
<th>recursive calls</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Another example

\[
\text{ACK } M \ N = \begin{cases} 
+ \ (N \ 0) & \text{if } M = 0 \\
(\text{ACK } (- M 1) N) & \text{otherwise}
\end{cases}
\]

(\text{IMPLIES (is-cons-cell } X\text{)} \\
\ (< \ (\text{length } (\text{cdr } X) \ (\text{length } X))))

<table>
<thead>
<tr>
<th>case</th>
<th>test</th>
<th>recursive calls</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Another example

• Lexicographical order of \((M, N)\)

\[
\text{LEX} \ L = \begin{cases} 
\text{cons } (- M 1) N & \text{if } M = 0 \\
\text{cons } M N & \text{otherwise}
\end{cases}
\]

First part summary

• When the function is defined, try to find measure
• Use induction lemmas to guide search
• Also allow programmer to provide the measure
• Once a measure has been found, the function is “admitted”
• As a side benefit, guarantees termination
Second part: applying induction

• Suppose term \((f \ t_1 \ldots t_n)\) appears in the goal \(G\).

• Assume \(t_1 \ldots t_n\) are all variables.
  – if they are not, things get more complicated.

• Assume "machine" for \(f\) looks like this:

\[
\begin{array}{cccc}
 r_k & 1 & \ldots & r_k \ h_k \\
 & \vdots & \ddots & \vdots \\
 r_2 & 1 & \ldots & r_2 \ h_2 \\
 r_1 & 1 & \ldots & r_1 \ h_1 \\
\end{array}
\]

Also, assume \(\sigma^h\) is subst such that \(\sigma^h\) applied to

\((f \ t_1 \ldots t_n)\) gives \(r_i \ h_i\)

Second part: applying induction

Goal \(G\) with sub-term \((f \ t_1 \ldots t_n)\) to \(k\).

Also, assume \(\sigma^h\) is subst such that \(\sigma^h\) applied to

\((f \ t_1 \ldots t_n)\) gives \(r_i \ h_i\).

Simple example from before

\[(\text{sum } N) = (\text{IF } (\leq N 0) 0 (+ N \text{ (sum } (- N 1))))\]

\[(\text{defthm } (\text{IMPLIES } (\leq N 0) (= (\text{sum } N) (/ (* N (+ N 1)) 2))))\]

Strengthening

• It is often advantageous, and sometimes even necessary to strengthen a conjecture before attempting an inductive proof.

• ACL2 does this with a generalization process, which runs before induction is applied.

• Tries to replace certain non-variable terms by fresh variables, which are implicitly universally quantified.
Strengthening example

• Suppose we want to prove (all free vars universally quantified):
  \[(\text{append} \ (\text{reverse} \ A) \ \text{nil}) = (\text{reverse} \ A)\]

• This conjecture is generalized
  – by replacing \((\text{reverse} \ A)\) with \(L\)
  \[(\text{append} \ L \ \text{nil}) = L\]

Strengthening gone wrong

• The problem with generalization is that the generalized conjecture may be “too strong”
  – and in fact may not hold even though the original conjecture did!

• Example:
  \[(= \ (\text{sum} \ (\text{fib} \ N)) \ (/ \ (* \ (\text{fib} \ N) \ (+ \ (\text{fib} \ N) \ 1)) \ 2)))\]

Strengthening gone wrong

• The problem with generalization is that the generalized conjecture may be “too strong”
  – and in fact may not hold even though the original conjecture did!

• Example:
  \[(= \ (\text{sum} \ (\text{fib} \ N)) \ (/ \ (* \ (\text{fib} \ N) \ (+ \ (\text{fib} \ N) \ 1)) \ 2)))\]

• Generalize:
  \[(= \ (\text{sum} \ M) \ (/ \ (* \ M \ (+ \ M \ 1)) \ 2)))\]
  This does not hold! We need to know that \(M\) is positive!

Strengthening fixed

• The generalization process constrains the newly introduced variables using theorems that have been labeled “generalize” by the user

• In the previous case, the following theorem could be used: \((\geq \ (\text{fib} \ N) \ 0)\)

• Once this theorem has been proven and labeled “generalize”, subsequent generalizations of \((\text{fib} \ N)\) to \(M\) will include the hypothesis \((\geq \ M \ 0)\).
  – Generalized conjecture in example becomes:
    \[(\text{IMPLIES} \ \ (\geq \ M \ 0) \ (= \ (\text{sum} \ M) \ (/ \ (* \ M \ (+ \ M \ 1)) \ 2)))\]

Induction in ACL2: summary

• The three difficulties, again
• When to apply induction
  – as a last resort
• What variables to apply induction on
  – analyze function at definition time to find decreasing measures
• How to strengthen induction hypothesis
  – replace certain terms with fresh vars

Similar ideas, but elsewhere

• Using recursive definitions to drive the application of induction appears in other theorem provers, for example Twelf

• Twelf is a theorem prover based on the LF logical framework, which can be used to encode logics, programming languages, and deductive systems

• Properties of such systems can then be proven by induction
Natural deduction

\[
\begin{align*}
\Gamma &\vdash A \\
\Gamma &\vdash A \land B \\
\Gamma &\vdash A \\
\Gamma &\vdash B \\
\Gamma &\vdash A \\
\Gamma &\vdash B \\
\end{align*}
\]

Proving properties of derivations

- Say we want to prove a property:
  - for every derivation \( D \), \( P(D) \) holds

Twelf

- Twelf can perform these kinds of proofs automatically
- The application of induction is interspersed with case splitting and the backward application of Twelf’s sequent inference rules