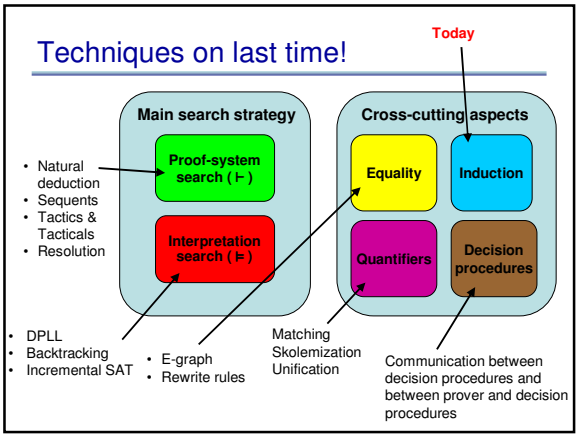
After today

- 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

$$\frac{P(0) \qquad \forall\, i{:}Nat\ P(i) \Rightarrow P(i+1)}{\forall\, i{:}Nat\ .\ P(i)}$$

Principle of induction

- More generally: well-founded induction

$$\frac{\forall\, j\ (\ [\forall\, i < j\ .\ P(i)] \Rightarrow P(j)\ ) \qquad (<, N) \text{ is a well-founded order}}{\forall\, i{:}N\ .\ P(i)}$$

- 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

Principle of induction

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

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

```
(sum N) = (IF (<= N 0)
              0
              (+ N (sum (- N 1))))
(defthm (IMPLIES (>= N 0)
                 (= (sum N)
                    (/ (* N (+ N 1)) 2))))
```

G(N)

G(0)

G(N) => G(N-1)

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 … x_n) = 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 … y_n), we must have (r (m y_1 … y_n) (m x_1 … x_n))
  - where r is a well-founded relation

Example

```
(funny-fact n i) =
    (IF (<= n 0)
        1
        (IF (= i 0) (* n (funny-fact (- n 1) i))
        (IF (> i 0) (* n (funny-fact (- n i) i))
                    (* n (funny-fact (+ n i) i))))))
```

Refine: identifying measures

- Suppose we have (f x_1 … x_n) = 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 … y_n), we must have:
    (IMPLIES (AND t_1 … t_k) (r (m y_1 … y_n) (m x_1 … x_n)))
  - where r is a well-founded relation

Build the "machine" for a function

```
(funny-fact n i) =
    (IF (<= n 0)
        1
        (IF (= i 0) (* n (funny-fact (- n 1) i))
        (IF (> i 0) (* n (funny-fact (- n i) i))
                    (* n (funny-fact (+ n i) i))))))
```

case	test	recursive calls
1	¬(n ≤ 0) ∧ i == 0	`(funny-fact (- n 1) i)`
2	¬(n ≤ 0) ∧ ¬(i == 0) ∧ i > 0	`(funny-fact (- n i) i)`
3	¬ ¬ ¬	`(funny-fact (+ n i) i)`

Build the "machine" for a function

```
(funny-fact n i) =
    (IF (<= n 0)
        1
        (IF (= i 0) (* n (funny-fact (- n 1) i))
        (IF (> i 0) (* n (funny-fact (- n i) i))
                    (* n (funny-fact (+ n i) i))))))
```

case	test	recursive calls
1	`(AND (NOT (<= n 0)) (= i 0))`	`(funny-fact (- n 1) i)`
2	`(AND (NOT (<= n 0)) (NOT (= i 0)) (> i 0))`	`(funny-fact (- n i) i)`
3	`(AND (NOT (<= n 0)) (NOT (= i 0)) (NOT (> i 0))`	`(funny-fact (+ n i) i)`

Use machine to test measure

- For example, for case 3:

```
(IMPLIES (AND (NOT (<= n 0))
              (NOT (= i 0)
              (NOT (> i 0))
         (< (+ 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:

```
(IMPLIES (is-cons-cell X)
         (< (length (cdr X)) (length X)))
```

How do you guess measure?

```
(IMPLIES (is-cons-cell X)
         (< (length (cdr X) (length X)))
```

- Suppose we have a function append:

```
(append L1 L2) =
   (IF (is-cons-cell L1)
       (cons (car L1) (append (cdr L1) L2))
       L2))
```

case	test	recursive calls
1		

How do you guess measure?

```
(IMPLIES (is-cons-cell X)
         (< (length (cdr X) (length X)))
```

- Suppose we have a function append:

```
(append L1 L2) =
   (IF (is-cons-cell L1)
       (cons (car L1) (append (cdr L1) L2))
       L2))
```

case	test	recursive calls
1	`(is-cons-cell L1)`	`(append (cdr L1) L2))`

Table and induction lemma immediately point out that length works as the measure and < works as the order

Another example

```
(ACK M N) = (IF (= M 0)
                (+ N 1)
                (IF (= N 0)
                    (ACK (- M 1) 1)
                    (ACK (- M 1) (ACK M (- N 1)))))
```

case	test	recursive calls
1	$\neg(M=0) \wedge (N=0)$	(ACK (-M 1) 1)
2	$\neg(M=0) \wedge \neg(N=0)$	ACK (-M 1) ...

Another example

```
(ACK M N) = (IF (= M 0)
                (+ N 1)
                (IF (= N 0)
                    (ACK (- M 1) 1)
                    (ACK (- M 1) (ACK M (- N 1)))))
```

case	test	recursive calls
1	`(AND (NOT (= M 0)) (= N 0))`	`(ACK (- M 1) 1)`
2	`(AND (NOT (= M 0)) (NOT (= N 0)))`	`(ACK (- M 1) (ACK M (- N 1)))` `(ACK M (- N 1))`

Measure?

Another example

- Lexicographical order of (M, N)

```
(LEX-< (cons (- M 1) N)
       (cons M N))

(LEX-< (cons (- M 1) (ACK ...))
       (cons M N))

(LEX-< (cons M (- N 1))
       (cons M N))
```

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 … t_n)** appears in the goal G
- Assume t_1 … t_n are all variables
  - if they are not, things get more complicated
- Assume "machine" for f looks like this:

case	test	recursive calls
1	q_1	r_1^1 ... r_1^{h1}
2	q_2	r_2^1 ... r_2^{h2}
…	…	…
k	q_k	r_k^1 ... r_k^{hk}

Also, assume σ_i^h is subst such that σ_i^h applied to **(f t_1 … t_n)** gives r_i^h

Second part: applying induction

Goal G with sub-term **(f t_1 … t_n)** = body

case	test	recursive calls
1	q_1	r_1^1 ... r_1^{h1}
2	q_2	r_2^1 ... r_2^{h2}
…	…	…
k	q_k	r_k^1 ... r_k^{hk}

Also, assume σ_i^h is subst such that σ_i^h applied to **(f t_1 … t_n)** gives r_i^h

$q_i \wedge \sigma_i'(G) \wedge \ldots \wedge \sigma_i^{h_i}(G) \Rightarrow G$ expanded with body of f

Second part: applying induction

Goal G with sub-term **(f t_1 … t_n)**

case	test	recursive calls
1	q_1	r_1^1 ... r_1^{h1}
2	q_2	r_2^1 ... r_2^{h2}
…	…	…
k	q_k	r_k^1 ... r_k^{hk}

Also, assume σ_i^h is subst such that σ_i^h applied to **(f t_1 … t_n)** gives r_i^h

$q_1 \wedge \sigma_1'(G) \wedge \cdots \wedge \sigma_1^{h_1}(G) \Rightarrow G$
$q_2 \wedge \sigma_2'(G) \wedge \cdots \wedge \sigma_2^{h_2}(G) \Rightarrow G$ → G with (f …) expanded to its body
$q_m \wedge \sigma_k'(G) \wedge \cdots \wedge \sigma_k^{h_k}(G) \Rightarrow G$
$\neg q_1 \wedge \neg q_2 \wedge \cdots \wedge \neg q_k \Rightarrow G$

Simple example from before

```
(sum N) = (IF (<= N 0)
              0
              (+ N (sum (- N 1))))
(defthm (IMPLIES (>= N 0)
                 (= (sum N)
                    (/ (* N (+ N 1)) 2))))
```

G

case	test	recursive calls
1	$\neg(N \leq 0)$	(sum (- N 1))

$\sigma = [N \rightarrow (- N 1)]$

$(\neg N \leq 0) \wedge G[N \rightarrow (- N 1)] \Rightarrow G$
$N \leq 0 \Rightarrow G$

Simple example from before

```
(sum N) = (IF (<= N 0)
              0
              (+ N (sum (- N 1))))
(defthm (IMPLIES (>= N 0)
                 (= (sum N)
                    (/ (* N (+ N 1)) 2))))
```

G

case	test	recursive calls
1	(NOT (<= N 0))	(sum (- N 1)))

$\sigma = [N \rightarrow (- N 1)]$

```
(IMPLIES (AND G[N → (- N 1)] (NOT (<= N 0)))
         G)
(IMPLIES (<= N 0) G)
```

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):

```
(append (reverse A) nil) = (reverse A)
```

- This conjecture is generalized
  - by replacing `(reverse A)` with `L`

```
(append L 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:

```
(= (sum (fib N))
   (/ (* (fib N) (+ (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:

```
(= (sum (fib N))
   (/ (* (fib N) (+ (fib N) 1)) 2))))
```

- Generalize:

```
(= (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: `(>= (fib N) 0)`
- Once this theorem has been proven and labeled "generalize", subsequent generalizations of `(fib N)` to `M` will include the hypothesis `(>= M 0)`.
  - Generalized conjecture in example becomes:

```
(IMPLIES (>= M 0)
         (= (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

$$\frac{}{\Gamma, A \vdash A}\ \text{Assume}$$

$$\frac{\Gamma \vdash A \quad \Gamma \vdash B}{\Gamma \vdash A \wedge B}\ \wedge I \qquad \frac{\Gamma \vdash A \wedge B}{\Gamma \vdash A}\ \wedge E_1 \qquad \frac{\Gamma \vdash A \wedge B}{\Gamma \vdash B}\ \wedge E_2$$

$$\frac{\Gamma \vdash A}{\Gamma \vdash A \vee B}\ \vee I_1 \qquad \frac{\Gamma \vdash B}{\Gamma \vdash A \vee B}\ \vee I_2 \qquad \frac{\Gamma, A \vdash C \quad \Gamma, B \vdash C \quad \Gamma \vdash A \vee B}{\Gamma \vdash C}\ \vee E$$

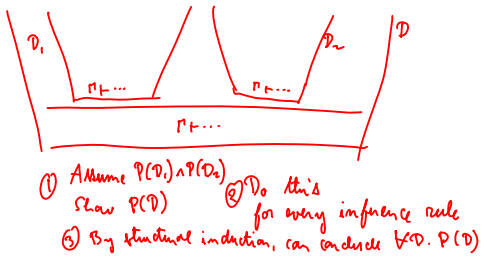
$$\frac{\Gamma, A \vdash B}{\Gamma \vdash A \Rightarrow B}\ \Rightarrow I \qquad \frac{\Gamma \vdash A \quad \Gamma \vdash A \Rightarrow B}{\Gamma \vdash B}\ \Rightarrow E$$

Proving properties of derivations

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

Proving properties of derivations

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



Twelf

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