Staff Solutions to In-Class Problems Week 5, Fri.

STAFF NOTE: Recursive games, Ch. 7.5

Problem 1.
We’re going to characterize a large category of games as a recursive data type and then prove, by structural induction, a fundamental theorem about game strategies. We are interested in two person games of perfect information that end with a numerical score. Chess and Checkers would count as value games using the values $1, -1, 0$ for a win, loss or draw for the first player. The game of Go really does end with a score based on the number of white and black stones that remain at the end.

Here’s the formal definition:

Definition. Let $V$ be a nonempty set of real numbers. The class $VG$ of $V$-valued two-person deterministic games of perfect information is defined recursively as follows:

Base case: A value $v \in V$ is a VG known as a payoff.

Constructor case: If $G$ is a nonempty set of VG’s, then $G$ is a VG. Each game $M \in G$ is called a possible first move of $G$.

STAFF NOTE: In all the games like this that we’re familiar with, there are only a finite number of possible first moves and a bound on the possible length of play. It’s worth noting that the definition of VG does not require this. An infinite VG might have plays of every finite length, but it can’t have an infinite play. Since finiteness is not needed to prove any of the results below, it would potentially even be misleading to assume it. Later, we’ll suggest how games with an infinite number of possible first moves might come up.

Infinite games do have their uses in the study of set theory and logic.

A strategy for a player is a rule that tells the player which move to make whenever it is their turn. That is, a strategy is a function $s$ from games to games with the property that $s(G) \in G$ for all games $G$. Given which player has the first move, a pair of strategies for the two players determines exactly which moves the players will choose. So the strategies determine a unique play of the game and a unique payoff.\footnote{We take for granted the fact that no VG has an infinite play. The proof of this by structural induction is essentially the same as that for win-lose games given in the text.}

The max-player wants a strategy that guarantees as high a payoff as possible, and the min-player wants a strategy that guarantees as low a payoff as possible.

The Fundamental Theorem for deterministic games of perfect information says that in any game, each player has an optimal strategy, and these strategies lead to the same payoff. More precisely,

Theorem (Fundamental Theorem for VG’s). Let $V$ be a finite set of real numbers and $G$ be a $V$-valued VG. Then there is a value $v \in V$, called a max-value $\max_G$ for $G$, such that if the max-player moves first,

- the max-player has a strategy that will finish with a payoff of at least $\max_G$, no matter what strategy the min-player uses, and
• the min-player has a strategy that will finish with a payoff of at most \( \max_G \), no matter what strategy the max-player uses.

It’s worth a moment for the reader to observe that the definition of \( \max_G \) implies that if there is one for \( G \), it is unique. So if the max-player has the first move, the Fundamental Theorem means that there’s no point in playing the game: the min-player may just as well pay the max-value to the max-player.

(a) Prove the Fundamental Theorem for VG’s.

*Hint:* VG’s are a recursively defined data type, so the basic method for proving that all VG’s have some property is structural induction on the definition of VG. Since the min-player moves first in whichever game the max-player picks for their first move, the induction hypothesis will need to cover that case as well.

**COMMENTS:**

- PS\_VG
- formerly CP\_VG
- revision of PS\_50\_point\_games, PS\_value\_games by ARM 3/4/16, revised 3/8/17
- S16.ps4, S17.cp5f

**keywords** = \[ recursive\_data structural\_induction games perfect\_information max-value min-value \]

**Solution.** The proof is by structural induction on the definition of a game \( G \in \text{VG} \). The induction hypothesis \( P(G) \) will be that

\[
P(G) \iff G \text{ has both a max-value } \max_G \text{ and a min-value } \min_G,\]

where a min-value for \( G \) is defined in the same way as a max-value except that max and min are switched. To be precise, a *min-value* for \( G \) is a value \( \min_G \in V \) such that if the min-player moves first,

• the min-player has a strategy that will finish with a payoff of *at most* \( \min_G \), no matter what strategy the max-player uses, and

• the max-player has a strategy that will finish with a payoff of *at least* \( \min_G \), no matter what strategy the min-player uses.

In general, the max-value and the min-value for \( G \) are different, since playing \( G \) with the max-player moving first is quite different from playing \( G \) with the min-player moving first.

**STAFF NOTE:** Add problem: Prove that the max-value of \( G \) is always \( \geq \) the min-value. ■

*Proof. Base case:* \( (G = v \in V) \). The max-value and min-value will be \( v \). The strategy for the players—regardless of who moves first—is “Suck it up, the game is over and you have to take \( v \) as your payoff.” This proves \( P(G) \).

**Constructor case:** \( (G \) is a nonempty set of VG’s). By structural induction we may assume that each \( M \in G \) has a a min-value \( \min_M \). Define

\[
v_{\text{max}} \iff \max\{|\min_M | M \in G\}.
\]

This max will exist because \( V \) is finite.

We claim that \( \max_G = v_{\text{max}} \). To prove this, we begin by describing a strategy with the max-player moving first that guarantees a payoff of at least \( v_{\text{max}} \). Namely, the max-player’s first move should be to some \( M \in G \) such that \( \min_M = v_{\text{max}} \). After this first move, the max-player has a strategy that, by definition of \( \min_M \), guarantees a payoff off of at least \( \min_M = v_{\text{max}} \) in the game \( M \) where the min-player moves first. The max-player should then use this strategy in the game \( M \).
Moreover, with the max-player moving first, the min-player has a strategy for \( G \) that guarantees a payoff of at most \( v_{\text{max}} \). Namely, if the first player's first move is to \( M \in G \), the min-player has a strategy that, by definition of \( \min_M \), guarantees a payoff of at most \( \min_M \) in the game \( M \) where the min-player moves first. The min-player should then use this strategy in the game \( M \). Since by definition \( v_{\text{max}} \geq \min_M \), this strategy guarantees the min-player a payoff in \( G \) of at most \( v_{\text{max}} \).

This proves that \( v_{\text{max}} \) has the properties required of \( \max_G \), and since \( \max_G \) is by definition unique, it follows that \( \max_G = v_{\text{max}} \).

Reasoning as above with max and min switched similarly proves that \( G \) has a min-value. This proves \( P(G) \), and we conclude by structural induction that \( P(G) \) is true for every \( VG \) with a finite number of payoff values. In particular every \( G \) has a max-value, as claimed.

\spiel{(b) (OPTIONAL). State some reasonable generalization of the Fundamental Theorem to games with an infinite set \( V \) of possible payoffs.}

\textbf{Solution.} A straightforward generalization is to redefine the max-value to be the least upper bound of the values for which the max-player has a guaranteed strategy. With an infinite number of possible payoffs, some games may now have an infinite max-player value, and even in the case that the max-value is finite, it might not be guaranteed exactly by player strategies. Rather, for any \( v < \max_G \), when the max-player moves first, there will be a strategy for the max-player that guarantees a payoff of at least \( v \), and there will be a strategy for the min-player that guarantees a payoff of at most \( v + (\max_G - v)/2 \).

\spiel{Problem 2.}

In this problem we’ll need to be careful about the propositional \textit{operations} that apply to truth values and the corresponding \textit{symbols} that appear in formulas. We’ll restrict ourselves to formulas with \textit{symbols} \texttt{And} and \texttt{Not} that correspond to the operations \texttt{AND}, \texttt{NOT}. We will also allow the constant symbols \texttt{True} and \texttt{False}.

\spiel{(a) Give a simple recursive definition of \textit{propositional formula} \( F \) and the set \texttt{pvar}(\( F \)) of propositional variables that appear in it.}

\textbf{COMMENTS:}

\begin{itemize}
  \item CP\_recursive\_prop\_form\_eval
  \item ARM 3/4/16
\end{itemize}

\textbf{keywords = [ propositional\_formula propositional\_variable recursive environment structural\_induction induction evaluate ]}

\textbf{Solution. Base cases:}

\begin{itemize}
  \item A propositional \texttt{variable} \( P \) is a propositional formula and the set \texttt{pvar}(\( P \)) of variables that appear in it is \( \{ P \} \).
  \item A propositional \texttt{constants} \texttt{True}, \texttt{False} are propositional formulas, and \( \texttt{pvar(True)} = \texttt{pvar(False)} ::= \emptyset \).
\end{itemize}

\textbf{Constructor cases:} If \( F, G \) are propositional formulas, then so are

\begin{itemize}
  \item \( (F \texttt{And} G) \), and \( \texttt{pvar((F And G)) ::= pvar(F) \cup pvar(G)} \).
  \item \( \texttt{Not}(F) \), and \( \texttt{pvar(Not(F)) ::= pvar(F)} \).
\end{itemize}
Let \( V \) be a set of propositional variables. A truth environment \( e \) over \( V \) assigns truth values to all these variables. In other words, \( e \) is a total function,

\[
e : V \to \{T, F\}.
\]

(b) Give a recursive definition of the truth value, \( \text{eval}(F, e) \), of propositional formula \( F \) in an environment \( e \) over a set of variables \( V \supseteq \text{pvar}(F) \).

**Solution.** Base cases:

\[
\begin{align*}
\text{eval}(P, e) & : = e(P), \\
\text{eval}(\text{True}, e) & : = T, \\
\text{eval}(\text{False}, e) & : = F.
\end{align*}
\]

Constructor cases: If \( F, G \) are propositional formulas, then

\[
\begin{align*}
\text{eval}(F \text{ And } G, e) & : = \text{eval}(F, e) \text{ AND } \text{eval}(G, e), \\
\text{eval}(\text{Not}(F), e) & : = \text{NOT}(\text{eval}(F, e)).
\end{align*}
\]

Clearly the truth value of a propositional formula only depends on the truth values of the variables in it. How could it be otherwise? But it’s good practice to work out a rigorous definition and proof of this assumption.

(c) Give an example of a propositional formula containing the variable \( P \) but whose truth value does not depend on \( P \). Now give a rigorous definition of the assertion that “the truth value of propositional formula \( F \) does not depend on propositional variable \( P \).”

**Hint:** Let \( e_1, e_2 \) be two environments whose values agree on all variables other than \( P \).

**Solution.** A simple example is the formula \( P \text{ And } \text{Not}(P) \) whose truth value is \( F \) in all environments.

Not “depending on \( P \)” means that whether \( P \) is \( T \) or \( F \), the truth value of \( F \) comes out the same. More precisely this means that if two environments agree on the truth values of all variables other than \( P \), then the value of \( F \) is the same in both environments:

**Definition.** The truth value of a propositional formula \( F \) does not depend on propositional variable \( P \) iff whenever \( e_1, e_2 \) are environments over \( V_0 \supseteq \text{pvar}(F) \cup \{P\} \) and \( e_1(Q) = e_2(Q) \) for all variables \( Q \in V_0 \) other than \( P \), then

\[
\text{eval}(F, e_1) = \text{eval}(F, e_2).
\]

(d) Give a rigorous definition of the assertion that “the truth value of a propositional formula only depends on the truth values of the variables that appear in it,” and then prove it by structural induction on the definition of propositional formula.

**Solution.**
**Definition.** The truth value of a propositional formula $F$ depends only on the truth values of $\text{pvar}(F)$ iff for all $P \notin \text{pvar}(F)$, $F$ does not depend on $P$.

To prove that all formulas $F$ have this property, suppose $P \notin \text{pvar}(F)$, and $e_1, e_2$ are environments over $V_0 \supseteq \text{pvar}(F) \cup \{P\}$ and $e_1(Q) = e_2(Q)$ for all variables $Q \in V_0$ other than $P$. Then we will prove by structural induction on $F$ that

$$\text{eval}(F, e_1) = \text{eval}(F, e_2).$$

**(1)**

**Base case** $F$ can’t be the variable $P$, so the only cases are when $F$ is a truth constant. Now if $F$ is $\text{True}$, then

$$\text{eval}(F, e_1) := \text{T} = \text{eval}(F, e_2).$$

by definition of eval. The same reasoning applies if $F$ is $\text{False}$, proving that (1) holds on the base case.

**Constructor case** ($F$ is $(G \text{ And } H)$). Then by structural induction hypothesis,

$$\text{eval}(G, e_1) = \text{eval}(G, e_2)$$
$$\text{eval}(H, e_1) = \text{eval}(H, e_2),$$

so

$$\text{eval}(F, e_1) = \text{eval}(G, e_1) \text{ AND } \text{eval}(H, e_1)$$
$$= \text{eval}(G, e_2) \text{ AND } \text{eval}(H, e_2)$$
$$= \text{eval}((G \text{ And } H), e_2)$$
$$= \text{eval}(F, e_2).$$

proving (1) in this case.

**STAFF NOTE:** We’ve made implicit use of the fact that

$$\text{pvar}(G) \subseteq \text{pvar}(G \text{ And } H)$$

by definition of $\text{pvar}(F)$.

**Constructor case** ($F$ is $\text{Not}(G)$). Similar, easier proof.

**(e)** Now we can formally define $F$ being valid. Namely, $F$ is valid iff

$$\forall e. \text{eval}(F, e) = \text{T}.$$

Give a similar formal definition of formula $G$ being unsatisfiable. Then use the definition of eval to prove that a formula $F$ is valid iff $\text{Not}(F)$ is unsatisfiable.

**Solution.** Let $e$ range over environments on $\text{pvar}(G)$. Then $G$ is unsatisfiable iff

$$\overline{\forall e. \text{eval}(G, e) = \text{T}}$$

$F$ is valid iff

$$\forall e. \text{eval}(F, e) = \text{T}$$

iff

$$\forall e. \overline{\text{eval}(F, e)} = \text{F}$$

(by def of operation $\text{Not}$)

iff

$$\forall e. \text{eval}(\overline{F}, e) = \text{F}$$

(by def of eval, $\overline{}$)

iff

$$\overline{\forall e. \text{eval}(\overline{F}, e) = \text{T}}$$

(by De Morgan)

iff

$$\overline{\text{Not}(F)}$$

is unsatisfiable

(by def of unsatisfiable).
Optional

Problem 3.

Nim is a two-person game that starts with some piles of stones. A player’s move consists of removing one or more stones from a single pile. Player-1 and player-2 alternate making moves, and whoever takes the last stone wins.

So if there is only one pile, then the first player to move wins by taking the whole pile. On the hand, if the game starts with just two piles, each with the same number of stones, then the player who moves second can guarantee a win simply by mimicking the first player. This means, for example, that if the first player removes three stones from one pile, then the second player removes three stones from the other pile. At this point, it’s worth thinking for a moment about why the mimicking strategy guarantees a win for the second player.

It turns out there is a winning strategy for one of the players that is easy to carry out but is not so obvious. To explain the winning strategy, we need to think of a number in two ways: as a nonnegative integer and as the bit string equal to the binary representation of the number—possibly with leading zeroes.

For example, the XOR of numbers \( r, s, \ldots \) is defined in terms of their binary representations: combine the corresponding bits of the binary representations of \( r, s, \ldots \) using XOR, and then interpret the resulting bit-string as a number. For example,

\[
2 \text{ XOR } 7 \text{ XOR } 9 = 12
\]

because, taking XOR’s down the columns, we have

\[
\begin{array}{cccc}
0 & 0 & 1 & 0 \ \\
0 & 1 & 1 & 1 \ \\
1 & 0 & 0 & 1 \ \\
1 & 1 & 0 & 0
\end{array}
\]

This is the same as doing binary addition of the numbers, but throwing away the carries (see Problem 3.6).

The XOR of the numbers of stones in the piles is called their Nim sum. In this problem we will verify that if the Nim sum is not zero on a player’s turn, then the player has a winning strategy. For example, if the game starts with five piles of equal size, then the first player has a winning strategy, but if the game starts with four equal-size piles, then the second player can force a win.

(a) Prove that if the Nim sum of the piles is zero, then any one move will leave a nonzero Nim sum.

Solution. When a player removes stones from a pile, the binary representation of the number of stones in the pile changes. Since the other piles stay the same, the bits in the Nim sum change at the positions where bits changed in the binary representation. Since all the bits in the Nim sum were initially equal to zero, the changed bits must have become ones. That makes the Nim sum nonzero.

(b) Prove that if there is a pile with more stones than the Nim sum of all the other piles, then there is a move that makes the Nim sum equal to zero.

Solution. If there is a size \( r \) greater than the the Nim sum \( n \) of the other piles, then remove stones from this pile until it is of size \( n \). Now the Nim sum of all the piles becomes \( n \text{ XOR } n = 0 \).
(c) Prove that if the Nim sum is not zero, then one of the piles is bigger than the Nim sum of all the other piles.

*Hint:* Notice that the largest pile may not be the one that is bigger than the Nim sum of the others; three piles of sizes 2,2,1 is an example.

**Solution.** Suppose the Nim sum of all the piles is $s$, and the high order digit of the Nim sum, that is, the leftmost 1 in the binary representation of $s$, occurs at the $n$th position. Since this bit of the Nim sum equals 1, there must be at least one pile of $r$ stones such that the $n$th bit of $r$ is also 1. This is the pile to choose.

Notice that, since $r \text{ XOR } r = 0$, we have $r \text{ XOR } (r \text{ XOR } s) = s$. This means that $r \text{ XOR } s$ is the Nim sum of all the piles besides the pile of $r$. Now $r \text{ XOR } s$ has $n$th bit 0, since both $r$ and $s$ have $n$th bit 1. Also, since each bit of $s$ above the $n$th is zero, each bit of $r$ above the $n$th must equal the corresponding bit of $r \text{ XOR } s$. So $r$ and $r \text{ XOR } s$ are the same above the $n$th bit, and $r$ is bigger at the $n$th bit. This implies that $r > r \text{ XOR } s$. That is, $r$ is bigger than the Nim sum of all the other piles, as claimed.

(d) Conclude that if the game begins with a nonzero Nim sum, then the first player has a winning strategy.

*Hint:* Describe a preserved invariant that the first player can maintain.

**Solution.** By part (a), whenever it is the second player’s turn to move, and the Nim sum is zero, the second player will leave a Nim sum that is not zero.

By parts (b) and (c), whenever it is the first player’s turn to move, and the Nim sum is not zero, the first player can leave the Nim sum equal to zero.

If the first player always moves to set a nonzero Nim sum to zero, then the Nim sum being nonzero on the first player’s turn and zero on the second player’s turn is a preserved invariant of the game.

Since the total number of stones decreases at every move, the game must eventually end with no stones are left. But the Nim sum of no stones is zero, so by the preserved invariant, it must happen on the second player’s turn. That is, the second player must lose.

(e) (Extra credit) Nim is sometimes played with winners and losers reversed, that is, the person who takes the last stone loses. This is called the *misère* version of the game. Use ideas from the winning strategy above for regular play to find one for *misère* play.

**Solution.** Follow the same strategy until just 2 piles remain, or there are only piles of size 1, and adapt for *misère.*