Editor: **Michael Wooldridge,** University of Oxford, michael.wooldridge@cs.ox.ac.uk

Thinking Backward with Professor Zermelo

Michael Wooldridge, *University of Oxford*

Much of game theory (and indeed much of economics in general) is concerned with devising models of strategic scenarios.[1] Because a key concern in game theory is to try to understand what constitutes a rational decision in strategic settings, the models constructed by game theorists attempt to capture all (and only) the information about the setting that is germane to the decisions that players must make, abstracting away as much irrelevant detail as possible. One of the most important and intuitively familiar game models is intended to capture situations where players make a series of decisions over time, and where players have information about previous players' moves. These games are called *extensive form games*. If the games in question are finite (that is, they do not go on forever), backward induction is a useful technique for analyzing them. Backward induction is an extremely powerful technique that has been applied not just to games but also to many other problems in computer science, where it is often known as Zermelo's algorithm, after Ernst Zermelo (1871–1953), who used it to analyze the game of chess.[2]

In this article, I will introduce extensive games of perfect information, the simplest and best-studied class of extensive form games, and show how backward induction can be used to analyze such games. I conclude by discussing some of the apparent paradoxes that can arise when using backward induction to analyze games.

Extensive Form Games

This column has shared many examples of games, but the most fundamental type of game is the strategic form game. In such a game, each player is presented with a set of possible choices and must select just one of these, in complete ignorance of the other players' choices. When all players have made their choices, the resulting collection of choices is the outcome of the game, which determines for each player a numeric utility indicating how good or bad that outcome is for that player. Strategic form games are usually described as games of simultaneous moves, and this assumption is often taken literally to mean that players make their choices at the same time. In fact, the assumption of simultaneous moves is really an informational assumption, which is intended to describe the fact that players have no knowledge of each other's choices when they make their own. Games like the prisoner's dilemma and the game of chicken are examples of strategic form games, and hence games of simultaneous moves. In an extensive form game, the assumption is that players do not make just one choice but possibly a series of choices over time. Moreover, extensive form games let us represent the information that agents have about the choices made by other players when they make their own choice. In the version of extensive form games discussed in this article, the assumption is that players have perfect information: that is, a player is completely and correctly aware of all the choices that were made preceding his choice. I leave the discussion of imperfect information to another day.

Extensive form games are perhaps best understood through the notion of a game tree (see Figure 1). In this game, there are two players, A and B. Each player makes just one move in the game, but unlike games of simultaneous moves, they take turns to move, just as in a game like chess. As shown in the right of the figure, A moves first, then B. There are four possible outcomes to the game, and in each outcome, one of the players is the winner (in this game, a draw is not possible).

More formally, the extensive form game in Figure 1 is defined by a game tree consisting of seven nodes, which are labeled $n0$ to $n6$:

- There is a single node in an extensive form game tree that has no incoming edges; in Figure 1, this node is labeled $n0$. This root node indicates

1541-1672/15/$31.00 © 2015 IEEE
Published by the IEEE Computer Society

where the game starts (that is, the first choice, or move, in the game).

- Edges that leave a node indicate the possible choices that are available when the game reaches that node. Each such edge is labeled with a possible choice. Thus, in the root node *n*0, the possible moves are *L* and *R*. In node *n*1, the choices are *l* or *r* (similarly for node *n*2).
- If a node has at least one outgoing edge, it is called a decision node. Each decision node is associated with a player—the player who makes the choice when the game reaches that node. In Figure 1, the players responsible for making choices are indicated at the right of the figure—thus, A moves first, followed by B. Agent A has just one decision node (*n*0), whereas agent B has two decision nodes (*n*1 and *n*2). Nodes that are not terminal are sometimes called interior nodes.
- A node that has no outgoing edges is called a terminal node, and it corresponds to an outcome of the game. There are four terminal nodes for the game in Figure 1: *n*3, *n*4, *n*5, and *n*6. Because terminal nodes correspond to a game's outcome, they are labeled with the payoffs that the players would receive if that outcome was reached. The game in Figure 1 is a "win-lose" game in the sense that in each outcome of the game, one player will be declared a winner. The labels in nodes *n*3 through *n*6 indicate which player wins in the corresponding outcome. Thus, in *n*3, player A wins, whereas in *n*4, player B wins.

Now, a strategy for each player *i* is a rule that says for each of *i*'s decision nodes what choice player *i* makes in that node. When playing an extensive form game, each player must choose such a strategy.

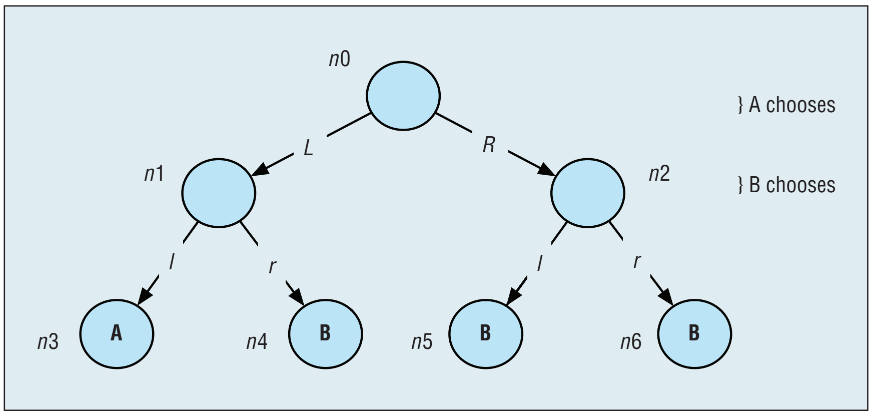

Figure 1. An extensive form game. The game is defined by a game tree comprising seven nodes (*n*0 to *n*6). Node *n*0 is a decision node for player A; nodes *n*1 and *n*2 are decision nodes for player B. All other nodes are terminal nodes.

Let's now analyze this game using backward induction. Because the game is a win-lose game, we will use backward induction to tell us which player has a winning strategy—that is, a strategy that guarantees to win the game for the corresponding player. The basic idea in backward induction is to analyze the game starting at the end, and then work backward. Recall from Figure 1 that each terminal node is labeled with the outcome of the game—in the game we are presently considering, the outcome is simply which player won the game. As we analyze the game, we gradually extend this labeling, node by node, working backward from the end of the game to the beginning, until we have finally labeled the root node of the game: the label of the root node indicates what the outcome of the game will be, assuming rational action by the players. The key idea behind this process is that, if we consider a node *n* in which all the children of that node have been labeled, from this we can deduce what the player who makes a choice in node *n* will do, and we can then label *n* with the outcome that would result if this choice was made. We can at that point effectively ignore everything below *n*—the label indicates what the outcome of the game would be, if the game ever reached *n*. By iterating this process, we progressively work from the end of the game back to the beginning, until we label the root node, at which point we know what the outcome of the game will be.

In more detail, the backward induction algorithm can be described as follows:

```
1. Repeat the following…
2. For each unlabeled node $n$ in the game:
3. If all the children of $n$ have been labeled with an outcome, then:
4. Label node $n$ with the outcome from the child of $n$ that would be best for the player who makes a decision at $n$ (if there are multiple equally good outcomes, we can choose any one of them—it doesn't matter which)
5. …until all nodes have been labeled.
```

Two observations are worth making. First, because the game is finite, and all terminal nodes are labeled, the algorithm I just described is guaranteed to terminate. Second, and perhaps less obviously, the algorithm works in time polynomial in the number of nodes in the graph. The latter, in particular, is an extremely desirable property.

So, let's apply this algorithm to Figure 1. To help visualize the algorithm at work, we follow common practice, and when we have introduced a new

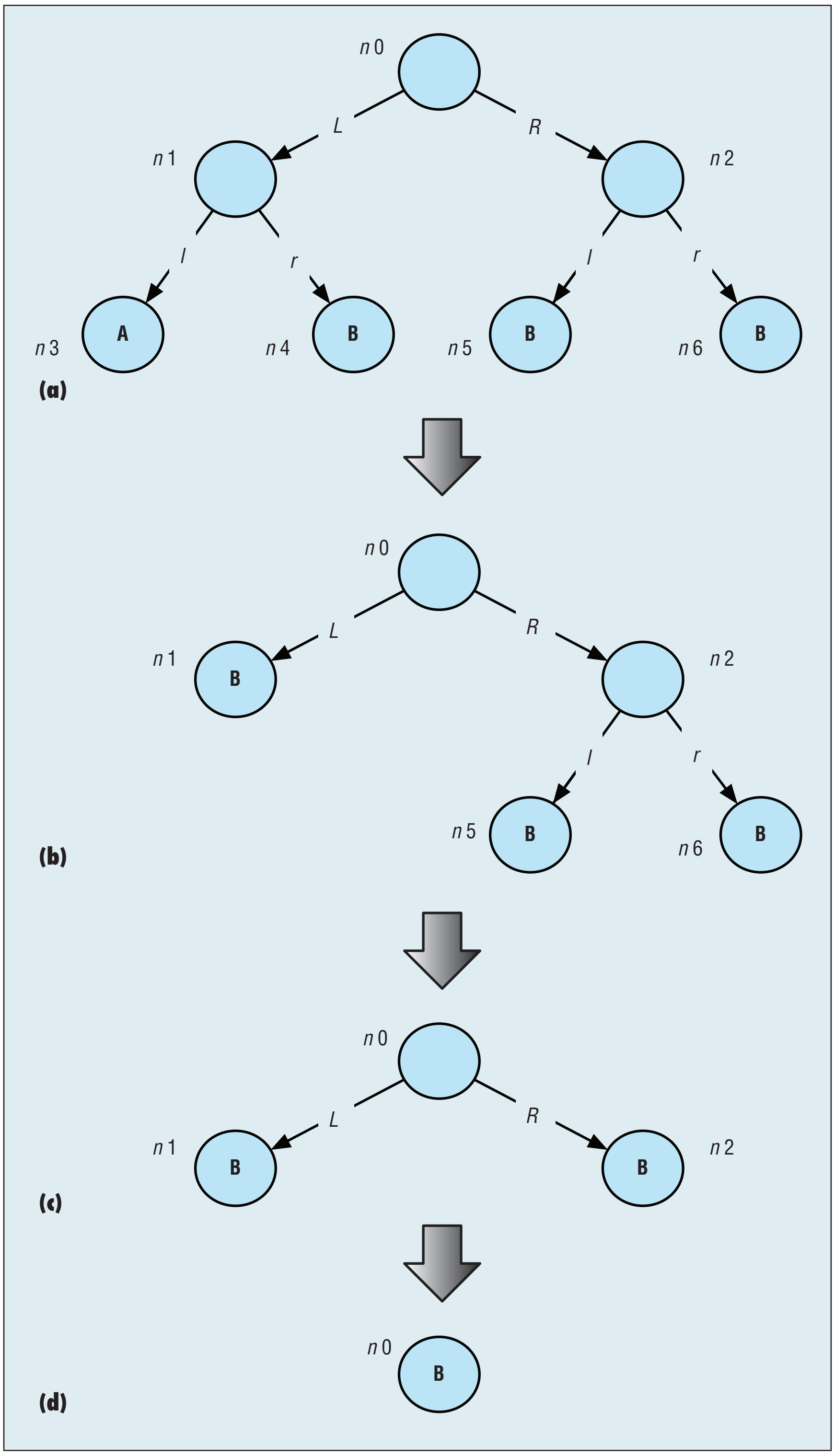

Figure 2. Backward induction at work. We systematically label each node with the outcome of the game that would result if we reach that node, considering each node in which all children have been labeled. (a) The original game tree; (b) after applying backward induction on node *n*1, nodes *n*3 and *n*4 are deleted; (c) after applying backward induction on node *n*2, nodes *n*5 and *n*6 are deleted; and (d) after applying backward induction on node *n*0, nodes *n*1 and *n*2 are deleted, and the outcome of the game is that B wins.

label for a node, we delete the game tree beneath that node, because it no longer needs to be considered in the algorithm. So, consider Figure 2:

- We start with node *n*1 (see Figure 2a). This is a decision node for player B, who must choose between actions *l* (leading to a win for player A) and *r* (leading to a win for player B). Clearly, player B would choose *r*, and so we label node *n*1 with B, and delete the tree beneath *n*1. The result is shown in Figure 2b.
- We then consider node *n*2. Again, this is a decision node for player B, who must choose between *l* (a win for B) and *r* (again, a win for B). It doesn't matter which action B chooses; they are both equally good for him, so let's say he chooses *l*. We thus label node *n*2 with B, and delete the tree below *n*2, resulting in Figure 2c.
- We now only have one unlabeled node left to consider: *n*0. This is a decision node for player A, who must choose between *L* (resulting in a win for B) and *R* (also resulting in a win for B). These both represent bad outcomes for A, of course, but nevertheless A must choose between them. A is clearly going to be indifferent about them, so let's say A chooses *L*; we therefore label *n*0 with B (see Figure 2d).

So, the backward induction tells us that the result of the game is a win for player B. We can read off B's winning strategy from the process of backward induction: if the game reaches *n*1, the player chooses *r*; if the game reaches *n*2, the player chooses *l*. This strategy will guarantee a win for player B no matter what strategy player A chooses.

As I noted in the introduction, mathematician Ernst Zermelo used backward induction to analyze the game of chess. The result he proved applies to all two-player finite extensive form games of perfect information: he showed that in every such game, one of the players must have a winning strategy. This result is now known as Zermelo's theorem.

The Kidnap Game

The example given earlier is a win-lose game, but more generally in game theory, we think of outcomes to games as being labeled not with winning players, but with payoff profiles—that is, a list of numeric values, one for each player, which indicates how good or bad the outcome is for that player. For such games, the relevant question is not whether a game is a "win" for a particular player but what strategies form a Nash equilibrium. Recall that a Nash equilibrium is a collection of strategies, one for each player, such that no player could benefit by doing anything other than his part of the strategy profile.

Let's see another extensive form game, this time one that includes payoffs for players. The game is called the kidnap game. The story used to illustrate it is slightly morbid and possibly not in the best of taste—apologies for that.

Anne has kidnapped Bob and demanded a ransom for his safe return. The ransom has been paid, and A now has to choose between releasing B as agreed or murdering him. If B is released, he can choose between telling the police about A's identity or keeping quiet. B has promised A that if he is freed, he will keep quiet: he will not inform the police about A's identity. Clearly, being murdered is the worst outcome for B; the best outcome for B would be to be released and to inform the police about A (because A's capture would prevent further kidnappings). The worst outcome for A would be if B was released and informed the police.

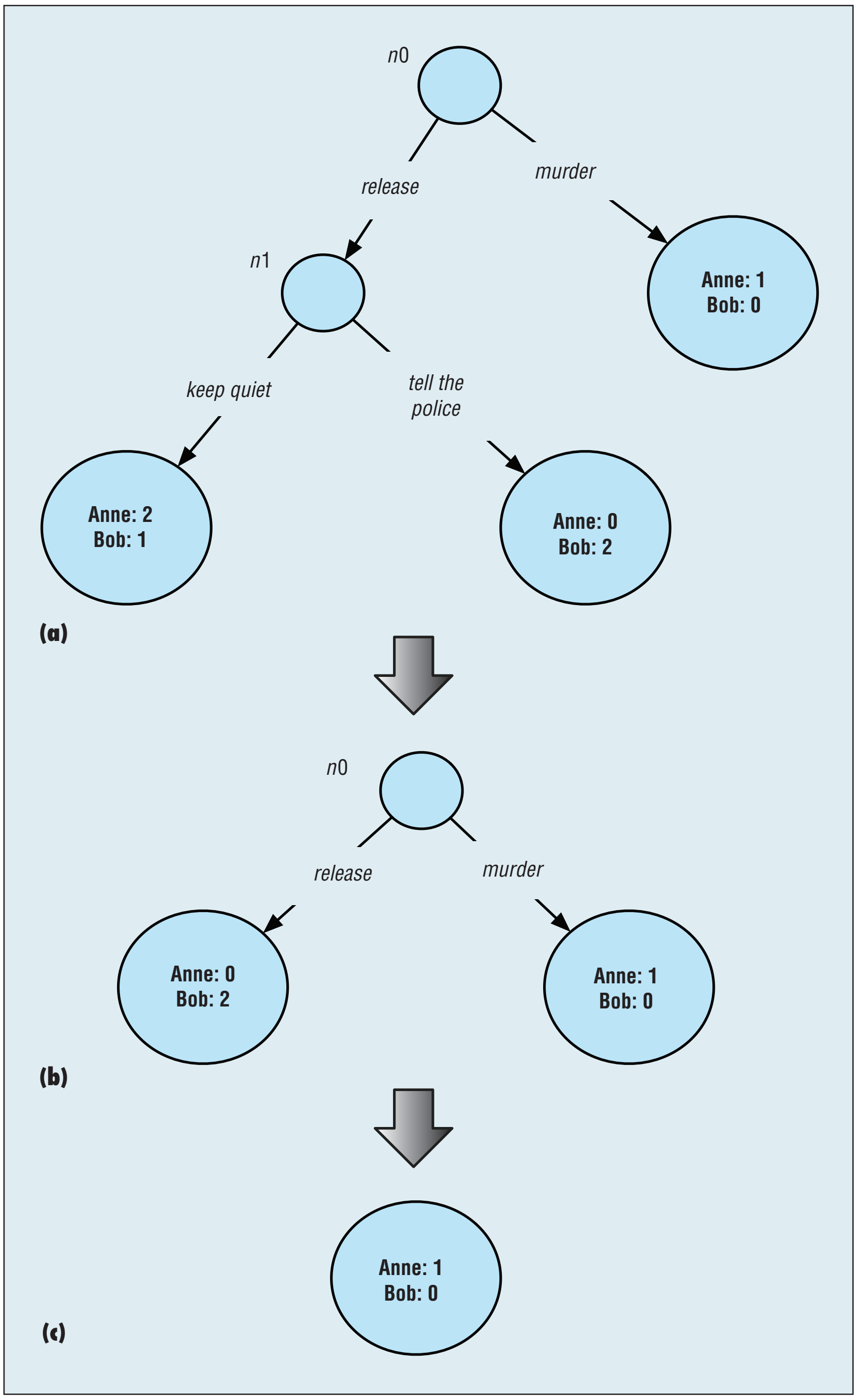

Figure 3. The kidnap game. Anne has kidnapped Bob and received a ransom payment for his safe return. Does it make sense for her to return him safely, or should she take a more grisly course of action? (a) The original game tree; (b) after applying backward induction on node *n*1; and (c) after applying backward induction on node *n*0, the outcome of the game is that Anne will murder Bob.

Figure 3a illustrates the game. There are just two decision nodes: $n0$ (a decision node for A) and $n1$ (a decision node for B). Backward induction proceeds exactly as before:

- Consider node $n1$. B has a choice between keeping quiet (giving him a payoff of 1) or telling the police

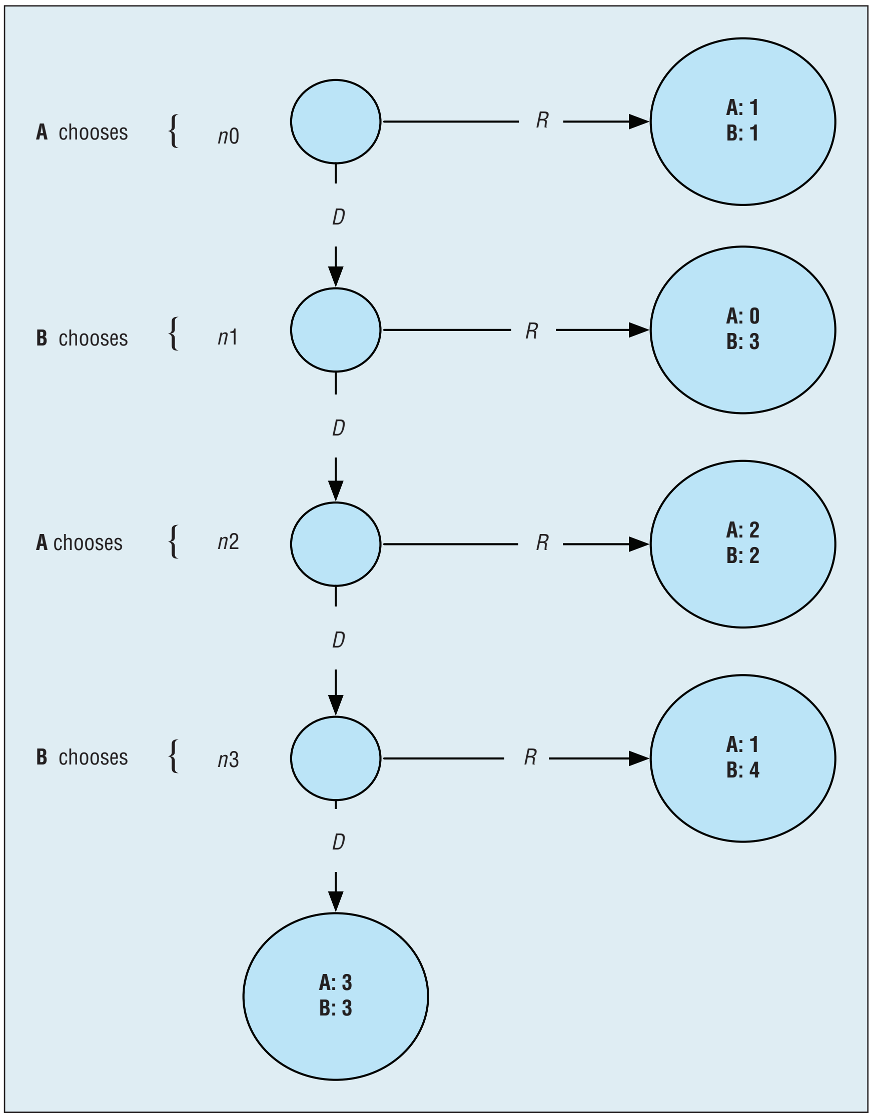

Figure 4. The centipede game. In this classic extensive form game, backward induction yields strategies that seem to be paradoxical.

(payoff of 2). Clearly, he would prefer the payoff of 2, so we label *n*1 with the payoffs corresponding to telling the police and delete the child nodes of *n*1.

- Now consider *n*0, a decision node for A (Figure 3b). The choices are "release" or "murder"; releasing would give A a payoff of 0, and murdering would give A a payoff of 1. Clearly, A would prefer a payoff of 1. We therefore label *n*0 with the payoffs corresponding to the murder outcome (Figure 3c).

So, the gruesome outcome of the kidnap game, according to backward induction, is that Anne will murder Bob. The strategies we obtain through backward induction (Anne murders; Bob tells the police) form a Nash equilibrium pair. And, indeed, an attractive property of backward induction is that the strategies we obtain by applying this technique are guaranteed to be Nash equilibria.

Alert readers will notice that the strategies we obtained when applying backward induction in this scenario have a slightly strange property: they say that Bob should tell the police in node *n*1 even though this node will not be reached when playing the game. Technically, we say that *n*1 is not on the equilibrium path. This situation seems odd, but the point is that when formulating a strategy for Anne, she needs to take into account what Bob would do if node *n*1 were reached. This kind of reasoning (about a scenario that does not in fact occur) is called *counterfactual reasoning*.

Note that a Nash equilibrium identifies a single path in the game—the equilibrium path—and only requires that the players' choices are rational for choices on the equilibrium path. Off the equilibrium path, Nash equilibria may suggest suboptimal moves. This observation motivated Reinhard Selten to propose a refinement of the Nash equilibrium solution concept for extensive form games, which is now known as *subgame perfect equilibrium*. Informally, a collection of strategies is a subgame perfect equilibrium if they form a Nash equilibrium at every node in the extensive form game tree (technically, if they form a Nash equilibrium at every subgame). For this work, Selten was a joint recipient of the 1994 Nobel Memorial Prize in Economic Sciences, along with John Nash and John Harsanyi. Now, an important property of backward induction when applied to extensive form games of the type we have considered here is that, not only does it guarantee to find Nash equilibrium strategies, backward induction in fact guarantees to give subgame perfect equilibrium strategies.

The Centipede Game

I hope by now I have convinced you that backward induction is a powerful and convincing technique for analyzing extensive form games. But, as with all game theoretic solution concepts, it is possible to find situations under which backward induction yields seemingly paradoxical results. To illustrate this point, consider the centipede game (see Figure 4). Our backward induction analysis proceeds as follows:

- Node *n*3 is a decision node for B, who has a choice of *R* (giving payoff of 4 to B) and *D* (giving payoff of 3). Clearly, B would choose *R*, and so we label *n*3 with payoffs (A: 1, B: 4).
- Node *n*2 is a decision node for A, who has a choice between *R* (payoff of 2) and *D* (payoff of 1); so, A will choose *R*, and we label *n*2 with (A: 2, B: 2).
- Node *n*1 is a decision node for B, who has a choice between *R* (payoff of 3) and *D* (payoff of 2). So, B will choose *R*, and we label *n*1 with (A: 0, B: 3).
- Finally, *n*0 is a decision node for A, who must choose between *R* (payoff of 1) and *D* (payoff of 0). Clearly, A will choose *R*, and so we label *n*0, the root node of the game, with (A: 1, B: 1).

Thus, the subgame perfect equilibrium strategies for this game result in the game ending after one move: player A chooses *R* on the first move, and both players receive a payoff of 1. Why is this paradoxical? First, and most obviously, they would both have benefited by reaching the bottommost outcome, in which they would both have received a payoff of 3. This is an example of where game theoretic analysis leads to the selection of an outcome that is socially undesirable (another well-known example is the prisoner's dilemma). Second, experimental results indicate that human players don't in fact play subgame perfect strategies in the centipede game. Although the version given earlier contains only four decisions, the basic game structure can be continued for an arbitrary number of rounds, and for large numbers of rounds, human players will tend to play *D* for at least a few rounds before someone plays *R*.

Zermelo's Algorithm beyond Game Theory

Zermelo's backward induction algorithm is not just one of the cornerstone algorithms in game theory—it has applications across the whole of computer science. It is a classic example of dynamic programming, in which a solution to an overall problem is systematically built up from the solutions to smaller problems. Two particularly important uses of Zermelo's algorithm are worth highlighting.

First, Zermelo's algorithm is one of the fundamental algorithms in model checking, which is a research domain concerned with verifying that computer systems satisfy specifications expressed as temporal logic formulas.[3] In model checking, a graph structure called a Kripke structure is used to represent a computer system's behavior, and the goal is to label states within this structure with the temporal logic formulae true at those states. Backward induction is used to build this labeling, working in exactly the same way as described earlier.

Second, Zermelo's algorithm corresponds to a technique called *value iteration*, which is a fundamental algorithm in the area of Markov decision processes (MDPs) and control theory.[4] Crudely, value iteration builds an optimal policy for controlling an MDP. Value iteration computes the policy backward: if we know what the optimal policy is for what we should do tomorrow onward, we can easily compute a policy for acting today onward. The equations that make this technique possible are known as Bellman equations.

Zermelo's algorithm can be used to compute, in polynomial time, subgame perfect equilibrium strategies in extensive form games. Although the scenarios examined in this article are rather small, precisely the same technique can be applied to extensive form games with arbitrary numbers of players and moves. Zermelo's backward induction algorithm is one of the cornerstone algorithms in game theory, but it also has applications in many areas of computer science. ■

References

1. M.J. Osborne and A. Rubinstein, *A Course in Game Theory,* MIT Press, 1994, p. 6.
2. U. Schwalbe and P. Walker, "Zermelo and the Early History of Game Theory," *Games and Economic Behavior,* vol. 34, no. 1, 2001, pp. 123–137.
3. E.M. Clarke, O. Grumberg, and D. Peled, *Model Checking,* MIT Press, 1999, pp. 35–39.
4. M.L. Puterman, *Markov Decision Processes,* Wiley, 1994, pp. 158–164.

Michael Wooldridge is a professor in the Department of Computer Science at the University of Oxford. Contact him at michael.wooldridge@cs.ox.ac.uk.