5

Games with Sequential Actions: Reasoning and Computing with the Extensive Form

In Chapter 3 we assumed that a game is represented in normal form: effectively, as a big table. In some sense, this is reasonable. The normal form is conceptually straightforward, and most see it as fundamental. While many other representations exist to describe finite games, we will see in this chapter and in Chapter 6 that each of them has an "induced normal form": a corresponding normal-form representation that preserves game-theoretic properties such as Nash equilibria. Thus the results given in Chapter 3 hold for all finite games, no matter how they are represented; in that sense the normal-form representation is universal.

In this chapter we will look at extensive-form games, a finite representation that does not always assume that players act simultaneously. This representation is in general exponentially smaller than its induced normal form, and furthermore can be much more natural to reason about. While the Nash equilibria of an extensive-form game can be found through its induced normal form, computational benefit can be had by working with the extensive form directly. Furthermore, there are other solution concepts, such as subgame-perfect equilibrium (see Section 5.1.3), which explicitly refer to the sequence in which players act and which are therefore not meaningful when applied to normal-form games.

5.1 Perfect-information extensive-form games

The normal-form game representation does not incorporate any notion of sequence, or time, of the actions of the players. The *extensive (*or *tree) form* is an alternative representation that makes the temporal structure explicit. We start by discussing the special case of *perfect information* extensive-form games, and then move on to discuss the more general class of *imperfect-information* extensive-form games in Section 5.2. In both cases we will restrict the discussion to finite games, that is, to games represented as finite trees.

5.1.1 *Definition*

Informally speaking, a perfect-information game in extensive form (or, more simply, a perfect-information game) is a tree in the sense of graph theory, in

which each node represents the choice of one of the players, each edge represents a possible action, and the leaves represent final outcomes over which each player has a utility function. Indeed, in certain circles (in particular, in artificial intelligence), these are known simply as game trees. Formally, we define them as follows.

Perfect-information game

Definition 5.1.1 (Perfect-information game) *A (finite) perfect-information game (in extensive form) is a tuple* $G = (N, A, H, Z, \chi, \rho, \sigma, u)$*, where:*

- N *is a set of* n *players;*
- A *is a (single) set of actions;*
- H *is a set of nonterminal choice nodes;*
- Z *is a set of terminal nodes, disjoint from* H*;*
- $\chi : H \mapsto 2^A$ *is the action function, which assigns to each choice node a set of possible actions;*
- $\rho : H \mapsto N$ *is the player function, which assigns to each nonterminal node a player* $i \in N$ *who chooses an action at that node;*
- $\sigma : H \times A \mapsto H \cup Z$ *is the successor function, which maps a choice node and an action to a new choice node or terminal node such that for all* $h_1, h_2 \in H$ *and* $a_1, a_2 \in A$*, if* $\sigma(h_1, a_1) = \sigma(h_2, a_2)$ *then* $h_1 = h_2$ *and* $a_1 = a_2$*; and*
- $u = (u_1, \ldots, u_n)$*, where* $u_i : Z \mapsto \mathbb{R}$ *is a real-valued utility function for player* i *on the terminal nodes* Z*.*

Since the choice nodes form a tree, we can unambiguously identify a node with its *history*, that is, the sequence of choices leading from the root node to it. We can also define the *descendants* of a node h, namely all the choice and terminal nodes in the subtree rooted at h.

An example of such a game is the *Sharing game*. Imagine a brother and sister following the following protocol for sharing two indivisible and identical presents from their parents. First the brother suggests a split, which can be one of three—he keeps both, she keeps both, or they each keep one. Then the sister chooses whether to accept or reject the split. If she accepts they each get their allocated present(s), and otherwise neither gets any gift. Assuming both siblings value the two presents equally and additively, the tree representation of this game is shown in Figure 5.1.

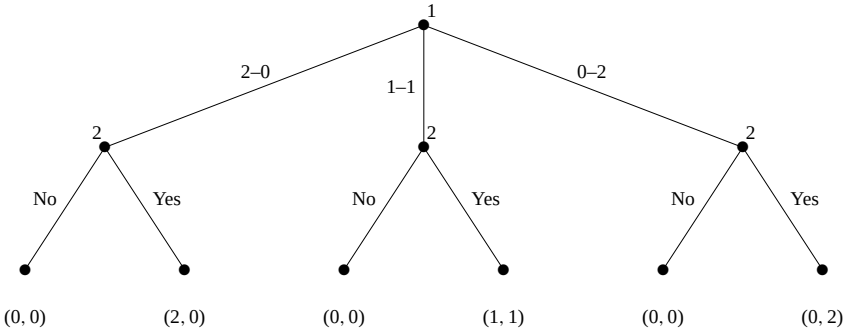

Figure 5.1 The Sharing game.

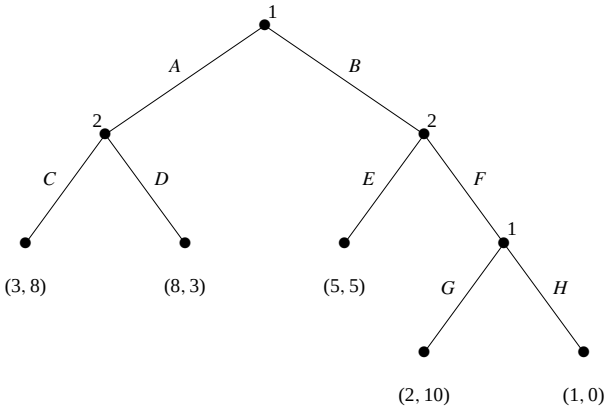

Figure 5.2 A perfect-information game in extensive form.

5.1.2 *Strategies and equilibria*

A pure strategy for a player in a perfect-information game is a complete specification of which deterministic action to take at every node belonging to that player. A more formal definition follows.

Definition 5.1.2 (Pure strategies) *Let* $G = (N, A, H, Z, \chi, \rho, \sigma, u)$ *be a perfect-information extensive-form game. Then the pure strategies of player* i *consist of the Cartesian product* $\prod_{h \in H, \rho(h)=i} \chi(h)$.

Notice that the definition contains a subtlety. An agent's strategy requires a decision at each choice node, regardless of whether or not it is possible to reach that node given the other choice nodes. In the Sharing game above the situation is straightforward—player 1 has three pure strategies, and player 2 has eight, as follows.

$S_1 = \{2\text{–}0, 1\text{–}1, 0\text{–}2\}$

$S_2 = \{(yes, yes, yes), (yes, yes, no), (yes, no, yes), (yes, no, no), (no, yes, yes), (no, yes, no), (no, no, yes), (no, no, no)\}$

But now consider the game shown in Figure 5.2.

In order to define a complete strategy for this game, each of the players must choose an action at each of his two choice nodes. Thus we can enumerate the pure strategies of the players as follows.

$S_1 = \{(A, G), (A, H), (B, G), (B, H)\}$

$S_2 = \{(C, E), (C, F), (D, E), (D, F)\}$

It is important to note that we have to include the strategies (A, G) and (A, H), even though once player 1 has chosen A then his own G-versus-H choice is moot.

	(C, E)	(C, F)	(D, E)	(D, F)
(A, G)	3, 8	3, 8	8, 3	8, 3
(A, H)	3, 8	3, 8	8, 3	8, 3
(B, G)	5, 5	2, 10	5, 5	2, 10
(B, H)	5, 5	1, 0	5, 5	1, 0

Figure 5.3 The game from Figure 5.2 in normal form.

The definition of best response and Nash equilibria in this game are exactly as they are for normal-form games. Indeed, this example illustrates how every perfect-information game can be converted to an equivalent normal-form game. For example, the perfect-information game of Figure 5.2 can be converted into the normal-form image of the game, shown in Figure 5.3. Clearly, the strategy spaces of the two games are the same, as are the pure-strategy Nash equilibria. (Indeed, both the mixed strategies and the mixed-strategy Nash equilibria of the two games are also the same; however, we defer further discussion of mixed strategies until we consider imperfect-information games in Section 5.2.)

In this way, for every perfect-information game there exists a corresponding normal-form game. Note, however, that the temporal structure of the extensive-form representation can result in a certain redundancy within the normal form. For example, in Figure 5.3 there are 16 different outcomes, while in Figure 5.2 there are only 5 (the payoff (3, 8) occurs only once in Figure 5.2 but four times in Figure 5.3 etc.). One general lesson is that while this transformation can always be performed, it can result in an exponential blowup of the game representation. This is an important lesson, since the didactic examples of normal-form games are very small, wrongly suggesting that this form is more compact.

The normal form gets its revenge, however, since the reverse transformation—from the normal form to the perfect-information extensive form—does not always exist. Consider, for example, the Prisoner's Dilemma game from Figure 3.3. A little experimentation will convince the reader that there does not exist a perfect-information game that is equivalent in the sense of having the same strategy profiles and the same payoffs. Intuitively, the problem is that perfect-information extensive-form games cannot model simultaneity. The general characterization of the class of normal-form games for which there exist corresponding perfect-information games in extensive form is somewhat complex.

The reader will have noticed that we have so far concentrated on pure strategies and pure Nash equilibria in extensive-form games. There are two reasons for this, or perhaps one reason and one excuse. The reason is that mixed strategies introduce a new subtlety, and it is convenient to postpone discussion of

	(C, E)	(C, F)	(D, E)	(D, F)
(A, G)	3, 8	**3, 8**	8, 3	8, 3
(A, H)	3, 8	**3, 8**	8, 3	8, 3
(B, G)	5, 5	2, 10	5, 5	2, 10
(B, H)	**5, 5**	1, 0	5, 5	1, 0

Figure 5.4 Equilibria of the game from Figure 5.2.

it. The excuse (which also allows the postponement, though not for long) is the following theorem.

Theorem 5.1.3 *Every (finite) perfect-information game in extensive form has a pure-strategy Nash equilibrium.*

This is perhaps the earliest result in game theory, due to Zermelo in 1913 (see the historical notes at the end of the chapter). The intuition here should be clear; since players take turns, and everyone gets to see everything that happened thus far before making a move, it is never necessary to introduce randomness into action selection in order to find an equilibrium. We will see this plainly when we discuss *backward induction* below. Both this intuition and the theorem will cease to hold when we discuss more general classes of games such as imperfect-information games in extensive form. First, however, we discuss an important refinement of the concept of Nash equilibrium.

backward induction

5.1.3 *Subgame-perfect equilibrium*

As we have discussed, the notion of Nash equilibrium is as well defined in perfect-information games in extensive form as it is in the normal form. However, as the following example shows, the Nash equilibrium can be too weak a notion for the extensive form. Consider again the perfect-information extensive-form game shown in Figure 5.2. There are three pure-strategy Nash equilibria in this game: $\{(A, G), (C, F)\}$, $\{(A, H), (C, F)\}$, and $\{(B, H), (C, E)\}$. This can be determined by examining the normal form image of the game, as indicated in Figure 5.4.

However, examining the normal form image of an extensive-form game obscures the game's temporal nature. To illustrate a problem that can arise in certain equilibria of extensive-form games, in Figure 5.5 we contrast the equilibria $\{(A, G), (C, F)\}$ and $\{(B, H), (C, E)\}$ by drawing them on the extensive-form game tree.

First consider the equilibrium $\{(A, G), (C, F)\}$. If player 1 chooses A then player 2 receives a higher payoff by choosing C than by choosing D. If player 2

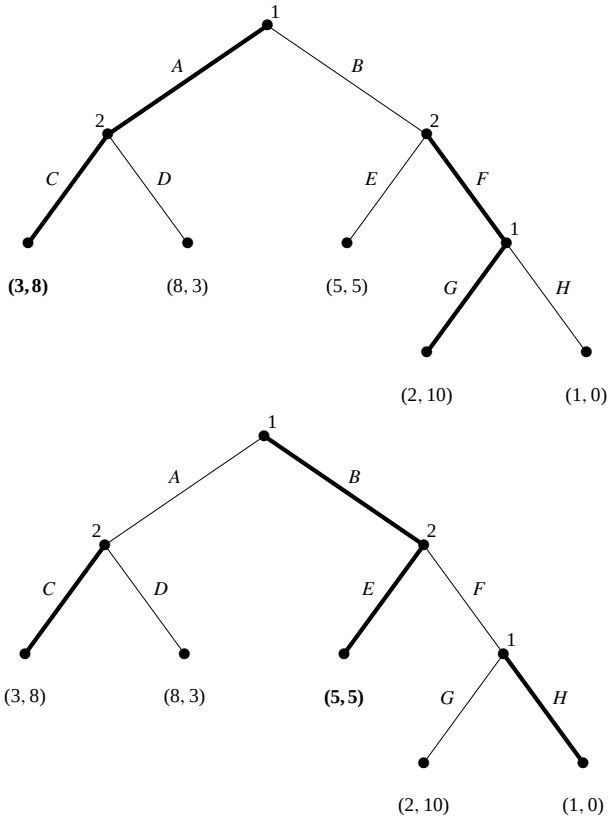

Figure 5.5 Two out of the three equilibria of the game from Figure 5.2: $\{(A, G), (C, F)\}$ and $\{(B, H), (C, E)\}$. Bold edges indicate players' choices at each node.

played the strategy (C, E) rather than (C, F) then player 1 would prefer to play B at the first node in the tree; as it is, player 1 gets a payoff of 3 by playing A rather than a payoff of 2 by playing B. Hence we have an equilibrium.

The second equilibrium $\{(B, H), (C, E)\}$ is less intuitive. First, note that $\{(B, G), (C, E)\}$ is *not* an equilibrium: player 2's best response to (B, G) is (C, F). Thus, the only reason that player 2 chooses to play the action E is that he knows that player 1 would play H at his second decision node. This behavior by player 1 is called a *threat*: by committing to choose an action that is harmful to player 2 in his second decision node, player 1 can cause player 2 to avoid that part of the tree. (Note that player 1 benefits from making this threat: he gets a payoff of 5 instead of 2 by playing (B, H) instead of (B, G).) So far so good. The problem, however, is that player 2 may not consider player 1's threat to be credible: if player 1 did reach his final decision node, actually choosing H over G would also reduce player 1's own utility. If player 2 played F, would player 1 really follow through on his threat and play H, or would he relent and pick G instead?

To formally capture the reason why the $\{(B, H), (C, E)\}$ equilibrium is unsatisfying, and to define an equilibrium refinement concept that does not suffer from this problem, we first define the notion of a subgame.

Definition 5.1.4 (Subgame) *Given a perfect-information extensive-form game G, the* subgame *of G rooted at node h is the restriction of G to the descendants of h. The set of subgames of G consists of all of subgames of G rooted at some node in G.*

Now we can define the notion of a *subgame-perfect equilibrium*, a refinement of the Nash equilibrium in perfect-information games in extensive form, which eliminates those unwanted Nash equilibria.[1]

subgame-perfect equilibrium (SPE)

Definition 5.1.5 (Subgame-perfect equilibrium) *The* subgame-perfect equilibria *(SPE) of a game G are all strategy profiles s such that for any subgame G' of G, the restriction of s to G' is a Nash equilibrium of G'.*

Since G is its own subgame, every SPE is also a Nash equilibrium. Furthermore, although SPE is a stronger concept than Nash equilibrium (i.e., every SPE is a NE, but not every NE is a SPE) it is still the case that every perfect-information extensive-form game has at least one subgame-perfect equilibrium.

This definition rules out "noncredible threats" of the sort illustrated in the above example. In particular, note that the extensive-form game in Figure 5.2 has only one subgame-perfect equilibrium, $\{(A, G), (C, F)\}$. Neither of the other Nash equilibria is subgame perfect. Consider the subgame rooted at player 1's second choice node. The unique Nash equilibrium of this (trivial) game is for player 1 to play G. Thus the action H, the restriction of the strategies (A, H) and (B, H) to this subgame, is not optimal in this subgame, and cannot be part of a subgame-perfect equilibrium of the larger game.

5.1.4 *Computing equilibria: backward induction*

n-player, general-sum games: the backward induction algorithm

backward induction

Inherent in the concept of subgame-perfect equilibrium is the principle of *backward induction*. One identifies the equilibria in the "bottom-most" subgame trees, and assumes that those equilibria will be played as one backs up and considers increasingly larger trees. We can use this procedure to compute a sample Nash equilibrium. This is good news: not only are we guaranteed to find a subgame-perfect equilibrium (rather than possibly finding a Nash equilibrium that involves noncredible threats), but also this procedure is computationally simple. In particular, it can be implemented as a single depth-first traversal of the game tree and thus requires time linear in the size of the game representation. Recall in contrast that the best known methods for finding Nash equilibria of general games require time exponential in the size of the normal form; remember as well that the induced normal form of an extensive-form game is exponentially larger than the original representation.

The algorithm BACKWARDINDUCTION is described in Figure 5.6. The variable *util_at_child* is a vector denoting the utility for each player at the child node; $util\_at\_child_{\rho(h)}$ denotes the element of this vector corresponding to the utility for

1. Note that the word "perfect" is used in two different senses here.

```
function BackwardInduction (node h) returns u(h)
if h ∈ Z then
    return u(h)                                   // h is a terminal node
best_util ← −∞
forall a ∈ χ(h) do
    util_at_child ← BackwardInduction(σ(h, a))
    if util_at_child_ρ(h) > best_util_ρ(h) then
        best_util ← util_at_child
return best_util
```

Figure 5.6 Procedure for finding the value of a sample (subgame-perfect) Nash equilibrium of a perfect-information extensive-form game.

player $\rho(h)$ (the player who gets to move at node h). Similarly, $best\_util$ is a vector giving utilities for each player.

Observe that this procedure does not return an equilibrium strategy for each of the n players, but rather describes how to label each node with a vector of n real numbers. This labeling can be seen as an extension of the game's utility function to the nonterminal nodes H. The players' equilibrium strategies follow straightforwardly from this extended utility function: every time a given player i has the opportunity to act at a given node $h \in H$ (i.e., $\rho(h) = i$), that player will choose an action $a_i \in \chi(h)$ that solves $\arg\max_{a_i \in \chi(h)} u_i(\sigma(a_i, h))$. These strategies can also be returned by BackwardInduction given some extra bookkeeping.

While the procedure demonstrates that in principle a sample SPE is effectively computable, in practice many game trees are not enumerated in advance and are hence unavailable for backward induction. For example, the extensive-form representation of chess has around 10^{150} nodes, which is vastly too large to represent explicitly. For such games it is more common to discuss the size of the game tree in terms of the average branching factor b (the average number of actions which are possible at each node) and a maximum depth m (the maximum number of sequential actions). A procedure which requires time linear in the size of the representation thus expands $O(b^m)$ nodes. Unfortunately, we can do no better than this on arbitrary perfect-information games.

Two-player, zero-sum games: minimax and alpha-beta pruning

minimax algorithm

We *can* make some computational headway in the widely applicable case of two-player, zero-sum games. We first note that BackwardInduction has another name in the two-player, zero-sum context: the *minimax algorithm*. Recall that in such games, only a single payoff number is required to characterize any outcome. Player 1 wants to maximize this number, while player 2 wants to minimize it. In this context BackwardInduction can be understood as propagating these single payoff numbers from the root of the tree up to the root. Each decision node for player 1 is labeled with the maximum of the labels of its child nodes (representing the fact that player 1 would choose the corresponding action), and each decision node for player 2 is labeled with the minimum of that node's

```
function AlphaBetaPruning (node h, real α, real β) returns u_1(h)
if h ∈ Z then
    return u_1(h)                                   // h is a terminal node
best_util ← (2ρ(h) − 3) × ∞                         // −∞ for player 1; ∞ for player 2
forall a ∈ χ(h) do
    if ρ(h) = 1 then
        best_util ← max(best_util, AlphaBetaPruning(σ(h, a), α, β))
        if best_util ≥ β then
            return best_util
        α ← max(α, best_util)
    else
        best_util ← min(best_util, AlphaBetaPruning(σ(h, a), α, β))
        if best_util ≤ α then
            return best_util
        β ← min(β, best_util)
return best_util
```

Figure 5.7 The alpha-beta pruning algorithm. It is invoked at the root node h as AlphaBetaPruning$(h, -\infty, \infty)$.

children's labels. The label on the root node is the value of the game: player 1's payoff in equilibrium.

How can we improve on the minimax algorithm? The fact that player 1 and player 2 always have strictly opposing interests means that we can *prune* away some parts of the game tree: we can recognize that certain subtrees will never be reached in equilibrium, even without examining the nodes in these subtrees. This leads us to a new algorithm called AlphaBetaPruning, which is given in Figure 5.7.

pruning

There are several ways in which AlphaBetaPruning differs from BackwardInduction. Some concern the fact that we have now restricted ourselves to a setting where there are only two players, and one player's utility is the negative of the other's. We thus deal only with the utility for player 1. This is why we treat the two players separately, maximizing for player 1 and minimizing for player 2.

At each node h either α or β is updated. These variables take the value of the previously encountered node that their corresponding player (player 1 for α and player 2 for β) would most prefer to choose *instead* of h. For example, consider the variable β at some node h. Now consider all the different choices that player 2 could make at ancestors of h that would prevent h from ever being reached, and that would ultimately lead to previously encountered terminal nodes. β is the best value that player 2 could obtain at any of these terminal nodes. Because the players do not have any alternative to starting at the root of the tree, at the beginning of the search $\alpha = -\infty$ and $\beta = \infty$.

We can now concentrate on the important difference between BackwardInduction and AlphaBetaPruning: in the latter procedure, the search can backtrack at a node that is not terminal. Let us think about things from the point of view of player 1, who is considering what action to play at node h. (As we

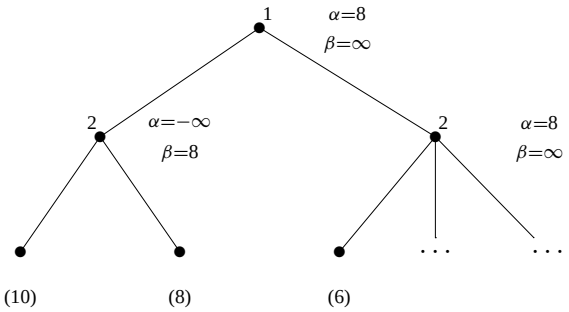

Figure 5.8 An example of alpha-beta pruning. We can backtrack after expanding the first child of the right choice node for player 2.

encourage you to check for yourself, a similar argument holds when it is player 2's turn to move at node h.) For player 1, this backtracking occurs on the line that reads "if *best_util* $\geq \beta$ then return *best_util*." What is going on here? We have just explored some, but not all, of the children of player 1's decision node h; the highest value among these explored nodes is *best_util*. The value of node h is therefore lower bounded by *best_util* (it is *best_util* if h has no children with larger values, and is some larger amount otherwise). Either way, if *best_util* $\geq \beta$ then player 1 knows that player 2 prefers choosing his best alternative (at some ancestor node of h) rather than allowing player 1 to act at node h. Thus node h cannot be on the equilibrium path[2] and so there is no need to continue exploring the game tree below h.

A simple example of AlphaBetaPruning in action is given in Figure 5.8. The search begins by heading down the left branch and visiting both terminal nodes, and eventually setting $\beta = 8$. (Do you see why?) It then returns the value 8 as the value of this subgame, which causes α to be set to 8 at the root node. In the right subgame the search visits the first terminal node and so sets *best_util* $= 6$ at the shaded node, which we will call h. Now at h we have *best_util* $\leq \alpha$, which means that we can backtrack. This is safe to do because we have just shown that player 1 would never choose this subgame: he can guarantee himself a payoff of 8 by choosing the left subgame, whereas his utility in the right subgame would be no more than 6.

The effectiveness of the alpha-beta pruning algorithm depends on the order in which nodes are considered. For example, if player 1 considers nodes in increasing order of their value, and player 2 considers nodes in decreasing order of value, then no nodes will ever be pruned. In the best case (where nodes are ordered in decreasing value for player 1 and in increasing order for player 2), alpha-beta pruning has complexity of $O(b^{\frac{m}{2}})$. We can rewrite this expression as $O(\sqrt{b}^m)$, making more explicit the fact that the game's branching factor would effectively be cut to the square root of its original value. If nodes are examined in random order then the analysis becomes somewhat more complicated;

2. In fact, in the case *best_util* $= \beta$, it *is* possible that h could be reached on an equilibrium path; however, in this case there is still always an equilibrium in which player 2 plays his best alternative and h is not reached.

when b is fairly small, the complexity of alpha-beta pruning is $O(b^{\frac{3m}{4}})$, which is still an exponential improvement. In practice, it is usually possible to achieve performance somewhere between the best case and the random case. This technique thus offers substantial practical benefit over straightforward backward induction in two-player, zero-sum games for which the game tree is represented implicitly.

Techniques like alpha-beta pruning are commonly used to build strong computer players for two-player board games such as chess. (However, they perform poorly on games with extremely large branching factors, such as go.) Of course, building a good computer player involves a great deal of engineering, and requires considerable attention to game-specific heuristics such as those used to order actions. One general technique is required by many such systems, however, and so is worth discussing here. The game tree in practical games can be so large that it is infeasible to search all the way down to leaf nodes. Instead, the search proceeds to some shallower depth (which is chosen either statically or dynamically). Where do we get the node values to propagate up using backward induction? The trick is to use an *evaluation function* to estimate the value of the deepest node reached (taking into account game-relevant features such as board position, number of pieces for each player, who gets to move next, etc., and either built by hand or learned). When the search has reached an appropriate depth, the node is treated as terminal with a call to the evaluation function replacing the evaluation of the utility function at that node. This requires a small change to the beginning of AlphaBetaPruning; otherwise, the algorithm works unchanged.

evaluation function

Two-player, general-sum games: computing all subgame-perfect equilibria

While the BackwardInduction procedure identifies one subgame-perfect equilibrium in linear time, it does not provide an efficient way of finding all of them. One might wonder how there could even *be* more than one SPE in a perfect-information game. Multiple subgame-perfect equilibria can exist when there exist one or more decision nodes at which a player chooses between subgames in which he receives the same utility. In such cases BackwardInduction simply chooses the first subgame it encountered. It could be useful to find the set of all subgame-perfect equilibria if we wanted to find a specific SPE (as we did with Nash equilibria of normal-form games in Section 4.2.4) such as the one that maximizes social welfare.

Here let us restrict ourselves to two-player perfect-information extensive-form games, but lift our previous restriction that the game be zero-sum. A somewhat more complicated algorithm can find the set of *all* subgame-perfect equilibrium values in worst-case cubic time.

Theorem 5.1.6 *Given a two-player perfect-information extensive-form game with ℓ leaves, the set of subgame-perfect equilibrium payoffs can be represented as the union of $O(\ell^2)$ axis-aligned rectangles and can be computed in time $O(\ell^3)$.*

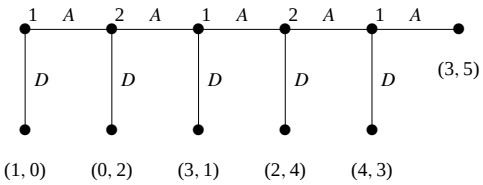

Figure 5.9 The Centipede game.

Intuitively, the algorithm works much like BackwardInduction, but the variable *util_at_child* holds a representation of all equilibrium values instead of just one. The "max" operation we had previously implemented through *best_util* is replaced by a subroutine that returns a representation of all the values that can be obtained in subgame-perfect equilibria of the node's children. This can include mixed strategies if multiple children are simultaneously best responses. More information about this algorithm can be found in the reference cited in the chapter notes.

An example and criticisms of backward induction

Centipede game

Despite the fact that strong arguments can be made in its favor, the concept of backward induction is not without controversy. To see why this is, consider the well-known *Centipede game*, depicted in Figure 5.9. (The game starts at the node at the upper left.) In this game two players alternate in making decisions, at each turn choosing between going "down" and ending the game or going "across" and continuing it (except at the last node where going "across" also ends the game). The payoffs are constructed in such a way that the only SPE is for each player to always choose to go down. To see why, consider the last choice. Clearly at that point the best choice for the player is to go down. Since this is the case, going down is also the best choice for the other player in the previous choice point. By induction the same argument holds for all choice points.

This would seem to be the end of this story, except for two pesky factors. The first problem is that the SPE prediction in this case flies in the face of intuition. Indeed, in laboratory experiments subjects in fact continue to play "across" until close to the end of the game. The second problem is theoretical. Imagine that you are the second player in the game, and in the first step of the game the first player actually goes across. What should you do? The SPE suggests you should go down, but the same analysis suggests that you would not have gotten to this choice point in the first place. In other words, you have reached a state to which your analysis has given a probability of zero. How should you amend your beliefs and course of action based on this measure-zero event? It turns out this seemingly small inconvenience actually raises a fundamental problem in game theory. We will not develop the subject further here, but let us only mention that there exist different accounts of this situation, and they depend on the probabilistic assumptions made, on what is common knowledge (in particular, whether there is common knowledge of rationality), and on exactly how one revises one's beliefs in the face of measure-zero events. The last question is intimately related to the subject of belief revision discussed in Chapter 14.

5.2 Imperfect-information extensive-form games

Up to this point, in our discussion of extensive-form games we have allowed players to specify the action that they would take at every choice node of the game. This implies that players know the node they are in, and—recalling that in such games we equate nodes with the histories that led to them—all the prior choices, including those of other agents. For this reason we have called these *perfect-information games*.

We might not always want to make such a strong assumption about our players and our environment. In many situations we may want to model agents needing to act with partial or no knowledge of the actions taken by others, or even agents with limited memory of their own past actions. The sequencing of choices allows us to represent such ignorance to a limited degree; an "earlier" choice might be interpreted as a choice made without knowing the "later" choices. However, so far we could not represent two choices made in the same play of the game in mutual ignorance of each other.

5.2.1 *Definition*

Imperfect-information games in extensive form address this limitation. An imperfect-information game is an extensive-form game in which each player's choice nodes are partitioned into information sets; intuitively, if two choice nodes are in the same information set then the agent cannot distinguish between them.[3]

Definition 5.2.1 (Imperfect-information game) *An imperfect-information game (in extensive form) is a tuple* $(N, A, H, Z, \chi, \rho, \sigma, u, I)$*, where:*

- $(N, A, H, Z, \chi, \rho, \sigma, u)$ *is a perfect-information extensive-form game; and*
- $I = (I_1, \ldots, I_n)$*, where* $I_i = (I_{i,1}, \ldots, I_{i,k_i})$ *is an equivalence relation on (i.e., a partition of)* $\{h \in H : \rho(h) = i\}$ *with the property that* $\chi(h) = \chi(h')$ *and* $\rho(h) = \rho(h')$ *whenever there exists a* j *for which* $h \in I_{i,j}$ *and* $h' \in I_{i,j}$*.*

Note that in order for the choice nodes to be truly indistinguishable, we require that the set of actions at each choice node in an information set be the same (otherwise, the player would be able to distinguish the nodes). Thus, if $I_{i,j} \in I_i$ is an equivalence class, we can unambiguously use the notation $\chi(I_{i,j})$ to denote the set of actions available to player i at any node in information set $I_{i,j}$.

Consider the imperfect-information extensive-form game shown in Figure 5.10. In this game, player 1 has two information sets: the set including the top choice node, and the set including the bottom choice nodes. Note that the two bottom choice nodes in the second information set have the same set of possible actions. We can regard player 1 as not knowing whether player 2 chose A or B when he makes his choice between ℓ and r.

3. From the technical point of view, imperfect-information games are obtained by overlaying a partition structure, as defined in Chapter 13 in connection with models of knowledge, over a perfect-information game.

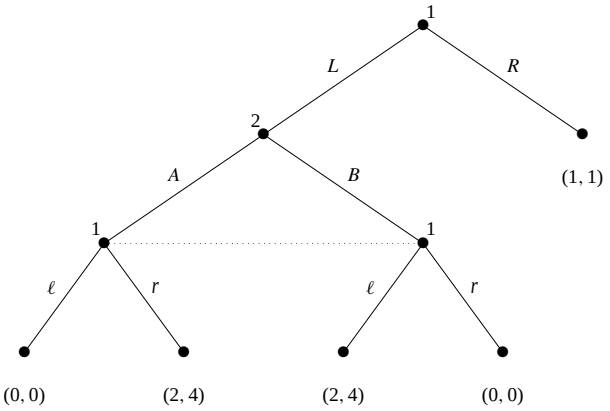

Figure 5.10 An imperfect-information game.

5.2.2 *Strategies and equilibria*

A pure strategy for an agent in an imperfect-information game selects one of the available actions in each information set of that agent.

Definition 5.2.2 (Pure strategies) *Let* $G = (N, A, H, Z, \chi, \rho, \sigma, u, I)$ *be an imperfect-information extensive-form game. Then the pure strategies of player* i *consist of the Cartesian product* $\prod_{I_{i,j} \in I_i} \chi(I_{i,j})$.

Thus perfect-information games can be thought of as a special case of imperfect-information games, in which every equivalence class of each partition is a singleton.

Consider again the Prisoner's Dilemma game, shown as a normal-form game in Figure 3.3. An equivalent imperfect-information game in extensive form is given in Figure 5.11.

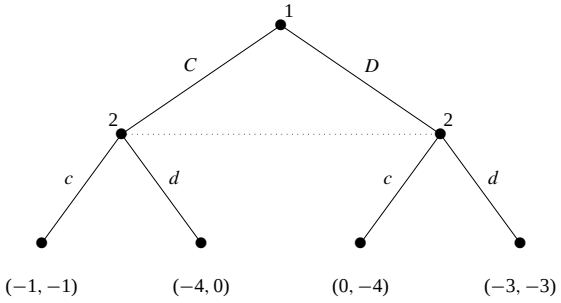

Figure 5.11 The Prisoner's Dilemma game in extensive form.

Note that we could have chosen to make player 2 choose first and player 1 choose second.

Recall that perfect-information games were not expressive enough to capture the prisoner's dilemma game and many other ones. In contrast, as is obvious from this example, any normal-form game can be trivially transformed into an equivalent imperfect-information game. However, this example is also special in

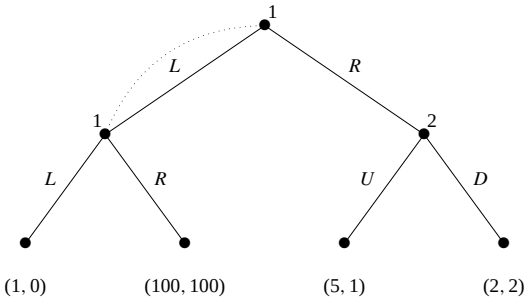

Figure 5.12 A game with imperfect recall.

that the Prisoner's Dilemma is a game with a dominant strategy solution, and thus in particular a pure-strategy Nash equilibrium. This is not true in general for imperfect-information games. To be precise about the equivalence between a normal-form game and its extensive-form image we must consider mixed strategies, and this is where we encounter a new subtlety.

As we did for perfect-information games, we can define the normal-form game corresponding to any given imperfect-information game; this normal game is again defined by enumerating the pure strategies of each agent. Now, we define the set of mixed strategies of an imperfect-information game as simply the set of mixed strategies in its image normal-form game; in the same way, we can also define the set of Nash equilibria.[4] However, we can also define the set of *behavioral strategies* in the extensive-form game. These are the strategies in which, rather than randomizing over complete pure strategies, the agent randomizes independently at each information set. And so, whereas a mixed strategy is a distribution over vectors (each vector describing a pure strategy), a behavioral strategy is a vector of distributions.

behavioral

We illustrate this distinction first in the special case of perfect-information games. For example, consider the game of Figure 5.2. A strategy for player 1 that selects A with probability .5 and G with probability .3 is a behavioral strategy. In contrast, the mixed strategy $(.6(A, G), .4(B, H))$ is not a behavioral strategy for that player, since the choices made by him at the two nodes are not independent (in fact, they are perfectly correlated).

In general, the expressive power of behavioral strategies and the expressive power of mixed strategies are noncomparable; in some games there are outcomes that are achieved via mixed strategies but not any behavioral strategies, and in some games it is the other way around.

Consider for example the game in Figure 5.12. In this game, when considering mixed strategies (but not behavioral strategies), R is a strictly dominant strategy for agent 1, D is agent 2's strict best response, and thus (R, D) is the unique Nash equilibrium. Note in particular that in a mixed strategy, agent 1 decides

4. Note that we have defined two transformations—one from any normal-form game to an imperfect-information game, and one in the other direction. However the first transformation is not one to one, and so if we transform a normal-form game to an extensive-form one and then back to normal form, we will not in general get back the same game we started out with. However, we will get a game with identical strategy spaces and equilibria.

probabilistically whether to play L or R in his information set, but once he decides he plays that pure strategy consistently. Thus the payoff of 100 is irrelevant in the context of mixed strategies. On the other hand, with behavioral strategies agent 1 gets to randomize afresh each time he finds himself in the information set. Noting that the pure strategy D is weakly dominant for agent 2 (and in fact is the unique best response to all strategies of agent 1 other than the pure strategy L), agent 1 computes the best response to D as follows. If he uses the behavioral strategy $(p, 1-p)$ (i.e., choosing L with probability p each time he finds himself in the information set), his expected payoff is

$$1 * p^2 + 100 * p(1-p) + 2 * (1-p).$$

The expression simplifies to $-99p^2 + 98p + 2$, whose maximum is obtained at $p = 98/198$. Thus $(R, D) = ((0, 1), (0, 1))$ is no longer an equilibrium in behavioral strategies, and instead we get the equilibrium $((98/198, 100/198), (0, 1))$.

There is, however, a broad class of imperfect-information games in which the expressive power of mixed and behavioral strategies coincides. This is the class of games of *perfect recall*. Intuitively speaking, in these games no player forgets any information he knew about moves made so far; in particular, he remembers precisely all his own moves. A formal definition follows.

perfect recall

Definition 5.2.3 (Perfect recall) *Player i has* perfect recall *in an imperfect-information game G if for any two nodes h, h' that are in the same information set for player i, for any path $h_0, a_0, h_1, a_1, h_2, \ldots, h_n, a_n, h$ from the root of the game to h (where the h_j are decision nodes and the a_j are actions) and for any path $h_0, a'_0, h'_1, a'_1, h'_2, \ldots, h'_m, a'_m, h'$ from the root to h' it must be the case that:*

1. *$n = m$;*
2. *for all $0 \leq j \leq n$, h_j and h'_j are in the same equivalence class for player i; and*
3. *for all $0 \leq j \leq n$, if $\rho(h_j) = i$ (i.e., h_j is a decision node of player i), then $a_j = a'_j$.*

G is a game of perfect recall if every player has perfect recall in it.

Clearly, every perfect-information game is a game of perfect recall.

Theorem 5.2.4 (Kuhn, 1953) *In a game of perfect recall, any mixed strategy of a given agent can be replaced by an equivalent behavioral strategy, and any behavioral strategy can be replaced by an equivalent mixed strategy. Here two strategies are equivalent in the sense that they induce the same probabilities on outcomes, for any fixed strategy profile (mixed or behavioral) of the remaining agents.*

As a corollary we can conclude that the set of Nash equilibria does not change if we restrict ourselves to behavioral strategies. This is true only in games of perfect recall, and thus, for example, in perfect-information games. We stress again, however, that in general imperfect-information games, mixed and behavioral strategies yield noncomparable sets of equilibria.

5.2.3 *Computing equilibria: the sequence form*

Because any extensive-form game can be converted into an equivalent normal-form game, an obvious way to find an equilibrium of an extensive-form game is to first convert it into a normal-form game and then find the equilibria using, for example the Lemke–Howson algorithm. This method is inefficient, however, because the number of actions in the normal-form game is *exponential* in the size of the extensive-form game. The normal-form game is created by considering all combinations of information set actions for each player, and the payoffs that result when these strategies are employed.

One way to avoid this problem is to operate directly on the extensive-form representation. This can be done by employing behavioral strategies to express a game using a description called the sequence form.

Defining the sequence form

The sequence form is (primarily) useful for representing imperfect-information extensive-form games of perfect recall. Definition 5.2.5 describes the elements of the sequence-form representation of such games; we then go on to explain what each of these elements means.

sequence form

Definition 5.2.5 (Sequence-form representation) *Let G be an imperfect-information game of perfect recall. The* sequence-form representation *of G is a tuple (N, Σ, g, C), where:*

- *N is a set of agents;*
- *$\Sigma = (\Sigma_1, \ldots, \Sigma_n)$, where Σ_i is the set of* sequences *available to agent i;*
- *$g = (g_1, \ldots, g_n)$, where $g_i : \Sigma \mapsto \mathbb{R}$ is the payoff function for agent i; and*
- *$C = (C_1, \ldots, C_n)$, where C_i is a set of linear constraints on the realization probabilities of agent i.*

Now let us define all these terms. To begin with, what is a sequence? The key insight of the sequence form is that, while there are exponentially many pure strategies in an extensive-form game, there are only a small number of nodes in the game tree. Rather than building a player's strategy around the idea of pure strategies, the sequence form builds it around paths in the tree from the root to each node.

sequence

Definition 5.2.6 (Sequence) *A* sequence *of actions of player $i \in N$, defined by a node $h \in H \cup Z$ of the game tree, is the ordered set of player i's actions that lie on the path from the root to h. Let $\emptyset$ denote the sequence corresponding to the root node. The set of sequences of player i is denoted Σ_i, and $\Sigma = \Sigma_1 \times \cdots \times \Sigma_n$ is the set of all sequences.*

A sequence can thus be thought of as a string listing the action choices that player i would have to take in order to get from the root to a given node h. Observe that h may or may not be a leaf node; observe also that the other players' actions that form part of this path are not part of the sequence.

	$\emptyset$	A	B
$\emptyset$	0, 0	0, 0	0, 0
L	0, 0	0, 0	0, 0
R	1, 1	0, 0	0, 0
$L\ell$	0, 0	0, 0	2, 4
Lr	0, 0	2, 4	0, 0

Figure 5.13 The sequence form of the game from Figure 5.10.

	A	B
$L\ell$	0, 0	2, 4
Lr	2, 4	0, 0
$R\ell$	1, 1	1, 1
Rr	1, 1	1, 1

Figure 5.14 The induced normal form of the game from Figure 5.10.

sequence-form payoff function

Definition 5.2.7 (Payoff function) *The* payoff function $g_i : \Sigma \mapsto \mathbb{R}$ *for agent i is given by $g(\sigma) = u(z)$ if a leaf node $z \in Z$ would be reached when each player played his sequence $\sigma_i \in \sigma$, and by $g(\sigma) = 0$ otherwise.*

Given the set of sequences Σ and the payoff function g, we can think of the sequence form as defining a tabular representation of an imperfect-information extensive-form game, much as the induced normal form does. Consider the game given in Figure 5.10 (see p. 126). The sets of sequences for the two players are $\Sigma_1 = \{\emptyset, L, R, L\ell, Lr\}$ and $\Sigma_2 = \{\emptyset, A, B\}$. The payoff function is given in Figure 5.13. For comparison, the induced normal form of the same game is given in Figure 5.14. Written this way, the sequence form is larger than the induced normal form. However, many of the entries in the game matrix in Figure 5.13 correspond to cases where the payoff function is defined to be zero because the given pair of sequences does not correspond to a leaf node in the game tree. These entries are shaded in gray to indicate that they could not arise in play. Each payoff that *is* defined at a leaf in the game tree occurs exactly once in the sequence-form table. Thus, if g was represented using a sparse encoding, only five values would have to be stored. Compare this to the induced normal form, where all of the eight entries correspond to leaf nodes from the game tree.

We now have a set of players, a set of sequences, and a mapping from sequences to payoffs. At first glance this may look like everything we need to describe our game. However, sequences do not quite take the place of actions. In particular, a player cannot simply select a single sequence in the way that he would select a pure strategy—the other player(s) might not play in a way that would allow him to follow it to its end. Put another way, players still need to define what they would do in every information set that could be reached in the game tree.

What we want is for agents to select behavioral strategies. (Since we have assumed that our game G has perfect recall, Theorem 5.2.4 tells us that any equilibrium will be expressible using behavioral strategies.) However, it turns out that it is not a good idea to work with behavioral strategies directly—if we did so, the optimization problems we develop later would be computationally harder to solve. Instead, we will develop the alternate concept of a *realization plan*, which corresponds to the probability that a given sequence would arise under a given behavioral strategy.

Consider an agent i following a behavioral strategy that assigned probability $\beta_i(h, a_i)$ to taking action a_i at a given decision node h. Then we can construct a *realization plan* that assigns probabilities to sequences in a way that recovers i's behavioral strategy β.

realization plan of β_i

realization probability

Definition 5.2.8 (Realization plan of β_i) *The* realization plan of β_i *for player* $i \in N$ *is a mapping* $r_i : \Sigma_i \mapsto [0, 1]$ *defined as* $r_i(\sigma_i) = \prod_{c \in \sigma_i} \beta_i(c)$. *Each value* $r_i(\sigma_i)$ *is called a* realization probability.

Definition 5.2.8 is not the most useful way of defining realization probabilities. There is a second, equivalent definition with the advantage that it involves a set of linear equations, although it is a bit more complicated. This definition relies on two functions that we will make extensive use of in this section.

To define the first function, we make use of our assumption that G is a game of perfect recall. This entails that, given an information set $I \in I_i$, there must be one single sequence that player i can play to reach all of his nonterminal choice nodes $h \in I$. We denote this mapping as $\text{seq}_i : I_i \mapsto \Sigma_i$, and call $\text{seq}_i(I)$ the sequence *leading to* information set I. Note that while there is only one sequence that leads to a given information set, a given sequence can lead to multiple different information sets. For example, if player 1 moves first and player 2 observes his move, then the sequence $\emptyset$ will lead to multiple information sets for player 2.

$\text{seq}_i(I)$: the sequence leading to I

The second function considers ways that sequences can be built from other sequences. By $\sigma_i a_i$ denote a sequence that consists of the sequence σ_i followed by the single action a_i. As long as the new sequence still belongs to Σ_i, we say that the sequence $\sigma_i a_i$ *extends* the sequence σ_i. A sequence can often be extended in multiple ways—for example, perhaps agent i could have chosen an action a_i' instead of a_i after playing sequence σ_i. We denote by $\text{Ext}_i : \Sigma_i \mapsto 2^{\Sigma_i}$ a function mapping from sequences to sets of sequences, where $\text{Ext}_i(\sigma_i)$ denotes the set of sequences that extend the sequence σ_i. We define $\text{Ext}_i(\emptyset)$ to be the set of all single-action sequences. Note that extension always refers to playing a *single* action beyond a given sequence; thus, $\sigma_i a_i a_i'$ does not belong to $\text{Ext}_i(\sigma_i)$, even if it is a valid sequence. (It *does* belong to $\text{Ext}_i(\sigma_i a_i)$.) Also note that not all sequences have extensions; one example is sequences leading to leaf nodes. In such cases $\text{Ext}_i(\sigma)$ returns the empty set. Finally, to reduce notation we introduce the shorthand $\text{Ext}_i(I) = \text{Ext}_i(\text{seq}_i(I))$: the sequences extending an information set are the sequences extending the (unique) sequence leading to that information set.

$\text{Ext}_i(\sigma_i)$: sequences extending σ_i

$\text{Ext}_i(I) = \text{Ext}_i(\text{seq}_i(I))$

realization plan

Definition 5.2.9 (Realization plan) *A* realization plan *for player $i \in N$ is a function $r_i : \Sigma_i \mapsto [0, 1]$ satisfying the following constraints.*

$$r_i(\emptyset) = 1 \tag{5.1}$$

$$\sum_{\sigma_i' \in \text{Ext}_i(I)} r_i(\sigma_i') = r_i(\text{seq}_i(I)) \qquad \forall I \in I_i \tag{5.2}$$

$$r_i(\sigma_i) \geq 0 \qquad \forall \sigma_i \in \Sigma_i \tag{5.3}$$

If a player i follows a realization plan r_i, we must be able to recover a behavioral strategy β_i from it. For a decision node h for player i that is in information set $I \in I_i$, and for any sequence $(\text{seq}_i(I)a_i) \in \text{Ext}_i(I)$, $\beta_i(h, a_i)$ is defined as $\frac{r_i(\text{seq}_i(I)a_i)}{r_i(\text{seq}_i(I))}$, as long as $r_i(\text{seq}_i(I)) > 0$. If $r_i(\text{seq}_i(I)) = 0$ then we can assign $\beta_i(h, a_i)$ an arbitrary value from $[0, 1]$: here β_i describes the player's behavioral strategy at a node that could never be reached in play because of the player's own previous decisions, and so the value we assign to β_i is irrelevant.

Let C_i be the set of constraints (5.2) on realization plans of player i. Let $C = (C_1, \ldots, C_n)$. We have now defined all the elements[5] of a sequence-form representation $G = (N, \Sigma, g, C)$, as laid out in Definition 5.2.5.

What is the space complexity of the sequence-form representation? Unlike the normal form, the size of this representation is linear in the size of the extensive-form game. There is one sequence for each node in the game tree, plus the $\emptyset$ sequence for each player. As argued previously, the payoff function g can be represented sparsely, so that each payoff corresponding to a leaf node is stored only once, and no other payoffs are stored at all. There is one version of constraint (5.2) for each edge in the game tree. Each such constraint for player i references only $|\text{Ext}_i(I)| + 1$ variables, again allowing sparse encoding.

Computing best responses in two-player games

The sequence-form representation can be leveraged to allow the computation of equilibria far more efficiently than can be done using the induced normal form. Here we will consider the case of two-player games, as it is these games for which the strongest results hold. First we consider the problem of determining player 1's best response to a fixed behavioral strategy of player 2 (represented as a realization plan). This problem can be written as the following linear program.

$$\text{maximize} \quad \sum_{\sigma_1 \in \Sigma_1} \left(\sum_{\sigma_2 \in \Sigma_2} g_1(\sigma_1, \sigma_2) r_2(\sigma_2) \right) r_1(\sigma_1) \tag{5.4}$$

$$\text{subject to} \quad r_1(\emptyset) = 1 \tag{5.5}$$

$$\sum_{\sigma_1' \in \text{Ext}_1(I)} r_1(\sigma_1') = r_1(\text{seq}_1(I)) \qquad \forall I \in I_1 \tag{5.6}$$

$$r_1(\sigma_1) \geq 0 \qquad \forall \sigma_1 \in \Sigma_1 \tag{5.7}$$

5. We do not need to explicitly store constraints (5.1) and (5.3), because they are always the same for every sequence-form representation.

This linear program is straightforward. First, observe that $g_1(\cdot)$ and $r_2(\cdot)$ are constants, while $r_1(\cdot)$ are variables. The LP states that player 1 should choose r_1 to maximize his expected utility (given in the objective function (5.4)) subject to constraints (5.5)–(5.7) which require that r_1 corresponds to a valid realization plan.

In an equilibrium, player 1 and player 2 best respond simultaneously. However, if we treated both r_1 and r_2 as variables in Equations (5.4)–(5.7) then the objective function (5.4) would no longer be linear. Happily, this problem does not arise in the dual of this linear program.[6] Denote the variables of our dual LP as v; there will be one v_I for every information set $I \in I_1$ (corresponding to constraint (5.6) from the primal) and one additional variable v_0 (corresponding to constraint (5.5)). For notational convenience, we define a "dummy" information set 0 for player 1; thus, we can consider every dual variable to correspond to an information set.

$\mathcal{I}_i(\sigma_i)$: the last information set encountered in σ_i

We now define one more function. Let $\mathcal{I}_i : \Sigma_i \mapsto I_i \cup \{0\}$ be a mapping from player i's sequences to information sets. We define $\mathcal{I}_i(\sigma_i)$ to be 0 iff $\sigma_i = \emptyset$, and to be the information set $I \in I_i$ in which the final action in σ_i was taken otherwise. Note that the information set in which each action in a sequence was taken is unambiguous because of our assumption that the game has perfect recall. Finally, we again overload notation to simplify the expressions that follow. Given a set of sequences Σ', let $\mathcal{I}_i(\Sigma')$ denote $\{\mathcal{I}_i(\sigma')|\sigma'_i \in \Sigma'_i\}$. Thus, for example, $\mathcal{I}_i(\text{Ext}_i(\sigma_1))$ is the (possibly empty) set of final information sets encountered in the (possibly empty) set of extensions of σ_i.

$\mathcal{I}_i(\text{Ext}_i(\sigma_1)) = \{\mathcal{I}_i(\sigma')|\sigma' \in \text{Ext}_i(\sigma_1)\}$

The dual LP follows.

$$\text{minimize} \quad v_0 \tag{5.8}$$

$$\text{subject to} \quad v_{\mathcal{I}_1(\sigma_1)} - \sum_{I' \in \mathcal{I}_1(\text{Ext}_1(\sigma_1))} v_{I'} \geq \sum_{\sigma_2 \in \Sigma_2} g_1(\sigma_1, \sigma_2) r_2(\sigma_2) \quad \forall \sigma_1 \in \Sigma_1 \tag{5.9}$$

The variable v_0 represents player 1's expected utility under the realization plan he chooses to play, given player 2's realization plan. In the optimal solution v_0 will correspond to player 1's expected utility when he plays his best response. (This follows from LP duality—primal and dual linear programs always have the same optimal solutions.) Each other variable v_I can be understood as the portion of this expected utility that player 1 will achieve under his best-response realization plan in the subgame starting from information set I, again given player 2's realization plan r_2.

There is one version of constraint (5.9) for every sequence σ_1 of player 1. Observe that there is always exactly one positive variable on the left-hand side of the inequality, corresponding to the information set of the last action in the sequence. There can also be zero or more negative variables, each of which corresponds to a different information set in which player 1 can end up after playing the given sequence. To understand this constraint, we will consider three different cases.

First, there are zero of these negative variables when the sequence cannot be extended—that is, when player 1 never gets to move again after $\mathcal{I}_1\sigma_1$, no

6. The dual of a linear program is defined in Appendix B.

matter what player 2 does. In this case, the right-hand side of the constraint will evaluate to player 1's expected payoff from the subgame beyond σ_1, given player 2's realization probabilities r_2. (This subgame is either a terminal node or one or more decision nodes for player 2 leading ultimately to terminal nodes.) Thus, here the constraint states that the expected utility from a decision at information set $\mathcal{I}_1(\sigma_1)$ must be at least as large as the expected utility from making the decision according to σ_1. In the optimal solution this constraint will be realized as equality if σ_1 is played with positive probability; contrapositively, if the inequality is strict, σ_1 will never be played.

The second case is when the structure of the game is such that player 1 will face another decision node no matter how he plays at information set $\mathcal{I}_1(\sigma_1)$. For example, this occurs if $\sigma_1 = \emptyset$ and player 1 moves at the root node: then $\mathcal{I}_1(\text{Ext}_1(\sigma_1)) = \{1\}$ (the first information set). As another example, if player 2 takes one of two moves at the root node and player 1 observes this move before choosing his own move, then for $\sigma_1 = \emptyset$ we will have $\mathcal{I}_1(\text{Ext}_1(\sigma_1)) = \{1, 2\}$. Whenever player 1 is guaranteed to face another decision node, the right-hand side of constraint (5.9) will evaluate to zero because $g_1(\sigma_1, \sigma_2)$ will equal 0 for all σ_2. Thus the constraint can be interpreted as stating that player 1's expected utility at information set $\mathcal{I}_1(\sigma_1)$ must be equal to the sum of the expected utilities at the information sets $\mathcal{I}_1(\text{Ext}_1(\sigma_1))$. In the optimal solution, where v_0 is minimized, these constraints are always be realized as equality.

Finally, there is the case where there exist extensions of sequence σ_1, but where it is also possible that player 2 will play in a way that will deny player 1 another move. For example, consider the game in Figure 5.2 from earlier in the chapter. If player 1 adopts the sequence B at his first information set, then he will reach his second information set if player 2 plays F, and will reach a leaf node otherwise. In this case there will be both negative terms on the left-hand side of constraint (5.9) (one for every information set that player 1 could reach beyond sequence σ_1) *and* positive terms on the right-hand side (expressing the expected utility player 1 achieves for reaching a leaf node). Here the constraint can be interpreted as asserting that i's expected utility at $\mathcal{I}_1(\sigma_1)$ can only exceed the sum of the expected utilities of i's successor information sets by the amount of the expected payoff due to reaching leaf nodes from player 2's move(s).

Computing equilibria of two-player zero-sum games

For two-player zero-sum games the sequence form allows us to write a linear program for computing a Nash equilibrium that can be solved in time polynomial in the size of the extensive form. Note that in contrast, the methods described in Section 4.1 would require time exponential in the size of the extensive form, because they require construction of an LP with a constraint for each pure strategy of each player and a variable for each pure strategy of one of the players.

This new linear program for games in sequence form can be constructed quite directly from the dual LP given in Equations (5.8)–(5.9). Intuitively, we simply treat the terms $r_2(\cdot)$ as variables rather than constants, and add in the constraints

from Definition 5.2.9 to ensure that r_2 is a valid realization plan. The program follows.

$$\begin{aligned}
&\text{minimize} && v_0 && && (5.10)\\
&\text{subject to} && v_{\mathcal{I}_1(\sigma_1)} - \sum_{I' \in \mathcal{I}_1(\text{Ext}_1(\sigma_1))} v_{I'} \geq \sum_{\sigma_2 \in \Sigma_2} g_1(\sigma_1, \sigma_2) r_2(\sigma_2) && \forall \sigma_1 \in \Sigma_1 && (5.11)\\
& && r_2(\emptyset) = 1 && && (5.12)\\
& && \sum_{\sigma_2' \in \text{Ext}_2(I)} r_2(\sigma_2') = r_2(\text{seq}_2(I)) && \forall I \in I_2 && (5.13)\\
& && r_2(\sigma_2) \geq 0 && \forall \sigma_2 \in \Sigma_2 && (5.14)
\end{aligned}$$

The fact that r_2 is now a variable means that player 2's realization plan will now be selected to minimize player 1's expected utility when player 1 best responds to it. In other words, we find a minmax strategy for player 2 against player 1, and since we have a two-player zero-sum game it is also a Nash equilibrium by Theorem 3.4.4. Observe that if we had tried this same trick with the primal LP given in Equations (5.4)–(5.7) we would have ended up with a quadratic objective function, and hence not a linear program.

Computing equilibria of two-player general-sum games

For two-player general-sum games, the problem of finding a Nash equilibrium can be formulated as a linear complementarity problem as follows.

$$\begin{aligned}
& r_1(\emptyset) = 1 && && (5.15)\\
& r_2(\emptyset) = 1 && && (5.16)\\
& \sum_{\sigma_1' \in \text{Ext}_1(I)} r_1(\sigma_1') = r_1(\text{seq}_1(I)) && \forall I \in I_1 && (5.17)\\
& \sum_{\sigma_2' \in \text{Ext}_2(I)} r_2(\sigma_2') = r_2(\text{seq}_2(I)) && \forall I \in I_2 && (5.18)\\
& r_1(\sigma_1) \geq 0 && \forall \sigma_1 \in \Sigma_1 && (5.19)\\
& r_2(\sigma_2) \geq 0 && \forall \sigma_2 \in \Sigma_2 && (5.20)\\
& \left(v^1_{\mathcal{I}_1(\sigma_1)} - \sum_{I' \in \mathcal{I}_1(\text{Ext}_1(\sigma_1))} v^1_{I'}\right) - \left(\sum_{\sigma_2 \in \Sigma_2} g_1(\sigma_1, \sigma_2) r_2(\sigma_2)\right) \geq 0 && \forall \sigma_1 \in \Sigma_1 && (5.21)\\
& \left(v^2_{\mathcal{I}_2(\sigma_2)} - \sum_{I' \in \mathcal{I}_2(\text{Ext}_2(\sigma_2))} v^2_{I'}\right) - \left(\sum_{\sigma_1 \in \Sigma_1} g_2(\sigma_1, \sigma_2) r_1(\sigma_1)\right) \geq 0 && \forall \sigma_2 \in \Sigma_2 && (5.22)\\
& r_1(\sigma_1)\left[\left(v^1_{\mathcal{I}_1(\sigma_1)} - \sum_{I' \in \mathcal{I}_1(\text{Ext}_1(\sigma_1))} v^1_{I'}\right) - \left(\sum_{\sigma_2 \in \Sigma_2} g_1(\sigma_1, \sigma_2) r_2(\sigma_2)\right)\right] = 0 && \forall \sigma_1 \in \Sigma_1 && (5.23)\\
& r_2(\sigma_2)\left[\left(v^2_{\mathcal{I}_2(\sigma_2)} - \sum_{I' \in \mathcal{I}_2(\text{Ext}_2(\sigma_2))} v^2_{I'}\right) - \left(\sum_{\sigma_1 \in \Sigma_1} g_2(\sigma_1, \sigma_2) r_1(\sigma_1)\right)\right] = 0 && \forall \sigma_2 \in \Sigma_2 && (5.24)
\end{aligned}$$

Like the linear complementarity problem for two-player games in normal form given in Equations (4.14)–(4.19) on Page 91, this is a feasibility problem

consisting of linear constraints and complementary slackness conditions. The linear constraints are those from the primal LP for player 1 (constraints (5.15), (5.17), and (5.19)), from the dual LP for player 1 (constraint (5.21)), and from the corresponding versions of these primal and dual programs for player 2 (constraints (5.16), (5.18), (5.20), and (5.22)). Note that we have rearranged some of these constraints by moving all terms to the left side, and have superscripted the v's with the appropriate player number.

If we stopped at constraint (5.22) we would have a linear program, but the variables v would be allowed to take arbitrarily large values. The complementary slackness conditions (constraints (5.23) and (5.24)) fix this problem at the expense of shifting us from a linear program to a linear complementarity problem. Let us examine constraint (5.23). It states that either sequence σ_1 is never played (i.e., $r_1(\sigma_1) = 0$) or that

$$v^1_{\mathcal{I}_1(\sigma_1)} - \sum_{I' \in \mathcal{I}_1(\text{Ext}_1(\sigma_1))} v^1_{I'} = \sum_{\sigma_2 \in \Sigma_2} g_1(\sigma_1, \sigma_2) r_2(\sigma_2). \tag{5.25}$$

What does it mean for Equation (5.25) to hold? The short answer is that this equation requires a property that we previously observed of the optimal solution to the dual LP given in Equations (5.8)–(5.9): that the weak inequality in constraint (5.9) will be realized as strict equality whenever the corresponding sequence is played with positive probability. We were able to achieve this property in the dual LP by minimizing v_0; however, this does not work in the two-player general-sum case where we have both v^1_0 and v^2_0. Instead, we use the complementary slackness idea that we previously applied in the LCP for normal-form games (constraint (4.19)).

This linear complementarity program cannot be solved using the Lemke–Howson algorithm, as we were able to do with our LCP for normal-form games. However, it *can* be solved using the Lemke algorithm, a more general version of Lemke–Howson. Neither algorithm is polynomial time in the worst case. However, it is exponentially faster to run the Lemke algorithm on a game in sequence form than it is to run the Lemke–Howson algorithm on the game's induced normal form. We omit the details of how to apply the Lemke algorithm to sequence-form games, but refer the interested reader to the reference given at the end of the chapter.

5.2.4 *Sequential equilibrium*

We have already seen that the Nash equilibrium concept is too weak for perfect-information games, and how the more selective notion of subgame-perfect equilibrium can be more instructive. The question is whether this essential idea can be applied to the broader class of imperfect-information games; it turns out that it can, although the details are considerably more involved.

Recall that in a subgame-perfect equilibrium we require that the strategy of each agent be a best response in every subgame, not only (rather than L). It is immediately apparent that the definition does not apply in imperfect-information games, if for no other reason than we no longer have a well-defined notion of a subgame. What we have instead at each information set is a "subforest" or a

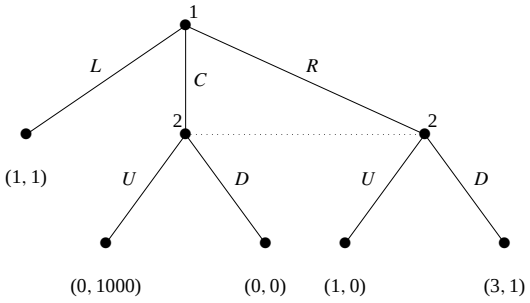

Figure 5.15 Player 2 knows where in the information set he is.

collection of subgames. We could require that each player's strategy be a best response in each subgame in each forest, but that would be both too strong a requirement and too weak. To see why it is too strong, consider the game in Figure 5.15.

The pure strategies of player 1 are $\{L, C, R\}$ and of player 2 $\{U, D\}$. Note also that the two pure Nash equilibria are (L, U) and (R, D). But should either of these be considered "subgame perfect?" On the face of it the answer is ambiguous, since in one subtree U (dramatically) dominates D and in the other D dominates U. However, consider the following argument. R dominates C for player 1, and player 2 knows this. So although player 2 does not have explicit information about which of the two nodes he is in within his information set, he can deduce that he is in the rightmost one based on player 1's incentives, and hence will go D. Furthermore player 1 knows that player 2 can deduce this, and therefore player 1 should go R. Thus, (R, D) is the only subgame-perfect equilibrium.

This example shows how a requirement that a sub strategy be a best response in all subgames is too simplistic. However, in general it is not the case that subtrees of an information set can be pruned as in the previous example so that all remaining ones agree on the best strategy for the player. In this case the naive application of the SPE intuition would rule out all strategies.

There have been several related proposals that apply the intuition underlying subgame-perfection in more sophisticated ways. One of the more influential notions has been that of *sequential equilibrium* (SE). It shares some features with the notion of trembling-hand perfection, discussed in Section 3.4.6. Note that indeed trembling-hand perfection, which was defined for normal-form games, applies here just as well; just think of the normal form induced by the extensive-form game. However, this notion makes no reference to the tree structure of the game. SE does, but at the expense of additional complexity.

sequential equilibrium

Sequential equilibrium is defined for games of perfect recall. As we have seen, in such games we can restrict our attention to behavioral strategies. Consider for the moment a fully mixed-strategy profile.[7] Such a strategy profile induces a positive probability on every node in the game tree. This means in particular

7. Again, recall that a strategy is fully mixed if, at every information set, each action is given some positive probability.

that every information set is given a positive probability. Therefore, for a given fully mixed-strategy profile, one can meaningfully speak of i's expected utility, given that he finds himself in any particular information set. (The expected utility of starting at any node is well defined, and since each node is given positive probability, one can apply Bayes' rule to aggregate the expected utilities of the different nodes in the information set.) If the fully mixed-strategy profile constitutes an equilibrium, it must be that each agent's strategy maximizes his expected utility in each of his information sets, holding the strategies of the other agents fixed.

All of the preceding discussion is for a fully mixed-strategy profile. The problem is that equilibria are rarely fully mixed, and strategy profiles that are not fully mixed do *not* induce a positive probability on every information set. The expected utility of starting in information sets whose probability is zero under the given strategy profile is simply not well defined. This is where the ingenious device of SE comes in. Given any strategy profile S (not necessarily fully mixed), imagine a probability distribution $\mu(h)$ over each information set. μ has to be *consistent* with S, in the sense that for sets whose probability is nonzero under their parents' conditional distribution S, this distribution is precisely the one defined by Bayes' rule. However, for other information sets, it can be any distribution. Intuitively, one can think of these distributions as the new beliefs of the agents, if they are surprised and find themselves in a situation they thought would not occur.[8] This means that each agent's expected utility is now well defined in any information set, including those having measure zero. For information set h belonging to agent i, with the associated probability distribution $\mu(h)$, the expected utility under strategy profile S is denoted by $u_i(S \mid h, \mu(h))$.

With this, the precise definition of SE is as follows.

Definition 5.2.10 (Sequential equilibrium) *A strategy profile S is a* sequential equilibrium *of an extensive-form game G if there exist probability distributions $\mu(h)$ for each information set h in G, such that the following two conditions hold:*

1. $(S, \mu) = \lim_{n\to\infty}(S^n, \mu^n)$ *for some sequence* $(S^1, \mu^1), (S^2, \mu^2), \ldots$, *where S^n is fully mixed, and μ^n is consistent with S^n (in fact, since S^n is fully mixed, μ^n is uniquely determined by S^n); and*
2. *For any information set h belonging to agent i, and any alternative strategy S_i' of i, we have that*

$$u_i(S \mid h, \mu(h)) \geq u_i((S', S_{-i}) \mid h, \mu(h)).$$

Analogous to subgame perfect equilibria in games of perfect information, sequential equilibria are guaranteed to always exist.

Theorem 5.2.11 *Every finite game of perfect recall has a sequential equilibrium.*

Finally, while sequential equilibria are defined for games of imperfect information, they are obviously also well defined for the special case of games of

8. This construction is essentially that of an LPS, discussed in Chapter 13.

perfect information. This raises the question of whether, in the context of games of perfect information, the two solution concepts coincide. The answer is that they almost do, but not quite.

Theorem 5.2.12 *Every subgame-perfect equilibrium is a sequential equilibrium, but the converse is not true in general.*[9]

5.3 History and references

As in Chapter 3, much of the material in this chapter is covered in modern game theory textbooks. Some of the historical references are as follows. The earliest game-theoretic publication is arguably that of Zermelo, who in 1913 introduced the notions of a game tree and backward induction and argued that in principle chess admits a trivial solution [Zermelo, 1913]. It was already mentioned in Chapter 3 that extensive-form games were discussed explicitly in von Neumann and Morgenstern [1944], as was backward induction. Subgame perfection was introduced by Selten [1965], who received a Nobel Prize in 1994. The material on computing all subgame-perfect equilibria is based on Littman et al. [2006]. The Centipede game was introduced by Rosenthal [1981]; many other papers discuss the rationality of backward induction in such games [Aumann, 1995, 1996; Binmore, 1996].

In 1953 Kuhn introduced extensive-form games of imperfect information, including the distinction and connection between mixed and behavioral strategies [Kuhn, 1953]. The sequence form, and its application to computing the equilibria of zero-sum games of imperfect information with perfect recall, is due to von Stengel [1996]. Many of the same ideas were developed earlier by Koller and Megiddo [1992]; see [von Stengel, 1996, pp. 242–243] for the distinctions. The use of the sequence form for computing the equilibria of general-sum two-player games of imperfect information is explained by Koller et al. [1996]. Sequential equilibria were introduced by Kreps and Wilson [1982]. Here, as in normal-form games, the full list of alternative solution concepts and connection among them is long, and the interested reader is referred to Hillas and Kohlberg [2002] and Govindan and Wilson [2005b] for a more extensive survey than is possible here.

9. For the record, the converse *is* true in so-called *generic games*, but we do not discuss those here. We do discuss genericity in *normal-form* games in Chapter 7, though there too only briefly.