CHAPTER 11

# Factoring

## 11.1 Introduction

A *compound game* is a single game consisting of two or more individual games. The state of a compound game is a composition of the states of the individual games. On each step of a compound game, the players perform actions in each of the individual games. A compound game is over when either one or all of the individual games are over (depending on the type of compound game).

As an example, consider Hodgepodge. One state of the game is illustrated below. Hodgepodge is a combination of Chess and Go. On each step, a player makes a move on each board.

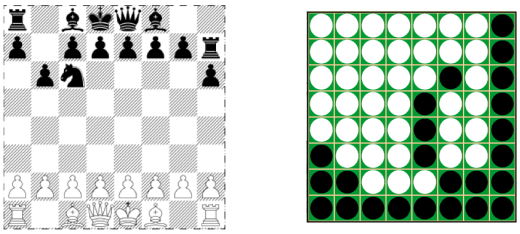

Figure 11.1 - One state in Hodgepodge

Using the techniques we have seen thus far, compound games can be quite expensive to play. Unless a player recognizes that there are independent subgames, it is likely to search a game tree that is far larger than it needs to be. If one subgame has branching factor $a$ and a second has branching factor $b$, then the branching factor of the joint game is $a \times b$, and the fringe of the game tree at depth $d$ is likely to be something like $(a \times b)^d$. This is wasteful if the two subgames are independent. In that case, there are two trees - one with branching factor $a$ and one with branching factor $b$, and the total size of the fringe of these trees at depth $d$ should be only $a^d + b^d$.

Factoring is the process of discovering independent games inside of larger games. Once discovered, game players can use these factors to play the individual games independently of each other and thus cut down on the combinatoric cost of playing such games. It turns out that it is often easier to discover independent subgames using the propnet representation of games rather than the original GDL.

In this chapter, we look at some elementary techniques for factoring games using propnets. In Section 2, we talk about discovering factors with completely independent subgames. In Section 3, we talk about factoring with interdependent goals and rewards; and, in Section 4, we talk about factoring with interdependent actions. In section 5, we discuss conditional factors, i.e. factors that appear only in certain states of games.

## 11.2 Compound Games With Independent Subgames

We begin our discussion of factoring with the simple case of compound games consisting of multiple completely independent subgames only one of which is relevant to the overall game. Without factoring, a player is likely to consider actions in all subgames when only one of the subgames matters. This is admittedly a very special case. It does not arise often, and it can be solved by means other than factoring (though not by the methods we have seen thus far). Nevertheless, it is worth considering because it prepares us for factoring more complicated games.

Multiple Buttons and Lights is an example of a game of this sort. See below. The overall game consists of multiple copies of Buttons and Lights. In each copy of Buttons and Lights, there are three base propositions (the lights) and three actions (the buttons). Pushing the first button in each group toggles the first light; pushing the second button in each group interchanges the first and second lights; and pushing the third button in each group interchanges the second and third lights. Initially, the lights are all off. The goal is to turn on all of the lights in the middle group. (The settings of the other lights are irrelevant.)

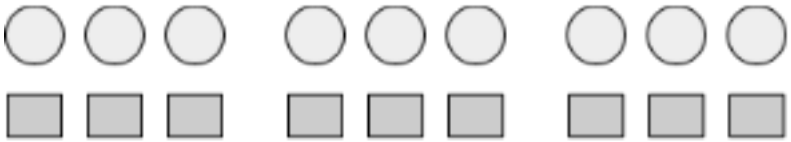

Figure 11.2 - Multiple Buttons and Lights

The GDL for this game is shown below. There is just one role. There are nine base propositions, and there are nine actions. All actions are legal at all times. Each `a(`*i*`)` toggles the corresponding `p(`*i*`)`. Each `b(`*i*`)` interchanges the values of `p(`*i*`)` to `q(`*i*`)`. And each `c(`*i*`)` interchanges the values of `q(`*i*`)` and `r(`*i*`)`. The game ends when `p(2)` and `q(2)` and `r(2)` are true. In this case, the player gets 100 points. (In violation of our rule about finiteness, there is no step counter, so this game could, in principle, run forever.)

```
role(white)

base(p(X)) :- index(X)
base(q(X)) :- index(X)
base(r(X)) :- index(X)

input(white,a(X)) :- index(X)
input(white,b(X)) :- index(X)
input(white,c(X)) :- index(X)

index(1)
index(2)
index(3)

legal(a(X)) :- index(X)
legal(b(X)) :- index(X)
legal(c(X)) :- index(X)

next(p(X)) :- does(white,a(X)) & ~true(p(X))
next(p(X)) :- does(white,b(X)) & true(q(X))
next(p(X)) :- does(white,c(X)) & true(p(X))
next(q(X)) :- does(white,a(X)) & true(q(X))
next(q(X)) :- does(white,b(X)) & true(p(X))
next(q(X)) :- does(white,c(X)) & true(r(X))
next(r(X)) :- does(white,a(X)) & true(r(X))
next(r(X)) :- does(white,b(X)) & true(r(X))
next(r(X)) :- does(white,c(X)) & true(q(X))

goal(white,100) :- terminal
goal(white,0) :- ~terminal

terminal :- true(p(2)) & true(q(2)) & true(r(2))
```

Figure 11.3 shows the propnet for Multiple Buttons and Lights. There are three disjoint parts of the propnet - one portion for the first group of buttons and lights, a second portion for the second group, and a third portion for the third group. Note that the goal and termination conditions are based entirely on the lights in the second group.

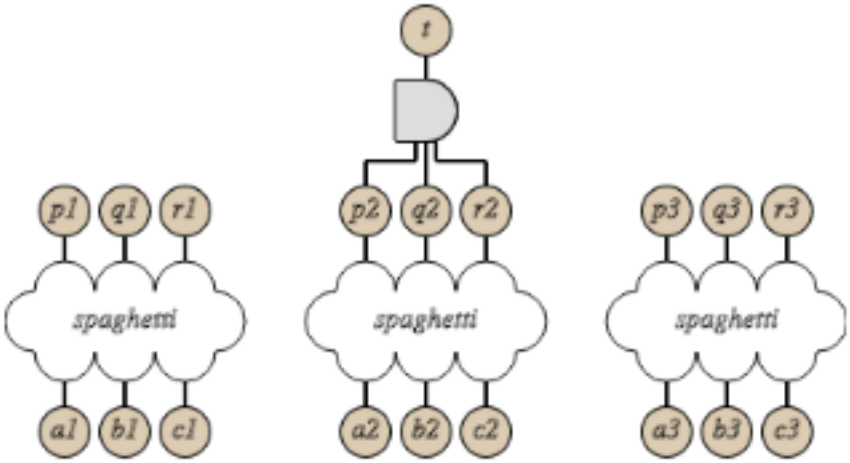


Figure 11.3 - Propnet for Multiple Buttons and Lights

Looking at the propnet for this game, it is easy to see that the game has a very special structure. The propnet consists of three completely disconnected subnets, one for each subgame. Finding factors in situations like this is easy - it can be computed in time that is polynomial in the size of the propnet.

Note that this technique can be applied equally well to multi-player games. Consider, for example, Multiple Tic Tac Toe, i.e. three games of Tic Tac Toe glued together in which only the middle game matters. See below.

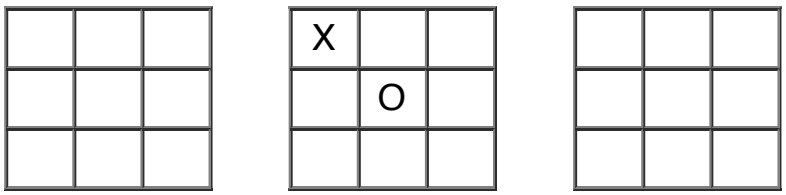


Figure 11.4 - Multiple TicTacToe

The propnet for Multiple TicTacToe is similar to the propnet for Multiple Buttons and Lights, and it is possible to find the structure for Multiple TicTacToe just as easily as for Multiple Buttons and Lights.

## 11.3 Compound Games With Interdependent Termination

In this section, we consider *compound games with interdependent termination*. As with the games discussed in the preceding section, actions are partitioned over distinct subgames and there are no incoming connections between those subgames. The main difference is that the termination condition for the overall game can depend on more than one and perhaps all of the subgames.

In games of this sort, the termination of the overall game can be defined as any boolean combination of conditions in the individual subgames. In the case where the combination is a disjunction, the game is said to have *disjunctive termination*. In the case where the combination is a conjunction, the game is said to have *conjunctive termination*.

As a simple example of a factorable game with disjunctive goals, consider Best Buttons and Lights. In Multiple Buttons and Lights, as defined in the preceding section, only one group of buttons and lights matters. In Best Buttons and Lights, the compound game terminates whenever *any* group terminates. This gives the player the freedom to play whichever subgame it likes and rest assured that, if it succeeds on that subgame, it succeeds on the overall game with the same score.

Figure 11.5 presents a sketch of the propnet for this game.

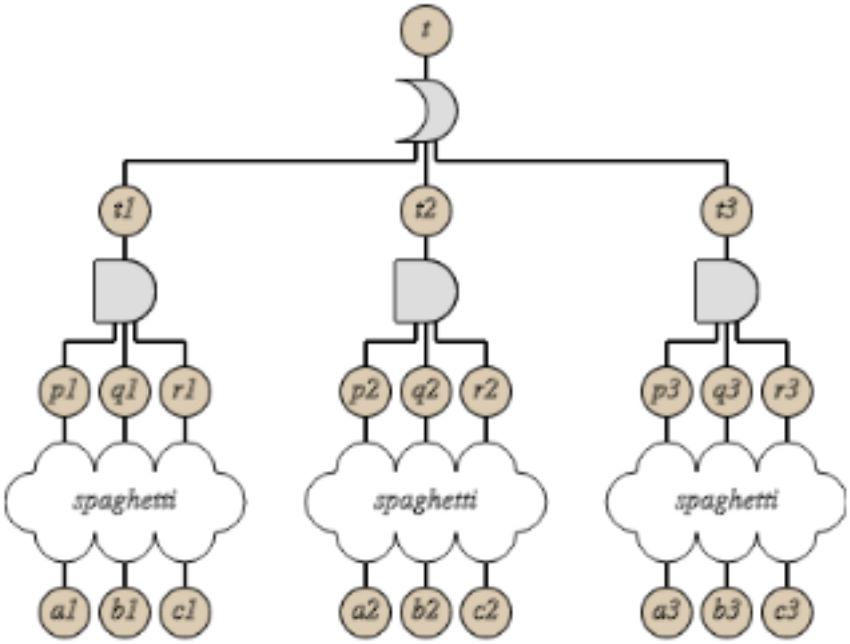


Figure 11.5 - Propnet for Best Buttons and Lights

The good news is that we can extend the techniques discussed in the preceding section to this case. Let's consider the disjunctive case. If the connective leading to a termination is a disjunction with its inputs in turn supplied by nodes in different subgames, then we simply cut off that node and inputs to the or-gate termination nodes for the overall game. We repeat this process so long as we encounter only disjunctions. If the game is truly disjunctive, this will lead to a separable propnet. See Figure 11.6.

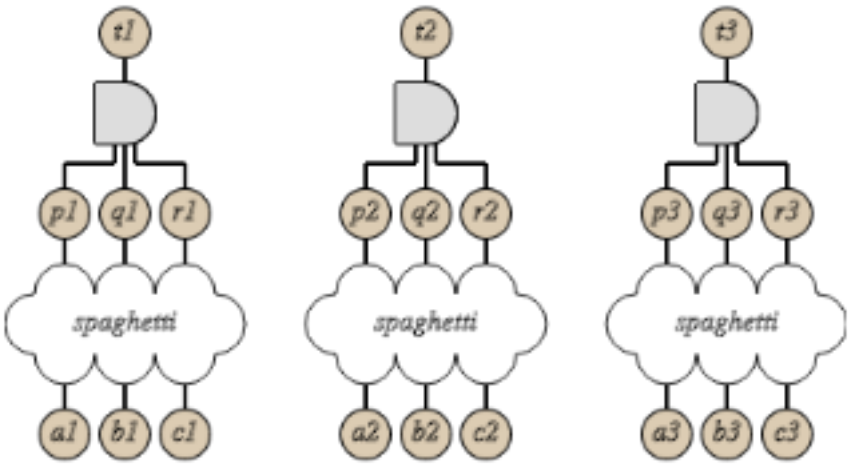


Figure 11.6 - Modified Propnet for Best Buttons and Lights

If this process succeeds in factoring the propnet, then the player simply picks one of the subgames and proceeds as with the case of independent subgames. Alternatively and better, the player tries playing all of the subgames and picks the one with the best score. Or at least that is the basic idea.

Unfortunately it is not quite that simple. There is a problem that does not arise in the case of completely independent subgames. The player may choose a subgame, find a winning strategy, and begin executing that strategy. Unfortunately, in the course of execution on the chosen subgame, one of the other subgames may terminate, terminating the game as a whole before the strategy in the chosen subgame is complete. This can lead to a lower score than the player might have expected.

The solution to this problem is to check each of the subgames for termination when no actions are played. We take the shortest time period and play each of the other subgames with that as step limit and take the one that provides the best result. For the subgame with the shortest termination, if there is no other game with that step limit, we try the next shortest step limit.

Although in this approach, the player searches all of the subgames more than once, this is usually a lot less expensive that searching the game tree for the unfactored game because it is not cross multiplying the branching factors of the independent games as it would if it did not use the game's factors.

## 11.4 Compound Games With Interdependent Actions

In this section, we look at compound games with interdependent actions. By interdependence of actions, we mean that those actions have effects on more than one of the subgames of the compound game.

On first blush, it might seem that games of this sort are not factorable. However, this is not necessarily true. Under certain circumstances, even with interdependent actions, it is possible to identify factors and use those factors to search the game tree more efficiently than without considering these factors.

As an example, consider the game of Joint Buttons and Lights. See the illustration in figure 11.7. As with the other Buttons and Lights variants that we have seen, the lights in this game are organized into groups of three. However, unlike the previous variants, buttons are not associated with specific groups. Instead, each button has effects on all groups. Button *aaa* toggles the first light in the first group *and* the first light in the second group*and* the first light in the third group. Button *aab* toggles the first light in the first group and the first light in the second group and interchanges the value of the first light and the second light in the third group. And so forth.

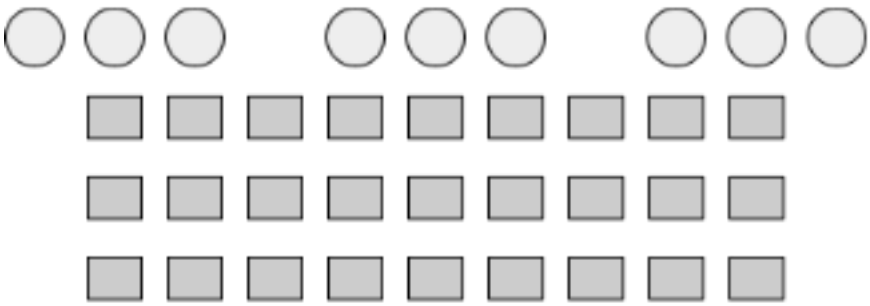

Figure 11.7 - Joint Buttons and Lights

Although all of the buttons in this game affect all of the subgames, the game is still factorable with just three actions per factor. The reason is that there is one button in the compound game for each combination of actions in the other two subgames. Thus the game trees for each subgame can be searched independently of each other and the actions chosen can be reassembled into overall actions of the compound game. For example, if action *a* is chosen in the first and second subgames and action *c* is chosen in the third subgame, then action *aac* can be performed in the compound game.

Recognizing when this can be done is not easy, but it is doable. There are two steps. In the first step, the player groups actions into equivalence classes of actions for each subgame. In the second step, the player determines whether these classes satisfy the lossless join property (defined below).

Finding equivalence classes is done by looking at the propnet for each subgame. Two actions are equivalent if they are used in the same way in the propnet for that subgame. For example, if each action is input to an and-gate with the same other input and if the outputs are connected by an or-gate, then the effects of one action cannot be distinguished from the effects of the other action; hence they are equivalent.

The process of finding equivalence classes is repeated for each potential subgame. In general, the equivalence classes for each subgame will be different. In fact, as we shall see, in order for the game to be factorable, they must be different.

Once a player has equivalence classes for each potential subgame, it then checks whether those equivalence classes are independent of each other. The criterion is simple. Each equivalence class in one potential subgame must have a non-empty intersection with each equivalence class of every other potential subgame. If this is true, then the partitions pass the lossless join criterion.

If a player finds equivalence classes for two potential subgames and they pass this lossless join test, then the player can factor the game into subgames. In order to benefit from the factoring, the player must modify each propnet so that the individual actions are replaced with the equivalence classes of which they are members. This cuts down on the number of possible actions to consider. Otherwise, the branching factor of the game trees for the subgames would be just as large as the branching factor for the overall game.

Once this is done, the player can then search the game trees for the different subgames to select actions to perform in each game. It can then find an action in the compound game for the particular combination of subgame actions. This is always doable provided that the partitioning of actions satisfies the lossless join property, which has already been confirmed. The player then performs any action in the intersection of the equivalence classes chosen in this process.

## 11.5 Conditional Independence

We now consider the class of games that, over time, become factorable. For example, a game might not initially be separable into independent games; but it may, after entering a certain state, become separable.

Consider the following game, called Joint Tic Tac Toe - two games of Tic Tac Toe connected by a single square that connects the two. The goal of the game for a player is to get two lines, a row, column or diagonal of that player's mark, with at least two of the marks residing in a specific tic tac toe domain. Diagonals through the middle square do not count. On each turn, the player in control can place two marks, either one in each distinct Tic Tac Toe domain or one mark in a Tic Tac Toe domain and one in the center square. Suppose that the state of the game is as follows.

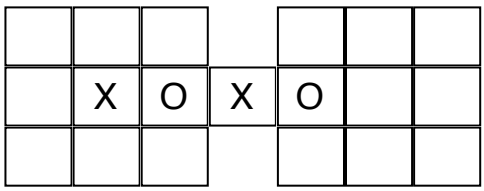


Figure 11.8 - Joint Tic Tac Toe

Once it is not possible for either player to achieve a row utilizing the center square, the only possible solutions lie in the domains of the Tic Tac Toe games that are joined by the center square. The states of the two Tic Tac Toe games can be considered independently to find the remaining optimal moves for the duration of the game. Only the game trees for each Tic Tac Toe game, modulo the center square, need to be searched to determine the remaining optimal moves. Given the current state, the game can be factored into two independent sub-games. However, from the initial state, this game cannot be factored into independent games, because the shared middle square intertwines the two domains as it is relevant to the satisfaction of goals in both sub-games.

While this game is not factorable in general, it is factorable contingent upon entering a state in which no row through the middle is possible for either player. We define the game as being contingently factorable, since it reduces to independent simultaneous sub-games, given that it enters a certain state. The raw computational benefit acquired from recognizing contingent factors is less than that of recognizing initial factors, because it requires that the game be in a specific state. However, the relative computational savings are still the same, because the number of accessible fringe nodes reduces to the sum of the remaining accessible fringe nodes of the individual games rather than the product.

At this point in time, there are no established techniques for discovering and exploiting cases of conditional independence. However, it seems likely that some of the techniques just discussed can be adapted to this case as well.