01 Feb 1987

A Logic Programming Model of the Game of Sprouts

Ralph M. Butler

Selden Y. Trimble
Missouri University of Science and Technology

Ralph W. Wilkerson
Missouri University of Science and Technology, ralphw@mst.edu

Recommended Citation

R. M. Butler et al., "A Logic Programming Model of the Game of Sprouts," *Proceedings of the Conference on Integrating Technology into Computer Science Education, ITiCSE*, pp. 319 - 323, Association for Computing Machinery, Feb 1987.
The definitive version is available at https://doi.org/10.1145/31820.31779

A LOGIC PROGRAMMING MODEL OF THE GAME OF SPROUTS

Ralph M. Butler, Selden Y. Trimble, Ralph W. Wilkerson

ABSTRACT

The game of Sprouts has intrigued mathematicians for nearly twenty years. This paper describes a representation scheme which simplifies much of the geometry of the game. Using this representation, we develop a Prolog program which will play Sprouts. It is hoped that the program will prove to be a useful research tool in finding the key to a winning strategy for Sprouts and that the representation will serve as a useful model for studying planar graphs.

1. INTRODUCTION

The game of Sprouts was introduced in 1967 by two British mathematicians, John H. Conway and Michael S. Paterson [1,p 564]. It is a two-player game involving the construction of a graph. (A graph is a non-empty set of vertices, i.e., points, and a set of edges, i.e., curves joining pairs of vertices.) The game begins with a graph that contains only vertices, the exact number being decided by mutual agreement. If n is this number, the game is called an n-point game.

The players take turns moving. A move always consists of joining two existing vertices by a curve or of joining one existing vertex to itself by a curve. In either case, a new vertex is then put somewhere in the middle of the curve. The net effect is that one new vertex and two new edges have been added to the graph. There are two restrictions: First, no vertex can have an order greater than three. (The order of a vertex is the number of edges coming into it.) Second, the graph must always be planar. (A planar graph is one which can be drawn on a plane so that its edges intersect only at vertices.)

It is convenient to think of a vertex as having three sockets into which the ends of edges may be plugged. Using this terminology, an n-point game begins with 3n sockets. Two sockets are used up each time a player puts in a new curve. However, since the player must put a new vertex in the middle of the curve, he creates one new socket. (The new vertex instantly has two edges coming into it, so it has only one socket left.) The result of a move is to reduce the number of sockets by one. Since there must be at least two sockets available in order to make a move, the n-point game cannot last more than 3n-1 moves. The game ends when it becomes impossible to move. The winner is the last to move. (A variant is that the loser is the last to move; this needs to be decided, of course, before the game begins.)

An example of a 2-point game, won by the second player, is given below:

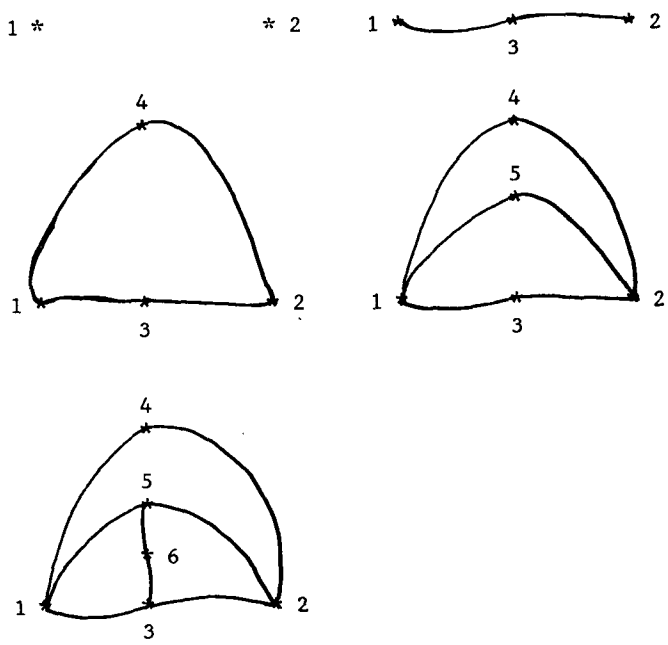

Figure 1.

Note that, in the final position, vertices 4 and 6 each have an unused socket. However, any curve joining them would have to cross an already existing edge, thereby violating the condition that the graph be planar or that no order of a vertex be greater than three.

It is known that the number of moves in an n-point game must be between 2n and 3n-1, and that these bounds are sharp [1,p 564]. However, it does not seem to be known in general whether the first player or the second player has a forced win. The following is known:

n	Player with forced win
1	2nd
2	2nd
3	1st
4	1st
5	1st
6	2nd

Table 1.

Some of this information has not come easily. The original proof that the 2nd player has a forced win in the 6-point game was forty seven pages long [1,p 568].

2. LIST MODEL OF SPROUTS

Before discussing the Prolog representation of Sprouts, our list model of Sprouts and a description of the legal moves based on this model must be presented in some detail. As an aid to understanding this, we shall refer to Figure 2, a 6-point game after six moves have been made.

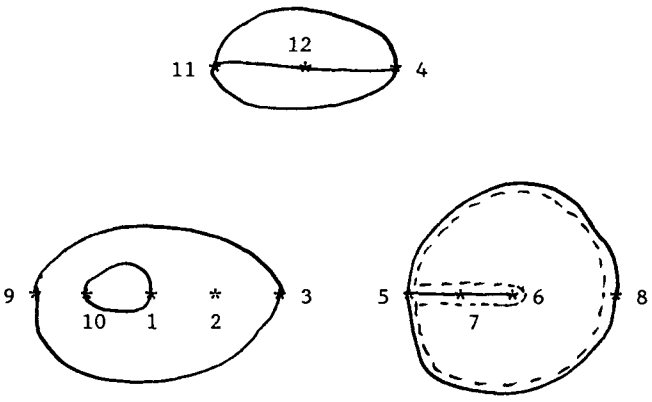

Figure 2.

The requirement that the graphs which appear in Sprouts should all be planar means that every move in Sprouts must be made inside a plane region, i.e., an open, connected set. In Figure 2, there appear to be six such regions. The first is the unbounded or outside region. Next, there are two on the left, one an annulus (doughnut) and the other a disk. Then there are the two half-disks in the middle. Finally, on the right there is a disk with a radius sticking into it. Any move made in Figure 2 must be made entirely inside one of these six regions. For example, vertex 1 could be joined to vertex 2 by drawing a curve inside the annulus. But vertex 2 could not be connected to vertex 8 because it would be necessary to draw a curve that crossed then outer circle of the annulus, thereby making the new graph non-planar.

However, it can be misleading in Sprouts to say there are six regions in Figure 2. Neither of the half-disks in the middle can have any moves made inside it because there are not enough available sockets. Vertices 4 and 11 have been completely used up, and vertex 12 has only one available socket left. So, as far as Sprouts is concerned, these two regions don't exist, i.e., Figure 2 consists of four regions.

In Sprouts, a region may be completely determined by specifying its boundary. The simplest region in Figure 2 is the disk on the left that is inside the annulus. It's boundary is just the circle on which are vertices 1 and 10. From the standpoint of Sprouts, the only significant thing about the boundary is that it contains these two vertices. So, we represent this boundary by the list (1 10).

The annulus is more complicated. It has three boundary components, i.e., its boundary is broken into three disconnected pieces. These components are represented as (9 3), (10 1), and (2).

The unbounded region has two boundary components, (9 3) and (8). This needs explaining. First of all, it is not necessary to list the two half-disks in the middle because vertices 4 and 11 cannot be used in making a move in Figure 2. Next, the representation, (8), is of the same type as the representation of the isolated vertex, 2, which is a boundary component of the annulus; yet (8) appears to be more than just an isolated vertex. But in fact, in the Sprouts game of Figure 2, (8) acts exactly as though it were just a point; so it is sufficient to represent it that way. Vertex 5 is topologically part of the boundary component, but it has no available sockets and so can be ignored.

The final region has only one boundary component, but that component is rather complicated. We represent it as (8 7 6 7) or as any cyclic permutation of this. This representation is determined by moving inside the region, but almost on the boundary, in either a clockwise or a counterclockwise direction. (Follow the dotted curve.) Notice that vertex 7 is listed twice. A later example will show the necessity for this. Also notice that vertex 5 is not listed at all.

Using the list representation of Sprouts, the position in Figure 2 is described as follows:

R1: (C1)
R2: (C1 C2 C3)
R3: (C2 C4)
R4: (C5)
C1: (1 10)
C2: (9 3)
C3: (2)
C4: (8)
C5: (8 7 6 7)

Table 2.

Here, "R" stands for region and "C" for (boundary) component.

To use this representation of Sprouts, we must reformulate the legal moves of Sprouts within this new context. While the "paper and pencil" mode of playing Sprouts is very straightforward (i.e. connect 2 points whose degree is less than 3 with a line that does not cross a line), the method of play now proceeds in a stepwise fashion.

First, a region must be specified in which the move is to take place. Two cases arise. In the first, two different components from this region and a point on each component are selected. The two points are joined creating a new component which replaces the two original components. No new regions are created. In our representation, the move is interpreted as follows: Let C1 = (P1 P2 ... Pn) and C2 = (Q1 Q2 ... Qm) be the boundary components being connected and suppose furthermore that Pj is being joined to Qi. Let R denote the newly added point on the curve joining Pj and Qi. Also let Pk ... Pt (or Qk ... Qt), for k and t any values, stand for all points from Pk to Pt except when Pk = Pt and in this case it will stand for just Pk. The order may be either increasing or decreasing depending upon whether k < t or not. There are four possible subcases which can occur. If C1 = (P1) and C2 = (Q1) consist of just singleton points then the new component will be the list (P1 R Q1 R). The next two cases are when only one of the components consists of a single point say C1 = (P1) and C2 = (Q1 ... Qm) then the new component will be the list (P1 R Qi ... Q1 Qm ... Qi R). Lastly, if both components contain more than one point then the new component will become the list (P1 ... Pj R Qi ... Q1 Qm ... Qi R Pj ... Pn).

After the new component has been constructed, the old components are deleted from the region in which the move was made. Furthermore, it helps to keep the representation free of unnecessary information by deleting all points whose socket counts are zero. Also, all regions that contain only 0 or 1 open sockets may be deleted since a move cannot be made in a region unless there are at least 2 open sockets available.

In the second case, one boundary component and two points on this component are selected. (In this case, the points may be the same provided that the point has at least two open sockets available.) Two new regions will be created, so begin by giving them names. The unused boundary components of the old region may be assigned in any manner to the two new regions, provided that each such component goes to exactly one of the new regions.

Again, relative to our representation suppose the boundary component selected is C = (P1 ... Pn) and we wish to join Pi to Pj where i = j is permissible. Three situations can occur in this type of move: (1) a point with a socket count of 3 is connected to itself, (2) a point with socket count of 2 is connected to itself, and (3) separate points are connected. In the first case, the new components formed are identical. That is, if C = (P1) then the two new boundary components are C1 = (P1 R) and C2 = (P1 R) where R is the new vertex on the curve joining P1 to itself. In each of the last two cases, the point(s) is (are) part of a larger component. In the second case, the first component formed is the same as the components formed in the first case,i.e. (Pi R). Then, to form the second new component, the list of points is split into a Head and a Tail (i.e., everything before and after the point being connected to itself). The second component is the list of points (Head Pi R Pi Tail) where either the Head or Tail may be null. Note that if both the Head and Tail are null then the second component is just (Pi R). In the third case, the two new boundary components will be C1 = (Pi ... Pj R) and
C2 = (Pj Tail Head Pi R) where Head is all elements on the list up to but not including Pi and the Tail is all elements after Pj. It is assumed that Pi comes before Pj on the list C and as before the Head and/or Tail may be null. As before, we remove all points whose socket counts are zero and all regions that contain only 0 or 1 open sockets.

Figure 3 shows a 4-point game after four moves have been made.

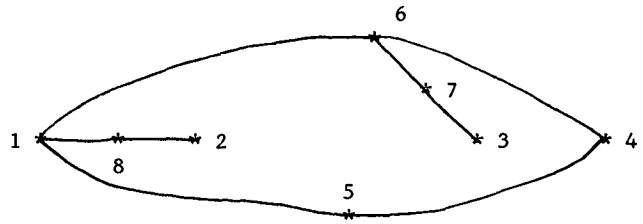

Figure 3.

The bounded region has only one component, (5 8 2 8 7 3 7 4). There are two ways to join vertices 4 and 8 in this region. Here, P8 = 4 and P4 = P2 = 8. Suppose P8 and P2 are joined. The result will be two new regions, each having one component. The new components, would be (8 2 8 7 3 7 4 9) and (4 5 8 9) before deletion of dead points or (2 7 3 7 9) and (5 9) after deletion. On the other hand, if P8 and P4 are joined, the new components, are (8 7 3 7 4 9) and (4 5 8 2 8 9) before deletion or (7 3 7 9) and (5 2 9) after. The results of the two ways of joining vertices 4 and 8 are very different. This example shows the necessity in certain cases of double-listing of a single vertex in the list representing a component.

3. THE PROLOG REPRESENTATION

Since our representation is intended to accurately model the game of Sprouts, a player should be able to draw a particular game on a piece paper as the game proceeds. Remember however, that the program does not represent the game strictly in terms of points and edges. Rather, the program maintains a database consisting of three entities: points, components, and regions. Each occurrence of an entity is represented by a sentence of a relation in the database. The regions represent the areas in which moves may be made and are defined by lists of components. The components consist of points and define the boundaries of regions. Likewise, the points on a component define that component. A move is made by selecting a region in which to move, then selecting two components (which may be the same), and finally, selecting two points (which may be the same) from the respective components.

Points, components, and regions are identified in the database by unique integers. Each point is said to have some number of "sockets". A socket is simply a location where an edge may be connected. At the beginning of a game, the relation describing the points shows each point to have three open sockets. If a point is used in a move, the open socket count for that point is decremented in the database. Table 3 is an example of how the relation representing the points may be viewed. It represents a 3-point game after point 1 has been connected to itself isolating point 2 from point 3, as in Figure 4.

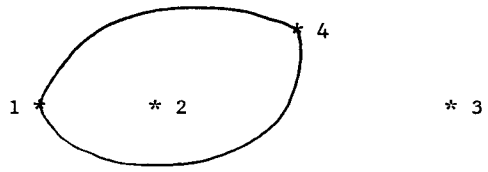

Figure 4.

point number	number of open sockets
1	1
2	3
3	3
4	1

Table 3. Relation: point-has-sockets-open

The components consist of points and form the regions. The relation describing the components in the database for the above example may be viewed as listed in Table 3.

component number	points in component
2	(2)
3	(3)
4	(1 4)

Table 4. Relation: component-has-points

Note that the points in each component are maintained in the database as a list.

The regions consist of components and define the areas in which moves may be made. The relation describing the regions in the database for the above example may be viewed as listed in Table 5.

region number	components in region
2	(2 4)
3	(3 4)

Table 5. Relation: region-has-component

The components in each region are maintained in a list much the same as points on a component.

Using these relations, it is possible to maintain a database that contains all points, components, and regions formed during the game. However, it is usually more convenient to play a game in which all "dead" points, components, and regions have been removed. For example, if a point has degree three, then that point no longer has any meaning in the game. The same is true for a region in which there is only one free socket. Our representation supports the ability to request that all such dead entities be deleted from the database.

Our implementation of the above representation is written in LPA micro-PROLOG Professional. The database consists of sentences that define the point, component, and region relations described in Tables 3, 4, and 5 above. The rules that define how to play a game and how to maintain the database during play are also implemented as Prolog relations. The current implementation supports several functions:

1. building the initial database for an n-point game,
2. updating the database each time a move is made in order to reflect the current status of the game,
3. deleting dead information from the database if that option has been selected by the user,

4. the ability to have the computer select and make a move heuristically or randomly, and

5. the ability to have the computer play an entire game by itself, using 4 above.

The code to implement these functions requires approximately 700 lines of Prolog. The code to maintain the representation without the deletion option is roughly half of the total. The program has been written and tested on an IBM PC personal computer. Thus, it is easily ported.

As stated earlier, the program currently selects a move at random if the user wishes. However, the code can be modified to include the user's own strategy. Separate strategies for the first and second players might even prove useful. The code is modularized so that such modifications should prove to be relatively simple.

We hope that others will develop an interest in attempting to discover a winning strategy for the game of Sprouts and will find our representation and program to be useful tools towards that end. For those who are interested, we would be happy to share a copy of our program.

REFERENCES

[1] E.R. Berlekamp, J. H. Conway, and R. K. Guy, Winning Ways for your mathematical plays, Academic Press, New York 1982.

[2] J.H. Briggs, LPA micro-PROLOG Professional User Guide, Logic Programming Associates Ltd, London 1985.

[3] K.L. Clark and F.G. McCabe, micro-PROLOG: Programming in Logic, Prentice/Hall, Englewood Cliffs, New Jersey 1984.

[4] Martin Gardner, Mathematical Games: Of sprouts and Brussels sprouts; games with a topological flavor, Sci. Amer. 217 1(July 1967) 112-115.

[5] Gordon Pritchett, The game of Sprouts, Two-Year Coll. Math. J. 7 4(Dec. 1976) 21-25.

Ralph M. Butler, Department of Computer Science, University of North Florida, Jacksonville, Florida, USA 32216

Selden Y. Trimble, Department of Mathematics, University of Missouri-Rolla, Rolla, Missouri, USA 65401

Ralph W. Wilkerson, Department of Computer Science, University of Missouri-Rolla, Rolla, Missouri, USA 65401