

## Chapter 3

# Informed Search

In this chapter we are going to discuss graph search algorithms and applications thereof for finding a *minimum cost* path from a start node to the goal node.

### 3.1 The Network Search Problem with Costs

The network search problem in Sect. 2.2 (Fig. 2.1) was devoid of any cost information. Let us now assume that the costs to traverse the edges of the graph in Fig. 2.1 are as indicated in Fig. 3.1.

There are two possible interpretations of the figures in Fig. 3.1: they can be thought of as costs of edge traversal or, alternatively, as edge lengths. (We prefer the latter interpretation in which case, of course, Fig. 3.1 is not to scale.) The task is to determine a minimum length path connecting  $s$  and  $g$ , or, more generally, minimum length paths connecting any two nodes.

The algorithms considered in this chapter assume the knowledge of an *heuristic distance* measure,  $H$ , between nodes. Values of  $H$  for the network in Fig. 3.1 are shown in Table 3.1. They are taken to be the estimated straight line distances between nodes and may be obtained by drawing the network in Fig. 3.1 to scale and taking measurements.

Three algorithms will be introduced here: the *A-Algorithm*, *Iterative Deepening A\** and *Iterative Deepening A\* $-\epsilon$* .

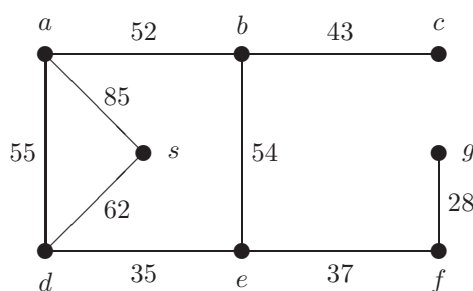


Figure 3.1: A Network with Costs

85	40	30	62	34	31	14	<i>s</i>
98	51	25	76	45	28	<i>g</i>	
109	71	54	73	37	<i>f</i>		
77	54	63	35	<i>e</i>			
55	61	88	<i>d</i>				
95	43	<i>c</i>					
52	<i>b</i>						
<i>a</i>							

Table 3.1: Straight Line Distances between Nodes in Fig. 3.1

### 3.1.1 Cost Measures

An estimated overall cost measure, calculated by the *heuristic evaluation function*  $F$ , will be attached to every path; it is represented as

$$F = G + H \quad (3.1)$$

where  $G$  is the *actual* cost incurred thus far by travelling from the start node to the current node and  $H$ , the

**ie** business school

#1 EUROPEAN BUSINESS SCHOOL  
FINANCIAL TIMES 2013

**#gobeyond**

**MASTER IN MANAGEMENT**

Because achieving your dreams is your greatest challenge. IE Business School's Master in Management taught in English, Spanish or bilingually, trains young high performance professionals at the beginning of their career through an innovative and stimulating program that will help them reach their full potential.

- Choose your area of specialization.
- Customize your master through the different options offered.
- Global Immersion Weeks in locations such as London, Silicon Valley or Shanghai.

*Because you change, we change with you.*

www.ie.edu/master-management | mim.admissions@ie.edu |

*heuristic*, is the *estimated* cost of getting from the current node to the goal node. Assume, for example, that in the network shown in Fig. 3.1 we start in  $d$  and want to end up in  $c$ . Equation (3.1) then reads for the path  $d \rightarrow s \rightarrow a$  (with obvious notation) as follows

$$\begin{aligned} F(d \rightarrow s \rightarrow a, c) &= G(d \rightarrow s \rightarrow a) + H(a, c) \\ &= (62 + 85) + 95 = 147 + 95 = 242 \end{aligned} \quad (3.2)$$

### 3.1.2 The A-Algorithm

We know from Chap. 2 that for blind search algorithms the updating of the *agenda* is crucial: Breadth First comes about by appending the list of extended paths to the list of open paths; Depth First requires these lists to be concatenated the other way round.

For the A-Algorithm, the updating of the agenda is equally important. The new agenda is obtained from the old one in the steps ① and ② below.

- ① Extend the head of the old agenda to get a list of successor paths. An intermediate, ‘working’ list will be formed by appending the tail of the old agenda to this list.
- ② The new agenda is obtained by sorting the paths in the working list from ① in ascending order of their  $F$ -values.
- ③ The steps ① and ② are iterated until the path at the head of the agenda leads to the goal node.

In the example shown in Fig. 3.2, the paths are prefixed by their respective  $F$ -values and postfixed by their respective  $G$ -values. Using this notation and the cost information, the example path in (3.2) is now denoted by  $242 - [a, s, d] - 147$ . Notice that this path also features in Fig. 3.2.

It can be shown (e.g. [23]) that if the heuristic  $H$  is *admissible*, i.e. it never overestimates the actual minimum distance travelled between two nodes, the A-Algorithm will deliver a minimum cost path if such a path exists.<sup>1</sup>In this case the A-Algorithm is referred to as an  $A^*$ -Algorithm and is termed *admissible*. (As the straight line distance is a minimum, the heuristic defined by Table 3.1 is admissible.)

### Implementation

The predicate `a_search(+Start, +Goal, -PathFound)` in `asearches.pl` implements the A-Algorithm. A few salient features of `a_search/3` will be discussed only; for details, the reader is referred to the source code which broadly follows the pattern of implementation of the blind search algorithms (Fig. 2.15, p. 65 and Fig. 2.20, p. 69).

The implementation of the A-Algorithm in `asearches.pl` uses the built-in predicate `keysort/2` to implement step ② (see inset on p. 108).

The module invoking `a_search/3` should have defined (or imported) the following predicates.

- The connectivity predicate `link/2`. For the network search problem, this is imported from `links.pl` (Fig. 2.2, p. 49).
- The estimated cost defined by `e_cost/3`. For the network search problem, this is defined in `graph_a.pl` by

---

<sup>1</sup>To be more precise, this holds only under some additional conditions which are satisfied, however, in most practical applications [23].

```
e_cost(Node,Goal,D) :- dist(Node,Goal,D).  
e_cost(Node,Goal,D) :- dist(Goal,Node,D).
```

with *dist/3* essentially implementing Table 3.1,

```
dist(s,a,85). ... dist(s,f,31). dist(s,g,14).  
dist(g,a,98). ... dist(g,f,28).  
...  
dist(b,a,52).
```



"I studied English for 16 years but...  
...I finally learned to speak it in just six lessons"  
Jane, Chinese architect

ENGLISH OUT THERE

Click to hear me talking before and after my unique course download



$[88-[d]-0] \xrightarrow{\textcircled{1}} [98-[e,d]-35, 92-[s,d]-62, 150-[a,d]-55] \xrightarrow{\textcircled{2}}$   
 $[92-[s,d]-62, 98-[e,d]-35, 150-[a,d]-55] \xrightarrow{\textcircled{1}} [242-[a,s,d]-147, 98-[e,d]-35, 150-[a,d]-55] \xrightarrow{\textcircled{2}}$   
 $[98-[e,d]-35, 150-[a,d]-55, 242-[a,s,d]-147] \xrightarrow{\textcircled{1}} [126-[f,e,d]-72, 132-[b,e,d]-89, 150-[a,d]-55, 242-[a,s,d]-147] \xrightarrow{\textcircled{2}}$   
 $[126-[f,e,d]-72, 132-[b,e,d]-89, 150-[a,d]-55, 242-[a,s,d]-147] \xrightarrow{\textcircled{1}}$   
 $[125-[g,f,e,d]-100, 132-[b,e,d]-89, 150-[a,d]-55, 242-[a,s,d]-147] \xrightarrow{\textcircled{2}}$   
 $[125-[g,f,e,d]-100, 132-[b,e,d]-89, 150-[a,d]-55, 242-[a,s,d]-147] \xrightarrow{\textcircled{1}}$   
 $[132-[b,e,d]-89, 150-[a,d]-55, 242-[a,s,d]-147] \xrightarrow{\textcircled{2}}$   
 $[132-[b,e,d]-89, 150-[a,d]-55, 242-[a,s,d]-147] \xrightarrow{\textcircled{1}}$   
 $[132-[c,b,e,d]-132, 236-[a,b,e,d]-141, 150-[a,d]-55, 242-[a,s,d]-147] \xrightarrow{\textcircled{2}}$   
 $[132-[c,b,e,d]-132, 150-[a,d]-55, 236-[a,b,e,d]-141, 242-[a,s,d]-147] \xrightarrow{\textcircled{3}} \text{success}$

Figure 3.2: Hand Computations: The Evolution of the Agenda for the A-Algorithm (from  $d$  to  $c$  in Fig 3.1)

- The actual edge costs defined by `edge_cost/3`. For the network search problem, this is defined in `graph_a.pl` by

```
edge_cost(Node1,Node2,Cost) :- link(Node1,Node2),
                               e_cost(Node1,Node2,Cost).
```

---

**Built-in Predicate:** `keysort(+List,-Sorted)`

Unifies `Sorted` with the sorted version of `List`. The entries in `List` have to be in the form `key-term` and they will be sorted in ascending order of the value of `key`.

Example: Sort a list of names with ages according to increasing values of age. (Facts for `age/2` to be entered manually.)

```
?- consult(user).
|: age(adam,34).
|: age(tracy,18).
|: age(george,15).
|: Ctrl+D
% user compiled 0.00 sec, 480 bytes
Yes
?- bagof(_Age-Name,age(_Name,_Age),L), keysort(L,Sorted).
L = [34-adam, 18-tracy, 15-george]
Sorted = [15-george, 18-tracy, 34-adam]
Yes
```

---

The interactive session below shows that the path  $d \rightarrow e \rightarrow b \rightarrow c$  is a shortest one from  $d$  to  $c$ .

```
?- consult(graph_a).
% asearches compiled into a_ida_ideaeps 0.00 sec, 7,736 bytes
% links compiled into edges 0.00 sec, 1,804 bytes
% graph_a compiled 0.00 sec, 16,584 bytes
?- a_search(d,c,PathFound), total_cost(PathFound,Cost).
PathFound = [d, e, b, c]
Cost = 132
```

### 3.1.3 Iterative Deepening $A^*$ and its $\epsilon$ -Admissible Version

Application of the  $A$ -Algorithm to a more substantial example in Sect. 3.2 will reveal that the  $A$ -Algorithm may fail due to excessive memory requirements.<sup>2</sup> Clearly, there is scope for improvement.

In the mid 1980s, a new algorithm was conceived by Korf [20] combining the idea of Iterative Deepening (Sect. 2.6) with a heuristic evaluation function; the resulting algorithm is known as *Iterative Deepening  $A^*$*  ( $IDA^*$ ).<sup>3</sup> The underlying idea is as follows.

- Use Depth First as the ‘core’ of the algorithm.

---

<sup>2</sup>We can see at this stage already that there is a special case of the  $A$ -Algorithm where lots of memory is required: the  $A$ -Algorithm specializes to Breadth First if unit edge costs and the zero heuristic are assumed.

<sup>3</sup>Noteworthy is also a more recent work by Korf [21] analysing  $IDA^*$ .



- Convert the core into a kind of Bounded Depth First Search with the bound (the horizon) now not being imposed on the length of the paths but on their  $F$ -values.
- Finally, imbed this ‘modified’ Bounded Depth First Search into a framework which repeatedly invokes it with a sequence of increasing bounds. The corresponding sequence of bounds in Iterative Deepening was defined as a sequence of multiples of some constant increment; a unit increment in the model implementation. The approach here is more sophisticated. Now, in any given phase of the iteration, the next value of the bound is obtained as the minimum of the  $F$ -values of all those paths which had to be ignored in the present phase. This approach ensures that in the new iteration cycle the least number of paths is extended.

The pseudocode of  $IDA^*$  won't be given here; it should be possible to reconstruct it from the above informal description. It can be shown that  $IDA^*$  is admissible under the same assumptions as  $A^*$ .

The so-called  $\epsilon$ -admissible version of  $IDA^*$  ( $IDA^*-\epsilon$ ) is a generalization of  $IDA^*$ . It is obtained by extending the  $F$ -horizon to

$\epsilon + \text{the minimum of all } F\text{-values of paths ignored}$

with some fixed  $\epsilon \geq 0$ . (It clearly specializes to  $IDA^*$  for  $\epsilon = 0$ .) This algorithm may ‘catch’ a solution which otherwise would fall just outside the current  $F$ -horizon.  $IDA^*-\epsilon$  may therefore find suboptimal solutions with

Excellent Economics and Business programmes at:



**university of  
 groningen**





“The perfect start  
 of a successful,  
 international career.”

CLICK HERE

to discover why both socially  
 and academically the University  
 of Groningen is one of the best  
 places for a student to be

[www.rug.nl/feb/education](http://www.rug.nl/feb/education)



```

?- consult(graph_a).
% asearches compiled into a_ida_ideaeps 0.00 sec, 7,736 bytes
% links compiled into edges 0.00 sec, 1,804 bytes
% graph_a compiled 0.00 sec, 16,584 bytes
Yes
?- path.
Select start node s, a, b, ..., f, g: d.
Select goal node s, a, b, ..., f, g: c.
Select algorithm (a/ida/ideaeps)... a.
% 586 inferences in 0.00 seconds (Infinite Lips)
Solution in 3 steps.
d -> e -> b -> c
Total cost: 132
Yes

```

Figure 3.3: An Interactive Session. (See Exercise 3.1.)

Node	1	2	3	4	5	6	7	8	9	10
Co-ordinates	(1, 4)	(2, 7)	(2, 9)	(3, 4)	(3, 5)	(3, 9)	(4, 1)	(4, 5)	(4, 9)	(5, 4)

Table 3.2: Node Co-ordinates in the Network in Fig. 3.4

broadly the same effort and memory as  $IDA^*$ .<sup>4</sup>

Both versions,  $IDA^*$  and  $IDA^*-\epsilon$ , are implemented in `asearches.pl`.

**Exercise 3.1.** Complete the definition of `graph_a.pl` to solve the network search problem in Fig. 3.1 as illustrated by the interactive session in Fig. 3.3. (The user should be able to run any of the three algorithms discussed here.) ■

**Exercise 3.2.** Fig. 3.4 shows a small directed network with the nodes' co-ordinates shown in Table 3.2. Let the length of an edge be the *city block* (or *Manhattan*) distance of its endpoints.<sup>5</sup>

- Find the shortest route from node 1 to node 10 *manually* by using the *A*-Algorithm with the straight line heuristic.
- Write a module (`graph_b.pl`, say), which uses `asearches.pl`, for finding the shortest route as before but now the user should be able to select the algorithm in the style shown in Fig. 3.3. ■

**Exercise 3.3.** (*Adjacency matrix*) To represent the network in Fig. 3.4, you will have directly defined the connectivity predicate `link/2` by a collection of facts.<sup>6</sup> A more flexible and elegant alternative to record the connectivity of a network is by using an *adjacency matrix*. The entries of this are zero everywhere except for

<sup>4</sup> $IDA^*-\epsilon$  may not return an optimal solution. An example for this will be seen in Sect. 3.2.

<sup>5</sup>The city block distance between two points is the shortest distance when measured in a zigzag parallel to the co-ordinate axes. Thus, for example, the nodes 6 and 8 are  $|3 - 4| + |9 - 5| = 5$  units apart.

<sup>6</sup>In all likelihood the same goes for the predicate that you will have used to record the nodes' co-ordinates from Table 3.2.





```
?- adj(1,_A), co_ord(1,_Co), path(_A,_Co).
Select start node 1, ..., 10: 1.
Select goal node 1, ..., 10: 10.
Select algorithm (a/ida/idaeps)... a.
% 561 inferences in 0.00 seconds (Infinite Lips)
Solution in 4 steps.
1 -> 2 -> 5 -> 8 -> 10
Total cost: 10
Yes
```

Notice in particular that the predicate `path(+A,+Co)` should initiate the search for the network with adjacency matrix `A` and list of node co-ordinates `Co`. Make use of `make_links/1` and `make_co_ordinates/1` from part (a) when defining `path/2`. Your implementation will be able to cope with any directed network specified in this manner. (Minor point: Display the correct number of nodes for the user to choose from.)

- (c) Use your implementation to determine the shortest path from node 1 to node 26 in the network in Fig. 3.6, p. 113. The node co-ordinates are given in Table 3.3, and, as before, the edge lengths should be calculated

## LIGS University

### based in Hawaii, USA

is currently enrolling in the  
Interactive Online **BBA, MBA, MSc,**  
**DBA and PhD** programs:

- ▶ enroll **by October 31st, 2014** and
- ▶ **save up to 11%** on the tuition!
- ▶ pay in 10 installments / 2 years
- ▶ Interactive **Online education**
- ▶ visit [www.ligsuniversity.com](http://www.ligsuniversity.com) to find out more!

**Note: LIGS University is not accredited by any nationally recognized accrediting agency listed by the US Secretary of Education. More info [here](#).**





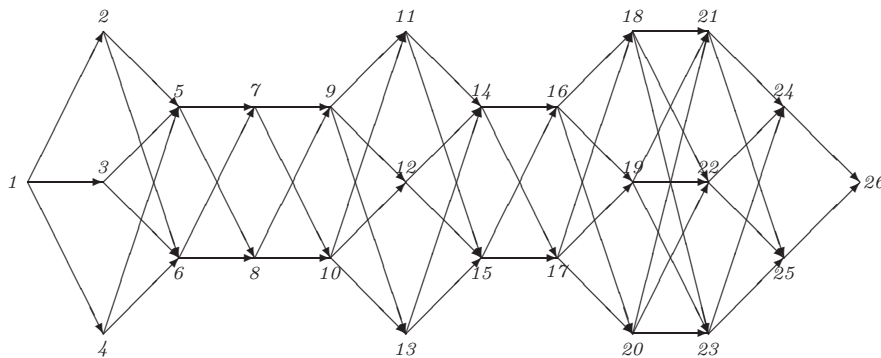


Figure 3.6: Network for Exercise 3.3, Part (c)

<i>Node</i>	1	2	3	4	5	6	7	8	9
<i>Co-ordinates</i>	(1, 2)	(2, 7)	(2, 14)	(2, 20)	(3, 2)	(3, 17)	(4, 5)	(4, 8)	(5, 2)
<i>Node</i>	10	11	12	13	14	15	16	17	18
<i>Co-ordinates</i>	(5, 20)	(6, 13)	(6, 17)	(6, 19)	(7, 2)	(7, 15)	(8, 7)	(8, 19)	(9, 4)
<i>Node</i>	19	20	21	22	23	24	25	26	
<i>Co-ordinates</i>	(9, 8)	(9, 18)	(10, 3)	(10, 16)	(10, 19)	(11, 3)	(11, 12)	(12, 5)	

Table 3.3: Node Co-ordinates in the Network in Fig. 3.6

by the city block distance.<sup>7</sup>

(The model solution for this exercise is in `graph_c.pl`.)

**Exercise 3.4.** (*Sparsity*) If the adjacency matrix of a network is *sparse*, i.e. most of its entries are zero (Fig. 3.5), it is a good idea to apply a compression scheme for storing it in the database. The following is a simple compression scheme. As each row can be thought of as a concatenation of lists comprising zeros and ones, we shall denote repetitions of the same character *C* by *N-C* where *N* is the number of times the character *C* appears. Thus, for example, [1-0, 2-1, 7-0] will stand for the first row of the matrix in (3.3). Define a predicate `decompress(+C,-A)` for converting a compressed matrix *C* into the corresponding adjacency matrix *A*.<sup>8</sup>

*Hint.* A concise definition may be achieved by adopting the *functional programming style*:

1. Define a predicate for converting terms of the form *N-C* to a list comprising *N* copies of *C*.
2. Define now a predicate by mapping the predicate in (1) followed by applying `flatten/2`.

<sup>7</sup>We shall meet this network in a different context in Sect. 3.4 as the search graph of the maze problem in Fig. 3.10, p. 122.

<sup>8</sup>The query in Exercise 3.3, part (b), may then equivalently be issued by

```
?- c_adj(1,_C), decompress(_C,_A), co_ord(1,_Co), path(_A,_Co).
```

if `c_adj/2` is used in an obvious manner for defining compressed adjacency matrices.

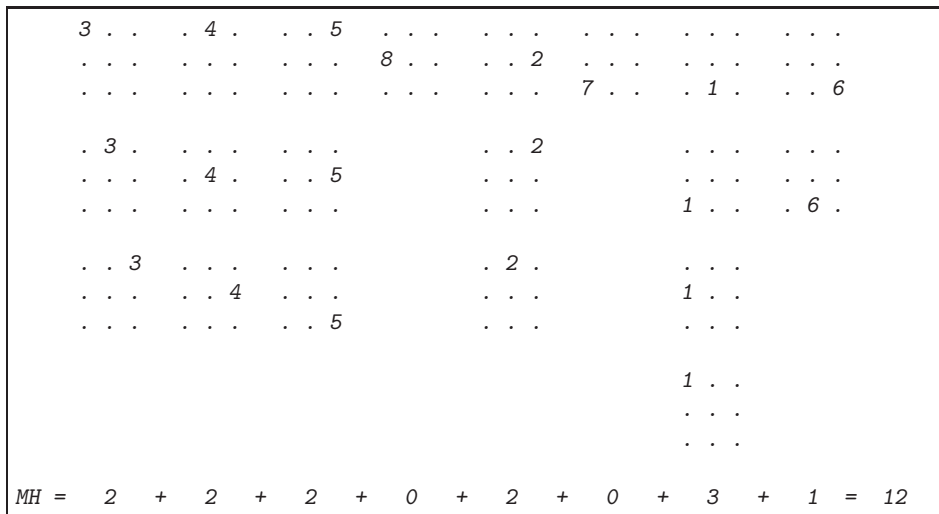


Figure 3.7: Calculating the Manhattan Distance between the tile arrangements in Fig. 2.45

3. Finally, implement decompression by mapping the predicate in (2) to the compressed matrix.

(The solution is in `graph.c.pl`.)



### 3.2 Case Study: The Eight Puzzle Revisited

For some choices of the terminal states for the Eight Puzzle we have not been able to find a solution using blind search (Table 2.1, p. 100). We are going to re-examine this puzzle here by informed search.

#### 3.2.1 The Heuristics

A popular heuristic for the Eight Puzzle is the *Manhattan Distance* (MH). For two tile arrangements, the MH is the minimum total number moves all eight tiles need to be moved *individually* from their initial to their respective final positions. Whereas in the original version of the puzzle prior to moving a tile we had to make space by moving tiles which were ‘in the way’, now in this *relaxed* problem the obstacle tiles are simply ignored. (As before, moves sideways and up and down are allowed only.) For example, the MH between the tile arrangements in Fig. 2.45, p. 99, is 12 as shown in Fig. 3.7. The MH never exceeds the actual distance (i.e. the minimum number of moves needed to convey one configuration to the other) which is 16 here (Fig. 2.46, p. 101). The MH is therefore an admissible heuristic.

The predicate `e_cost(mh,+State1,+State2,-C)`<sup>9</sup> returns the estimated cost between *State1* and *State2* as measured by the MH; for the states in Fig. 2.45 we have, for example,

```
?- e_cost(mh,state(3,4,5,8,0,2,7,1,6),state(1,2,3,8,0,4,7,6,5),C).
C = 12
```

<sup>9</sup>In the first argument we indicate the heuristic employed. (In Exercise 3.5 we will be considering another heuristic too.)

To implement this predicate, we first represent the system's states in matrix form, i.e. by a list comprising three lists.

```
matrix_form(state(T11,T12,T13,T21,T22,T23,T31,T32,T33),
            [[T11,T12,T13],[T21,T22,T23],[T31,T32,T33]]).
```

Given now two matrix representations, *Matrix1* and *Matrix2*, we find the number of steps *D* needed to convey the tile located at  $(i, j)$  in *Matrix1* to its new position in *Matrix2* by applying *mh\_distance/5*, defined by

```
mh(I,J,Matrix1,Matrix2,D) :- ijth(I,J,Matrix1,E),
                             (E \= 0,
                              ijth(K,L,Matrix2,E),
                              D is abs(I - K) + abs(J - L));
                             D = 0), !.10
```

For example, the number of steps in the seventh sequence of tile moves in Fig. 3.7 is verified by

```
?- mh(3,2,[[3,4,5],[8,0,2],[7,1,6]],[[1,2,3],[8,0,4],[7,6,5]],D).
D = 3
```

Finally, as seen in Fig. 3.7, the MH between any two tile arrangements (in matrix notation) is the sum of the number of moves for each individual tile.

```
mh(Matrix1,Matrix2,D) :- mh(1,1,Matrix1,Matrix2,D11),
                          ...
                          mh(3,3,Matrix1,Matrix2,D33),
                          D is D11 + D12 + ... + D33.
```

**Exercise 3.5.** Another heuristic for the eight puzzle is the *number of misplaced tiles* (MP): each tile already in the right position will contribute zero whereas each of the other tiles will contribute unity. Implement this heuristic by *e\_cost(mp,+State1,+State2,-C)*. Example:

```
?- e_cost(mp,state(3,4,5,8,0,2,7,1,6),state(1,2,3,8,0,4,7,6,5),C).
C = 6
```

Thus, this heuristic does not exceed the MH<sup>11</sup> which itself is admissible. Hence MP is admissible. (MP is defined in `eight_puzzle.a.pl`.) ■

### 3.2.2 Prolog Implementation

The Prolog implementation is in the file `eight_puzzle.a.pl`. A sample run is shown in Fig. 3.8, p. 116. For example, case 9 is now solvable while previously it was not viable (Table 2.1, p. 100). Table 3.4 shows the CPU times for the heuristic searches using a 300 MHz machine. (Unsuccessful cases and those with excessive computing times have been omitted.) Comparing Table 3.4 with Table 2.1 shows the dramatic benefit of using

<sup>10</sup>The predicate *ijth(?I,?J,+Matrix,?Entry)* is defined here by

```
ijth(I,J,ListOfRows,E) :- nth1(I,ListOfRows,Row), nth1(J,Row,E).
```

It is used in two modes. First, to get access to the  $(i, j)$ th entry of a matrix, use the mode *ijth(+I,+J,+Matrix,-Entry)*. Then, to identify the position of *Entry* in *Matrix*, use *ijth/4* in the mode *ijth(-I,-J,+Matrix,+Entry)*.

<sup>11</sup>In fact, MP is at most the number of tiles, i.e. 8. Since MH is 12 here, we know without checking further that MP is less than MH.

```

?- consult(eight_puzzle_a).
% asearches compiled into a_ida_ideaeps 0.00 sec, 7,704 bytes
% eight_links compiled into links 0.00 sec, 4,100 bytes
% eight_puzzle_a compiled 0.00 sec, 22,288 bytes
Yes
?- tiles.
Start state for test case number 1:
8 1 2
7 3
6 4 5
-----
...
-----
Start state for test case number 9:
5 6 7
4 8
3 2 1
-----
...
Select test case (a number between 1 and 10)... 9.
Select heuristic (mh/mp)... mh.
Select algorithm (a/ida/ideaeps)... a.
Solution in 30 steps.
Show result in full? (y/n) y.
5 6 7
4 8
3 2 1
-----
5 6 7
4 2 8
3 1
-----
...
-----
1 3
8 2 4
7 6 5
-----
1 2 3
8 4
7 6 5
-----
Yes

```

Figure 3.8: Solving the Eight Puzzle by Heuristic Search



Test Case Number			1	2	3	4	5	6	7	8	9	10
Goal Node at Depth			8	8	10	12	13	16	16	20	30	30
CPU Seconds	mp	a	0.1	0.1	0.0	0.3	0.7	26.8	14.3	-	-	-
		ida	0.1	0.1	0.1	0.5	1.0	4.2	5.1	59.9	-	-
	mh	a	0.0	0.0	0.1	0.1	0.1	0.9	0.7	38.0	42.0	-
		ida	0.1	0.1	0.0	0.1	0.1	0.3	0.8	8.1	2.8	52.9

Table 3.4: CPU Times (in Seconds) for the Eight Puzzle with Heuristic Search

heuristic search. It confirms furthermore that MH is better than MP and that  $IDA^*$  is preferable to the  $A^*$ -Algorithm.

Case 9 becomes viable for the number of misplaced tiles heuristic for  $IDA^*-\epsilon$ . With  $\epsilon = 25$ , we get a solution in 32 steps in 30.4 CPU seconds.

**Exercise 3.6.** (*Other Algorithms*) As precursors to the  $A$ -Algorithm, in many AI books two other algorithms are also discussed: *Hill Climbing* and *Best First Search* (e.g. [34]).

.....Alcatel-Lucent 

[www.alcatel-lucent.com/careers](http://www.alcatel-lucent.com/careers)



What if you could build your future and create the future?

One generation's transformation is the next's status quo. In the near future, people may soon think it's strange that devices ever had to be "plugged in." To obtain that status, there needs to be "The Shift".



Hill Climbing is a modification of Depth First in that the nodes obtained by expanding a parent node will be, prior to them being put to the front of the agenda, sorted in ascending order of their estimated distances to the goal node.<sup>12</sup>

Best First is an extension of the previous idea in that now, prior to choosing the node to be expanded next, *all* open paths in the agenda are sorted in ascending order of their estimated distances to the goal node.<sup>13</sup>

You should implement these two algorithms.

*Notes.*

- (a) Model your implementation of the search algorithms on `asearches.pl`. As in `asearches.pl`, represent the estimated cost of a path by a prefix; no postfix is needed now.
- (b) Model your solution of the Eight Puzzle on `eight_puzzle_a.pl`.
- (c) Run the implementation and interpret the results.

(The model solution will be found in `bsearches.pl` and `eight_puzzle_b.pl`.) ■

### 3.3 Project: Robot Navigation<sup>14</sup>

Develop a Prolog program that can be used to guide a robot in the matrix shown in Fig. 3.9 along a *shortest* route from any cell to any other cell.<sup>15</sup> The robot should be able to move parallel to the walls but not diagonally.

*Notes.*

1. Use the search algorithms' implementations in `asearches.pl`.
2. Use the city block distance as a heuristic  $H$ .
3. There are several possibilities to model the 'cost' of a path. The simplest is to take its length as a measure of cost, i.e.

$$G = \text{path length} \tag{3.4}$$

The length is the sum of the edge costs each of which is in our application unity; we therefore declare

```
edge_cost(_,_,1).
```

Using this measure, the cost of the path found in Fig. 3.9 is 14.

4. Experiments using the cost measure in (3.4) suggest that the problem cannot always be solved by the  $A$ -Algorithm as the agenda may become excessively large. This will happen if there are too many paths of the same length sharing the same endpoints. The cost defined by

$$G = \text{path length} + \delta \times \text{path tortuosity} \tag{3.5}$$

---

<sup>12</sup>The underlying intuitive expectation is here that expanding nodes that are deemed closer to the goal node will lead faster to the goal node.

<sup>13</sup>Best First is therefore a kind of  $A$ -Algorithm with the  $G$  component in (3.1) set to zero.

<sup>14</sup>A simplified version of the problem described in this section served as a coursework problem in the late Prof. Imad Torsun's Prolog lectures in the late 1990s.

<sup>15</sup>The matrix layout (robot floorplan on Fig. 3.9) is taken from [23, p. 83].

```

?- consult(robot).
% rsearches compiled into rsearches 0.00 sec, 7,924 bytes
% floorplan compiled into floorplan 0.05 sec, 9,524 bytes
% robot compiled 0.05 sec, 25,116 bytes
Yes
?- robot.
  1  2  3  4  5  6  7  8  9 10 11 12 13 14
  .  .  .  .  .  .  .  .  .  .  .  .  .  .
+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
1 | | | | | | | | | | | | | | | | | | | | | | | | | | 1
+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
2 | | | | | | | | | | | | | | | | | | | | | | | | | | 2
+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
3 | | | | | | | | | | | | | | | | | | | | | | | | | | 3
+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
4 | | | | | | | | | | | | | | | | | | | | | | | | | | 4
+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
5 | | | | |XXX|XXX|XXX|XXX|XXX|XXX| | | | | | | | | | 5
+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
6 | | | | |XXX|XXX| | | |XXX|XXX| | | | | | | | | | 6
+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
7 | | | | |XXX|XXX| | | |XXX|XXX| | | | | | | | | | 7
+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
8 | | | | | | | | | | | | | | | | | | | | | | | | | | 8
+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
9 | | | | | | | | | | | | | | | | | | | | | | | | | | 9
+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
10 | | | | | | | | | | | | | | | | | | | | | | | | | | 10
+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
11 | | | | | | | | | | | | | | | | | | | | | | | | | | 11
+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
  1  2  3  4  5  6  7  8  9 10 11 12 13 14

Select start cell ... cell(5,11).
Select goal cell ... cell(7,3).
Select algorithm (a/ida/idaeps)... a.

% 842,633 inferences in 5.66 seconds (148875 Lips)
From cell(5, 11) to cell(7, 3) in 14 moves:
  1  2  3  4  5  6  7  8  9 10 11 12 13 14
  .  .  .  .  .  .  .  .  .  .  .  .  .  .
+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
1 | | | | | | | | | | | | | | | | | | | | | | | | | | 1
+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
2 | | | | | | | | | | | | | | | | | | | | | | | | | | 2
+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
3 | | | | | | | | | | | * * * * * | | | | | | | | | | 3
+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
4 | | | | | | | | | | | | | * | | | | | | | | | | | | 4
+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
5 | | | | |XXX|XXX|XXX|XXX|XXX|XXX| * | | | | | | | | | 5
+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
6 | | | | |XXX|XXX| | | |XXX|XXX| * | | | | | | | | | 6
+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
7 | | | | |XXX|XXX| | | |XXX|XXX| * | | | | | | | | | 7
+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
8 | | | | | | | | | | | * * * * * * * * * | | | | | | 8
+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
9 | | | | | | | | | | | * | | | | | | | | | | | | | | 9
+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
10 | | | | | | | | | | | * | | | | | | | | | | | | | | 10
+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
11 | | | | | | | | | | | * | | | | | | | | | | | | | | 11
+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
  1  2  3  4  5  6  7  8  9 10 11 12 13 14

Yes

```

Figure 3.9: Robot Navigation

where

$$\text{path tortuosity} = \text{number of turns}$$

will differentiate between such paths sufficiently enough for excessive growth of the agenda to be avoided. To guarantee that all least cost paths are also shortest paths, choose  $\delta > 0$  small enough such that a shorter path, however tortuous, will always be assigned a smaller cost. Assuming that no path will have more than, say, nine turns,  $\delta = 0.1$  will do. Using this measure, the cost of the path found in Fig. 3.9 is 14.3.<sup>16</sup>

5. Your implementation using (3.4) will always succeed if Iterative Deepening  $A^*$  is used but may run out of memory for the  $A$ -Algorithm.
6. A more ambitious implementation will use (3.5), and this will always succeed, also for the  $A$ -Algorithm. The implementations in `asearches.pl` can cope with the usual cost structure only, i.e. where each edge is assigned a fixed cost. To cater for the more complex cost structure in (3.5), you should devise a modified version of `asearches.pl`. (The model solution uses `rsearches.pl` that is an adaptation of `asearches.pl`.)
7. The predicate defining the floor layout, called `cell/2` in the model implementation, may be defined by facts as follows.

```
cell(1,1). cell(1,2). ... cell(11,11).
```

It would be rather tedious, however, to enter these facts into the database manually and therefore they are *asserted* ([9, p. 80]) by invoking a rule-based equivalent, `position/2`, prior to running the main body of the program. For example, the upper block of cell positions may be defined by

```
position(X,Y) :- between(1,11,X), between(1,4,Y).
```

which then is followed by the *assertion* of the facts defining `cell/2` by `layout/0` as shown below.

```
layout :- retractall(cell(-,-)),
           position(X,Y),
           assert(cell(X,Y)),
           fail.
layout. } failure driven loop ([9, p. 77])
        } catch-all clause
```

This is a simple form of *memoization* (e.g. [19], p. 179 and [28], p. 181), aimed at saving computing time during the search process. In addition, it introduces some flexibility, as the suggested arrangement allows the floor layout to be easily modified if required.

8. The top level module of the model implementation is in `robot.pl`. It uses the modules in `rsearches.pl` (or `asearches.pl`, depending on which cost measure is being employed) and `floorplan.pl`. The latter implements the path's display on the terminal as shown in Fig. 3.9. (A less ambitious solution will display the path by showing its co-ordinates only.)

---

<sup>16</sup>By contrast, the path from `cell(5,11)` to `cell(7,3)` and having turns at `cell(5,8)`, `cell(9,8)`, `cell(9,4)` and `cell(7,4)` has the same length as the one found in Fig. 3.9 but it is more tortuous as it changes directions four times rather than thrice. It will be assigned the cost of 14.4.

### 3.4 Project: The Shortest Route in a Maze

Develop a Prolog program for searching for a shortest path in a maze of a specific kind with the following features.

- The program should search in mazes exemplified in Fig. 3.10 whereby
  - The gates are arranged in groups parallel to each other;
  - Adjacent groups of gates are a unit distance apart;
  - Groups of gates are numbered 1, 2, ... (up to 12 in Fig. 3.10);
  - Group number 1 comprises the IN gate only;
  - The group with the highest number (here: 12) comprises the OUT gate only;
  - Gates are of unit width;
  - The position of the gates relative to the left wall is recorded by a number (1, ..., 20 in Fig. 3.10) and the overall width of the maze is determined by the position of the rightmost gate;
- The program should display on the terminal the maze and the shortest path found.

Furthermore, as seen in Fig. 3.10, the program should also have the following features.

**Maastricht University** *Leading in Learning!*

**Join the best at the Maastricht University School of Business and Economics!**

**Top master's programmes**

- 33<sup>rd</sup> place Financial Times worldwide ranking: MSc International Business
- 1<sup>st</sup> place: MSc International Business
- 1<sup>st</sup> place: MSc Financial Economics
- 2<sup>nd</sup> place: MSc Management of Learning
- 2<sup>nd</sup> place: MSc Economics
- 2<sup>nd</sup> place: MSc Econometrics and Operations Research
- 2<sup>nd</sup> place: MSc Global Supply Chain Management and Change

Sources: Keuzegids Master ranking 2013; Elsevier 'Beste Studies' ranking 2012; Financial Times Global Masters in Management ranking 2012

**Maastricht University is the best specialist university in the Netherlands (Elsevier)**

**Visit us and find out why we are the best!**  
**Master's Open Day: 22 February 2014**

[www.mastersopenday.nl](http://www.mastersopenday.nl)





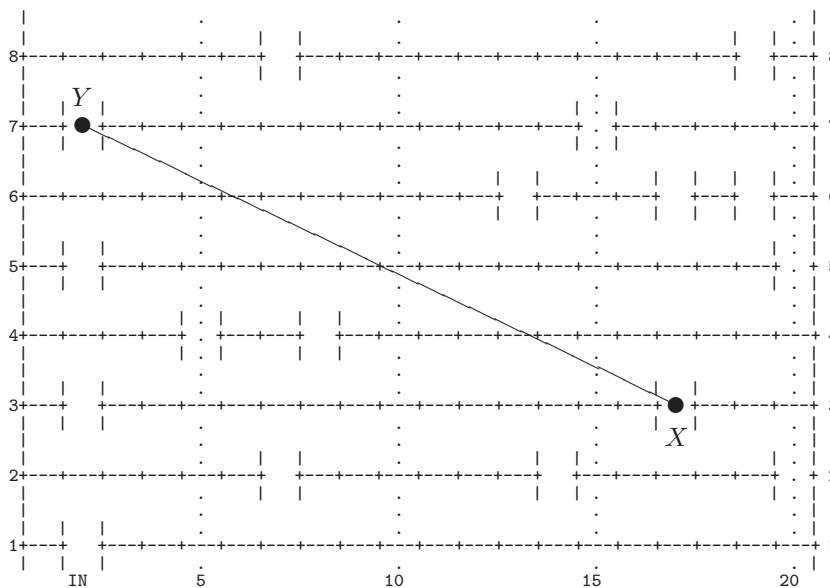


Figure 3.11: Calculating the Euclidean Heuristic  $H_1$

- The user should choose between three evaluation functions (of the form  $F = G + H$ ), whereby the heuristic component,  $H$ , is one of the following: zero (**zero**), the Euclidean distance (**ed**), or, an alternative distance (**alt**) which will be described in Sect. 3.4.1. (All three suggested choices of  $H$  will be seen admissible.)
- The user should choose between three algorithms:  $A^*$ , Iterative Deepening  $A^*$  and Iterative Deepening  $A^* - \epsilon$ .
- The program should return a display of the shortest path found and its length.

### 3.4.1 Suggested Implementation Details

The predicate `gates/2` will be used to specify the structure of a maze. For example,

```
gates(2, [[2], [7,14,20], [2,17], [5,8], [2,20], [13,17,19],
         [2,15], [7,19], [4,8,18], [3,16,19], [3,12], [5]]).
```

specifies the maze shown in Fig. 3.10. The first argument of `gates/2` stands for the ‘test case number’; its second argument takes a list-of-lists defining the structure of the maze in an obvious manner.

#### Heuristics

*The zero heuristic  $H_0$ .* Put simply  $H_0 \equiv 0$ .

*The Euclidean heuristic  $H_1$ .* This is the straight line (‘Euclidean’) distance  $e$  between any two gates. Fig. 3.11 illustrates  $H_1$ : to estimate the distance between two gates  $X$  and  $Y$ , simply use Pythagoras (3.6).

$$H_1(X, Y) = e(X, Y) = \sqrt{(17 - 2)^2 + (3 - 7)^2} = 15.52 \tag{3.6}$$

The alternative heuristic  $H_2$ . If  $X$  and  $Y$  are in adjacent rows then put  $H_2(X, Y) = e(X, Y)$ . Assume now that  $X$  and  $Y$  are at least two rows apart.  $H_2(X, Y)$  is then defined with reference to Fig. 3.12. Take for each row of gates between  $X$  and  $Y$  every gate in that row as an intermediate gate in a two-stage ‘flight’ between  $X$  and  $Y$ . Keep the row fixed and compute the minimum of such ‘flight distances’ — each such minimum ‘flight distance’ is obviously a lower bound on the true maze distance between  $X$  and  $Y$ . The alternative heuristic  $H_2(X, Y)$  is defined as the maximum of all such minimum flight distances, obtained by varying the in-between rows of gates. Equations (3.7)-(3.8) illustrate the computation of  $H_2$ .

$$H_2(X, Y) = \max \left\{ \begin{array}{l} \min \{ e(X, U_1) + e(U_1, Y), \\ e(X, U_2) + e(U_2, Y) \}, \\ \min \{ e(X, V_1) + e(V_1, Y), \\ e(X, V_2) + e(V_2, Y) \}, \\ \min \{ e(X, W_1) + e(W_1, Y), \\ e(X, W_2) + e(W_2, Y), \\ e(X, W_3) + e(W_3, Y) \} \end{array} \right\} \quad (3.7)$$



**> Apply now**

REDEFINE YOUR FUTURE  
**AXA GLOBAL GRADUATE  
PROGRAM 2015**

redefining / standards 

agence cdtg - © Photonstop



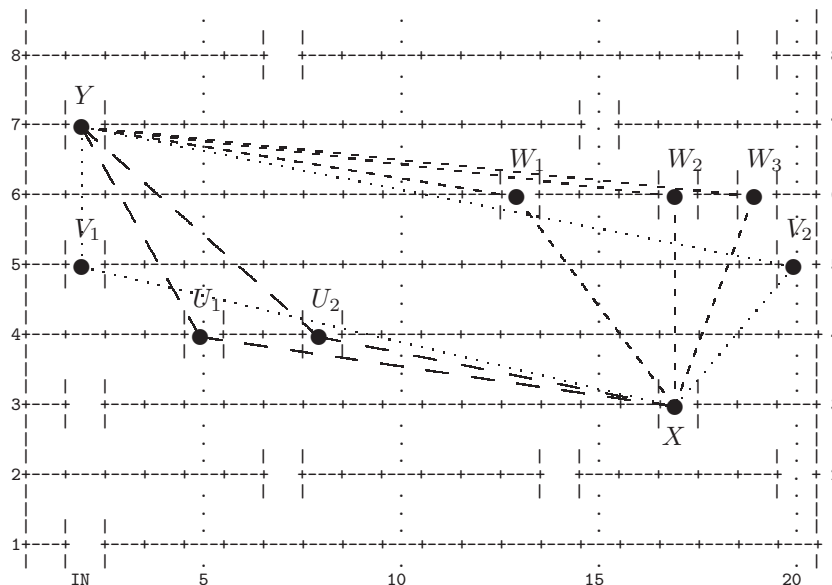


Figure 3.12: Calculating the Alternative Heuristic  $H_2$

$$\begin{aligned}
 H_2(X, Y) = \max \{ & \min \{ 16.28, 15.77 \}, \\
 & \min \{ 17.13, 21.72 \}, \\
 & \min \{ 16.05, 18.03, 20.62 \} \} = 17.13
 \end{aligned}
 \tag{3.8}$$

The result is an admissible heuristic. Equations (3.6) and (3.7)-(3.8) show that  $H_2$  is not worse than the Euclidean heuristic  $H_1$ , i.e.

$$H_1(X, Y) \leq H_2(X, Y) \leq \text{true distance between } X \text{ and } Y$$

$H_2$  will be, however, more expensive to compute than either  $H_0$  or  $H_1$ .

### Manual Implementation

As a first step towards a full implementation, the problem shall be solved for the maze in Fig. 3.10 with the zero heuristic  $H_0$  and without returning a pictorial display of the path found. In this initial phase we won't be making use of *gates/2* directly. Instead, the necessary information about the maze will be represented by a collection of facts defining *edge\_cost/3* thus

```

edge_cost(state(1,2),state(2,7),6).
edge_cost(state(1,2),state(2,14),13).
...

```

(The filename chosen to hold these clauses, *tedious.pl*, reflects the effort involved.) The above definition of *edge\_cost/3* can be derived from the search graph indicated in Fig. 3.13 below. We define *link/2* in terms of *edge\_cost/3* by

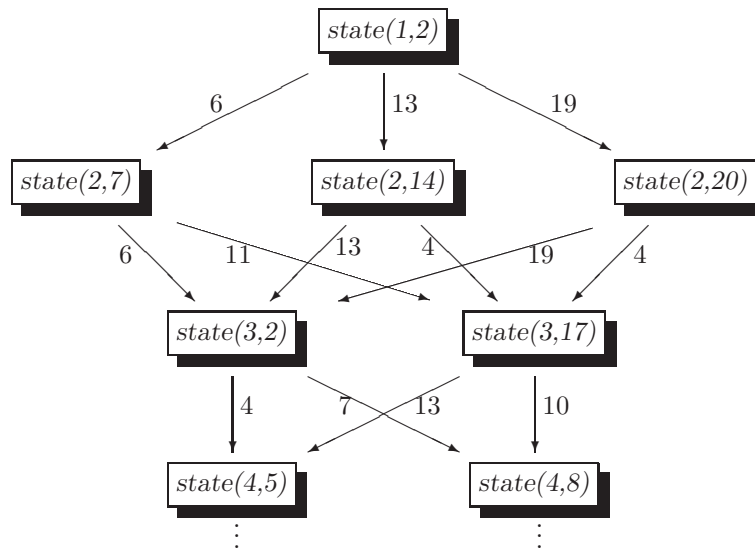


Figure 3.13: Search Graph for the Gates' Position

**BI NORWEGIAN BUSINESS SCHOOL**

EFMD EQUIS ACCREDITED

## Empowering People. Improving Business.

BI Norwegian Business School is one of Europe's largest business schools welcoming more than 20,000 students. Our programmes provide a stimulating and multi-cultural learning environment with an international outlook ultimately providing students with professional skills to meet the increasing needs of businesses.

BI offers four different two-year, full-time Master of Science (MSc) programmes that are taught entirely in English and have been designed to provide professional skills to meet the increasing need of businesses. The MSc programmes provide a stimulating and multi-cultural learning environment to give you the best platform to launch into your career.

- MSc in Business
- MSc in Financial Economics
- MSc in Strategic Marketing Management
- MSc in Leadership and Organisational Psychology

[www.bi.edu/master](http://www.bi.edu/master)



```
link(Node1,Node2) :- edge_cost(Node1,Node2,_).
```

The positions of the terminal gates will be recorded in `tedious.pl` by

```
start_state(state(1,2)). final_state(state(12,5)).
```

Finally, the zero heuristic will be implemented by the definition

```
e_cost(_,_,0).
```

We are now in a position to find interactively the path shown in Fig. 3.10:

```
?- consult(tedious).
% asearches compiled into a_aida_ideaeps 0.00 sec, 7,704 bytes
% tedious compiled 0.00 sec, 15,544 bytes
Yes
?- start_state(_S), final_state(_G), a_search(_S,_G,_PathFound), write_term(_PathFound, []).
[state(1, 2), state(2, 7), state(3, 2), state(4, 5),
 state(5, 2), state(6, 13), state(7, 15), state(8, 7),
 state(9, 4), state(10, 3), state(11, 3), state(12, 5)]
Yes
```

**Exercise 3.7.** Complete the file `tedious.pl` and run the search for the maze in Fig. 3.10 by using the heuristic  $H_0$ . ■

### Full Implementation

The predicates which will be used by the search algorithms in `asearches.pl` should be defined in the top module, `maze.pl`, say. Below you will find some *guidelines* for these and another predicate used to display the result.

A *rule-based* version (in one clause) of `link/2` will define the node connectivity; then, for example, for the maze shown in Fig. 3.10 we get

```
?- consult(maze).
...
?- maze.17
Select test case (a number between 1 and 5)... 2.
Select heuristic (zero/ed/alt)... ed.
Select algorithm (a/ida/ideaeps)... a.
...
?- link(state(3,17),Gate).
Gate = state(4, 5) ;
Gate = state(4, 8) ;
No
```

<sup>17</sup>This predicate, among other things, writes to the database the gates' arrangement chosen by the user. The predicate `gates/1` will be used to hold this information.

```
maze :- (retractall(gates(-));true),
        select_testcase(N),
        assert((gates(AllGates) :- gates(N,AllGates))),
        ...
```

Now you should define `link/2` for extracting the connectivity information from `gates/1`.

The predicate `e_cost(+Heur,+G1,+G2,-Est)` should return in `Est` the estimated distance of the gates `G1` and `G2`. Equations (3.6) and (3.7)-(3.8) are confirmed for example by

```
?- e_cost(ed,state(3,17),state(7,2),Est).
Est = 15.5242
?- e_cost(alt,state(3,17),state(7,2),Est).
Est = 17.1327
```

The pictorial display of the maze and the path found is accomplished by the predicate `show_picture(+Pic)`, defined in the module `maze_disp.pl`, with `Pic` specifying the maze and the path. To produce for example the display in Fig. 3.10, `Pic` will be unified with the list of pairs

```
[(5, [5]), (3, [3,12]), (3, [3,16,19]), ..., (2, [2])]
```

(`Pic` allows to identify for each row the gate through which the path passes and the position of all the gates in that row.)

**Exercise 3.8.** Complete the implementation of the maze search problem as described above. ■

**Exercise 3.9.** The model implementation uses the straight line distance to derive heuristics. Modify the implementation by basing the heuristics on the city block distance and observe and interpret changes in the CPU time. ■

**Exercise 3.10.** The idea of the alternative heuristic function  $H_2$  can be refined. For example,  $H_3(X, Y)$  may be defined for gates  $X$  and  $Y$  at least three rows apart by maximizing the minimum flight distances between  $X$  and  $Y$  with two intermediate gates. Put  $H_3(X, Y) = H_2(X, Y)$  if  $X$  and  $Y$  are less than three rows apart.  $H_n$  ( $n \geq 4$ ) may be defined in an analogous manner.  $H_n$  is a better heuristic than  $H_{n-1}$ , i.e.  $H_n \geq H_{n-1}$  but it will be more expensive to compute. Experiment with these heuristics to find out whether the computational benefit in the search process outweighs the increased computing time for the heuristics themselves. ■

**Exercise 3.11.** The search graph of the maze problem is *acyclic*, i.e. no node can be visited more than once (e.g. Fig. 3.13). Path checking is therefore not required in this case. Disable path checking in `asearches.pl` and confirm that the resulting implementation uses less CPU time. ■

## 3.5 Project: Moving a Knight

Write a Prolog program which, given two positions on the chessboard, will find a shortest sequence of moves a knight needs between these two positions.<sup>18</sup> Your program will behave as indicated in Fig. 3.14. You should experiment with the suggested heuristics to find out how long the search takes with each.

The model solution is in `knight.pl` and it uses `asearches.pl`.

<sup>18</sup>The present search problem originates from [10].



```

?- consult(knight).
% asearches compiled into a_ida_ideaeps 0.00 sec, 7,704 bytes
% knight compiled 0.05 sec, 19,104 bytes
Yes
?- jumps.
Select heuristic (min/mh/ed/co)... ed.
Select algorithm (a/ida)... ida.
Select initial position of knight ([a-h][1-8])... a8.
Select final position of knight ([a-h][1-8])... h1.
cost limit/CPU time: 1/399.3
cost limit/CPU time: 4.42719/399.35
cost limit/CPU time: 4.49285/399.35
cost limit/CPU time: 4.52982/399.35
cost limit/CPU time: 4.60768/399.35
cost limit/CPU time: 4.61245/399.41
cost limit/CPU time: 4.63246/399.46
cost limit/CPU time: 4.84391/399.52
cost limit/CPU time: 4.89443/399.63
cost limit/CPU time: 5.2249/399.74
cost limit/CPU time: 5.23607/399.9
cost limit/CPU time: 5.26491/400.06
cost limit/CPU time: 5.40588/400.23
cost limit/CPU time: 5.40832/400.39
cost limit/CPU time: 5.41421/400.61
cost limit/CPU time: 5.44721/400.94
cost limit/CPU time: 5.72029/401.33
cost limit/CPU time: 5.78885/401.77
cost limit/CPU time: 5.84708/402.26
cost limit/CPU time: 5.86356/402.76
cost limit/CPU time: 5.89737/403.31
cost limit/CPU time: 6/403.91
% 474,024 inferences in 4.66 seconds (101722 Lips)
Solution in 6 steps:
a8 b6 a4 b2 d1 f2 h1

+---+---+---+---+---+---+---+---+
8 | X | | | | | | | | |
+---+---+---+---+---+---+---+---+
7 | | | | | | | | | |
+---+---+---+---+---+---+---+---+
6 | | X | | | | | | | |
+---+---+---+---+---+---+---+---+
5 | | | | | | | | | |
+---+---+---+---+---+---+---+---+
4 | X | | | | | | | | |
+---+---+---+---+---+---+---+---+
3 | | | | | | | | | |
+---+---+---+---+---+---+---+---+
2 | | X | | | | X | | | |
+---+---+---+---+---+---+---+---+
1 | | | | X | | | | X | |
+---+---+---+---+---+---+---+---+
  a  b  c  d  e  f  g  h
Yes

```

Figure 3.14: Sample Session: Moving a Knight

### Suggested Heuristics

Let the letters annotating the board's columns be replaced by  $1, \dots, 8$  and refer to the knight's position by a pair  $P = (x, y)$  with co-ordinates  $x, y \in \{1, \dots, 8\}$ . Define two heuristics  $H_1$  and  $H_2$  by

$$H_q(P, P') = \begin{cases} \frac{d_1(P, P')}{3}, & \text{when } q = 1 \\ \frac{d_2(P, P')}{\sqrt{5}}, & \text{when } q = 2 \end{cases} \quad (3.9)$$

where  $d_1$  and  $d_2$  denote respectively the city block distance (also called 'Manhattan distance') and the Euclidean distance:

$$d_q((x, y), (x', y')) = \begin{cases} |x - x'| + |y - y'|, & \text{when } q = 1 \\ \sqrt{(x - x')^2 + (y - y')^2}, & \text{when } q = 2 \end{cases}$$

$H_1$  and  $H_2$  are referred to in Fig. 3.14 by *mh* and *ed*, respectively.

## Need help with your dissertation?

Get in-depth feedback & advice from experts in your topic area. Find out what you can do to improve the quality of your dissertation!

Get Help Now



Go to [www.helpmyassignment.co.uk](http://www.helpmyassignment.co.uk) for more info



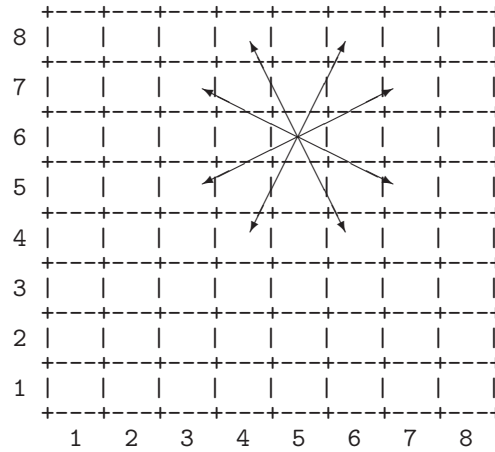


Figure 3.15: The Knight Moves One Step

An interesting property of these heuristics is that none dominates the other.<sup>19</sup>

*Admissibility.* We show that both  $H_1$  and  $H_2$  are admissible. For pairs of positions one step apart, it is

$$d_q(P, P') = \begin{cases} 3, & \text{when } q = 1 \\ \sqrt{5}, & \text{when } q = 2 \end{cases}$$

(This is illustrated in Fig. 3.15 for  $P = (4, 6)$ .) In general, if the sequence of positions

$$P = P_0, P_1, \dots, P_n = P'$$

takes the knight from  $P$  to  $P'$  in the *minimum* number of moves  $n$ , say, then, by the *Triangle Inequality* for  $d_q$  it is

$$\begin{aligned} d_q(P, P') &= d_q(P_0, P_n) \\ &\leq d_q(P_0, P_1) + \dots + d_q(P_{n-1}, P_n) = \begin{cases} 3n, & \text{when } q = 1 \\ \sqrt{5}n, & \text{when } q = 2 \end{cases} \end{aligned} \quad (3.10)$$

From (3.10) we have by the definition of  $H_q$  in (3.9) that

$$H_q(P, P') \leq n$$

*Generalization.* We note in passing that for any  $q \geq 1$ ,  $H_q$ , defined by

$$H_q(P, P') = \frac{d_q(P, P')}{(1 + 2^q)^{1/q}}$$

with

$$d_q((x, y), (x', y')) = (|x - x'|^q + |y - y'|^q)^{1/q}$$

<sup>19</sup>By this we mean that there are positions  $P, P', Q$  and  $Q'$  such  $H_1(P, P') < H_2(P, P')$  and  $H_1(Q, Q') > H_2(Q, Q')$ . This holds for example for  $P = (4, 3)$ ,  $P' = (7, 4)$ ,  $Q = (4, 3)$  and  $Q' = (6, 1)$ .

is an admissible heuristic.<sup>20</sup>

*Combined heuristic.* This we define by

$$H_{co}(P, P') = \max\{H_1(P, P'), H_2(P, P')\}$$

It is of course also admissible and it is a genuine improvement on both  $H_1$  and  $H_2$  since, as we have seen earlier, none dominates the other.

*A Non-Admissible Heuristic.* Define  $H_{min}$  by

$$H_{min}((x, y), (x', y')) = \min\{|x - x'|, |y - y'|\}$$

This is not admissible since  $H_{min}((7, 2), (1, 8)) = 6$  but  $(7, 2) \rightarrow (5, 3) \rightarrow (3, 4) \rightarrow (2, 6) \rightarrow (1, 8)$  is a sequence of 4 moves from  $(7, 2)$  to  $(1, 8)$ .  $IDA^*$  will indeed find this non-optimal sequence of moves if it is used with  $H_{min}$ .

---

<sup>20</sup>The reasoning is as before with the following addenda. It is

$$d_q(P, P') = \|P - P'\|_q$$

with the  $q$ -norm  $\|\cdot\|_q$  defined by

$$\|(x, y)\|_q = (|x|^q + |y|^q)^{1/q}$$

The Triangle Inequality for  $d_q$  follows from the *Minkowski Inequality* for the  $q$ -norm

$$\|P + P'\|_q \leq \|P\|_q + \|P'\|_q$$

See, e.g. [31].

# Chapter 4

## Text Processing

Whereas the problems considered thus far were taken from Artificial Intelligence, we are going now to apply Prolog to problems in text processing.

The present chapter is in three parts.

First, the Prolog implementation is described of a tool for removing from a file sections of text situated between marker strings. (The tool is therefore a primitive static *program slicer*; [32] and [12].) This tool then is used in a practical context for removing sample solutions from the L<sup>A</sup>T<sub>E</sub>X source code of a solved exam script. It is also shown in this context how SWI-Prolog code can be embedded into a LINUX shell script.

The second part addresses the question of how Prolog can be used to generate L<sup>A</sup>T<sub>E</sub>X code for drawing *parametric curves*. Some new features of Prolog will thereby also be introduced.

The final part comprises a sequence of solved Prolog exercises, implementing a tool for drawing *families* of parametric curves in L<sup>A</sup>T<sub>E</sub>X. The exercises are of increasing complexity and finally describe how SWI-Prolog can interact with LINUX through a shell script.

### 4.1 Text Removal

#### 4.1.1 Practical Context

I use L<sup>A</sup>T<sub>E</sub>X on LINUX for preparing examination papers. This is done in the following steps.

1. Create a L<sup>A</sup>T<sub>E</sub>X source file in a text editor.
2. Translate the L<sup>A</sup>T<sub>E</sub>X file into a a DVI file.
3. Translate the DVI file into a PDF file.
4. View the PDF file.

These steps are performed for `exam.tex` by running the LINUX commands in Fig. 4.1.<sup>1</sup> Upon execution of the last line in Fig. 4.1, a new window will pop up and the exam paper may be viewed.

External examiners require examination papers *with model answers*. I create therefore a PDF file with model solutions in the first instance where answers are appended to each subquestion. The answers are placed between

---

<sup>1</sup>`bash-3.1$` is the system prompt in Fig. 4.1.

```
bash-3.1$ latex exam.tex
bash-3.1$ dvipdf exam.dvi
bash-3.1$ kpdf exam.pdf
```

Figure 4.1: Processing the File `exam.tex`

some *marker strings* enabling me eventually to *locate and remove* all text between them when creating the final  $\text{\LaTeX}$  source leading to the printed PDF for students. It is this *text removal* process which is automated by the Prolog implementation to be discussed here.

#### 4.1.2 Specification

Write a predicate `sieve(+Infile, -Outfile, +Startmarker, +Endmarker)` of arity 4 for removing all text in the file named in *Infile* in between all occurrences of lines starting with text in *Startmarker* and those starting with text in *Endmarker*. The result should be saved in the file named in *Outfile*. *Outfile* is without marker lines. If *Outfile* already exists, its old version should be overwritten, if it does not exist, it should be newly created. The file shown in Fig. 4.2 is an example of *Infile* with the marker phrases ‘water st’ and

**Brain power**

By 2020, wind could provide one-tenth of our planet's electricity needs. Already today, SKF's innovative know-how is crucial to running a large proportion of the world's wind turbines.

Up to 25 % of the generating costs relate to maintenance. These can be reduced dramatically thanks to our systems for on-line condition monitoring and automatic lubrication. We help make it more economical to create cleaner, cheaper energy out of thin air.

By sharing our experience, expertise, and creativity, industries can boost performance beyond expectations. Therefore we need the best employees who can meet this challenge!

The Power of Knowledge Engineering

Plug into The Power of Knowledge Engineering.  
Visit us at [www.skf.com/knowledge](http://www.skf.com/knowledge)

**SKF**



'water e', say. (The file comprises a random collection of geographical names.) After the Prolog query

```

birningham
new york
lake district

las vegas
grand canaria

london
water starts } ← Line starting with Startmarker
pacific ocean
loch ness

north sea
water ends } ← Line starting with Endmarker
kalahari desert
st andreas fault
north pole
water starts } ← Line starting with Startmarker
mediterranean sea
lake balaton
lake konstanz
river thames
river danube
water ends } ← Line starting with Endmarker
britain
europe

```

Figure 4.2: The File `with_waters`

```
?- sieve('with\_waters', 'without\_waters', 'water st', 'water e').2
Yes
```

the file `without_waters` will have been created. This is shown in Fig. 4.3.

### 4.1.3 Implementation

#### Definition of Predicates

The main predicate `sieve/4` is defined in terms of `sieve/2`, both are shown in (P-4.1).

<sup>2</sup>Notice that the sequence of *two* characters '`\_`' represents the underscore. Likewise, '`\.`' will have to be typed for the dot in a filename or marker string.



```

bermingham
new york
lake district

las vegas
grand canaria

london
kalahari desert
st andreas fault
north pole
britain
europe

```

Figure 4.3: The File without\_waters

*Prolog Code P-4.1: Definition of sieve/4 and sieve/2*

```

1 sieve(File_In, File_Out, Start_String, End_String) :-
2     see(File_In),
3     tell(File_Out),
4     told,
5     append(File_Out),
6     switch_off,
7     sieve(Start_String, End_String),
8     told,
9     seen, !.

10 sieve(Start_String, End_String) :-
11     atom_chars(Start_String, Start_List),
12     atom_chars(End_String, End_List),
13     get_line(Line),
14     ((append(Start_List,_,Line), switch_on); true),
15     (Line = [end_of_file];
16     atom_codes(A,Line),
17     ((switch(off), write(A)); true),
18     ((append(End_List,_,Line), switch_off); true),
19     sieve(Start_String, End_String)).

```

The predicates *get\_line/1* (and its auxiliary *get\_line/2*), *switch\_off/1* and *switch\_on/1* are defined in (P-4.2).

## Prolog Code P-4.2: Auxiliaries for (P-4.1)

```

1 :- dynamic(switch/1).
2 switch_off :- retractall(switch(_)),
3               assert(switch(off)).
4 switch_on  :- retractall(switch(_)),
5               assert(switch(on)).
6 get_line(List) :- get_line([], List).
7 get_line(Acc, List) :- get_char(Next),
8                       ((Next = '\n', reverse([Next|Acc], List));
9                       (Next = end_of_file, List = [Next]));
10                      get_line([Next|Acc], List)).

```

For the SWI-Prolog built-ins *atom\_chars/2* and *atom\_codes/2*, the reader is referred respectively to pages 126 and 19 of [9].

“I studied English for 16 years but...  
...I finally learned to speak it in just six lessons”  
Jane, Chinese architect

ENGLISH OUT THERE

Click to hear me talking before and after my unique course download



Noteworthy are three more built-in predicates used here: the standard Prolog predicates *see/1*, *seen/0* (respectively for directing the input stream to a file and redirecting it) and *get\_char/1* for reading a character; the example below illustrates their use by reading the first three characters of the file *with\_waters* in Fig. 4.2.

```
?- see(with_waters), get_char(First), get_char(Sec), get_char(Third), seen.
First = b
Sec = i
Third = r
```

Yes

### Details of Implementation

- The predicate *get\_line/1* in (P-4.2) is defined in terms of *get\_line/2* by the accumulator technique. It reads into its argument the next line from the input stream. Example:

```
?- set_prolog_flag(toplevel_print_options, [max_depth(20)]).
Yes
?- see(with_waters), get_line(First), get_line(Sec), seen.
First = [b, i, r, m, i, n, g, h, a, m,
]
Sec = [n, e, w, , y, o, r, k,
]
Yes
```

The following observations apply.

1. It is seen from the above query that a line read by *get\_line/1* is represented as a list of the characters it is composed of.
  2. By definition the last character of each line in a file is the new line character ‘\n’. That explains the line break seen in the above query.
  3. Finally (not demonstrated here), each file ends with the end-of-file marker ‘end\_of\_file’. The one-entry list [end\_of\_file] is deemed to be the last line of every file by the definition in (P-4.2).
- The switches *switch\_off/0* and *switch\_on/0* are used, writing respectively *switch(off)* and *switch(on)* in the Prolog database, respectively for removal and retention of lines from the input file.
  - The main predicates are *sieve/4* and *sieve/2* in (P-4.1), the latter defined by recursion and called by the former.

*sieve/4*: this is the top level predicate.

1. Line 2 opens the input file.
2. The goals in lines 3-4 in (P-4.1) make sure that the earlier version of the output file (if there is such a file) is deleted.
3. In line 5, the new output stream is opened via *append/1*<sup>3</sup>.
4. In line 6, the switch is set to the position (‘off’), anticipating that initially lines will be retained.

<sup>3</sup>Not to be confused with the predicate *append/3*!

5. In line 7, *sieve/2* is invoked and processing is carried out.
6. Lines 8 and 9 close respectively output and input.

*sieve/2*: this is called from *sieve/4*.

1. Lines 14 and 18 contain the most interesting feature of this predicate: *append/3* is used in them for *pattern matching*. For example, the goal

```
append(Start_List,_,Line)
```

succeeds if the initial segment of the list *Line* is *Start\_List*.

2. *atom\_chars/2* is used in *sieve/2* to disassemble the start and end markers into lists in preparation for pattern matching.
3. Notice that the built-in predicate *atom\_codes/2* can be used in *two roles* as the interactive session below demonstrates.

```
?- atom_codes(A,[b, i, r, m, i, n, g, h, a, m]).
A = birmingham
Yes
?- atom_codes(birmingham, L).
L = [98, 105, 114, 109, 105, 110, 103, 104, 97, 109]
Yes
```

In line 16 of (P-4.1), *atom\_codes/2* is used in its first role, i.e. to convert a list of characters to an atom. This atom is the current line, it is written to the output file.

4. Recursion is stopped in *sieve/2* (and control is returned to line 8 of *sieve/4*) when the end-of-file marker is read (line 15).

#### 4.1.4 Using a LINUX Shell Script

##### Specification

Imbed the Prolog implementation from Sect. 4.1.3 into a LINUX shell script for providing the same functionality as the predicate *sieve/4* does. The application obtained thereby will run without explicitly having to use the SWI-Prolog system. The intended behaviour of the script is illustrated in Fig. 4.4. The dialogue shown in Fig. 4.4 has the same effect as the Prolog session envisaged in Sect. 4.1.2.

[22] is an accessible introduction to LINUX and the beginnings of shell scripting.

##### Implementation

*Plan*

```

bash-3.1$ ./sieve with\_waters without\_waters water\_st water\_e
% /home/acsenki/scripts/sieve.pl compiled 0.00 sec, 4,284 bytes
Input file : 'with\_waters'
Output file: 'without\_waters'
Text removal between the phrases 'water st' and 'water e'
bash-3.1$ cat without\_waters
bermingham
new york
lake district

las vegas
grand canaria

london
kalahari desert
st andreas fault
north pole
britain
europe

```

Figure 4.4: Running the Shell Script `sieve`


What do you want to do?

No matter what you want out of your future career, an employer with a broad range of operations in a load of countries will always be the ticket. Working within the Volvo Group means more than 100,000 friends and colleagues in more than 185 countries all over the world. We offer graduates great career opportunities – check out the Career section at our web site [www.volvogroup.com](http://www.volvogroup.com). We look forward to getting to know you!

**VOLVO**  
 AB Volvo (publ)  
[www.volvogroup.com](http://www.volvogroup.com)

VOLVO TRUCKS | RENAULT TRUCKS | MACK TRUCKS | VOLVO BUSES | VOLVO CONSTRUCTION EQUIPMENT | VOLVO PENTA | VOLVO AERO | VOLVO IT  
 VOLVO FINANCIAL SERVICES | VOLVO 3P | VOLVO POWERTRAIN | VOLVO PARTS | VOLVO TECHNOLOGY | VOLVO LOGISTICS | BUSINESS AREA ASIA

The shell script should

1. Receive four arguments from the user (two filenames and two pattern strings),
2. Write them to a temporary file `temp`,
3. Invoke SWI-Prolog in the batch mode, which then
  - Should open the temporary file `temp`,
  - Should read the strings from `temp`,
  - Should call `sieve/4` to perform text removal,
  - Should close `temp`
4. Close the Prolog system,
5. Report on the actions performed,
6. Delete `temp`.

#### *Shell Script and Additional Prolog Predicates*

The LINUX shell script `sieve` in (S-4.1) is an implementation of the plan.

```


LINUX Shell Script S-4.1: sieve



```

1  #!/bin/bash
2  if [ $# -ne 4 ]; then
3      echo "Error: supply four arguments"
4  else
5      if [ -e $1 ]; then
6          echo $1 > temp
7          echo $2 >> temp
8          echo $3 >> temp
9          echo $4 >> temp
10 #
11     pl -f sieve.pl -g go -t halt
12 #
13     echo "Input file : '$1'"
14     echo "Output file: '$2'"
15     echo "Text removal between the phrases '$3' and '$4'"
16 #
17     rm temp
18 else
19     echo "Error: file '$1' does not exist"
20 fi
21 fi

```


```

In line 11 of (S-4.1), the Prolog source `sieve.pl` is invoked as a command line argument [33, Sect. 2.3]. `sieve.pl` comprises (P-4.1), (P-4.2) from Sect. 4.1.3 and the code in (P-4.3).

**Prolog Code P-4.3: Definition of `go/0` and `get_string/1`**

```

1 go :- see(temp),
2     get_string(File_In),
3     get_string(File_Out),
4     get_string(Start_String),
5     get_string(End_String),
6     sieve(File_In, File_Out, Start_String, End_String),
7     seen.
8
9 %
10 % auxiliary predicate get_string/1 ...
11 %
12
13 get_string(String) :- get_line(List),
                       append(ShortList, ['\n'],List),
                       atom_chars(String, ShortList).

```

In `go/0` from `sieve.pl` the existence of a file named `temp` is assumed, comprising four lines, the two file names (input and output files) and the two marker patterns, forming one line each. The top level predicate is now `go/0` which then uses `sieve/4`.

*Running the Script*

The script `sieve` makes (and eventually deletes) a temporary file `temp`, holding the four strings read by the predicate `go/0`. The script invokes the Prolog source `sieve.pl`, effecting a result as specified in Sect. 4.1.2. Some additional features are also demonstrated in the LINUX command window Fig. 4.5.

```

bash-3.1$ chmod -x sieve
bash-3.1$ ls -l sieve
-rw--w----+ 1 acsenki 2042 426 Sep 2 16:11 sieve
bash-3.1$ ./sieve with\_waters without\_waters water\ st water\ e
bash: ./sieve: Permission denied
bash-3.1$ chmod +x sieve
bash-3.1$ ./sieve with\_waters without\_waters water\ st
Error: supply four arguments
bash-3.1$ ./sieve with\_waters without\_waters water\ st water\ e
Input file : 'with_waters'
Output file: 'without_waters'
Text removal between the phrases 'water st' and 'water e'
bash-3.1$ ls temp
ls: temp: No such file or directory

```

Figure 4.5: Another Run of the Shell Script `sieve`*Comments on Fig. 4.5.*

1. The first three commands illustrate what happens if initially `sieve` is not executable.
2. The fourth command makes `sieve` executable.
3. The fifth command illustrates the script's response if less than four arguments are supplied.



4. The next command shows the normal mode of operation. The response has to be read in conjunction with (S-4.1). The output file created is `without_waters`; it is of course identical to that in Fig. 4.3.
5. The last command confirms that the temporary file `temp` has been removed.

### 4.1.5 Application: Removing Model Solutions

`part_sln.tex` (shown in Fig. 4.6) is a file forming part of a collection of  $\text{\LaTeX}$  source files to be assembled to a single  $\text{\LaTeX}$  source. Text between the user-defined  $\text{\LaTeX}$  commands `\solstart` and `\solend` forms part of a

```

...
\definecolor{hellgrau}{gray}{0.85}
\newcommand{\solstart}{\begin{center}\textbf{- - - - -}}
  \fcolorbox{black}{hellgrau}{Start Solution}- - - - -}\end{center}}
\newcommand{\solend}{\begin{center}\textbf{- - - - -}}
  \fcolorbox{black}{hellgrau}{End Solution}- - - - -}\end{center}}
...
\begin{itemize}
\item
First question.
\item
Second question.
\end{itemize}
\solstart
\begin{itemize}
\item
Answer to first question.
\item
Answer to second question.
\end{itemize}
\solend
Further questions.
...

```

Figure 4.6: The File `part_sln.tex`

model solution of exam questions, not to be shown to students in the final version. Fig. 4.7 shows the structure of the printed version of the exam script *with* solutions.

The task is to use the shell script `sieve` for producing the file `part.tex` from `part_sln.tex`; the latter is

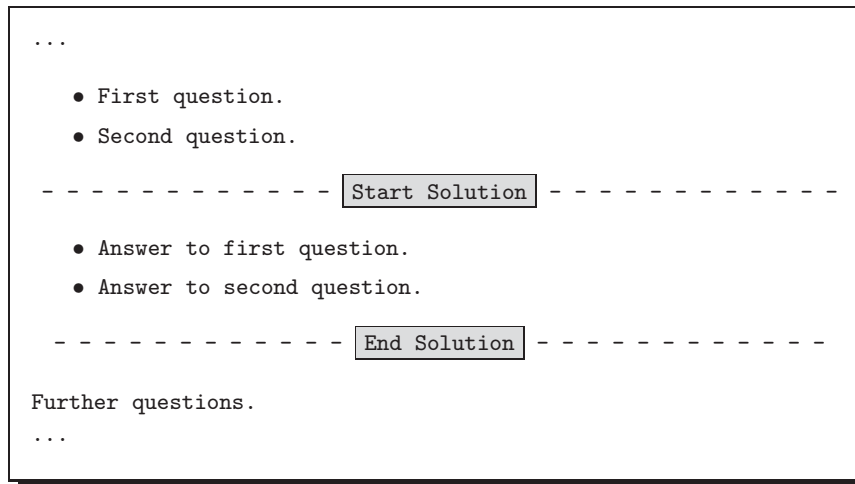


Figure 4.7: Structure of the Printed Exam Script with Solutions

**gaiteye**  
*Challenge the way we run*

**EXPERIENCE THE POWER OF  
FULL ENGAGEMENT...**

.....

**RUN FASTER.  
RUN LONGER..  
RUN EASIER...**

**READ MORE & PRE-ORDER TODAY  
WWW.GAITEYE.COM**

shown in Fig. 4.8. In `part.tex`, all lines between `\solstart` and `\solend` have been removed, including the marker lines themselves.

```

...
\definecolor{hellgrau}{gray}{0.85}
\newcommand{\solstart}{\begin{center}\textbf{- - - - -}}
  \fcolorbox{black}{hellgrau}{Start Solution}- - - - -\end{center}}
\newcommand{\solend}{\begin{center}\textbf{- - - - -}}
  \fcolorbox{black}{hellgrau}{End Solution}- - - - -\end{center}}
...
\begin{itemize}
\item
First question.
\item
Second question.
\end{itemize}
Further questions.
...

```

Figure 4.8: The File `part.tex`

It is seen in Fig. 4.8 in particular that the text between the marker phrases (`\solstart` and `\solend`) is removed only if they are the first phrase of their respective lines. (This is why the command definitions in Fig. 4.8 are still there.)

```

bash-3.1$ ./sieve part\_sln\.tex part\.tex \

```

Figure 4.9: Running the Shell Script `sieve`

The task was achieved by running the shell script as shown in Fig. 4.9. Fig. 4.9 illustrates how string arguments containing the backslash character or the dot are used when running the shell script.

## 4.2 Text Generation and Drawing with L<sup>A</sup>T<sub>E</sub>X

### 4.2.1 Cycloids

Cycloids are a class of plain curves, well known from the Calculus of Variations (see e.g. the early classic [13, p. 26] or [26, Ch. 22, p. 844]). A cycloid is described by a point  $P$  attached to a disc rolling on a straight line (the *base line*) (Fig. 4.10). The following notation will be used.

- $r$  is the radius of the disc,
- $a$  is the distance of  $P = (x, y)$  from the disc's centre  $C$ ,
- $\phi$  is the angle of rotation of the disc, measured in radians, clockwise positive.

The disc rests initially on the co-ordinate origin, therefore,  $C = (0, r)$  and  $P = (0, r - a)$  for  $\phi = 0$ ; this is the disc on the left in Fig. 4.10. If  $P$  is outside the disc ( $a > r$ ) the curve generated is a *prolate* cycloid (Fig. 4.11); if it is inside ( $a < r$ ) a *curtate* cycloid is obtained (Fig. 4.12); and, if it is on the perimeter of the disc ( $a = r$ ) a *common* cycloid (Fig. 4.13) is obtained. (For cycloids and other plane curves a good reference is [11, p. 165].) The co-ordinates of a point on the cycloid are given by

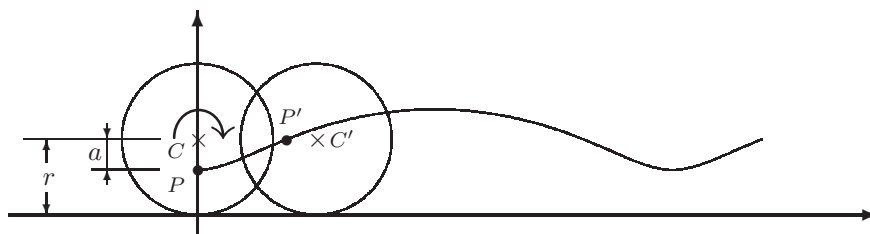


Figure 4.10: Drawing a Cycloid ( $\phi = \pi/2$ )

$$x = r\phi - a \sin \phi, \tag{4.1}$$

$$y = r - a \cos \phi. \tag{4.2}$$

The disc on the right in Fig. 4.10 is obtained by rotating the initial disc clockwise by  $\phi = \pi/2$ . According to (4.1)-(4.2),  $P$ 's new position is  $P' = (r\phi - a \sin \phi, r - a \cos \phi) = (r\pi/2 - a, r)$ , whereas  $C$  obviously moves to  $C' = (r\phi, r) = (r\pi/2, r)$ .

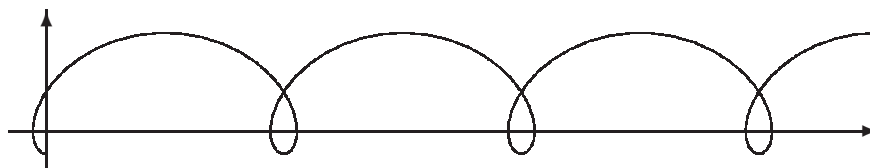


Figure 4.11: Prolate Cycloid Drawn with `\writecurve` from Fig. 4.14 ( $r = 5, a = 8, 3.5$  revs)

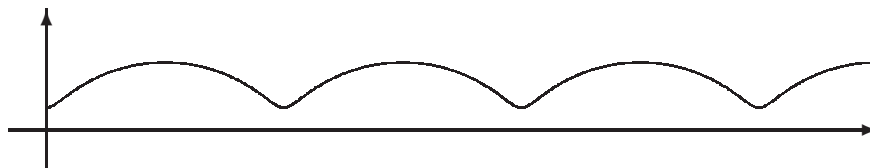


Figure 4.12: Curtate Cycloid Drawn with `\writecurve` similar to Fig. 4.14 ( $r = 5$ ,  $a = 3$ , 3.5 revs)

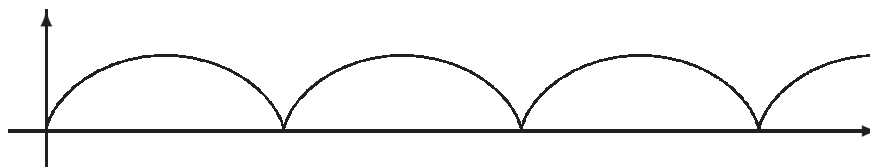
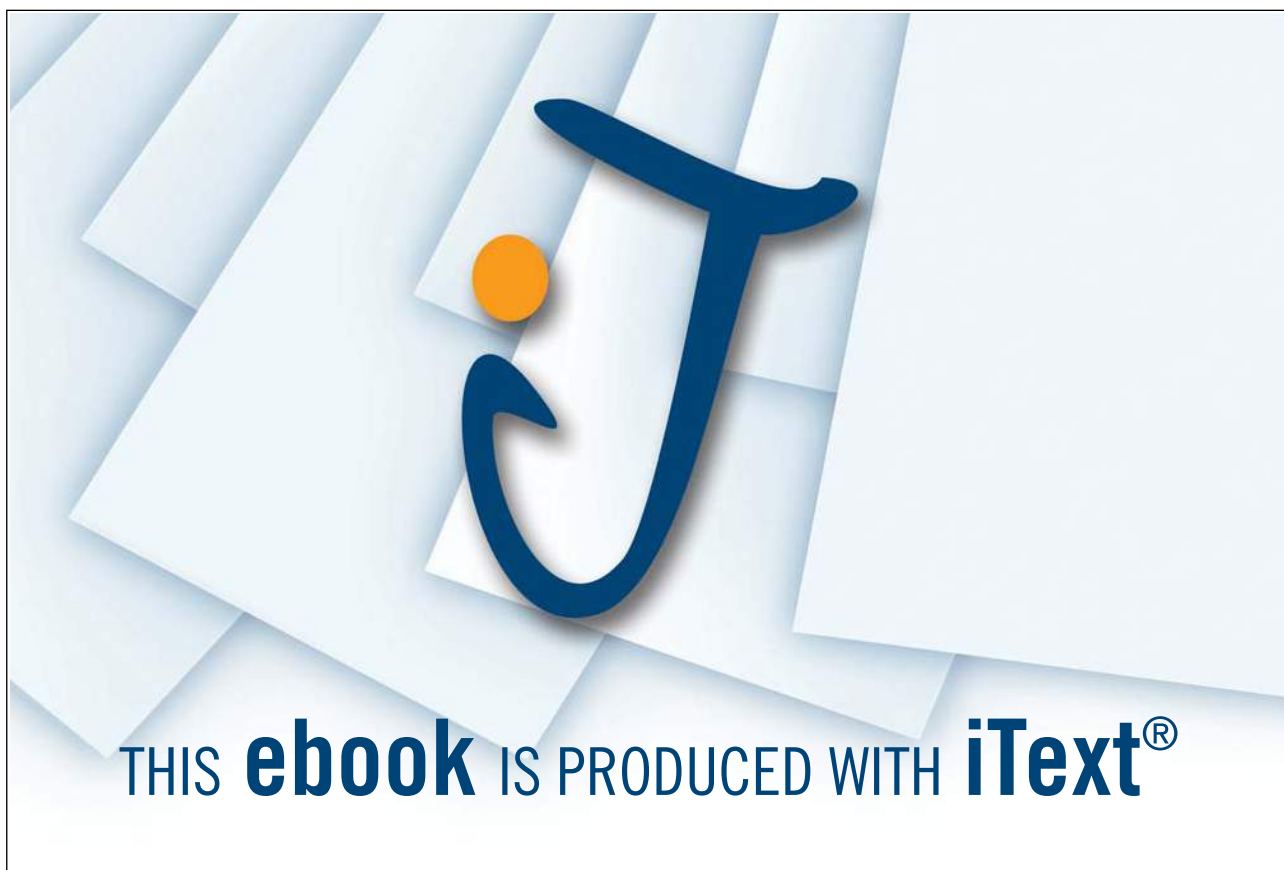


Figure 4.13: Common Cycloid Drawn with `\writecurve` similar to Fig. 4.14 ( $r = 5$ ,  $a = 5$ , 3.5 revs)

#### 4.2.2 Task

Define a Prolog predicate which will generate a  $\text{\LaTeX}$  command for drawing a cycloid of a given description.



The only tool available is the  $\text{\LaTeX}$  package `epic` (e.g. [14]).

The package `epic` provides the command `\drawline` for connecting a sequence of points by a straight line segment. The syntax of this command is

$$\backslash\text{drawline}[stretch](x_1, y_1)(x_2, y_2)\dots(x_n, y_n)$$

where *stretch* is an optional parameter (not used here) and  $(x_1, y_1)(x_2, y_2)\dots(x_n, y_n)$  is the sequence of coordinates of the points to be connected. The task is to define a Prolog predicate `define_command/4` for displaying on the terminal text which is essentially the  $\text{\LaTeX}$  command sought. This is illustrated in Fig. 4.14. The text so obtained is then pasted (after possibly some minor modifications) into the desired location in the

```
?- define_command(5, 8, 3.5, 100).
\newcommand{\writecurve}{\drawline(0,-3)(-0.645588,-2.80733)
(-1.20712,-2.23862)(-1.60458,-1.32124)(-1.76588,-0.099392)(-1.63027,1.36808)
...
(101.754,11.3212)(104.35,12.2386)(107.111,12.8073)(109.956,13.0)}
Yes
```

Figure 4.14: Generating the  $\text{\LaTeX}$  Command `\writecurve` with `define_command/4`

$\text{\LaTeX}$  source file. The curve thus drawn will comprise a sequence of straight line segments, an approximation to the specified cycloid, looking like as a smooth curve if the subdivision of the parameter interval is fine enough. Fig. 4.11, for example, was drawn by applying the  $\text{\LaTeX}$  code (L-4.1). (The  $\text{\LaTeX}$  command `\writecurve`, as generated by Prolog in Fig. 4.14, is used in line 9 of (L-4.1).)

***$\text{\LaTeX}$  Code L-4.1: Drawing Fig. 4.11***

```
1 \begin{figure}[h]
2 \begin{center}
3 \setlength{\unitlength}{1mm}
4 \begin{picture}(118,16)(0,0)
5 \thicklines
6 \put(5,-5){\vector(0,1){21}}
7 \put(0,0){\vector(1,0){115}}
8 \thinlines
9 \put(5,5){\makebox(0,0){\writecurve}}
10 \end{picture}
11 \end{center}
12 \caption{Prolate Cycloid Drawn with \texttt{\writecurve} from
13 Fig.~\ref{textprocessing:cycloids:generatecommand}
14 ($r=5$, $a=8$, $3.5$ revs)}
15 \label{textprocessing:cycloids:fig:prolate}
16 \end{figure}
```

### 4.2.3 Solution

The Prolog predicates for generating the  $\text{\LaTeX}$  command `\writecurve` are shown in (P-4.4).

**Prolog Code P-4.4: Prolog Code Generating `\writecurve`**

```

1 cyc(R, A, Alpha, Pair) :- Pi is 3.1415926,
2                           Rad is Alpha * Pi / 180,
3                           S is sin(Rad),
4                           C is cos(Rad),
5                           X is R * Rad - A * S,
6                           Y is R - A * C,
7                           concat_atom(['(',X,',',',Y,')'], Pair).
8
9 mesh(Revs, NInt, List) :- mesh(Revs, NInt, NInt, List, []), !.
10
11 mesh(_, _, 0, [0|Acc], Acc).
12 mesh(Revs, NInt, NumInt, List, Acc) :-
13   H is NumInt * (Revs * 360 / NInt),
14   NewNumInt is NumInt - 1,
15   mesh(Revs, NInt, NewNumInt, List, [H|Acc]).
16
17 pairs(R, A, Revs, NInt, Pairs) :- mesh(Revs, NInt, Mesh),
18   maplist(cyc(R,A), Mesh, Pairs).
19
20 define_command(R, A, Revs, NInt) :-
21   pairs(R, A, Revs, NInt, Pairs),
22   concat_atom(['\newcommand{\writecurve}{\drawline'|Pairs], Atom),
23   concat_atom([Atom,']'], C),
24   write(C).

```

Comments on, and Exemplification of (P-4.4).

- ① Let  $r = 10$ ,  $a = 4$  and  $C = (0, 10)$ . A counterclockwise rotation by  $\alpha = 90^\circ$  (& associated roll of the disc to the right) moves the point  $P = (0, 6)$  to  $P' = (11.708, 10.0)$ .

```

?- cyc(10, 4, 0, Pair).
Pair = '(0,6)'
Yes
- cyc(10, 4, 90, Pair).
Pair = '(11.708,10.0)'
Yes

```

`cyc/3` is essentially an implementation of (4.1)-(4.2) with the proviso that rotations are measured in degrees. The output of `cyc/3` is an atom.

- ② Let us assume that we want to plot the path of  $P$  between the two positions from ①, involving a quarter turn clockwise. A crude approximation will take snapshots corresponding to the positions  $0^\circ$ ,  $15^\circ$ ,  $30^\circ$ ,  $45^\circ$ ,  $60^\circ$ ,  $75^\circ$  and  $90^\circ$ . The number of intervals involved is therefore 6 (each of length  $15^\circ$ ). The 7 gridpoints are generated as a list by `mesh/3` thus

```

?- mesh(0.25, 6, List).
List = [0, 15, 30, 45, 60, 75, 90]
Yes

```



- ③ A sequence of points on the path of  $P$  is generated by `pairs/5`. For example, the 7 pairs of co-ordinates of  $P$  in ② are obtained by

```
?- pairs(10, 4, 0.25, 6, Pairs).
Pairs = ['(0,6)', '(1.58272,6.1363)', '(3.23599,6.5359)', '(5.02555,7.17157)',
        '(7.00787,8.0)', '(9.22627,8.96472)', '(11.708,10.0)']
```

Yes

`pairs/5` uses `mesh/3` as an auxiliary. Furthermore, `cyc/5` is used in *partial application* in the second goal in the definition of `pairs/5` in the first argument of `maplist/3`. The output of `pairs/5` is a list of atoms. They represent the co-ordinates of the points which will form the vertices of the approximating polygon. `\drawline` from `epic` will be used to connect them.

- ④ `define_command/4` essentially concatenates the list entries from ② thus

```
?- define_command(10, 4, 0.25, 6).
\newcommand{\writecurve}{\drawline(0,6)(1.58272,6.1363)(3.23599,6.5359)(5.02555,7.17157)
(7.00787,8.0)(9.22627,8.96472)(11.708,10.0)}
```

Yes



www.sylvania.com

We do not reinvent  
the wheel we reinvent  
light.

Fascinating lighting offers an infinite spectrum of possibilities: Innovative technologies and new markets provide both opportunities and challenges. An environment in which your expertise is in high demand. Enjoy the supportive working atmosphere within our global group and benefit from international career paths. Implement sustainable ideas in close cooperation with other specialists and contribute to influencing our future. Come and join us in reinventing light every day.

Light is OSRAM

OSRAM  
SYLVANIA




- ⑤ Numbers whose modulus is very small or very large are displayed by default in Prolog in the scientific number format (the ‘exponential notation’). **If applicable**, change such numbers to be displayed in the floating point format using the ‘non-exponential notation’. For example,  $1/888888$  will be displayed as  $1.125e-06$ . Change this to  $0.000001125$  in the  $\text{\LaTeX}$  file.<sup>4</sup> (Notice that this point does not apply to the output generated in ④.)
- ⑥ Now the  $\text{\LaTeX}$  command `\writecurve` is ready to be used inside a figure and it will draw the desired cycloid. Fig. 4.15 was drawn with the `\writecurve`  $\text{\LaTeX}$  command from ④; the code for Fig. 4.15 is not shown here as it is very similar to that shown in (L-4.1).

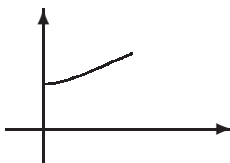


Figure 4.15: ‘Quarter’ Cycloid Drawn with `\writecurve` ( $r = 10$ ,  $a = 4$ ,  $1/4$  revs)

### 4.3 Exercises

**Exercise 4.1.** The predicate `sieve/4` was defined in Sect. 4.1 for *removing* text situated between some specified pairs of markers. Define now a predicate `retain/4` for *retaining* text between some specified pairs of markers. (Such a predicate could be used, for example, for extracting all figures from a  $\text{\LaTeX}$  document.) Use your Prolog implementation in a shell script for solving the same task. ■

**Exercise 4.2.** The two circles shown in Fig. 4.10 were drawn with the user-defined  $\text{\LaTeX}$  command `\defcirc`. The definition of `\defcirc` was generated interactively by running the predicate `circ_command/4` as shown in Fig. 4.16. (L-4.2) shows a partial view of the  $\text{\LaTeX}$  `picture` environment defining Fig. 4.10: lines

```
?- circ_command(10, 0, 0, 100).
\newcommand{\defcirc}{\drawline(10,0)(9.98027,0.627905)
(9.92115,1.25333)(9.82287,1.87381)(9.68583,2.4869)(9.51057,3.09017)
...
(9.82287,-1.87381)(9.92115,-1.25333)(9.98027,-0.627906)(10.0,-1.0718e-06)}
Yes
```

Figure 4.16: Generating the  $\text{\LaTeX}$  Command `\defcirc` with `circ_command/4`

9 and 11 illustrate the use of `\defcirc`.

<sup>4</sup>The alternative is using `sformat/3` (formatted write) in (P-4.4) for displaying numbers in non-exponential notation; see Exercise 4.3.

**LaTeX Code L-4.2:** Partial view of the LaTeX code for Fig. 4.10

```

1 \begin{figure}[h]
2 \begin{center}
3 \setlength{\unitlength}{1mm}
4 \begin{picture}(118,25)(0,0)
5 \thicklines
6 \put(25,-5){\vector(0,1){30}}
7 \put(0,-2){\vector(1,0){115}}
8 \thinlines
9 \put(25,8){\makebox(0,0){\defcirc}}
10 ...
11 \put(40.707963,8){\makebox(0,0){\defcirc}}
12 ...
13 \end{picture}
14 \end{center}
15 \caption{Drawing a Cycloid}\label{textprocessing:fig:definingcycloid}
16 \end{figure}

```

Define the Prolog predicate *circ\_command(+Radius, +CentreX, +CentreY, +NInt)*<sup>5</sup> for displaying on the terminal LaTeX code defining `\defcirc`.

As before, assume that only *basic* LaTeX and the *epic* package are available.<sup>6</sup> ■

**Exercise 4.3.** You will have defined in Exercise 4.2 a Prolog predicate *circ\_command/4* the output of which may have to be put through the manual processing step described in ⑤ of Sect. 4.2.3. This exercise is about writing an *improved* implementation of *circ\_command/4*, called *imp\_circ\_command/4*, that will obviate this since its output will contain pairs of numbers in non-exponential notation only.

The ‘old’ version of the predicate may be used to define a command for a circle of radius 10 with centre (0, 10) by approximating the circle with a regular 20 sided polygon (Fig. 4.17). Both entries of the sixteenth

```

?- circ_command(10, 0, 10, 20).
\newcommand{\defcirc}{\drawline(10,10)(9.51057,13.0902)
(8.09017,15.8779)(5.87785,18.0902)(3.09017,19.5106)(2.67949e-07,20.0)
(-3.09017,19.5106)(-5.87785,18.0902)(-8.09017,15.8779)(-9.51057,13.0902)
(-10.0,10.0)(-9.51057,6.90983)(-8.09017,4.12215)(-5.87785,1.90983)
(-3.09017,0.489435)(-8.03847e-07,3.19744e-14)(3.09017,0.489435)(5.87785,1.90983)
(8.09017,4.12215)(9.51056,6.90983)(10.0,10.0)}
Yes

```

Figure 4.17: Generating the LaTeX Command `\defcirc` with *circ\_command/4*

pair in Fig. 4.17 are in the exponential notation, something LaTeX won’t accept. The modified version produces essentially the same output with all the numbers in the floating point notation (Fig. 4.18).

You should define *imp\_circ\_command/4* by using the SWI-Prolog built-in predicate *sformat/3*.

*Hint.*

The predicate *sformat/3* is there for producing formatted output returned as a string. Use the ‘f’ format (for floating point, non-exponential) in the second argument of *sformat/3*. For further information, see [6, p. 493]

<sup>5</sup>*NInt* denotes the number of intervals used when discretising a full revolution.

<sup>6</sup>In basic LaTeX `\circle` is used to draw circles. It allows, however, to draw circles up to a certain size only.

```
?- imp_circ_command(10, 0, 10, 20).
newcommand{\defcirc}{\drawline(10.000000,10.000000)
(9.5105652,13.0901699)(8.0901700,15.8778524)(5.8778527,18.0901698)
(3.0901701,19.5105651)(0.0000003,20.0000000)(-3.0901696,19.5105653)
(-5.8778522,18.0901702)(-8.0901697,15.8778529)(-9.5105650,13.0901704)
(-10.0000000,10.0000005)(-9.5105653,6.9098306)(-8.0901703,4.1221480)
(-5.8778531,1.9098305)(-3.0901707,0.4894351)(-0.0000008,0.0000000)
(3.0901691,0.4894346)(5.8778518,1.9098295)(8.0901694,4.1221467)
(9.5105648,6.9098291)(10.0000000,9.9999989)}
Yes
```

Figure 4.18: Generating the L<sup>A</sup>T<sub>E</sub>X Command `\defcirc` with `imp_circ_command/4`

and [33].

**Exercise 4.4.** We are now in a position to address the generation of L<sup>A</sup>T<sub>E</sub>X code for *any* parametric two-dimensional curve. The aim is to define a predicate

$$\text{gen\_command2}(+CName, +Fun, +Lower, +Upper, +NInt, +Pars) \quad (4.3)$$

The arguments and the intended working of `gen_command2/6` are best explained with reference to an example.



360°  
thinking.

**Deloitte.**

Discover the truth at [www.deloitte.ca/careers](http://www.deloitte.ca/careers)

© Deloitte & Touche LLP and affiliated entities.



The curve we are going to use is the improved circle *imp\_circ/5* from (P-A.11), p. 193 (solution of Exercise 4.3).

The L<sup>A</sup>T<sub>E</sub>X command for drawing a polygonal approximation with four sides to the lower half of a circular arc with radius 10, centre (0, 10) should be generated thus

```
?- gen_command2('\halfcirc', imp_circ, 180, 360, 4, [10,0,10]).
\newcommand{\halfcirc}{\drawline(-10.0000000,10.0000005)(-7.0710683,2.9289327)(-0.0000008,0.0000000)
(7.0710671,2.9289315)(10.0000000,9.9999989)}
```

Once this command definition is in the L<sup>A</sup>T<sub>E</sub>X code, `\halfcirc` is ready to be used in a figure. (The output may then look like the polygon in Fig. 4.19.) The arguments in (4.3) are easily matched to their respective values

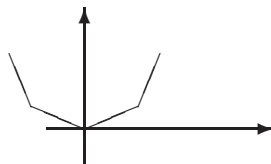


Figure 4.19: Polygon Drawn with `\halfcirc`

in the query. On the other hand, *imp\_circ(+R, +X, +Y, +Alpha, -Pair)*, the predicate from (P-A.11), has

1. Three fixed (input) parameters: radius *R*, and the two co-ordinates of the centre *X* and *Y*;
2. One argument: angle of rotation *Alpha*, measured counterclockwise positive from the circle's rightmost point;
3. One output: *Pair*, returned as a string.

The following is taking place in the query above.

- The command name *CName* in (4.3) is unified with the string '`\halfcirc`';
- The predicate name *Fun* is unified with '`imp_circ`';
- The domain of the argument *Alpha* is the interval  $[Lower, Upper] = [180, 360]$ . It is subdivided into *NInt* (= 4) intervals of equal length. The function values (pairs) are calculated internally for all interval endpoints, i.e. the 5 values of *Alpha*,  $[180, 225, 270, 315, 360]$ ;
- The argument *Pars* (list of parameters) is unified with  $[10, 0, 10]$ , amounting to the unifications  $R = 10$ ,  $X = 0$ ,  $Y = 10$ ;
- And, finally, after some processing, the command definition is written to the terminal.

---

**Built-in Predicate: *apply(+Pred, +List)***

Uses the entries of *List* as arguments to the predicate *Pred*. Partial application of *Pred* is possible. The examples below refer to a polynomial defined by the predicate *pol/5*,

```
pol(A, B, C, X, Y) :- Y is A + B * X + C * X^2.
```

```
?- pol(4, 3, 2, 10, Y).
Y = 234
Yes
?- apply(pol, [4, 3, 2, 10, Y]).
Y = 234
Yes
?- apply(pol(4, 3), [2, 10, Y]).
Y = 234
Yes
```

*apply/2* is a *higher order* predicate. Use *apply(+Pred, +List)* to invoke *Pred* whose arity is not known at compile time.

---

*Detailed Plan.*

The main point is to recognize the need to be able to pass on a predicate name as an argument. The built-in predicate *apply/2* is used to accomplish that. The implementation described here has a ‘functional flavour’.

1. Write a predicate *gen\_mesh(+Lower, +Upper, +NInt, -Mesh)* for generating a list of meshpoints.

```
?- gen_mesh(180, 360, 4, Mesh).
Mesh = [180, 225, 270, 315, 360]
Yes
```

2. Define a predicate *applic(+Fun, +Pars, +Argument, -Outcome)* for calculating values of a function, defined by a predicate. For example, instead of having

```
?- imp_circ(10, 0, 10, 225, Outcome).
Outcome = '(-7.0710683,2.9289327)'
Yes
```

we may now equivalently do

```
?- applic(imp_circ, [10, 0, 10], 225, Outcome).
Outcome = '(-7.0710683,2.9289327)'
Yes
```

The two queries may deliver the same but the second one will be preferable in our context as it allows the predicate name to be passed on as an *argument*; *applic/4* is therefore a *higher order* predicate. Notice that the order of the arguments supplied to *Fun* is replicated by the entries of the list *Pars* and the arguments *Argument* and *Outcome*.

*Hint.* Use the built-in predicate *apply/2*. (See inset.)

3. Define a predicate *gen\_vals(+Fun, +Lower, +Upper, +NInt, +Pars, -Vals)* for calculating the list of values taken by a given function at equidistant gridpoints. Example:

```
?- gen_vals(imp_circ, 180, 360, 4, [10,0,10], Vals).
Vals = [(-10.0000000,10.0000005)', '(-7.0710683,2.9289327)', '(-0.0000008,0.0000000)',
        '(7.0710671,2.9289315)', '(10.0000000,9.9999989)']
Yes
```


Use here *gen\_mesh/4* and *applic/4* from above. Furthermore, use also the built-in predicate *maplist/3*.

4. Finally define *gen\_command2(+CName, +Fun, +Lower, +Upper, +NInt, +Pars)*; it should behave as exemplified on p. 154.


■

**Exercise 4.5.** The *logarithmic spiral* in Fig. 4.20 was drawn with the L<sup>A</sup>T<sub>E</sub>X command `\spiral` the definition of which was generated with Prolog by using *gen\_command2/6* from Exercise 4.4.

SIMPLY CLEVER

**ŠKODA**  


We will turn your CV into  
an opportunity of a lifetime



Do you like cars? Would you like to be a part of a successful brand?  
 We will appreciate and reward both your enthusiasm and talent.  
 Send us your CV. You will be surprised where it can take you.

Send us your CV on  
[www.employerforlife.com](http://www.employerforlife.com)



```
?- gen_command2('\spiral', log_spiral, 0, 2160, 300, [85, 0, 0]).
\newcommand{\spiral}{\drawline(1.0000000,0.0000000)(1.0030823,0.1267188)(0.9901165,0.2542187)
...
(25.6446869,-6.5844539)(26.5581065,-3.3550864)(27.0651201,-0.0000174)}
Yes
```

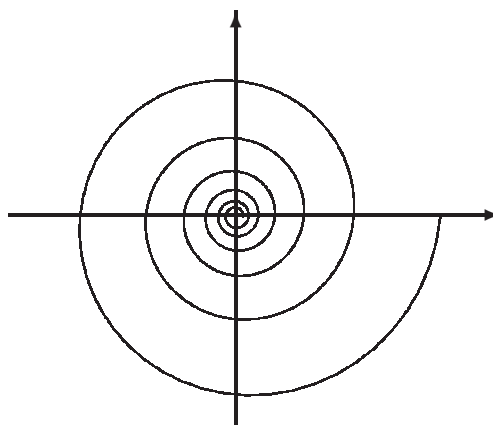


Figure 4.20: Logarithmic Spiral Drawn with `\spiral`

Define the predicate `log_spiral(+Alpha, +CentreX, +CentreY, +RotAngle, -Pair)` and then redraw in  $\text{\LaTeX}$  the spiral on Fig. 4.20.

*Hint.* As is well known (e.g. [2]), a point on the logarithmic spiral with Cartesian co-ordinates  $(r \cos \phi, r \sin \phi)$  is defined by  $r = e^{k\phi}$  with  $k = \cot \alpha$ , where  $(r, \phi)$  are the point's polar co-ordinates and  $\alpha$  is the constant (acute) angle at which the spiral cuts all rays emitted from the origin. ( $\phi$  and  $\alpha$  are both measured in radians in these formulae.) In the above query, we have made  $2160^\circ/360^\circ = 6$  revolutions, subdivided the interval  $[0^\circ, 2160^\circ]$  into 300 intervals of equal length, and, the angle  $\alpha$  measured  $85^\circ$ . (Obviously, the arguments `Alpha` and `RotAngle` in `log_spiral/5` are both measured in degrees.) The pole was taken to be the origin  $(0, 0)$ .

*Note.* An entire section is devoted to spirals in the beautiful book [25]. Questions concerning their self-similarity occupy the authors' attention. ■

**Exercise 4.6.** You are asked to defined the predicate `curves/2` in this exercise. It will simplify and automate the command definitions considered in Exercise 4.4.

Assume that we want to draw possibly *several* parametric curves in  $\text{\LaTeX}$  each of which we can in isolation specify, generate and draw as described in Exercise 4.4. The pasting-in from the terminal of the  $\text{\LaTeX}$  codes generated is cumbersome and error prone as it is a manual step. Therefore, we want to be able to create a file where all the  $\text{\LaTeX}$  code will be deposited, ready to be included into our  $\text{\LaTeX}$  document via `\include`. Furthermore, the curves' *interactive* specifications (via the keyboard) is also best avoided for the same reason; the preferred way of doing this is via some input file.

*Illustrative Example.*

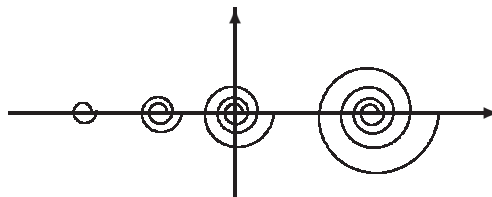


Figure 4.21: Growing Spirals

We want to generate Fig. 4.21 containing four spirals. The L<sup>A</sup>T<sub>E</sub>X command for each of the four spirals can be generated by *gen\_command2/6* from Exercise 4.4. (It is assumed of course that the predicate *log\_spiral/5* from Exercise 4.5 is available.) Once *curves/2* is available, we can solve this task in the following three steps.

- ① Create a file stating the four curves' specifications in terms of *gen\_command2/6*; this has been done here in *spirals* shown in Fig. 4.22. The lines in *spirals* whose first character is % serve as comment lines.

```

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% Spirals specified via gen_command2/6 ...
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% gen_command2('\tinyspiral', log_spiral, 0, 360, 36, [85, 0, 0]). ...
%
gen_command2('\tinyspiral', log_spiral, 0, 360, 36, [85, 0, 0])
%
% gen_command2('\smallspiral', log_spiral, 0, 720, 72, [85, 0, 0]). ...
%
gen_command2('\smallspiral', log_spiral, 0, 720, 72, [85, 0, 0])
%
% gen_command2('\normalspiral', log_spiral, 0, 1080, 108, [85, 0, 0]). ...
%
gen_command2('\normalspiral', log_spiral, 0, 1080, 108, [85, 0, 0])
%
% gen_command2('\largespiral', log_spiral, 0, 1440, 144, [85, 0, 0]). ...
%
gen_command2('\largespiral', log_spiral, 0, 1440, 144, [85, 0, 0])
%

```

Figure 4.22: The File *spirals*

- ② Perform now the following Prolog dialogue.

```

?- consult(draw).
% draw compiled 0.00 sec, 11,432 bytes
Yes
?- curves('spirals', 'spirals.tex').
Yes

```

- ③ The file `spirals.tex` will have been created in step ②. This is shown in Fig. 4.23. Notice that

```

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% Spirals specified via gen_command2/6 ...
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% gen_command2('\tinyspiral', log_spiral, 0, 360, 36, [85, 0, 0]). ...
%
\newcommand{\tinyspiral}{\drawline(1.000000,0.000000)(0.9999608,0.1763201)
...
(1.6805635,-0.2963289)(1.7327464,-0.0000002)}
%
...
...
%
% gen_command2('\largespiral', log_spiral, 0, 1440, 144, [85, 0, 0]). ...
%
\newcommand{\largespiral}{\drawline(1.000000,0.000000)(0.9999608,0.1763201)
...
(8.7429878,-1.5416285)(9.0144653,-0.0000039)}
%

```

Figure 4.23: The File `spirals.tex`

`spirals.tex` is a valid L<sup>A</sup>T<sub>E</sub>X file best included into the L<sup>A</sup>T<sub>E</sub>X source by means of `\include{spirals}`. Lines starting in `spirals` with `%` are copied unchanged by `curves/2` into `spirals.tex`, becoming thereby L<sup>A</sup>T<sub>E</sub>X comment lines. `curves/2` uses `gen_command/6` to generate the commands specifying the curves, here the four spirals.

#### Define the predicate `curves/2`!

*Hint.* Use `apply/2` to call a predicate whose name is known at runtime only. For example, in the query below, after defining the predicate `pol/5` the variable `Pred` is unified with `pol(4, 3, 2, 10, Y)` and then the goal `pol(4, 3, 2, 10, Y)` is satisfied via the call `apply(Pred, [])`.

```

?- consult(user).
|: pol(A, B, C, X, Y) :- Y is A + B * X + C * X * X.
|: 
% user://1 compiled 0.01 sec, 392 bytes
Yes
?- Pred = pol(4, 3, 2, 10, Y), apply(Pred, []).
Pred = pol(4, 3, 2, 10, 234)
Y = 234
Yes

```

**Exercise 4.7.** Embed the predicate `curves/2` from Exercise 4.6 into a LINUX shell script called ‘`curves`’ for creating a L<sup>A</sup>T<sub>E</sub>X file for defining parametric curves. The shell script will use two arguments corresponding to those of `curves/2`. (This solution will have the benefit of the underlying Prolog application remaining hidden

from the user.)

*Illustrative Example.*

Running the script `curves` as shown in Fig. 4.24 will have the same effect as applying the predicate `curves/2` in step ② of Exercise 4.6. The file `spirals.tex` created thereby was copied by means of the last line of Fig. 4.24

```
csenki@linux:~/scripts> ./curves spirals spirals.tex
% /home/csenki/scripts/draw.pl compiled 0.00 sec, 11,800 bytes
Input file : 'spirals'
Output file: 'spirals.tex'
LaTeX source 'spirals.tex' created
csenki@linux:~/scripts> cp spirals.tex ~/texmatter/ventus
```

Figure 4.24: Running the Shell Script `curves`

into a directory where all  $\text{\LaTeX}$  source for the present document is kept. (This copy was made subsequently part of the  $\text{\LaTeX}$  source by writing `\include{spirals}` in the source's top level file.) ■

I joined MITAS because  
I wanted **real responsibility**

The Graduate Programme  
for Engineers and Geoscientists  
[www.discovermitas.com](http://www.discovermitas.com)





**Month 16**  
I was a construction  
supervisor in  
the North Sea  
advising and  
helping foremen  
solve problems

Real work  
International opportunities  
Three work placements





