Appendix A

Examples

A.1 Alpha-Beta Example

To give an idea of how Alpha-Beta works, this appendix illustrates how it traverses the tree of figure 2.4 in detail and concept. For convenience, figure A.1 shows the code of the Alpha-Beta function again. The two items of most interest are how and why cutoffs are performed, and seeing how Alpha-Beta constructs the minimal tree that proves f_{root}. In this example Alpha-Beta will find some cutoffs, but it will traverse more than the minimal tree, since the children of some nodes are not ordered best first. (For example, in max node a the left-most child b is not the highest and in min node k the left-most child is not the lowest.) The values for α, β and g as Alpha-Beta traverses them are shown next to the nodes in figure A.2. Children that are cut off are shown as a small black dot.

The table in figure A.3 gives a step-by-step account of Alpha-Beta's progress. Cutoffs are indicated in the table, and explained in the text below. Since Alpha-Beta works towards finding a max and a min solution tree of equal value, we have shown these in the table as well. Recall from the postcondition of Alpha-Beta that if a node returns a g value that lies within the search window, then both a T^+ and a T^- have been traversed. This case applies to most of the nodes. Only nodes j, p and u return a value outside their window. Their value is determined by a single solution tree. Upper bounds for a node are indicated as f^+, lower bounds as f^-. Max solution trees are shown as T^+, min solution trees as T^-. Figure A.4 gives the final minimal tree that proves the minimax value.

The root is called with $\alpha = -\infty$ and $\beta = +\infty$. Its first child b is expanded with the same parameters. The same holds for node c, d and e. Node e is a leaf, which calls the evaluation function. It returns its minimax value of 41 to its parent. Here, at node d, the values of g and β are updated to 41. Node d performs a cutoff check; since $g > \alpha$ (because $41 > -\infty$) the search continues to the next child, f. Since β was updated in its parent, this node is searched with a window of $\langle -\infty, 41 \rangle$. Node f returns its minimax value.

Parent d returns 5, the minimum of 41 and 5. Parent c updates g and α to 5. Node c

```
function alphabeta(n, α, β) → g;
    if n = leaf then return eval(n);
    else if n = max then
        g := −∞;
        c := firstchild(n);
        while g < β and c ≠ ⊥ do
            g := max(g, alphabeta(c, α, β));
            α := max(α, g);
            c := nextbrother(c);
    else /* n is a min node */
        g := +∞;
        c := firstchild(n);
        while g > α and c ≠ ⊥ do
            g := min(g, alphabeta(c, α, β));
            β := min(β, g);
            c := nextbrother(c);
    return g;
```

Figure A.1: The Alpha-Beta Function

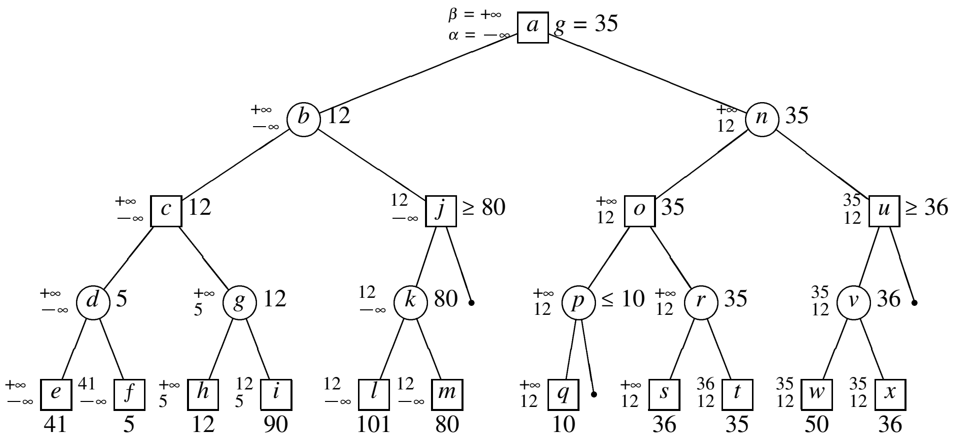

Figure A.2: Example Tree for Alpha-Beta

#	n	α_n	β_n	g_n	cutoff?	f_n^-	T_n^-	f_n^+	T_n^+
1	a	$-\infty$	$+\infty$	$-\infty$		$-\infty$		$+\infty$	
2	b	$-\infty$	$+\infty$	$+\infty$		$-\infty$		$+\infty$	
3	c	$-\infty$	$+\infty$	$-\infty$		$-\infty$		$+\infty$	
4	d	$-\infty$	$+\infty$	$+\infty$		$-\infty$		$+\infty$	
5	e	$-\infty$	$+\infty$	41		41	e	41	e
6	d	$-\infty$	41	41	$41 > -\infty$	$-\infty$		41	d, e
7	f	$-\infty$	41	5		5	f	f	f
8	d	$-\infty$	5	5	$\perp$	5	d, e, f	5	d, f
9	c	5	$+\infty$	5	$5 < +\infty$	5	c, d, e, f	$+\infty$	
10	g	5	$+\infty$	$+\infty$		$-\infty$		$+\infty$	
11	h	5	$+\infty$	12		12	h	12	h
12	g	5	12	12	$12 > 5$	$-\infty$		12	g, h
13	i	5	12	90		90	i	90	i
14	g	5	12	12	$\perp$	12	g, h, i	12	g, h
15	c	12	$+\infty$	12	$\perp$	12	c, g, h, i	12	c, d, f, g, h
16	b	$-\infty$	12	12	$12 > -\infty$	$-\infty$		12	b, c, d, f, g, h
17	j	$-\infty$	12	$-\infty$		$-\infty$		$+\infty$	
18	k	$-\infty$	12	$+\infty$		$-\infty$		$+\infty$	
19	l	$-\infty$	12	101		101	l	101	l
20	k	$-\infty$	12	101	$101 > -\infty$	$-\infty$		101	k, l
21	m	$-\infty$	12	80		80	m	80	m
22	k	$-\infty$	12	80	$\perp$	80	k, l, m	80	k, m
23	j	80	12	80	$80 \not< 12$	80	j, k, l, m	$+\infty$	
24	b	$-\infty$	12	12	$\perp$	12	$b, c, g, h,$ i, j, k, l, m	12	b, c, d, f, g, h
25	a	12	$+\infty$	12	$12 < +\infty$	12	$a, b, c, g,$ h, i, j, k, l, m	$+\infty$	
26	n	12	$+\infty$	$+\infty$		$-\infty$		$+\infty$	
27	o	12	$+\infty$	$-\infty$		$-\infty$		$+\infty$	
28	p	12	$+\infty$	$+\infty$		$-\infty$		$+\infty$	
29	q	12	$+\infty$	10		10	q	10	q
30	p	12	10	10	$10 \not> 12$	$-\infty$		10	p, q
31	o	12	$+\infty$	10	$10 < +\infty$	$-\infty$		$+\infty$	
32	r	12	$+\infty$	$+\infty$		$-\infty$		$+\infty$	
33	s	12	$+\infty$	36		36	s	36	s
34	r	12	36	36	$36 > 12$	$-\infty$		36	r, s
35	t	12	36	35		35	t	35	t
36	r	12	35	35	$\perp$	35	r, s, t	35	r, t
37	o	35	$+\infty$	35	$\perp$	35	o, r, s, t	35	o, p, q, r, t
38	n	12	35	35	$35 > 12$	$-\infty$		35	n, o, p, q, r, t
39	u	12	35	$-\infty$		$-\infty$		$+\infty$	
40	v	12	35	$+\infty$		$-\infty$		$+\infty$	
41	w	12	35	50		50	w	50	w
42	v	12	35	50	$50 > 12$	$-\infty$		50	v, w
43	x	12	35	36		36	x	36	x
44	v	12	35	36	$\perp$	36	v, w, x	36	v, x
45	u	36	35	36	$36 \not< 35$	36	u, v, w, x	$+\infty$	
46	n	12	35	35	$\perp$	35	$n, o, r, s,$ t, u, v, w, x	35	n, o, p, q, r, t
47	a	35	$+\infty$	35	$\perp$	35	$a, n, o, r,$ s, t, u, v, w, x	35	$a, b, c, d, f, g,$ h, n, o, p, q, r, t

Figure A.3: Alpha-Beta Example

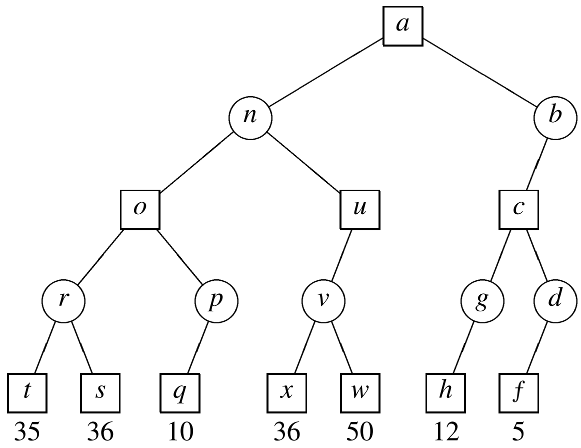

Figure A.4: Minimal Alpha-Beta Tree

continues to search node g, since $5 < +\infty$. The lower bound is used to search node g and its children with the smaller search window of $\langle 5, +\infty \rangle$. Node g returns the minimum of 12 and 90 to c, which returns the maximum of 5 and 12 to b. In node b the search is continued to expand j. Node b is a min node. The g-value 12 is an upper bound, substantiated by a max solution tree containing node c, d, f, g and h. See line 16 in the table. The search window for node j is reduced to $\langle -\infty, 12 \rangle$, indicating that parent b already has an upper bound of 12, so that if in any of the children of b a lower bound ≥ 12 appears, the search can be stopped. Node j expands the sub-tree rooted in its child k, which returns 80. This causes a cutoff of its brother in node j, since $80 \not< 12$. The search of node j, whose value, being a max node, is a lower bound of 80 that can only increase, is no longer useful. The value of parent b is already as low as 12, and since it is a min node, it will never increase. Because b has no other children to lower his value further, it returns his value too. The return value of b is defined by min solution tree $b, c, g, h, i, j, k, l, m$ and max solution tree b, c, d, f, g, h.

At the root the value g is updated to the new lower bound of 12. Consequently, α is updated to 12. Since $12 < +\infty$, no cutoff occurs. Searching the sub-tree below n can still increase the g value of the root. Node n is expanded. Node o, p and q are searched with window $\langle 12, +\infty \rangle$. Node q returns 10. At its parent p this causes a cutoff, since $10 \not> 12$. In contrast to the previous cutoff, the reason for it cannot be found by looking at the value of the direct parent, since o does not have a g value yet other than $-\infty$. This kind of cutoff is known as a *deep* cutoff (in contrast to the previous cutoff, which is said to be shallow). It is called deep because the reason for the cutoff lies further away than the direct parent of the node. The sub-tree below b has returned 12, so the value of a, which is a max node, will be at least 12. Node p, a min node, can at most return a value of 10. The value of node q ensures that none of its brothers can ever influence the value of the root. Node p returns 10.

Next nodes r, s, t, u, v, w and x are searched. The sub-tree below v returns 36. This causes a cutoff (one of the shallow kind) in node u, since $36 \not< 35$. Node u returns 36,

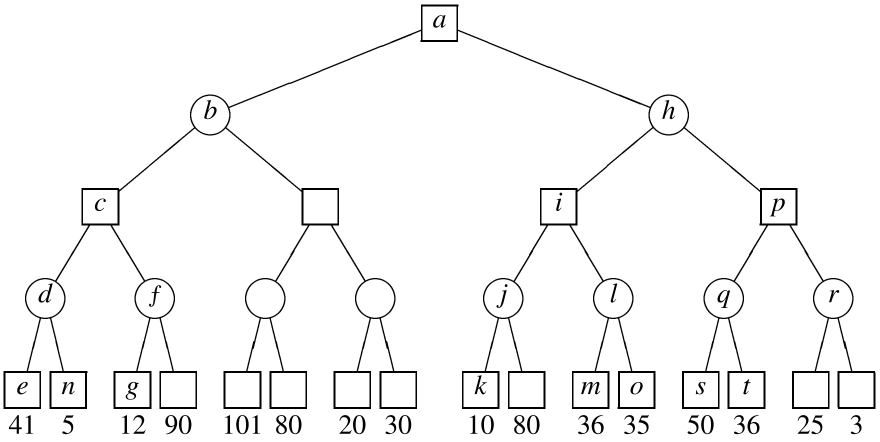

Figure A.5: Example Tree for SSS*

node *n* returns the minimum of 35 and 36, and the root returns the maximum of 12 and 35. The minimax value of the tree has been found, it is 35.

The example illustrates that Alpha-Beta can miss some cutoffs. Although pruning occurs in the example, Alpha-Beta builds a tree that is larger than the minimal tree, as can be seen by comparing figures A.2 and A.4. Alpha-Beta expands 11 leaves, less than the minimax tree of 16, but more than the minimal tree of 7.

A.2 SSS* Example

In the next example we will give an idea of how SSS* works.

Stockman originally used min solution trees to explain his algorithm. Here SSS* is explained using upper bounds and max solution trees, since we think that the algorithm is easier to understand that way and it makes the connection to MT-SSS* easier to see. The two key concepts in the explanation are an *upper bound* on the minimax value, and a *max solution tree*. SSS* starts off with an upper bound of $+\infty$, and works by successively lowering this upper bound until it is equal to the minimax value. The max solution trees are constructed to compute the value of each upper bound.

SSS* works by manipulating a list of nodes, the OPEN list. The nodes have a status associated with them, either *live* (L) or *solved* (S), and a merit, denoted $\hat{h}$. The OPEN list is sorted in descending order, so that the entry with highest merit (the "best" node) is at the front and will be selected for expansion.

We will examine how SSS* searches the tree in figure A.5 for its minimax value (the same tree as in the Alpha-Beta example). Since the tree is almost the same as the one used by Pearl in his explanation of SSS* [99], it may be instructive to compare his min with our max solution tree explanation. In the trees in figures A.6–A.12 the nodes are numbered *a* to *t* in the order in which SSS* visits them first. A number of stages, or passes, can be distinguished in the traversal of this tree. At the end of each pass the OPEN list consists of *solved* nodes only. In appendix B we will see that after

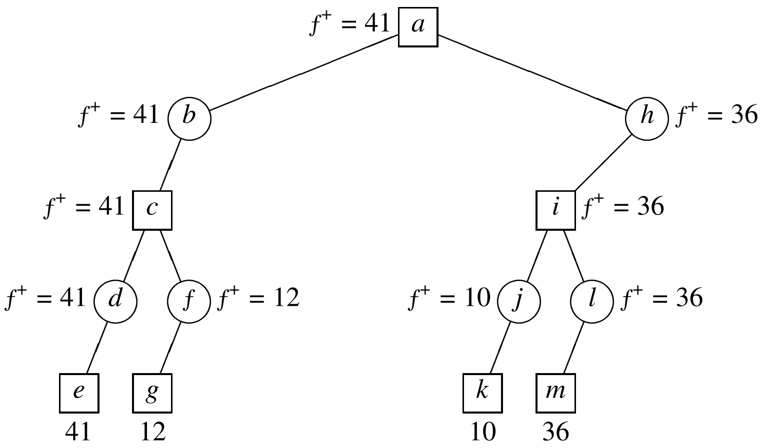

Figure A.6: SSS* Pass 1

Γ operations 1 and 3 SSS* has constructed a min solution tree for the solved node. Appendix B also stresses the fact that at any time the nodes in the OPEN list are the leaves of a (partial) max solution tree rooted at the root, defining an upper bound on it. These two solution trees are shown in the tables. We will start by examining how SSS* traverses the first pass.

First pass: (see figures A.6 and A.7)
In the first pass the left-most max solution tree is expanded to create the first non-trivial upper bound on the minimax value of the root. The code in figure 2.17 places an entry in the OPEN list containing the root node a, with a trivial upper bound: $+\infty$. This entry, containing an internal max node which has as first child a min node, matches Γ case 6, causing a to be replaced by its children (in left-first order). The OPEN list now contains nodes b and h, with a value of $+\infty$. The left-most of these children, b, is at the front of the list. It matches Γ case 5, causing it to be replaced by its left-most child, c. The list now contains nodes c and h. Next, case 6 replaces c by d and f, and case 5 replaces d by e, giving as list nodes e, f and h. Now a new Γ-case comes into play, since node e has no children. Node e does not match case 6, but case 4, causing its state to change from *live* to *solved*, and its $\hat{h}$ value to go from $+\infty$ to 41. Since the list is kept sorted in descending order, the next entry on the list appears at the front, f, the left-most entry with highest $\hat{h}$ value. It matches case 5, g is inserted, which matches case 4, causing it to enter the list with value 12. The list is now $(\langle h, L, \infty\rangle, \langle e, S, 41\rangle, \langle g, S, 12\rangle)$. Next the right subtree below h is expanded in the same way, through a sequence of Γ cases 5 (node i is visited), 6 (j and l enter the list), 5 (k replaces j), 4 (k gets value 10), 5 (l is replaced by m), and 4 (l gets value 36). The OPEN list is now $(\langle e, S, 41\rangle, \langle m, S, 36\rangle, \langle g, S, 12\rangle, \langle k, S, 10\rangle)$.

We have seen so far that at max nodes all children were expanded (case 6), while at min nodes only the first child was added to the OPEN list (case 5). Case 4 evaluated the leaf nodes of the tree. Maintaining the list in sorted order guaranteed that the entry with the highest upper bound was at front. Note that the sub-tree expanded thus far is

#	n	Γ	OPEN list after Γ	f^-_{top}	T^-_{top}	f^+_{root}	T^+_{root}
0			$\big(\langle a, L, +\infty\rangle\big)$			$+\infty$	a
1	a	6	$\big(\langle b, L, +\infty\rangle, \langle h, L, +\infty\rangle\big)$	$-\infty$		$+\infty$	a, b, h
2	b	5	$\big(\langle c, L, +\infty\rangle, \langle h, L, +\infty\rangle\big)$	$-\infty$		$+\infty$	a, b, c, h
3	c	6	$\big(\langle d, L, +\infty\rangle, \langle f, L, +\infty\rangle, \langle h, L, +\infty\rangle\big)$	$-\infty$		$+\infty$	a, b, c, d, f, h
4	d	5	$\big(\langle e, L, +\infty\rangle, \langle f, L, +\infty\rangle, \langle h, L, +\infty\rangle\big)$	$-\infty$		$+\infty$	a, b, c, d, e, f, h
5	e	4	$\big(\langle f, L, +\infty\rangle, \langle h, L, +\infty\rangle, \langle e, S, 41\rangle\big)$	$-\infty$		$+\infty$	a, b, c, d, e, f, h
6	f	5	$\big(\langle g, L, +\infty\rangle, \langle h, L, +\infty\rangle, \langle e, S, 41\rangle\big)$	$-\infty$		$+\infty$	a, b, c, d, e, f, g, h
7	g	4	$\big(\langle h, L, +\infty\rangle, \langle e, S, 41\rangle, \langle g, S, 12\rangle\big)$	$-\infty$		$+\infty$	a, b, c, d, e, f, g, h
8	h	5	$\big(\langle i, L, +\infty\rangle, \langle e, S, 41\rangle, \langle g, S, 12\rangle\big)$	$-\infty$		$+\infty$	$a, b, c, d,$ e, f, g, h, i
9	i	6	$\big(\langle j, L, +\infty\rangle, \langle l, L, +\infty\rangle,$ $\langle e, S, 41\rangle, \langle g, S, 12\rangle\big)$	$-\infty$		$+\infty$	$a, b, c, d, e,$ f, g, h, i, j, l
10	j	5	$\big(\langle k, L, +\infty\rangle, \langle l, L, +\infty\rangle,$ $\langle e, S, 41\rangle, \langle g, S, 12\rangle\big)$	$-\infty$		$+\infty$	$a, b, c, d, e,$ f, g, h, i, j, k, l
11	k	4	$\big(\langle l, L, +\infty\rangle, \langle e, S, 41\rangle,$ $\langle g, S, 12\rangle, \langle k, S, 10\rangle\big)$	$-\infty$		$+\infty$	$a, b, c, d, e,$ f, g, h, i, j, k, l
12	l	5	$\big(\langle m, L, +\infty\rangle, \langle e, S, 41\rangle,$ $\langle g, S, 12\rangle, \langle k, S, 10\rangle\big)$	$-\infty$		$+\infty$	$a, b, c, d, e,$ f, g, h, i, j, k, l, m
13	m	4	$\big(\langle e, S, 41\rangle, \langle m, S, 36\rangle,$ $\langle g, S, 12\rangle, \langle k, S, 10\rangle\big)$	41	e	41	$a, b, c, d, e,$ f, g, h, i, j, k, l, m

Figure A.7: SSS* Table Pass 1

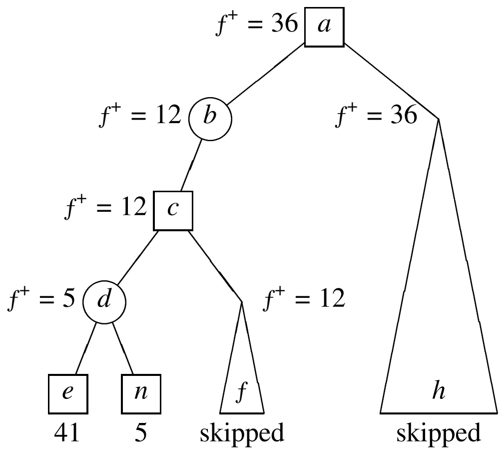

Figure A.8: SSS* Pass 2

#	n	Γ	OPEN list after Γ	f^-_{top}	T^-_{top}	f^+_{root}	T^+_{root}
14	e	2	$\big(\langle n,L,41\rangle, \langle m,S,36\rangle, \langle g,S,12\rangle, \langle k,S,10\rangle\big)$	$-\infty$		41	$a,b,c,d,e,f,g,h,i,j,k,l,m$
15	n	4	$\big(\langle m,S,36\rangle, \langle g,S,12\rangle, \langle k,S,10\rangle, \langle n,S,5\rangle\big)$	36	m	36	$a,b,c,d,e,n,f,g,h,i,j,k,l,m$

Figure A.9: SSS* Table Pass 2

a max solution tree (compare figure A.6 to figure 2.6). The minimax value of this tree is 41, which is also the $\hat{h}$ value of the first entry of the OPEN list.

Second pass: (see figures A.8 and A.9)
In the second pass, SSS* will try to lower the upper bound of 41 to come closer to f. The next upper bound will be computed by expanding a brother of the critical leaf e. The critical leaf has a min parent, node d, so expanding this brother can lower d's value, which will, in turn, lower the minimax value at the root of the max solution tree. Since this value is the maximum of its leaves, there is no point in expanding brothers of non-critical leaves, since then node e will keep the value of the root at 41. Thus, node n is in a sense the best node to expand. The entry for node e matches Γ case 2, which replaces e by the brother n, giving it state *live*, and the value 41, the sharpest (lowest) upper bound of the previous pass. The n entry matches Γ case 4. Case 4 evaluates the leaf, and assigns to $\hat{h}$ either this value (5), or the sharpest upper bound so far, if that happens to be lower. Node n gets value 5. In general, Γ case 4 performs the minimizing operation of the minimax function, ensuring that the $\hat{h}$ of the first (highest) entry of the OPEN list will always be the sharpest upper bound on the minimax value of the root, based on the previously expanded nodes. The OPEN list has become $\big(\langle m,S,36\rangle, \langle g,S,12\rangle, \langle k,S,10\rangle, \langle n,S,5\rangle\big)$. Thus, the upper bound on the root has been lowered to 36. Its value is determined by a new (sharper) max solution tree,

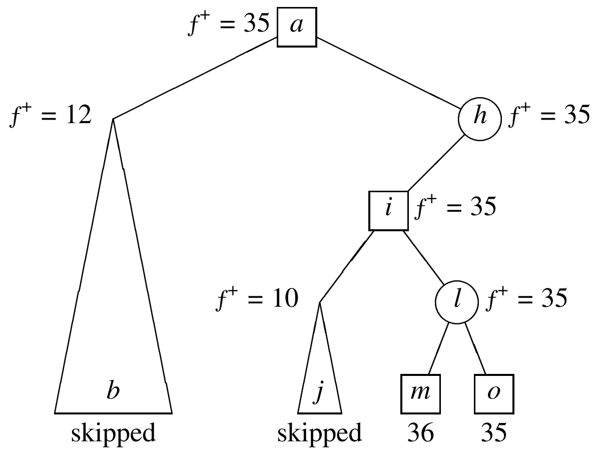

Figure A.10: SSS* Pass 3

#	n	Γ	OPEN list after Γ	f^-_{top}	T^-_{top}	f^+_{root}	T^+_{root}
16	m	2	$(\langle o, L, 36\rangle, \langle g, S, 12\rangle,$ $\langle k, S, 10\rangle, \langle n, S, 5\rangle)$	$-\infty$		36	$a, b, c, d, e, n, f, g,$ h, i, j, k, l, m, o
17	o	4	$(\langle o, S, 35\rangle, \langle g, S, 12\rangle,$ $\langle k, S, 10\rangle, \langle n, S, 5\rangle)$	35	o	35	$a, b, c, d, e, n, f, g,$ h, i, j, k, l, m, o

Figure A.11: SSS* Table Pass 3

whose leaves are contained in the OPEN list.

Third Pass: (see figures A.10 and A.11)
In the third pass, the goal of the search is to get the upper bound below 36. Just as in the second pass, the first entry of the OPEN list, m, matches Γ case 2, and its brother is inserted. It matches Γ case 4, so it gets evaluated. The new brother is node o, with value 35. Again, a sharper upper bound has been found. The new OPEN list is $(\langle o, S, 35\rangle, \langle g, S, 12\rangle, \langle k, S, 10\rangle, \langle n, S, 5\rangle)$.

Fourth Pass: (see figures A.12 and A.13)
The previous search lowered the bound from 36 to 35. In the fourth pass the first entry has no immediate brother. It matches a Γ case that is used to backtrack, case 3, which replaces node o by its parent l. Case 3 is always followed by case 1, which replaces l by its parent i and, in addition, deletes all child-entries from the list—only node k in this case. Each time case 1 applies, all children of a min node and its max parent have been expanded and the search of the subtree has been completed. To avoid having old nodes interfere with the remainder of the search, they must be removed from the OPEN list. The list now contains: $(\langle i, S, 35\rangle, \langle g, S, 12\rangle, \langle n, S, 5\rangle)$. Next, case 2 matches entry i, and expansion of the brother of i commences. Node p is inserted into the list with state *live*. It matches case 6, which inserts q and r into the list. Node q matches case 5,

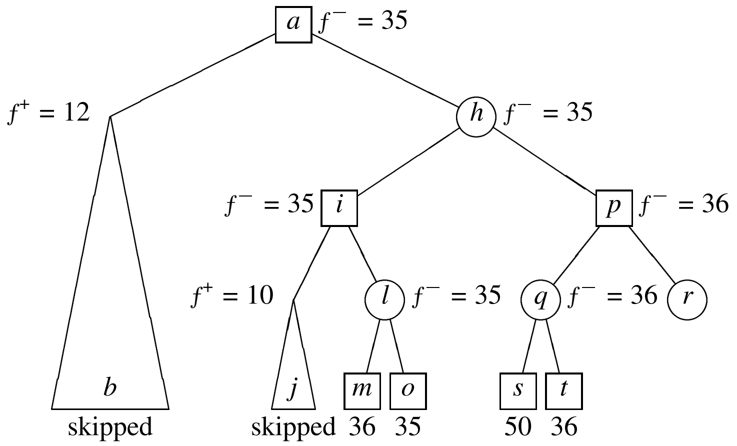

Figure A.12: SSS* Pass 4

#	n	Γ	OPEN list after Γ	f^-_{top}	T^-_{top}	f^+_{root}	T^+_{root}
18	o	3	$(\langle l,S,35\rangle, \langle g,S,12\rangle, \langle k,S,10\rangle, \langle n,S,5\rangle)$	35	l,m,o	35	$a,b,c,d,e,n,f,g,$ h,i,j,k,l,m,o
19	l	1	$(\langle i,S,35\rangle, \langle g,S,12\rangle, \langle n,S,5\rangle)$	35	i,l,m,o	35	$a,b,c,d,e,n,f,g,$ h,i,j,k,l,m,o
20	i	2	$(\langle p,L,35\rangle, \langle g,S,12\rangle, \langle n,S,5\rangle)$	$-\infty$		35	$a,b,c,d,e,n,f,g,$ h,i,j,k,l,m,o,p
21	p	6	$(\langle q,L,35\rangle, \langle r,L,35\rangle, \langle g,S,12\rangle, \langle n,S,5\rangle)$	$-\infty$		35	$a,b,c,d,e,n,f,g,$ h,i,j,k,l,m,o,p,q,r
22	q	5	$(\langle s,L,35\rangle, \langle r,L,35\rangle, \langle g,S,12\rangle, \langle n,S,5\rangle)$	$-\infty$		35	$a,b,c,d,e,n,f,g,$ h,i,j,k,l,m,o,p,q,r,s
23	s	4	$(\langle s,S,35\rangle, \langle r,L,35\rangle, \langle g,S,12\rangle, \langle n,S,5\rangle)$	50	s	35	$a,b,c,d,e,n,f,g,$ h,i,j,k,l,m,o,p,q,r,s
24	s	2	$(\langle t,L,35\rangle, \langle r,L,35\rangle, \langle g,S,12\rangle, \langle n,S,5\rangle)$	$-\infty$		35	$a,b,c,d,e,n,f,g,$ h,i,j,k,l,m,o,p,q,r,s,t
25	t	4	$(\langle t,S,35\rangle, \langle r,L,35\rangle, \langle g,S,12\rangle, \langle n,S,5\rangle)$	36	t	35	$a,b,c,d,e,n,f,g,$ h,i,j,k,l,m,o,p,q,r,s,t
26	t	3	$(\langle q,S,35\rangle, \langle r,L,35\rangle, \langle g,S,12\rangle, \langle n,S,5\rangle)$	36	q,s,t	35	$a,b,c,d,e,n,f,g,$ h,i,j,k,l,m,o,p,q,r,s,t
27	q	1	$(\langle p,S,35\rangle, \langle g,S,12\rangle, \langle n,S,5\rangle)$	35	p,q,s,t	35	$a,b,c,d,e,n,f,g,$ h,i,j,k,l,m,o,p,q,r,s,t
28	p	3	$(\langle h,S,35\rangle, \langle g,S,12\rangle, \langle n,S,5\rangle)$	35	$h,i,l,m,$ o,p,q,s,t	35	$a,b,c,d,e,n,f,g,$ h,i,j,k,l,m,o,p,q,r,s,t
29	h	1	$(\langle a,S,35\rangle)$	35	$a,h,i,l,m,$ o,p,q,s,t	35	$a,b,c,d,e,n,f,g,$ h,i,j,k,l,m,o,p,q,r,s,t

Figure A.13: SSS* Table Pass 4

which inserts its left-most child s, still with $\hat{h}$ value 35. This leaf is then evaluated by case 4. The evaluation of 50 is not less than the sharpest upper bound of 35, so $\hat{h}$ is not changed. The OPEN list is now: $(\langle s,S,35\rangle,\langle r,L,35\rangle,\langle g,S,12\rangle,\langle n,S,5\rangle)$. Node s is a max node with a brother. It matches case 2, which replaces s by its brother t. Node t is evaluated to value 36 by case 4, which again does not lower the sharpest upper bound of 35. The OPEN list is now: $(\langle t,S,35\rangle,\langle r,L,35\rangle,\langle g,S,12\rangle,\langle n,S,5\rangle)$. Node t matches case 3, which is followed by case 1, inserting p and purging the OPEN list of entry r. The list is now: $(\langle p,S,35\rangle,\langle g,S,12\rangle,\langle n,S,5\rangle)$. Since max node p has no brothers, case 3 applies, which is followed by case 1. Case 1 inserts the root a into the list, and purges the list of all the children of a. The list now becomes the single *solved* entry $\langle a,S,35\rangle$, which satisfies the termination condition of SSS*. The minimax value is 35.

A.3 MT-SSS* Example

Now we will use the tree from the previous examples to show how the reformulation of SSS* works.

SSS* finds f_{root} by determining a sequence of upper bounds on it. The idea behind MT-SSS* is that these upper bounds can also be found using a null-window Alpha-Beta call. The null-window call creates the solution tree. This solution tree is stored in memory, so that it can be refined in later passes. It turns out that in this way Alpha-Beta will expand the same solution trees as SSS*. We will show how this works in detail using an example. Comparing it to the SSS* example illustrates that both formulations expand the same trees. (In appendix B the equivalence of the two formulations is discussed in some detail.)

Alpha-Beta is used to construct solution trees. The postcondition of the Alpha-Beta procedure in section 2.1.3 suggests that using outcome 2, we can have it return an upper bound if we make it fail low. To create a fail low, Alpha-Beta must be called with a search window greater than any possible leaf node value. Alpha-Beta, when called with such a window, will find the same upper bound, and expand the same max solution tree, as SSS*. This can be seen intuitively because both Alpha-Beta and SSS* expand the children of a node in a left-to-right order.

The example tree in figure A.2 is searched to determine its minimax value. A number of stages, or passes, can be distinguished in the traversal of this tree. At the end of each pass a full max solution tree exists, which determines a better upper bound on the minimax value. Also, solved nodes represent min solution trees. These two solution trees are shown in the tables. Note that in the figures the nodes are still numbered a to t in the order in which SSS* visits them first, not MT-SSS*.

First pass: (see figures A.14 and A.15)
In the first pass the left-most max solution tree is expanded to create the first non-trivial upper bound on the minimax value of the root. SSS* builds the max solution tree shown in figure A.14, using cases 4, 5, and 6 of the Γ operator. Instead of using Γ cases 4, 5 and 6 and a sorted OPEN list, MT-SSS* uses MT to compute the bound, by traversing

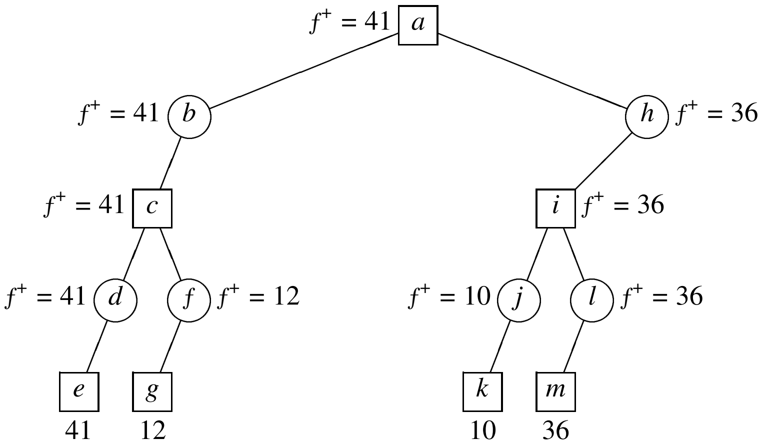

Figure A.14: MT-SSS* Pass 1

#	n	γ_n	g_n	cutoff?	f_n^-	T_n^-	f_n^+	T_n^+
1	a	$+\infty$	$-\infty$		$-\infty$		$+\infty$	
2	b	$+\infty$	$+\infty$		$-\infty$		$+\infty$	
3	c	$+\infty$	$-\infty$		$-\infty$		$+\infty$	
4	d	$+\infty$	$+\infty$		$-\infty$		$+\infty$	
5	e	$+\infty$	41		$-\infty$	e	41	e
6	d	$+\infty$	41	$41 \not\geq +\infty$	$-\infty$		41	d, e
7	c	$+\infty$	41	$41 < +\infty$	$-\infty$		$+\infty$	
8	f	$+\infty$	$+\infty$		$-\infty$		$+\infty$	
9	g	$+\infty$	12		$-\infty$	g	12	g
10	f	$+\infty$	12	$12 \not\geq +\infty$	$-\infty$		12	f, g
11	c	$+\infty$	41	$\perp$	$-\infty$		41	c, d, e, f, g
12	b	$+\infty$	41	$41 \not\geq +\infty$	$-\infty$		41	b, c, d, e, f, g
13	a	$+\infty$	41	$41 < +\infty$	$-\infty$		$+\infty$	
14	h	$+\infty$	$+\infty$		$-\infty$		$+\infty$	
15	i	$+\infty$	$-\infty$		$-\infty$		$+\infty$	
16	j	$+\infty$	$+\infty$		$-\infty$		$+\infty$	
17	k	$+\infty$	10		$-\infty$	k	10	k
18	j	$+\infty$	10	$10 \not\geq +\infty$	$-\infty$		10	j, k
19	i	$+\infty$	10	$10 < +\infty$	$-\infty$		$+\infty$	
20	l	$+\infty$	$+\infty$		$-\infty$		$+\infty$	
21	m	$+\infty$	36		$-\infty$	m	36	m
22	l	$+\infty$	36	$36 \not\geq +\infty$	$-\infty$		36	l, m
23	i	$+\infty$	36	$\perp$	$-\infty$		36	i, j, k, l, m
24	h	$+\infty$	36	$36 \not\geq +\infty$	$-\infty$		36	h, i, j, k, l, m
25	a	$+\infty$	41	$\perp$	$-\infty$		41	$a, b, c, d, e, f, g, h, i, j, k, l, m$

Figure A.15: MT-SSS* Table Pass 1

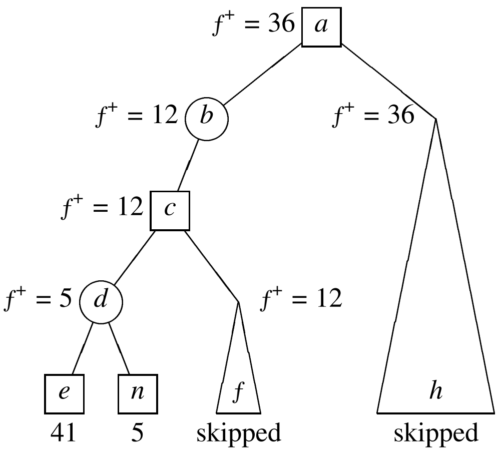

Figure A.16: MT-SSS* Pass 2

#	n	γ_n	g_n	cutoff?	f_n^-	T_n^-	f_n^+	T_n^+
26	a	41	$-\infty$		$-\infty$		41	$a,b,c,d,e,f,g,h,i,j,k,l,m$
27	b	41	$+\infty$		$-\infty$		41	b,c,d,e,f,g
28	c	41	$-\infty$		$-\infty$		41	c,d,e,f,g
29	d	41	$+\infty$		$-\infty$		41	d,e
30	e	41	41	$41 \neq +\infty$	41	e	$+\infty$	e
31	n	41	5		$-\infty$	n	5	n
32	d	41	5	$\perp$	$-\infty$	d,e,n	5	d,n
33	c	41	5	$5 < 41$	$-\infty$		$+\infty$	
34	f	41	12	$12 \ngeq 41$	$-\infty$		12	f,g
35	c	41	12	$\perp$	$-\infty$	c,d,e,n	12	c,d,n,f,g
36	b	41	12	$12 \ngeq 41$	$-\infty$		12	b,c,d,n,f,g
37	a	41	12	$12 < 41$	$-\infty$		12	$a,b,c,d,n,f,g,h,i,j,k,l,m$
38	h	41	36	$36 \ngeq 41$	$-\infty$		36	h,i,j,k,l,m
39	a	41	36	$\perp$	$-\infty$		36	$a,b,c,d,n,f,g,h,i,j,k,l,m$

Figure A.17: MT-SSS* Table Pass 2

solution trees. At unexpanded nodes, f^- is $-\infty$, and f^+ is $+\infty$. A call MT(G, ∞) will cause an alpha cutoff at all min nodes, since all internal calls return values $g < \gamma = \infty$. No beta cutoffs at max nodes will occur, since all $g < \gamma$. We see that the call MT(a, ∞) on the tree in figure A.2 will traverse the tree in figure A.14, conforming to Alpha-Beta's postcondition. Due to the "store" operation in figure 3.2, this tree is saved in memory so that its backed-up values can be used in a later pass. The max solution tree stored at the end of this pass consists of the nodes $a, b, c, d, e, f, g, h, i, j, k, l$ and m, yielding an upper bound of 41.

Second pass: (see figures A.16 and A.17)
This pass lowers the upper bound on f from 41 to 36. How can we use Alpha-Beta to do this? Since the max solution tree defining the upper bound of 41 has been stored by the previous MT call, Alpha-Beta can re-traverse the nodes on the principal variation

(a, b, c, d, e) to find the critical leaf e, and see whether expanding its brother will yield a search tree with a lower minimax value. Finding this critical leaf, and selecting its brother for expansion is the essence of the "best-first" behavior of SSS* (and MT-SSS*). The critical leaf e has a min parent, node d, so expanding the brother can lower its value, which will, in turn, lower the minimax value at the root of the max solution tree. Since this value is the maximum of its leaves, there is no point in expanding brothers of non-critical leaves, because then node e will keep the value of the root at 41. Thus, based on the information that the max solution tree provides, there is only one node (n) whose expansion makes sense. Other nodes are worse, since they cannot change the bound at the root.

To give Alpha-Beta the task of returning a value lower than $f^+ = 41$, we give it a search window which will cause it to fail low. The old window of $\langle \infty - 1, \infty \rangle$ will not do, since the code in figure 3.2 will cause it to return from both nodes b and h, with a value of 41, lower than $+\infty$, but not the lower upper bound. A better choice is the search window $\langle f^+ - 1, f^+ \rangle$, or $\langle 40, 41 \rangle$, which prompts MT to descend the principal variation and return as soon as a lower f^+ on node a is found. It descends to nodes b, c, d, e and continues to search node n. It will back up value 5 to node d and cause a cutoff. The value of d is no longer determined by e but by n. Node e is no longer part of the max solution tree that determines the sharpest upper bound. It has been proven that e can be erased from memory as long as we remember that n is the new best child (not shown in the MT code). The value 5 is backed up to c. No beta cutoff occurs at c, so f's bound is retrieved. Since $f^+ \not\geq \gamma$ at node f, MT does not enter it, but uses $c.f^+ = 12$ for g'. 12 is backed up to b, where it causes an alpha cutoff. Next, 12 is backed up to a. Since $g < \gamma$, node h is probed, but since $c.f^+ \not\geq \gamma$ $(36 \not\geq 41)$ it is not entered. The call MT$(a, 41)$ fails low with value 36, the sharper upper bound. The max solution tree defining this bound consists of nodes $a, b, c, d, n, f, g, h, i, j, k, l$ and m (that is, node e has been replaced with n).

By storing previously expanded nodes in memory, and calling MT with the right search window, we can make it traverse the principal variation, and expand brothers of the critical leaf, to get a better upper bound on the minimax value of the root, in exactly the same way as SSS* does.

Third Pass: (see figures A.18 and A.19)
In the previous pass, the upper bound was lowered from 41 to 36. A call MT$(a, 36)$ is performed. From the previous search, we know that b has an $f^+ \not\geq 36$ so it is not entered; $h.f^+ \geq 36$, so it is. The algorithm follows the principal variation to the node giving the 36 (h to i to l to m). The brother of m is expanded. The bound on the minimax value at the root has now been improved from 36 to 35. The max solution tree defining this bound consists of nodes $a, b, c, d, n, f, g, h, i, j, k, l$ and o.

Fourth Pass: (see figures A.20 and A.21)
This is the last pass of MT-SSS*. We will find that the upper bound cannot be lowered. A call with window $\langle f^+ - 1, f^+ \rangle$, or MT$(a, 35)$, is performed. In this pass we will not

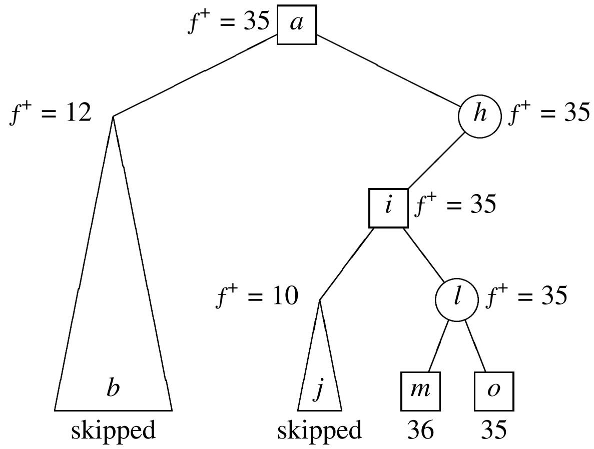

Figure A.18: MT-SSS* Pass 3

#	n	γ_n	g_n	cutoff?	f_n^-	T_n^-	f_n^+	T_n^+
40	a	36	$-\infty$		$-\infty$		36	$a, b, c, d, n, f, g, h, i, j, k, l, m$
41	b	36	12	$12 \not\geq 36$	$-\infty$		12	b, c, d, n, f, g
42	h	36	$+\infty$		$-\infty$		36	h, i, j, k, l, m
43	i	36	$-\infty$		$-\infty$		36	i, j, k, l, m
44	j	36	10	$10 \not\geq 36$	$-\infty$		10	j, k
45	l	36	$+\infty$		$-\infty$		36	l, m
46	m	36	36	$36 \not\leq +\infty$	36	m	$+\infty$	m
47	o	36	35		$-\infty$	o	35	o
48	l	36	35	$\perp$	$-\infty$	l, m, o	35	l, o
49	i	36	35	$\perp$	$-\infty$	i, l, m, o	35	i, j, k, l, o
50	h	36	35	$35 \not\geq 36$	$-\infty$		35	h, i, j, k, l, o
51	a	36	35	$\perp$	$-\infty$		35	$a, b, c, d, n, f, g, h, i, j, k, l, o$

Figure A.19: MT-SSS* Table Pass 3

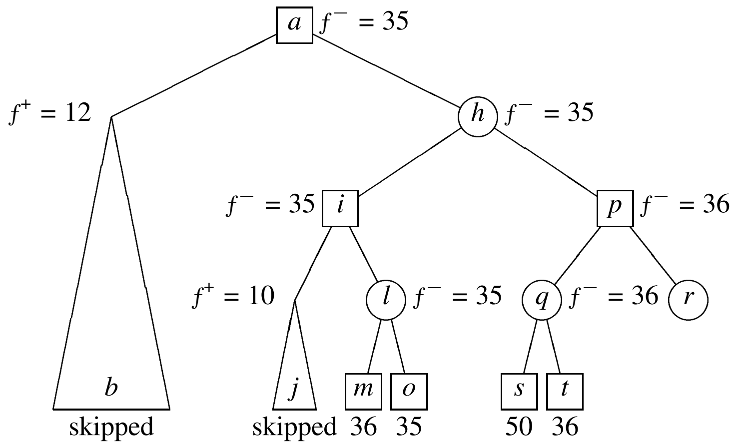

Figure A.20: MT-SSS* Pass 4

#	n	γ_n	g_n	cutoff?	f_n^-	T_n^-	f_n^+	T_n^+
52	a	35	$-\infty$		$-\infty$		35	$a, b, c, d, n, f, g, h, i, j, k, l, o$
53	b	35	12	$12 \not\geq 35$	$-\infty$		12	b, c, d, e, f, g
54	h	35	$+\infty$		$-\infty$		35	h, i, j, k, l, o
55	i	35	$-\infty$		$-\infty$	i, l, m, o	35	i, j, k, l, o
56	j	35	34	$34 \not\geq 35$	$-\infty$		34	j, k
57	l	35	$+\infty$		$-\infty$		36	l, m
58	m	35	36	$36 \not< 35$	36	m	$+\infty$	m
59	o	35	35	$35 \not= +\infty$	35	o	$+\infty$	o
60	l	35	35	$\perp$	35	l, m, o	$+\infty$	l, o
61	i	35	35	$\perp$	35	i, l, m, o	$+\infty$	i, j, k, l, o
62	p	35	$-\infty$		$-\infty$		$+\infty$	
63	q	35	$+\infty$		$-\infty$		$+\infty$	
64	s	35	50		50	s	$+\infty$	s
65	t	35	36		36	t	$+\infty$	t
66	q	35	36	$\perp$	36	q, s, t	$+\infty$	q, t
67	p	35	36	$36 \not< 35$	36	p, q, s, t	$+\infty$	
68	h	35	35	$\perp$	35	$h, i, l, m, o, p, q, s, t$	$+\infty$	h, i, j, k, l, o
69	a	35	35	$\perp$	35	$a, h, i, l, m, o, p, q, s, t$	$+\infty$	$a, b, c, d, n, f, g, h, i, j, k, l, o$

Figure A.21: MT-SSS* Table Pass 4

find a fail low as usual, but a fail high with return value 35. The return value is now a lower bound, backed-up by a min solution tree (all children of a min node included, only one for each max node).

How does Alpha-Beta traverse this min solution tree? The search follows the critical path a, h, i, l and o. At node l, child m has $c.f^- \not< \gamma$, so it is not entered. Now o is entered but not evaluated since it is no longer open. Node o returns the stored bound $o.f^+ = 35$. The value of the children is retrieved from storage. Note that the previous pass stored an f^+ value for l and o, while this pass stores an f^-. Node i cannot lower h's value ($g \geq \gamma, 35 \geq 35$, no cutoff occurs), so the search explores p. Node p expands q which, in turn, searches s and t. Since p is a maximizing node, the value of q (36) causes a cutoff: $g \not< \gamma$, node r is not searched. Both of h's children are ≥ 35. Node h returns 35, and so does a. Node a was searched attempting to show whether its value was $<$ or ≥ 35. Node h provides the answer: greater than or equal. This call to MT fails high, meaning we have a lower bound of 35 on the search. The previous call to MT established an upper bound of 35. Thus the minimax value of the tree is proven to be 35.

We see that nothing special is needed to have Alpha-Beta traverse the min solution tree $a, h, i, l, m, o, p, q, s$ and t. The ordinary cutoff decisions cause its traversal, when $\alpha = f^+(a) - 1$ and $\beta = f^+(a)$.

In the previous four passes we called MT (Alpha-Beta) with a special search window to have it emulate SSS*. This sequence of calls, creating a sequence of fail lows until the final fail high, can be captured in a single loop, given by the pseudo code of figure 3.1.

Appendix B

Equivalence of SSS* and MT-SSS*

This appendix discusses the ideas behind the equivalence of Stockman's SSS* and the new reformulation MT-SSS*. We do not present a formal proof in this appendix. That can be found in [106]. An outline of the full proof can be found in appendix A of [113, 112]. The aim of the current appendix is to convey the idea behind the equivalence. It can be regarded as an "informal proof," or as an illustration supporting the formal proof, which requires a certain amount of familiarity with the old formulation of SSS*.

B.1 MT and List-ops

Figure B.1 shows an extended version of MT, to be called by MT-SSS*, shown in figure B.2. The list operations (list-ops) between **{*** and ***}** are inserted to show the equivalence of MT-SSS* and Stockman's SSS*. The value of $\hat{h}$ is the current value of γ. (In implementations of MT-SSS* the list-ops should not be included.) The claim is that, when called by MT-SSS*, the list operations in MT cause the same Γ operations to be applied as in Stockman's original formulation. These list-ops are the core of the equivalence proof [106]. Their place in the MT code shows in a clear and concise way where and how the SSS* operators fit in the way Alpha-Beta traverses a tree. The example in appendix A.3 can be used to check this. Without the list-ops, the call MT(n, γ) is an ordinary null-window Alpha-Beta$(n, \gamma - 1, \gamma)$ search, except that MT uses one bound, making the code a bit simpler.

MT-SSS* and SSS* are equivalent in the sense that they evaluate the same leaf nodes in the same order, when called on the same minimax tree. In the previous chapters and examples SSS* and MT-SSS* have been treated primarily as algorithms that manipulate max solution trees. The original SSS* formulation stresses the min-solution-tree view. The full story is, of course, that both max and min solution trees are manipulated by SSS*, just as Alpha-Beta's postcondition in section 2.1.3 shows that Alpha-Beta also constructs both types of trees.

The critical tree that proves the minimax value of a game tree is a union of the max solution tree with the lowest possible upper bound, and the min solution tree with

```
/* For equivalence with SSS* this function must be called by MT-SSS*   */
/* (see figure B.2)                                                    */
/* MT: storage enhanced null-window Alpha-Beta(n, γ − 1, γ).           */
/* n is the node to be searched, γ − 1 is α , γ is β in the call.      */
function MT(n, γ) → g;
  if n = leaf then
      retrieve n.f⁻, n.f⁺; /* non-existing bounds are ± ∞ */
      if n.f⁻ = −∞ and n.f⁺ = +∞ then
          {* List-op(4, n); *}
          g := eval(n);
      else if n.f⁺ = +∞ then g := n.f⁻; else g := n.f⁺;
  else if n = max then
      {* retrieve n.f⁻, n.f⁺; if n.f⁺ = +∞ and n.f⁻ = −∞ then List-op(6, n); *}
      g := −∞;
      c := firstchild(n);
      /* g ≥ γ causes a beta cutoff (β = γ) */
      while g < γ and c ≠ ⊥ do
          retrieve c.f⁺;
          if c.f⁺ ≥ γ then
              g′ := MT(c, γ);
              {* if g′ ≥ γ then List-op(1, c); *}
          else g′ := c.f⁺;
          g := max(g, g′);
          c := nextbrother(c);
  else /* n is a min node */
      {* retrieve n.f⁻, n.f⁺; if n.f⁺ = +∞ and n.f⁻ = −∞ then List-op(5, n); *}
      g := +∞;
      c := firstchild(n);
      /* g < γ causes an alpha cutoff (α = γ − 1) */
      while g ≥ γ and c ≠ ⊥ do
          retrieve c.f⁻;
          if c.f⁻ < γ then
              g′ := MT(c, γ);
              {* if g′ ≥ γ then
                  if c < lastchild(n) then List-op(2, c); else List-op(3, c); *}
          else g′ := c.f⁻;
          g := min(g, g′);
          c := nextbrother(c);
  /* Store one bound per node. Delete possible old bound (see page 53). */
  if g ≥ γ then n.f⁻ := g; store n.f⁻;
        else n.f⁺ := g; store n.f⁺;
  return g;
```

Figure B.1: MT with SSS*'s List-Ops

function MT-SSS*(n) $\rightarrow f$;
    $g := +\infty$;
    repeat
        $\gamma := g$;
        $g :=$ MT(n, γ);
    until $g = \gamma$;
    return g;

Figure B.2: SSS* as a Sequence of MT Searches

the highest possible lower bound. SSS* can be regarded as an interleaving of two processes, one working downward on the max solution tree part of the final minimal tree, and one working upward on the min solution tree part. First a max solution tree is expanded downwards from the root. Next, SSS* constructs a min solution tree growing upwards off of the critical leaf. The aim is to create a min solution tree that reaches all the way up to the root, and has the same value as the max solution tree, because that signals that the critical tree that proves the minimax value has been constructed.

However, often this process is ended prematurely, when SSS* finds that the value of the min solution tree drops below that of another node/min solution tree, which causes the whole process to start over again, giving SSS* its interleaved nature.

B.2 MT and the Six Operators

Turning to figures B.2, B.1, and 2.17, the construction of solution trees by the six operators will be discussed in a little more detail, focusing on the circumstances in which SSS* and MT invoke the six Γ operators.

Many of the operations that SSS* performs have a minor, local, effect: a max solution tree is being expanded, or SSS* backtracks to the node where the next one will be grown. However, at some points operations with a more major, global, nature are performed. When operator 4 has evaluated the last leaf of a max solution tree, and all the entries in the OPEN list are *solved*, the highest $\hat{h}$ value drops. The selection of the next entry, through SSS*'s sorting of the OPEN list, is a major decision point, where SSS* selects a node which is in a certain sense globally "best" (figure 2.18).

In MT-SSS* the minor, local, operations are performed by the MT code, by having a null-window Alpha-Beta search construct a solution tree. The major decision to find the next node which is globally "best" occurs in MT-SSS* when MT is called at the root to traverse the critical path to the critical leaf. In MT-SSS* the global decisions are more visible than in SSS*, since they coincide with a new pass of the main loop.

A number of features of SSS* play an important role in this discussion. At any time, the highest $\hat{h}$ value in the OPEN list is an upper bound on the minimax value of the root of the game tree. The entries are the leaves of a partial max solution tree. When the OPEN list consists only of *solved* entries, a pass of the main loop of MT-SSS* has ended (assuming that only one critical path exists). The entries in the list are the leaves of a total max solution tree, that is rooted at the root of the game tree.

Concerning MT, the following pre- and postcondition hold (following Alpha-Beta's postcondition in section 2.1.3): at a fail high, MT has traversed a min solution tree, whose value $f^-(n)$ is stored with node n. At a fail low, MT has traversed a max solution tree, whose value $f^+(n)$ is stored with node n. Since MT is called by MT-SSS* with $\gamma = f^+(root)$, the node on which MT is called is either open, or part of a max solution tree.

The SSS* notions *live* and *solved* correspond to the Alpha-Beta notions *open* (children not yet expanded) and *closed* (all children expanded). The term *sub* max solution tree is used for a max solution tree whose root is an interior node, not necessarily the root of the game tree, in other words, a smaller, deeper, max solution tree. A sub max solution tree may even be as small as a single leaf. The same applies to the term sub min solution trees.

It will be shown that the list-ops in MT cause the same operations to be performed on the OPEN list as the original SSS* operators. One of the key points is that SSS* always selects the entry with the highest $\hat{h}$ value. (Other preconditions of the operators, such as being applied to an open node or to a max node, are easily verified from the MT code in figure B.1.) Therefore, the focus of attention in the following discussion will be on how MT performs all the list-ops on the (left-most) node with the highest $\hat{h}$ value.

The key circumstances in which the six operators are invoked are discussed from the viewpoint of SSS* and MT.

SSS:* At the start of the algorithm, the OPEN list is empty. The entry with the highest $\hat{h}$ value is the root. SSS* now expands the left-most max solution tree using the operators 4, 5, and 6.

MT: MT selects this entry, below which the left-most max solution tree is expanded (by Alpha-Beta's postcondition), applying list-op's 4, 5, and 6. Examination of the SSS* operators on the one hand, and the MT code on the other, shows easily that the list-op's in MT follow the same expansion sequence as the operators in SSS*.

SSS:* At the start of another pass of MT-SSS*, the OPEN list contains only *solved* entries, that form the leaves of a total max solution tree. The highest entry is an upper bound on the minimax value of the root, and contains the critical leaf of the max solution tree.

MT: The conventional Alpha-Beta cutoff decisions cause MT to traverse the left-most critical path of the max solution tree, along which $\gamma = f^+(n) = f^+(root)$ holds, until the left-most critical leaf is reached. (In closed max nodes the left-most child with $c.f^+ = \gamma$ is entered; the highest child among the children in the max solution tree. In closed min nodes the left-most child with $c.f^- = -\infty < \gamma$ is entered; the only child in the max solution tree.) Thus, both SSS* and MT select the critical leaf to be operated upon.

Next, SSS* selects the node where a sub max solution tree will be expanded, in order to try to lower the upper bound on the minimax value of the root.

SSS:* Depending on the parity of the depth of the tree, either operator 1, or 2/3 are performed on the critical leaf. (Turning to the MT code, the critical leaf's value is equal to γ, so the inner MT call fails high with $g' \geq \gamma$.) Assuming an even depth of the tree, operator 2 or 3 are performed. These operators expand the "best" node: the node (or rather, the sub max solution tree which happens to be only a single node) that can influence the value at the root (see also figure 2.18). If the critical leaf has an unexpanded brother, then operator 2 selects it.

MT: In MT the unexpanded brother is selected because the return value $g' \geq \gamma$ does not cause a cutoff. (The MT-SSS* code ensures that $\gamma = f^+(root)$ is equal to the highest $\hat{h}$ value).

SSS:* If all open brothers of the critical leaf have been expanded with decreasing their $\hat{h}$ value, operator 3 applies, which performs half of the back-up operation to start expanding the next smallest sub max solution tree. Operator 1 applies next, which does the other half of the back-up motion, putting a max node at the front of the list, to be replaced by operator 2 by its next brother, starting the expansion of the sub max solution tree by operator 4, 5 and 6 just as when the sub max solution tree were only a single node, only it is bigger. (For odd depth trees, the part of the back-up process starting with operator 1 also applies.)

MT: In MT the back-up motion of operator 3 occurs because there are no more children, causing MT to automatically back-up to its parent. The back-up motion of operator 1 occurs because the return value of the inner MT call is still a fail high ($g' \geq \gamma$), which causes a cutoff in the max parent because $g \not< \gamma$, causing MT to back-up. In the min node where MT ends up, the return value is still a fail high, which causes no cutoff, so the next child is expanded, just like operator 2 does. (Of course, if no brother exists for operator 2, operator 3 applies. And if operator 1 puts the root on the list, which has no brothers, MT/MT-SSS* and SSS* both terminate.) Thus MT and SSS* also select the same nodes when the place to expand the sub max solution tree is to be determined.

Expansion of the sub max solution tree succeeds if the value of $\hat{h}$ is lowered, or in MT terms, if the inner MT calls fail low. If it fails high, a min solution tree will be constructed. (In the MT case this follows from the Alpha-Beta postcondition.)

SSS:* The question of failing high or low occurs when operator 4 is performed. A "fail high" causes the $\hat{h}$ value of the first entry to stay the same, so the backing-up operators for solved entries 1, 2 and 3 are applied, in the same fashion as described previously. A "fail low" causes another entry to have the highest $\hat{h}$ value. This entry has been inserted by operator 6, the only operator that inserts multiple entries into the list. All of the entries inserted by operator 6 have the same $\hat{h}$ value. The only operator that can lower an $\hat{h}$ value is number 4. When this happens, the next brother (or rather, (great) uncle) appears at the front. (No entries from outside the sub max solution tree to be constructed can enter, because their $\hat{h}$ value is lower.) In this way, as long as the expansions "fail low", a max solution tree is expanded in the same way as the left-most max solution tree in the first pass.

MT: In MT, construction of a sub max solution tree after a fail low is easy to see from Alpha-Beta's postcondition, or from the code. (The same applies for a sub min solution tree after a fail high).

This concludes the description of a single pass of MT-SSS*. Next, the globally "best" node is selected—in SSS* since the OPEN list is sorted, in MT-SSS* because the stored $f^+(n)$ values together with the γ value cause MT to traverse the critical path.

The previous explanation indicated that SSS* and MT perform the same operations on the OPEN list for the following cases:

- At the first call to MT from MT-SSS*, where the left-most max solution tree is constructed.
- When finding the critical leaf in subsequent calls to MT from the main loop of MT-SSS*, where the OPEN list contains the leaves of a max solution tree.
- When further expanding parts of the tree in order to get a better upper bound on the root (fail low), or finding that the current upper bound is the minimax value (fail high).

Comparing SSS* to MT/MT-SSS*, the ingenuity with which the six Γ cases emulate Alpha-Beta's behavior in MT-SSS* is extraordinary. Why SSS* was created in its old form is easier to see if one realizes that its roots lie in AO*, an algorithm for the search of problem-reduction spaces [93].