Partial Information Endgame Databases

Yngvi Björnsson[1], Jonathan Schaeffer[2], and Nathan R. Sturtevant[2]

[1] Department of Computer Science,
Reykjavik University, Reykjavik, Iceland
yngvi@ru.is
[2] Department of Computing Science,
University of Alberta, Edmonton, Alberta, Canada
{jonathan, nathanst}@cs.ualberta.ca

Abstract. Endgame databases have previously been built based on complete analysis of endgame positions. In the domain of Checkers, where endgame databases consisting of 39 *trillion* positions have already been built, it would be beneficial to be able to build select portions of even larger databases, without fully computing portions of the database that will almost never be needed. We present a new win-loss-draw value algorithm that can build endgame databases when unknown (partial information) values are present, showing that significant portions of these databases can be resolved using these methods.

1 Introduction

Endgame databases were pioneered over 20 years ago [8,9]. The basic idea, computing the value of positions at the end of the game and backing up the values towards the start of the game, is both simple and powerful. These databases have provided numerous insights to human analysts (e.g., chess [3]), have been useful for solving games (e.g., awari [4], nine men's morris [1]), and have been instrumental in building super-human programs (e.g., CHINOOK [5]).

The biggest (and longest) endgame database computation is that from the CHINOOK checkers project. The databases currently contain 39 *trillion* (3.9×10^{13}) positions—all positions with 10 or fewer pieces on the board. These databases have been instrumental in an on-going effort to solve the game of checkers. In January 2005, the CHINOOK team achieved their first milestone, announcing a proof of the infamous White Doctor opening (it is a draw) [6]. Endgame databases introduce perfect knowledge into the proof, replacing the traditional heuristic evaluations. The attempt to solve checkers would be greatly accelerated if the 11-piece databases could be computed.

Endgame databases need to be computed in a certain order, to preserve the dependencies inherent in the calculation. For example, the database for all positions with two kings need to be computed before the 3-king database can be computed. Ideally, one would like to extend this computation to include *all* the pieces on the board (e.g., the 32-piece database for chess!). However, there are several limitations to extending the database calculations as far as possible. As

H.J. van den Herik et al. (Eds.): ACG11, LNCS 4250, pp. 11–22, 2006.

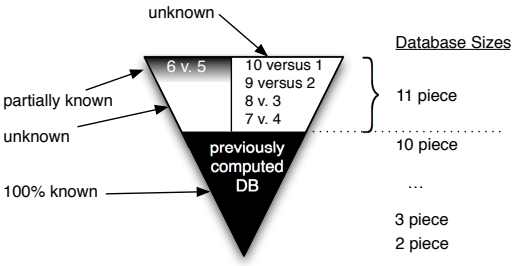

Fig. 1. Our goal: skip seldom used database and only compute more commonly used database positions

the number of pieces in the database increase, one encounters limits in data size (the databases become too large), computation time (there are more positions and longer winning sequences), memory (databases quickly become too big for RAM, resulting in costly I/O), and correctness (the computations need to be replicated to verify correctness).

A completed 11-piece checkers database would encompass 259 trillion positions, over six times larger than all existing checkers databases, which already took many years to build. The complete 6-piece vs. 5-piece subset (all combinations of checkers and kings) is a large portion of this, 118 trillion positions. Unfortunately, most of the computation is practically irrelevant for furthering the checkers proof. The first 11-piece database that must be computed by standard retrograde analysis, 6 kings vs. 5 kings, is never reached in tournament checkers games and never arises in our checkers proof trees. In effect, because of the computation dependencies, the least useful databases get computed before the most useful. The 11-piece database that would be most useful is the 6-checker vs. 5-checker database, but this is the very last calculation that gets done. This database has only 25 billion positions. Even the 6 vs. 5 endgames with a maximum of 1 king on the board come to 300 billion positions—easily doable using current technology. The problem is that we need to compute 118 trillion 6-piece vs. 5-piece positions before we get to the very small useful portion.

The goal of this research is illustrated in Fig. 1. Checkers databases that have been fully computed, such as the 2-piece through 10-pieces databases, are shaded in black. We would like to avoid computing the large and rarely used portions of the 11-piece database, and concentrate our efforts on a portion of the 6 vs. 5 database. Without calculating the full database this portion cannot be computed exactly, but even partial bounds on the values will be quite useful.

Instrumentation of the checkers prover used in [6] suggests that roughly 20% of non-database positions encountered in the search are 11-piece positions. Of these, over 90% are positions with 6 pieces vs. 5, with the total number of kings being 0 or 1. In the attempt to solve checkers, we do not need to prove the exact value of every position. In many cases a lower-bound or upper-bound on the value of a position will be adequate, for instance, to prove that a line of play

is at least a draw. Thus, databases that only provide partial information about a set of positions are still useful. Similar methods to what we present here for backing up information in end-game databases are explored in [2].

This paper makes the following contributions.

1. A win-loss-draw value ordering when unknown (partial information) values are present.
2. The implementation of a retrograde analysis algorithm that correctly propagates partial information values.
3. Experimental results and validation for the 10-piece partial information databases.

2 Partial Information Endgame Databases

In this section we provide some background (2.1), introduce database values for partial information databases (2.2), explain the building of partial information databases (2.3), and sketch a proof of correctness (2.4).

2.1 Background: Perfection Information Endgame Databases

Endgame databases traditionally contain three possible outcomes, win (W), loss (L) or tie/draw (T). A new database is computed by repeatedly iterating through all positions in the database, filling in values for positions that are known, and repeating until all values are known. Note that we do not store the number of moves needed to resolve each position, which may be needed to successfully play out some positions.

Initially, all positions in the database are marked with an extra value, unknown (U), which does not remain in the database past the retrograde analysis. The only way a position can be assigned a value is if one of its children is a win, or if all children have been resolved to a loss or a tie.

There is an exception for games like Checkers which have rules for draw-by-repetition. It is possible that there are cycles of states for which the best move is a draw-by-repetition. When doing retrograde analysis these positions will never be resolved exactly because every node within the repeated cycle of moves will always have one unknown child. But, once no other changes can be propagated through the database, remaining positions with unknown values are given the value T, which effectively handles these positions.

This process of retrograde analysis, for which pseudo-code is shown in Table 1, relies on three operators, *completed*, *decide*, and *max*. In perfect-information endgame databases a position is *completed* as long as the value is not U, *decide* is the identity function except on input of U which becomes T, and *max* uses the following ordering on values: $L < T < U < W$.

The high-level pseudo-code and definitions above omit many practical considerations (e.g., for reducing I/O). For example, from a practical standpoint, the database values are always from the perspective of the player that has the move in the given position. Therefore, we must always negate the value of a child

Table 1. Pseudo-code for standard retrograde analysis

```
Retrograde_Analysis(database)
  Initialize database to unknown, positionsUpdated = 1
  while (positionsUpdated != 0)
    positionsUpdated = 0
    for each position in database
      if (!completed(position))
        if (value(position) != max(children of position))
          positionsUpdated += 1
          value(position) = max(children of position)
  for each position in database
    value(position) = decide(position)
```

before applying the max operator. In the above example, the negate operator maps L to W and vice verse, but T and U remain the same. Also, if no move is available for the player to move (player has no piece or all pieces are blocked) the position is labeled a loss for the player. However, this high-level view provides an elegant conceptual model of retrograde analysis, and allows us introduce our new ideas simply by redefining a few operators.

2.2 Database Values for Partial Information Databases

If we want to build partial information bounds into an endgame database, we will need more values than just $W/L/T$ in the database. Instead, we need an upper and lower-bound on the value of a position. So, we write possible values of each position as two letters, the lower-bound followed by the upper-bound.

First, the standard values of $W/L/T$ will be replaced with $WW/LL/TT$, meaning that the upper-bound and lower-bound for these positions is identical. Then, we introduce three new values, LT, LW and TW. LT means the position's value has a lower-bound of loss and an upper-bound of tie. Definitions for LW and TW follow similarly. We will refer to the new three values as partial-information values, because they do not provide perfect information about the value of a position, only bounds.

Given partial-information values, the first thing we need is an ordering on the values which can be used by the max operation. This ordering is defined in Fig. 2. A directed edge between any two states A and B means that B is strictly greater than A. This property is transitive, so the max of any two states A and B is the first common node on all paths between A and WW and B and WW. For most states this is straightforward. For instance, $max(LT, TW) = TW$. But, this is only a partial ordering, because TT is neither greater than or less than LW. So $max(TT, LW) = TW$. Another way to view this is that the lower-bound on the max of two values is the max of the lower-bounds on these values, and the upper-bound on the max of two values is the max of the upper-bounds.

These six bounded values are the only values that will be found in a completed partial information database. But, they are not adequate in themselves, because we need a richer definition for unknown value(s) when we actually build a partial information database.

2.3 Building Partial Information Databases

When we begin to build a partial information database there are several places where partial information values will enter the database. As before, we can initialize positions to a value of *unknown* before we begin. Positions on the fringe of the database being built can have children that are found in either a previously computed database, or a database that we have not computed. For instance, if we do not compute the 2-king 11-piece databases and we have a move that makes the second king on the board, we will not find that position in a database. Such positions are given the value LW, because we have no information about their values, and we will never attempt to calculate them.

Given the new definition of *max* and previous definitions of *decide* and *completed*, it may seem that we have enough information to build partial-information databases. But, there is one additional case that arises, which is the interaction of draw-by-repetition positions with other unknown values.

Consider the four positions which are part of a draw-by-repetition cycle in part (a) of Fig. 3. In a regular database, the player to move at the highlighted node would prefer a draw-by-repetition to the loss available as an alternate move. So, the final value of this state after retrograde analysis will be T.

Now consider part (b) of Fig. 3, where the alternate value to the draw-by-repetition is completely unknown, LW. In this situation, the first player is guaranteed at least a tie because of the draw-by-repetition, but there is a chance that the alternate move will lead to a win. Thus, the actual value of the state is TW. That is, there is one possible move that will lead to a tie by repetition, and there is another move that might lead to a win. At the parent of this position (using a nega-max formulation) the bound is then LT, and similarly in the rest of the cycle.

But this will not actually happen unless we modify the values used by retrograde analysis. If we have a single value for draw-by-repetition which is not

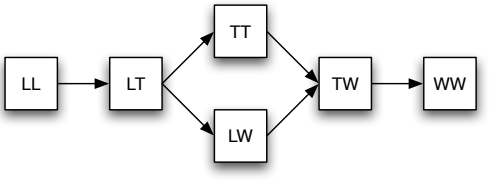

Fig. 2. The partial ordering of values used for database generation

resolved until after all passes through the database, none of the positions in this cycle will resolve. So, when all other positions have stopped updating, the *unknown* on the positions in the cycle will incorrectly be converted to TT. This process ignores the interactions between draw-by-repetition and positions that have partial information bounds.

Thus, in the same way that we introduced partial information bounds on the value of a state, we need to introduce new unknown values for use during retrograde analysis. These values are not bounds on the final value of a state, but instead represent the value a state would take if the retrograde analysis were to stop without further updates.

These new values are uLW, uTW, uTT and uLT. uTT is the same as the previous *unknown* value, meaning that a state is a draw-by-repetition. The additional values arise as a result of ways that uTT can combine with partial bounds in the game. We demonstrate this in Fig. 4.

In this figure we demonstrate how values are propagated through a draw-by-repetition loop. At the far left of this diagram we have the initial values in the database, with all values initialized to uTT. The only node that can update its value is the highlighted node, which is subsequently updated to uTW. In the next step, the parent of this state, which is highlighted, can now be updated to uLT. This process continues until no further updates are possible. If retrograde analysis ends at this point, all the unknown values will be converted to exact values.

Figure 4 shows how the values uTW and uLT can arise in the database. The other unknown value, uLW, is introduced when we have a state that has one child with a value of uLT and another child with the value LW.

We can now define the *completed*, *decide*, and *max* operators for partial information databases. A position is *completed* once it has a value of WW, TT, LL, LW, TW, or LT. We *decide* the final value of an unknown state by converting it to its corresponding known value. So, $decide(uLW) = LW$, and likewise for other unknown values. Finally, the maximization function is defined for all possible values by Table 2. Utilizing these new definitions, we can use

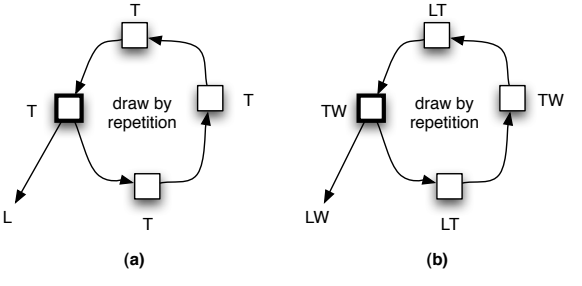

Fig. 3. Draw-by-repetition interactions

Table 2. The maximum of each possible combination of states that can be encountered in the process of building partial-information databases

max	WW	TT	LL	TW	LT	LW	uLW	uTW	uTT	uLT
WW	WW	WW	WW	WW	WW	WW	WW	WW	WW	WW
TT	WW	TT	TT	TW	TT	TW	uTW	uTW	uTT	uTT
LL	WW	TT	LL	TW	LT	LW	uLW	uTW	uTT	uLT
TW	WW	TW	TW	TW	TW	TW	uTW	uTW	uTW	uTW
LT	WW	TT	LT	TW	LT	LW	uLW	uTW	uTT	uLT
LW	WW	TW	LW	TW	LW	LW	uLW	uTW	uTW	uLW
uLW	WW	uTW	uLW	uTW	uLW	uLW	uLW	uTW	uTW	uLW
uTW	WW	uTW	uTW	uTW	uTW	uTW	uTW	uTW	uTW	uTW
uTT	WW	uTT	uTT	uTW	uTT	uTW	uTW	uTW	uTT	uTT
uLT	WW	uTT	uLT	uTW	uLT	uLW	uLW	uTW	uTT	uLT

the same methodology as shown in pseudo-code in Table 1 to compute partial information databases, except that the initialized *unknown* value is now uTT.

2.4 Proof of Correctness

In this section we sketch a proof of correctness; but many details are omitted for clarity. We first show that any positions which become *decided* during retrograde analysis must have correct bounds. Then, we show that draw-by-repetition values are computed correctly. Finally, we show that draw-by-repetition subsets of the database cannot improperly effect other parts of the database.

First, consider positions that will become *decided* during the process of retrograde analysis. This means that either all of the children of these positions are decided, or they have one child that leads to a win. If a position has one child that leads to a proven win, this will be a proven win no matter whether we are using partial information databases or perfect information databases. Similarly, if a position is resolved to exactly TT or LL during retrograde analysis, this value must have been calculated in the exact same manner as in a standard endgame database.

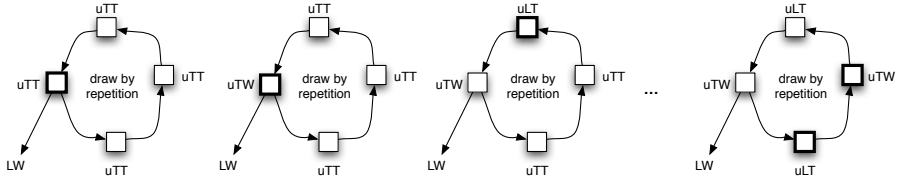

Fig. 4. The propagation of values combined with uTT

Now, consider positions that resolved to LW, LT or TW. There are two places that partial information bounds are introduced into the database. The first is when a child is found in a database that has not been computed, in which case it is given the value LW. This is the most general bound possible, so the actual value of this state must fall within these partial information bounds (loss to win). The second place we can get partial information bounds is from another partial information database, which we assume has already been calculated correctly. From Table 2 we can verify that the combination of LW, LT, and TW with any other *known* value correctly preserves upper- and lower-bounds on a particular state. Thus, any value that becomes decided during retrograde analysis is calculated correctly.

Next, consider positions involved in draw-by-repetition, which are not resolved until retrograde analysis completes. If they are part of a closed loop (all positions outside the loop lead to a loss or a tie) then these positions are handled no differently by retrograde analysis with partial information values then they would be handled by standard retrograde analysis.

In the case where a position near a draw-by-repetition loop has a partial information value that could possibly be a win, it will be incorporated into the bounds of positions in the draw-by-repetition loop, as in Fig. 4. Again, from Table 2 one can verify that there is no way to combine values in such a loop to eliminate the draw-by-repetition.

Finally, is it possible for unknown values from a draw-by-repetition to adversely effect other positions in the database, now that we have expanded the range of possible values that can occur in a draw-by-repetition loop? It is not possible because of the following observation: Any values which are not *known* besides uTT must trace to at least one position where there is a uTT value for one child and a partial information value (TW or LW) at the other child. The value of such states then will either be a draw-by-repetition or the result of the partial information value from the other child—but there is not enough information in the analysis to resolve which. In either case lower- and upper-bounds for such states will be correct. Presenting a formal proof of this would be quite detailed, but simply relies again on the values in Table 2.

While we have not shown in a completely formal manner that our partial information databases are correct, these description should give the reader a feel of the general correctness retrograde analysis using both partial information values and extended values for unknown positions.

3 Results

The partial information database algorithm has been built into the CHINOOK database construction program [7]. To validate the algorithm we constructed the 5-checker vs. 5-checker subset of the 10-piece databases. For this computation, we removed all the previously computed 10-piece databases. In other words, the database program only had access to the 2-piece through 9-piece databases. After constructing the partial information 10-piece databases we verified that

Table 3. Max table for partial databases, 5 vs. 5 checkers. Each entry is the percentage of resolved positions.

	7	6	5	4	3	2
7	16	25	33	37	44	70
6	26	38	49	56	68	84
5	35	48	62	71	81	88
4	35	53	66	80	86	88
3	40	63	79	86	91	95
2	62	81	84	91	97	100

all bounds were consistent with the actual values in the previously computed 10-piece databases.

Table 3 shows some results for the 5-checker vs. 5-checker database. The database is broken into smaller pieces based on the leading rank of the checker for each side. For example, the 7/7 table entry has both Black and White having a checker on the seventh rank. It is not possible to have a checker on the eighth rank (it already has become a king). The 6/4 entry has all of White's checkers on or before the sixth rank with at least one on the sixth rank, while all of Black's checkers are on the fourth rank or before (with at least one on the fourth rank). Note that there is no result for rank 1—since each side has 5 checkers, it is not possible to have them all on the first rank.

The table shows the fraction of (non-capture) positions that have been resolved. A position proven to be a win, loss or tie counts as one point, while a lower- or upper-bound of a tie counts as a half point. The total database score divided by the number of positions in the computation gives the score. For example, the 7/7 entry shows that when both sides have a leading checker on the seventh rank, then the database program was able to resolve 16% of the values. The 7/7 entry is low for two reasons.

1. Most positions in the 5 vs. 5 databases are drawn. This makes it difficult for the database construction algorithm, since a draw cannot be proven until the values (or bounds on values) of all children of a position are known.
2. This database computation has both sides "on the boundary". When the leading checker advances, it becomes a king—and there are no database results for that position. Hence many of the 7/7 computation are unresolved because they lead to unresolved (partial information) positions.

As one would expect, as one moves away from the boundary, a higher percent of positions are resolved. This is easily explained since when the leading checker in these databases advances, it moves into a database that has already been computed and has (partial) results. It is particularly noteworthy that 100% of the positions with both players leading checkers at the second rank are resolved correctly and completely. So, from one perspective, this portion of the databases is no longer a partial information database, as all values are exact.

Also of interest is that the database construction algorithm took as many as 36 passes over the data to compute the values in Table 3. This means that some of the values represent proven wins in 36 ply. If ever one of these positions comes up in the checkers proof, the perfect value from the partial information databases will replace 36-ply of search!

The above results point to an obvious way to improve the results. Since the boundary introduces the unknown values, an effort should be made to resolve as many boundary positions as possible. For example, one could do a small (say, 5-ply) search for each boundary position. This would increase the number of boundary positions that have useful data, either proven or partial values. These values, of course, get propagated into the database calculation, which would increase the percentage of overall positions resolved. We have not yet incorporated this into our program, but it is a point of future work.

So, how will this impact the 11-piece databases? The 11-piece databases are less likely to have draws—one side is up a checker, so we would expect the side with more pieces to win most positions. Thus, to provide additional insight into how the 11-piece database results will look, we also computed partial 4 vs. 5-checker 9-piece databases.

Table 4. Max table for 9 piece databases, 4 vs. 5 checkers

	7	6	5	4	3	2
7	7	10	13	15	25	74
6	46	55	57	61	73	98
5	64	71	78	80	91	97
4	74	81	87	93	96	99
3	82	88	94	97	98	100
2	86	93	97	99	99	100
1	87	94	95	99	100	100

Table 4 shows the 9-piece results when the player to move has four checkers against an opponent with five checkers. Table 5 shows the similar table when the player to move has five checkers against an opponent with four.

These numbers are best explained using specific examples from the table. In the first row, second column (7/6) of Table 4 we find the entry 10. This means that only 10% of the positions have been resolved when the player with 4 checkers is to move and has his most advanced checker on the seventh rank. But, in Table 5 in the first row, second column we find the entry 66, meaning 66% of positions have been resolved when the player with 5 checkers is to move with his most advanced checker on the seventh rank. So, each entry is the percent of resolved positions relative to the most advanced checker on the board, which player is to move next, and the number of checkers that each player has.

The first item of interest is that the ratio of resolved positions is noticeably higher than in the 10-piece database, which we expected, because of the uneven material value on the board. We see also that the stronger side can generally

Table 5. Max table for 9 piece databases, 5 vs. 4 checkers

	7	6	5	4	3	2	1
7	44	66	78	85	92	91	90
6	51	73	84	91	95	96	95
5	51	79	90	95	98	99	99
4	52	82	94	98	99	100	100
3	57	90	98	99	100	100	100
2	86	99	100	100	100	100	100

resolve a larger percentage of positions. The only exceptions to this is where the weaker side is just about to promote a checker to a king, because the databases containing the result of such a move has not been computed.

Also of interest is the asymmetric nature of the table. The entry in the sixth row and seventh (6/7) column for the weak player (46%) entry is much stronger for the weak side than the 7/6 entry (10%). This occurs because from the 6/7 entry the weak player can move his most advanced checker forward into the 7/7 database. Because it will then be the other player's turn, many of these positions (44%) will already be resolved.

4 Conclusions

The goal of this research has been to compute the important parts of the 11-piece checkers endgame databases. At the time of this writing the 6-piece vs. 5-piece endgames with one or fewer kings are currently being computed, followed by the 6-checker vs. 6-checker database. When complete, these will be added to the CHINOOK databases and used in the checkers prover. Although this computation may only cause roughly 10% of the positions examined by the prover to be resolved by the new databases, our experience is that in many cases this will result in many ply being eliminated from some of the sub-proofs. This will result in a substantial reduction in the computational effort needed to solve checkers.

We also plan to compute some 7-piece vs. 4-piece and 8-piece vs. 3-piece partial information endgames. Although these lopsided endgames are almost always won for the stronger side, having the proven database value will be very useful. Sadly, any checkers solver must always explore the path of maximum resistance, since it must prefer an unknown value over a loss or draw. Consequently, the prover tends to make moves to postpone resolving the value of a position—an "unknown" position down 5 pieces (it could be a win!) is preferred over a known draw.

In conclusion, partial information databases are not a complete set of resolved values, but they still provide useful information. They allow us to probe further forward in the game search space, allowing us to uncover new secrets of the game—without having to pay the full price of computational resources and time.

References

1. R. Gasser. Solving Nine Men's Morris. *Computational Intelligence*, 12:24–41, 1996.
2. T.R. Lincke. *Exploring the Computational Limits of Large Exhaustive Search Problems*. Ph.D. thesis, Swiss Federal Institute of Technology, 2002.
3. John Nunn. *Secrets of Rook Endings*. Gambit Books, B.T. Batsford Ltd., London, 1999.
4. J. Romein and H. Bal. Solving the Game of Awari Using Parallel Retrograde Analysis. *IEEE Computer*, 36(10):26–33, 2003.
5. J. Schaeffer. *One Jump Ahead*. Springer-Verlag, New York, 1997.
6. J. Schaeffer, Y. Bjornsson, N. Burch, A. Kishimoto, M. Müller, R. Lake, P. Lu, and S. Sutphen. Solving Checkers. In *International Joint Conference on Artificial Intelligence (IJCAI)*, pages 292–297, 2005.
7. J. Schaeffer, Y. Bjornsson, Neil Burch, Robert Lake, Paul Lu, and Steve Sutphen. Building the 10-piece Checkers Endgame Databases. In *10th Advances in Computer Games (ACG10), Many Games, Many Challenges* (eds. H. J. van den Herik, H. Iida, and E. A. Heinz), Kluwer Academic Publishers, Boston, pages 193–210, 2004.
8. T. Ströhlein. *Untersuchungen uber Kombinatorische Spiele*. Dissertation, Fakultät für Allgemeine Wissenschaften der Technischen Hochschule München, 1970.
9. K. Thompson. Retrograde Analysis of Certain Endgames. *ICCA Journal*, 9(3):131–139, 1986.