Alpha-Beta Pruning Verified

Tobias Nipkow
Department of Computer Science, Technical University of Munich, Germany

Abstract

Alpha-beta pruning is an efficient search strategy for two-player game trees. It was invented in the late 1950s and is at the heart of most implementations of combinatorial game playing programs. We have formalized and verified a number of variations of alpha-beta pruning, in particular fail-hard and fail-soft, and valuations into linear orders, distributive lattices and domains with negative values.

2012 ACM Subject Classification Software and its engineering → Formal software verification; Theory of computation → Algorithmic game theory

Keywords and phrases Verification, Algorithmic Game Theory, Isabelle

Digital Object Identifier 10.4230/LIPIcs.ITP.2024.1

Category Invited Talk

1 Introduction

In this article, we sketch the results of our formalization [9] of a number of variants of alpha-beta pruning in the proof assistant Isabelle [12, 11]. A more detailed presentation can be found in a (forthcoming) book [10]. We restrict ourselves to functional correctness and do not cover quantitative results

We consider two-player games with the usual notion of game trees. Let $val(t)$ denote the *value* of a game tree t. Alpha-beta pruning is an efficient strategy for determining the value of a tree. In this extended abstract we assume that the reader is familiar with the basic idea underlying alpha-beta pruning. We do not show any code but all our variants f of alpha-beta pruning follow the calling convention $f\ a\ b\ t$ where (a, b) is the search window and t the game tree.

2 Linear Orders

In the literature it is often assumed that values are numbers extended with $-\infty$ and ∞. In this section we merely assume that the type of values is a bounded linear order with $\bot$ and $\top$.

The first thorough mathematical analysis of alpha-beta pruning is due to Knuth and Moore [6]. They employ the relation $x \cong y\ (a, b)$ defined as follows:

$$(y \le a \longrightarrow x \le a) \wedge (a < y < b \longrightarrow x = y) \wedge (y \ge b \longrightarrow x \ge b)$$

The notation $x \cong y\ (a, b)$ is ours and emphasizes the fact that the relation is symmetric if $a < b$. Knuth and Moore consider the so-called *fail-hard* variant of alpha-beta pruning, which we denote by $hard\ a\ b\ t$, and prove that $val(t) \cong hard\ a\ b\ t\ (a, b)$ which implies overall correctness: $hard\ \bot\ \top\ t = val(t)$.

Fishburn [3] suggested the *fail-soft* variant and states that it searches the same part of the tree as fail-hard and satisfies the relation $soft\ a\ b\ t \le val(t)\ (a, b)$ (the notation is again ours) where $y \le x\ (a, b)$ is defined as follows:

$$(y \le a \longrightarrow x \le y) \wedge (a < y < b \longrightarrow x = y) \wedge (y \ge b \longrightarrow x \ge y)$$

We verified both that the search space is the same and that $soft\ a\ b\ t \leq val(t)\ (a, b)$. Because $y \leq x\ (a, b)$ implies $x \cong y\ (a, b)$ (but not the other way around), it appears that *soft* satisfies a stronger property than *hard*. However, we could also prove $hard\ a\ b\ t \leq val(t)\ (a, b)$, which begs the question in what sense *soft* is better than *hard*. The answer: we could also prove $hard\ a\ b\ t \leq soft\ a\ b\ t\ (a, b)$. In summary: *soft* is as least as close to *val* as *hard* (and sometimes closer).

2.1 Negative Values

In the literature, it is often assumed that values are positive and negative numbers and that the value v for one player is $-v$ for the other player. In this situation the definition of *val* and of alpha-beta pruning can be streamlined: instead of having one function for each player, now one function in total is needed. We found that this works for arbitrary linear orders with unary negation under these two assumptions:

$$-min\ x\ y = max\ (-x)(-y) \qquad -(-x) = x$$

3 Lattices

Bird and Hughes [1] were the first to generalize alpha-beta pruning from linear orders to lattices. The generalization of the code, including game tree evaluation, is easy: simply replace *min* and *max* by $\sqcap$ and $\sqcup$. It turns out that this version of alpha-beta pruning works for bounded distributive lattices (Bird and Hughes confusingly talk about Boolean algebra). In order to prove this, Bird and Hughes invent the following relation (the notation $\simeq$ is ours)

$$x \simeq y\ (a, b) \iff a \sqcup (x \sqcap b) = a \sqcup (y \sqcap b)$$

and prove

$$bh\ a\ b\ t \simeq val(t)\ (a, b)$$

where *bh* is their variation of fail-hard. Top-level correctness $bh\ \bot\ \top\ t = val(t)$ follows immediately.

For linear orders (but not for distributive lattices) $\cong$ and $\simeq$ coincide if $a < b$:

$$x \cong y\ (a, b) \iff x \simeq y\ (a, b)$$

It is also possible to rephrase Fishburn's predicate $\leq$ with *min* and *max* (for linear orders) if $a < b$:

$$y \leq x\ (a, b) \iff min\ x\ b \leq y \leq max\ x\ a$$

We rephrase the r.h.s. for disttributive lattices and define

$$y \sqsubseteq x\ (a, b) \iff x \sqcap b \leq y \leq x \sqcup a$$

It coincides with $\leq$ for linear orders (but not for distributive lattices) if $a < b$

$$y \leq x\ (a, b) \iff y \sqsubseteq x\ (a, b)$$

In distributive lattices $\sqsubseteq$ implies $\simeq$

$$y \sqsubseteq x\ (a, b) \implies x \simeq y\ (a, b)$$

but the opposite direction does not hold.

Fail-hard and fail-soft carry over to distributive lattices (mutadis mutandis) and satisfy the stronger $\sqsubseteq$

$$f\ a\ b\ t \sqsubseteq val(t)\ (a, b)$$

where f can be fail-hard, fail-soft, and *bh* (Bird and Hughes' version),

3.1 Negative Values

The same extension to negative values that we sketched in Section 2.1 also works for distributive lattices. Now we assume that unary negation satisfies

$$-(x \sqcap y) = (-x) \sqcup (-y) \qquad -(-x) = x$$

The result is a so-called *de Morgan algebra* [8]. Again, the definitions of *val* and the variants of alpha-beta pruning can be streamlined as sketched before.

4 Related Work

Ginsburg and Jaffray [4] rediscover (independently of Bird and Hughes) that pruning also works for distributive lattices but stop short of formulating and proving an actual algorithm. Li *et al.* [7] build on the work of Ginsburg and Jaffray, formulate and prove an actual algorithm correct (rediscovering the correctness notion $\simeq$ by Bird and Hughes) and extend the algorithm to also cache and reuse earlier calls, an aspect not covered above.

In our formalization of variants of alpha-beta pruning we also considered the program that Hughes (in a much cited paper [5]) derived from a specification of *val*. Because his program looks simpler than alpha-beta pruning, we suspected that it may not prune enough. Hughes (private communication) used Haskell's Quickcheck [2] to compare the search space of his program with that of our implementation (which is the canonical alpha-beta algorithm) and confirmed our suspicion.

5 Conclusion

We have formally verified a number of variants of alpha-beta pruning, both for linear orders and distributive lattices, have clarified the relationship between different correctness notions, have expressed precisely (and proved) in what sense fail-soft is better than fail-hard, and discovered a "suboptimal" version of alpha-beta pruning in a much cited paper.

References

1 Richard S. Bird and John Hughes. The alpha-beta algorithm: An exercise in program transformation. *Information Processing Letters*, 24(1):53–57, 1987. doi:10.1016/0020-0190(87)90198-0.

2 Koen Claessen and John Hughes. Quickcheck: a lightweight tool for random testing of Haskell programs. In Martin Odersky and Philip Wadler, editors, *International Conference on Functional Programming (ICFP '00)*, pages 268–279. ACM, 2000. doi:10.1145/351240.351266.

3 John P. Fishburn. Another optimization of alpha-beta search. *SIGART Newsl.*, 84:37–38, 1983. doi:10.1145/1056623.1056628.

4 Matthew L. Ginsberg and Alan Jaffray. Alpha-beta pruning under partial orders. In Richard J. Nowakowski, editor, *More Games of No Chance*, volume 42 of *MSRI Publications*, pages 37–48. Cambridge University Press, 2002. URL: http://library.msri.org/books/Book42/files/ginsberg.pdf.

5 J. Hughes. Why Functional Programming Matters. *The Computer Journal*, 32(2):98–107, 1989. doi:10.1093/comjnl/32.2.98.

6 Donald E. Knuth and Ronald W. Moore. An analysis of alpha-beta pruning. *Artif. Intell.*, 6(4):293–326, 1975. doi:10.1016/0004-3702(75)90019-3.

7 Junkang Li, Bruno Zanuttini, Tristan Cazenave, and Véronique Ventos. Generalisation of alpha-beta search for AND-OR graphs with partially ordered values. In Luc De Raedt, editor, *International Joint Conference on Artificial Intelligence, IJCAI 2022*, pages 4769–4775. ijcai.org, 2022. doi:10.24963/ijcai.2022/661.

8 Gr. C. Moisil. Recherches sur l'algèbre de la logique. *Annales scientifiques de l'Université de Jassy*, 122:1–118, 1936.

9 Tobias Nipkow. Alpha-beta pruning. *Archive of Formal Proofs*, June 2024. , Formal proof development. URL: https://isa-afp.org/entries/Alpha_Beta_Pruning.html.

10 Tobias Nipkow, editor. *Functional Data Structures and Algorithms. A Proof Assistant Approach.* ACM Books. self-published, 2024. URL: https://functional-algorithms-verified.org/.

11 Tobias Nipkow and Gerwin Klein. *Concrete Semantics with Isabelle/HOL.* Springer, 2014. URL: http://concrete-semantics.org.

12 Tobias Nipkow, Lawrence C. Paulson, and Markus Wenzel. *Isabelle/HOL — A Proof Assistant for Higher-Order Logic*, volume 2283 of *LNCS*. Springer, 2002.