arXiv:2611.32320v1 [cs.AI] 24 Nov 2026

通过单子延续实现阿尔法-贝塔剪枝

Martín Hötzel Escardó[1] 和 Paulo Oliva[2]

[1] 计算机科学系 伯明翰大学
m.escardo@bham.ac.uk[2] 电子工程与计算机科学学院 伦敦玛丽女王大学 p.oliva@qmul.ac.uk

摘要。 阿尔法-贝塔剪枝是博弈人工智能中的一项基础技术，它通过避免探索不会影响最优决策的分支，实现高效的极小极大搜索。我们提出了一种基于延续单子与读取器单子组合的阿尔法-贝塔剪枝单子化表述。不同于以往将阿尔法-贝塔界限编码到可用行动类型中的方法，我们的构造将这些界限记录在单子结果中，同时保持走法类型不变。我们进一步表明，通过扩展结果类型以包含相应的着法序列，普通延续单子不仅足以计算最优结果，也足以计算最优对局。这为处理阿尔法-贝塔搜索中选择单子相关低效问题而提出的延续单子推广提供了一种替代方案。我们通过井字棋的实现展示了这一方法：与穷举搜索相比，在计算最优对局时，所评估的对局序列数量减少了39倍。最后，我们通过证明单子化阿尔法-贝塔搜索计算出的最优结果和最优对局与穷举极小极大搜索相同，正式确立了该构造的正确性。我们的结果表明，人工智能中的基础算法可以借助标准的单子抽象加以表达和验证，并为进一步发展组合式对抗搜索方法奠定基础。

关键词： 极小极大搜索 · 阿尔法-贝塔剪枝 · 读取器单子 · 选择单子 · 延续单子 · 量词乘积 · 单子编程 · Agda · Haskell

ACM CCS： 博弈树搜索 · 函数式语言 · 范畴语义 · 软件形式化验证 · 类型论

MSC 2020： 68T20（主要）；91A18、91A05、68N18、18C15、68V15、03B38、68Q55、91A46（次要）

1 引言

阿尔法-贝塔剪枝是人工智能领域的基础搜索技术之一，尤其适用于对抗性博弈。最早对该方法作出明确描述的文献之一，是 Hart 和 Edwards 于 1961年在

麻省理工学院人工智能项目备忘录树剪枝（*TP*）算法 [7]。随后，该方法被称为阿尔法-贝塔启发式，由此形成了如今普遍使用的术语。阿尔法-贝塔剪枝通过维护最大化玩家和最小化玩家可获得值的界，并终止对那些不会影响最终决策的分支的探索，从而改进极小极大搜索。该基本算法在保留极小极大搜索结果的同时，避免评估已证明无关的位置。后来，Knuth 和 Moore [10], 对其理论基础进行了更为正式的阐述，确立了阿尔法-贝塔剪枝作为博弈树搜索核心技术的地位。

我们已经在 [2,3] 中展示，最小最大类搜索可以使用单子编程来执行：通过延续单子计算最优结果，并通过选择单子计算最优对局和策略（另见 [1,4,5]）。由于延续单子的类型

$$K_R X \equiv (X \to R) \to R$$

推广了布尔量词的类型 $\exists, \forall$: $(X \to \mathbb{B}) \to \mathbb{B}$，因此我们将 $K_R X$ 的元素称为量词，并用它们来为所谓的高阶博弈中的玩家建模。在 $K_R X$ 中，我们将 X 视为玩家的可用行动 类型，而将 R 视为博弈结果类型。

在最近的一项工作中，Hartmann [8]提出了这一构造的扩展，以适应 alpha-beta 类优化。该工作建立在 Hartmann 等人关于延续单子一般化的工作之上 [9]。在 Hartmann [8], 中， α 和 β 的值被“编码”到走法的类型中，因此扩展后的走法类型变为 $X \times \mathbb{N} \times \mathbb{N}$，其中 X 是走法的实际类型，而 $\mathbb{N} \times \mathbb{N}$ 记录 alpha 和 beta 的当前值。

本文提出一种替代方法，通过结合延续单子与读取器单子来实现单子式阿尔法-贝塔优化。我们的主要贡献如下：

(1) 不同于 [8], 中考虑类型为 $X \times \mathbb{N} \times \mathbb{N}$ 的走法，我们将证明（第 3 节） alpha 和 beta 的值可以“存储”在单子结果上。形式化地，设 T 是一个 单子。我们将类型为 $K_{TR} X$ 的量词称为 单子量词 或 单子式延续算子，因为结果类型 TR 本身就是单子的。我们将证明，取 T 为 读取器单子

$$TX = R \times R \to X$$

我们便可以使用 TR 中的单子结果来跟踪 alpha-beta 的值。

(2) 在第 4 节中，我们给出形式化证明，证明使用读取器单子的单子延续确实能计算出与穷举搜索相同的最优结果（推论 1）。

(3) Hartmann [8]和 Hartmann 等人 [9] 正确指出，在将 alpha 和 beta 的额外信息沿着搜索树向下传递时，选择单子 似乎会一遍又一遍地重新计算子博弈，因为惰性

求值阻碍了高效选择搜索。他们建议对延续单子进行推广（使其在结果类型上具有多态性），从而可以用它来计算最优走法。我们将在此展示，只需扩展结果类型 R，使其包含导向该结果的博弈过程（见第5节），就能直接使用通常的延续单子来实现这些目的。

（4）最后，在第6节中，我们将该方法及其正确性证明扩展到依赖历史的博弈，其中每一轮可用的走法取决于此前已经进行的走法 [3,5]。随后，我们应用α-β剪枝搜索来计算井字棋的最优结果和博弈过程，并衡量使用剪枝机制所带来的性能提升。

第4节和第5节中的定义、结果与示例，以及第2节和第3节示例的结果，已经在证明助手Agda中形式化，并收录于TypeTopology库的Games.AlphaBeta.index模块及其导入的模块中 [6]。该形式化是在第6.1节所述的依赖历史的博弈这一一般框架下开发的。其Games.AlphaBeta.Article模块逐项对应本文内容，而其Games.AlphaBeta.Summary模块则陈述了定理1和推论1和2，以及这些定理和推论的陈述所使用的每个定义，因此无需阅读证明即可将这些陈述与本文进行比较。

2 使用延续单子进行穷举搜索

我们首先概述如何使用延续单子，在交替使用最大值量词和最小值量词的情况下，对博弈树进行穷举搜索（参见 [2]）。我们先定义最小值量词和最大值量词，然后给出一个示例博弈树，最后将这些（局部）量词组合成适用于整个博弈的（全局）量词。

2.1 最小值量词和最大值量词

作为贯穿全文的示例，我们将整数用作走法，用整数列表表示可用走法列表和对局序列，并将整数用作结果：

```
导入 K

类型 走法  =整数类型 路
径  = [走法]类型 走法
列表 = [整数]类型 R  =
整数
```

然后，我们可以定义作用于这些类型的最小值和最大值量词（玩家）：

```
   -- 最大化玩家量词phiMax:: 走法 ->
K R 走法
```

```
phiMax xs  =K (\p -> 最大值 $ map p xs)

    -- 最小化玩家量词phiMin::走法-> K R 走法
phiMin xs  =K (\p -> 最小值 $ map p xs)
```

2.2 一个示例博弈树

为了说明并测试搜索过程，我们考虑以下简单的博弈树（图1），该博弈树基于维基百科有关阿尔法-贝塔剪枝的文章中给出的示例 [11]:

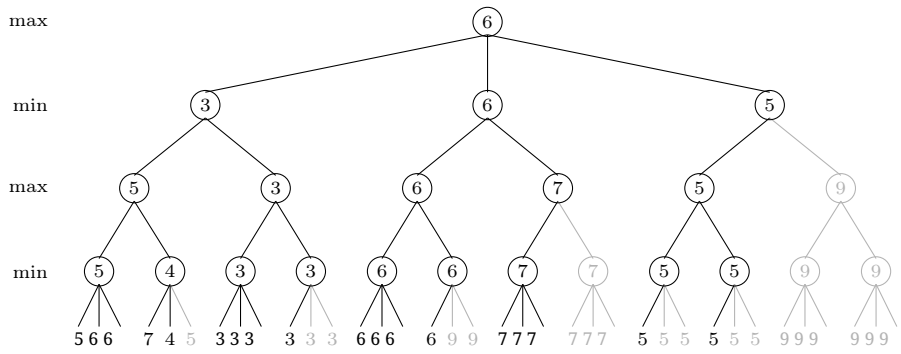


图1. 示例博弈树。每个节点都显示其最优结果。灰色的结果是第 3 节中的 α-β搜索未评估的结果。

```
导入 Debug.Trace

    -- 示例，基于：--
https://en.wikipedia.org/w/index.php?oldid=1380302874-- 示例
中博弈树的结果函数q' :: 路径-> Rq'  [0,0,0,0] = 5q'
 [0,0,0,_] = 6q'  [0,0,1,0] = 7q'  [0,0,1,1] = 4q'
 [0,0,1,2] = 5 -- 可以剪枝q'  [0,1,_,_] = 3 -- 可以部分剪枝
q'  [1,0,0,_] = 6q'  [1,0,1,0] = 6q'  [1,0,1,_] = 9 -- 可
以剪枝
```

```
q'  [1,1,_,_] = 7 -- 可以部分剪枝q'
 [2,0,_,_] = 5 -- 可以部分剪枝q'
 [2,_,_,_] = 9 -- 可以剪枝

-- 添加跟踪以监控已求值分支
-- 以及求值顺序
q :: 路径-> R
q xs  =跟踪 (show xs  ++ " -> "  ++ show (q' xs)) (q' xs)
```

2.3 组合量词

接着可以将这些量词“序列化”，从而得到一个作用于博弈树中对局序列（路径）的全局量词:

```
    -- 极大极小实验博弈树的博弈量词phis ::  [KR Move]phis  = [
phiMax  [0,1,2], phiMin  [0,1],phiMax  [0,1], phiMin  [0,1,2]
                                                                ]

    -- 将量词序列化以得到'全局' 博弈量词bigPhi :: KR PathbigPhi =
```

序列 phis

注*1*（*Haskell* 的“*sequence*”和量词乘积）。对于单子 M，Haskell 的“sequence”操作可以将一序列单子效应组合成单一效应，从而“生成”一个列表:

$$\text{sequence} : (MX)^n \to MX^n.$$

我们已在 [2] 中说明，延续单子的这一特定实例（即 $M = K_R$ 时）可以计算一般高阶博弈中的最优结果（另见 [3,4]）。如上所述，我们将延续单子的元素称为 K_R 量词，并将上述构造称为量词乘积，记作

$$\bigotimes : (K_R X)^n \to K_R X^n$$

其定义为

$$(\bigotimes[])(q) := q([])$$
$$(\bigotimes(\phi_0 : \phi))(q) := \phi_0(\lambda x.(\bigotimes \phi)(\lambda t.q(x \cdot t))),$$

其中，$x \cdot t$ 将元素 x 添加到序列 t的前面。

然后，我们可以对（可追踪的）结果函数应用全局博弈量词，从而获得博弈的 最优结果 ，以及实验结果函数 q 的求值序列:

```
最优_结果:: R最优_结果 = 博弈量词
bigPhi q

-- 测试运行（考虑所有路径）：-- ghci> 最优_
结果 -- [0,0,0,0]-> 5-- [0,0,0,1]-> 6-- ... --
 [2,1,1,1]-> 9-- [2,1,1,2]-> 9-- 6
```

在这种情况下，我们看到最大最小博弈的最优结果计算正确，但遍历了全部 36 条路径。

3 通过 Reader 单子添加阿尔法-贝塔剪枝

在 Hartmann [8], 的工作基础上，我们现在展示如何利用标准延续单子 $K_R X \equiv (X \to R) \to R$ 实现 Alpha-Beta优化，方法是将结果类型设为 TR，其中 T 是读取器单子，因此我们处理的是 $K_{TR}X$。而在 [8], 中，alpha 和 beta 的值被编码在走法的类型中。在我们的例子中，搜索中的走法类型对应于博弈中的实际走法，但单子结果则捕获了更新 alpha 和 beta 值所产生的副作用。我们先定义经过优化的最小值和最大值量词，然后将它们组合起来，最后在示例博弈树上运行 α-β搜索。

3.1 优化后的最小值和最大值量词

我们先导入一些有用的函数、标准的延续单子（见附录 B）以及读取器单子（来自模块 Control.Monad.Trans.Reader）。走法、路径和结果（R）的类型与上文相同。

```
导入 K导入
Control.Monad.Trans.Reader导入
Debug.Trace

导入 Utils

-- 用作 -inf 和 +inf inf :: 整数inf
 = 1000


    -- 读取器单子类型 T =
Reader (Int, Int)
```

然后，我们可以为最大化玩家和最小化玩家定义量词，将 (α，β) 的值纳入考量。最大化玩家会在当前考察的值大于或等于 β 时停止，并更新 α （最高保证结果）：

```
phiMax :: Moves -> (Move -> T R) -> T R
phiMax xs p = do (_, b) <-ask rs <- l xs return $
maximum $ takeUntil (>= b) rs where l ::
Moves-> T [R]l [] =return []l (x:xs) =do (a,
b) <-ask r <- p x rs <- local(const (max a r, b)) (l
xs) return (r :rs)
```

另一方面，最小化玩家量词会在当前考察的值小于或等于 α时停止，并更新 β （最低保证结果）：

```
phiMin :: Moves -> (Move -> T R) -> T R phi
Min xs p = do (a, _) <-ask rs <- l xs return $ 最小
值 $ takeUntil (<= a) rs where l :: Moves-> T [R]
l [] =return []l (x:xs) =do (a, b) <-ask r <- p x
rs <- local(const (a, min b r)) (l xs) return (r :rs)
```

3.2 组合优化后的量词

与第 2.3 节一样，我们在此也使用单子“序列”，为整个博弈构造一个单一的（全局）量词，不过现在它作用于单子结果 TR:

```
-- 最大-最小博弈树的博弈量词phisT :: [K(T R)
走法]phisT = [K(phiMax [0,1,2]), K (phiMin
[0,1]),K(phiMax [0,1]), K (phiMin [0,1,2])]
```

```
    -- 用于获得'全局' 博弈量词bigPhi :: K(T R) 路径bigPhi =序
列 phisT
```

3.3 示例博弈树的 α-β搜索

然后，我们可以在第2.2节的示例博弈中测试这一点。我们需要将结果函数 q “单子化”：

```
-- 将其单子化并添加跟踪，以监控已求值分支及其求值顺序

q :: 路径-> T Rq xs  =返回 $ 令 r  = q' xs
于跟踪（显示 xs  ++ " -> "  ++ 显示 r）r


最优_结果:: R
最优_结果 = (runReader $ 量词 bigPhi q) (-inf, inf)

-- 输出: -- ghci> 最优_结果 --
 [0,0,0,0]-> 5--  [0,0,0,1]-
> 6--  [0,0,0,2]-> 6--
 [0,0,1,0]-> 7--  [0,0,1,1]-
> 4--  [0,1,0,0]-> 3--
 [0,1,0,1]-> 3--  [0,1,0,2]-
> 3--  [0,1,1,0]-> 3--
 [1,0,0,0]-> 6--  [1,0,0,1]-
> 6--  [1,0,0,2]-> 6--
 [1,0,1,0]-> 6--  [1,1,0,0]-
> 7--  [1,1,0,1]-> 7--
 [1,1,0,2]-> 7--  [2,0,0,0]-
> 5--  [2,0,1,0]-> 5-- 6
```

我们可以看到，无关分支确实被剪枝了，在搜索过程中只考虑了 18 个关键对局。

4 Alpha-Beta 优化的正确性

现在，我们给出一个形式化证明，证明第3节中通过单子延续实现的 Alpha-Beta优化确实（总是）计算出与原始（未经优化的）构造相同的最优博弈结果。

我们通过映射 val: $R \to V$ 到具有可判定严格弱序 $<$ 的值类型 V 来比较结果；也就是说，这是一个可判定关系，具有反自反性、传递性和余传递性，其中余传递性意味着 $a < b$ 蕴含 $a < c$或 $c < b$。我们用 $a \leq b$ 表示 $\neg(b < a)$。可判定线性序（例如整数上的线性序）就是一类例子。一般而言，严格弱序是对相等价类构成的线性序；当 a 和 b 时，二者的值相等，也就是说，$a < b$ 和 $b < a$均不成立。界 α 和 β 都是值。在第2–4节中，$V = R$ 且 val 是恒等映射；在第5节中，$R = V \times$Path，且 val 是第一投影。结果之间以及结果与值之间的比较，都是对其值进行比较；而 max(α, r) 等于 max(α, val(r))。非空结果列表的最大值是其中值最大的首个元素，由附录A中的 pickMax 计算；最小值同理。在每个非终局位置，可用行动列表都非空。

令 $[X]^+$ 表示由 X构成的非空列表，并记

$$\phi_{\min}, \phi_{\max} \colon [X]^+ \to (X \to R) \to R$$

为标准量词 min 和 max（第2.1节），并记

$$\phi^*_{\min}, \phi^*_{\max} \colon [X]^+ \to (X \to TR) \to TR$$

为执行 α-β 剪枝的单子化版本，其中 T由 $TX = V \times V \to X$给出，即读取器单子（第 3.1 节）。我们还将用 ϕ 或 ϕ^*作为相应量词中 min 或 max 版本的占位符。

给定博弈 $G = (\phi, q)$，其中 ϕ是由 n 个量词构成的序列，且 $q\colon X^n \to R$ 是结果函数，则最优博弈结果可计算为（参见注1）

$$\mathrm{out}(G) := \left(\bigotimes \phi\right)(q).$$

单子博弈 $G^* = (\phi^*, \eta \circ q)$也同理，其中 $\eta(r)(\alpha, \beta) = r$是读取器单子的单位。给定 (α, β) 的具体值，我们也可以得到博弈结果：

$$\mathrm{out}^*_{\alpha,\beta}(G) := \left(\bigotimes \phi^*\right)(\eta \circ q)(\alpha, \beta),$$

我们将其称为单子结果。我们将证明，最优博弈结果确实与单子结果一致。由于我们将通过对博弈树进行归纳法来证明这一点，因此，对于初始博弈过程 s，我们采用记号 G_s 表示博弈 (ϕ_s, q_s)，其中 ϕ_s 是执行走法 s 后剩余的量词列表；而由 $q_s(t) = q(s \cdot t)$ 给出的 q_s，则是由 s 定义的子博弈的结果函数。这里，$s \cdot t$ 是 s 与 t 的连接（记号重载）。根据 $\bigotimes$ 的定义，如果 ϕ 是位置 s 处的量词，那么

$$\begin{aligned}\mathrm{out}(G_s) &= \phi(\lambda x.\mathrm{out}(G_{s\cdot x})),\\ \mathrm{out}^*_{\alpha,\beta}(G_s) &= \phi^*\left(\lambda x, \alpha', \beta'.\mathrm{out}^*_{\alpha',\beta'}(G_{s\cdot x})\right)(\alpha, \beta).\end{aligned}$$

第6节将讨论如何轻松地将这些定义和证明推广到这样一类博弈：某个位置处的量词取决于迄今为止已执行的走法 s 。

由于我们希望证明第3.1节中单子量词 ϕ^* 的正确性，因此也用标准数学记法定义它们:

定义1（$\phi^*_{\min}$ **和** $\phi^*_{\max}$）。我们先定义

$$\phi^*_{\max} \colon [X]^+ \to (X \to TR) \to TR$$

其中， $[X]^+$ 是 X 上非空列表的类型，如下所示:

$$\phi^*_{\max}(xs, p)(\alpha, \beta) := \max(\text{takeUntil } (\geq \beta)\ \ell(xs)(\alpha, \beta))$$

其中 ℓ: $[X] \to T[R]$

$$\begin{aligned} \ell([\,])(\alpha, \beta) &:= [\,] \\ \ell(x : xs)(\alpha, \beta) &:= v \colon \ell(xs)(\max(\alpha, v), \beta), \quad \textit{where } v = p(x)(\alpha, \beta). \end{aligned}$$

这里，takeUntil $c\ ys$ 是 ys 的初始片段，取至并包括第一个满足 c 的元素；若不存在这样的元素，则取整个 ys 。因此，当 ys 非空时，结果也非空。 $\phi^*_{\min}$ 的定义与上述定义对偶（参见第*3.1*节的 *Haskell* 代码）。

从现在起，我们聚焦于 $\phi^*_{\max}$。对偶结果对 $\phi^*_{\min}$ 也成立，因为对于反向序列，交换 α 和 β 后， $\phi^*_{\min}$是 $\phi^*_{\max}$ ，这会交换下文定理 1 的条款 (i) 和 (ii)，并保留条款 (iii)。

请注意，在 $\phi^*_{\max}$ 的定义中，列表 $\ell(xs)(\alpha, \beta)$ 包含形如 $p(x)(\alpha', \beta)$ 的元素，它们对应不同的 α' 值。对于搜索按此顺序考察的列表 $xs = [x_0, \ldots, x_{n-1}]$，定义 α_0、.....、α_{n-1} 如下:

$$\alpha_0 := \alpha, \qquad \alpha_{i+1} := \max(\alpha_i, p(x_i)(\alpha_i, \beta)).$$

引理1（ $\ell(xs)(\alpha, \beta)$**的刻画**）。给定 p: $X \to TR$以及 $xs = [x_0, \ldots, x_{n-1}]$: $[X]$和 α, β，则

$$\ell(xs)(\alpha, \beta) = [p(x_0)(\alpha_0, \beta), \ldots, p(x_{n-1})(\alpha_{n-1}, \beta)].$$

上述引理表明， $\ell(xs)(\alpha, \beta)$ 的第 i 个成员是 $p(x_i)(\alpha_i, \beta)$。这对于我们通过对博弈树进行归纳法来证明结果至关重要。下面的引理可由上述对 $\ell(xs)(\alpha, \beta)$ 的刻画直接得出:

引理 2. 设给定 p: $X \to TR$、一个非空列表 $xs = [x_0, \ldots, x_{n-1}]$ 以及 (α, β)。

(i) 若对所有 $i < n$，都有 $p(x_i)(\alpha, \beta) \leq \alpha$，则对所有 $i < n$，都有 $\alpha_i = \alpha$ ，且$\phi^*_{\max}(xs, p)(\alpha, \beta) \leq \alpha$; *(ii)* 若存在某个 $i < n$，使得 $p(x_i)(\alpha, \beta) \geq \beta$，则 $\phi^*_{\max}(xs, p)(\alpha, \beta) \geq \beta$。

如上所述，我们还可以证明关于 $\phi^*_{\min}$ 和一个序列 β_i 的类似（对偶）引理。

定理 1. 对于每个部分对局 s 以及每一对 (α, β)

(i) $\mathrm{out}(G_s) \leq \alpha$ 意味着 $\mathrm{out}^*_{\alpha,\beta}(G_s) \leq \alpha$, *(ii)* $\mathrm{out}(G_s) \geq \beta$ 意味着 $\mathrm{out}^*_{\alpha,\beta}(G_s) \geq \beta$, *(iii)* $\alpha < \mathrm{out}(G_s) < \beta$ 意味着 $\mathrm{out}^*_{\alpha,\beta}(G_s) = \mathrm{out}(G_s)$。

证明。 对博弈树使用归纳法。

(1) 若 s 是终局位置，则 $\mathrm{out}^*_{\alpha,\beta}(G_s) = q(s) = \mathrm{out}(G_s)$，且 (i)–(iii) 显然成立。

(2) 设 s 是一个中间位置，并且不失一般性地假设 max 在 s处走棋。设 $xs = [x_0, \ldots, x_{n-1}]$ 为 s处走法的列表，且 $p = \lambda x, \alpha', \beta'.\mathrm{out}^*_{\alpha',\beta'}(G_{s\cdot x})$。我们分三种情况讨论：

(i) $\mathrm{out}(G_s) \leq \alpha$。这是最简单的情况，因为对于所有走法 x （因为我们考虑的是最大化玩家），如果 $\mathrm{out}(G_s) \leq \alpha$，那么 $\mathrm{out}(G_{s\cdot x}) \leq \alpha$。根据归纳假设，对于所有 x，我们有 $\mathrm{out}^*_{\alpha,\beta}(G_{s\cdot x}) \leq \alpha$。在这种情况下，在递归函数 ℓ 的定义中，α 和 β 的值不会更新。因此

$$\begin{aligned}\mathrm{out}^*_{\alpha,\beta}(G_s) &= \phi^*_{\max}(xs, p)(\alpha, \beta)\\ &\overset{\text{L2}\,(i)}{\leq} \alpha.\end{aligned}$$

(ii) $\mathrm{out}(G_s) \geq \beta$。设 i 满足 $\mathrm{out}(G_{s\cdot x_i}) \geq \beta$。由于 max 会更新 α 而不会更新 β，在 (α_i, β) 处应用归纳假设 (ii)，可得 $\mathrm{out}^*_{\alpha_i,\beta}(G_{s\cdot x_i}) \geq \beta$，因此根据引理 2(ii)，

$$\begin{aligned}\mathrm{out}^*_{\alpha,\beta}(G_s) &= \phi^*_{\max}(xs, p)(\alpha, \beta)\\ &\overset{\text{L2}\,(ii)}{\geq} \beta.\end{aligned}$$

(iii) $\alpha < \mathrm{out}(G_s) < \beta$。设 $v = \mathrm{out}(G_s)$，并设 k 为首次取得该最大值的索引，因此 $\mathrm{out}(G_{s\cdot x_k}) = v$。那么，对于每个 i，$\mathrm{out}(G_{s\cdot x_i}) \leq v < \beta$；而对于 $i < k$，$\mathrm{out}(G_{s\cdot x_i}) < v$。每个索引 i 都属于以下某一种情况：

(a) $\mathrm{out}(G_{s\cdot x_i}) \leq \alpha_i$，然后根据归纳法 (i)，$\mathrm{out}^*_{\alpha_i,\beta}(G_{s\cdot x_i}) \leq \alpha_i$；(b) $\alpha_i < \mathrm{out}(G_{s\cdot x_i})$，然后，由于 $\mathrm{out}(G_{s\cdot x_i}) < \beta$，归纳法 (iii) 得出 $\mathrm{out}^*_{\alpha_i,\beta}(G_{s\cdot x_i}) = \mathrm{out}(G_{s\cdot x_i})$。

由于 $\alpha < v$，对 i 进行归纳可得 $\alpha_i < v$ 以及 $\mathrm{out}^*_{\alpha_i,\beta}(G_{s\cdot x_i}) < v$ 对每个 $i < k$ 均成立。因此 $\alpha_k < v$，所以 (b) 适用于 k，并且

$$\mathrm{out}^*_{\alpha_k,\beta}(G_{s\cdot x_k}) = \mathrm{out}(G_{s\cdot x_k}) = v.$$

对于 $k < i$，我们有 $\alpha_i = v$，因此根据 (a) 或 (b)，$\mathrm{out}^*_{\alpha_i,\beta}(G_{s\cdot x_i}) \leq v$。因此，列表中没有任何元素是 $\geq \beta$，takeUntil ($\geq \beta$) 会保留整个列表，而它的

最大值为 v，首次在索引 k 处取得。因此，

$$\begin{aligned}
\mathrm{out}^*_{\alpha,\beta}(G_s) &= \phi^*_{\max}(xs, p)(\alpha, \beta) \\
&= \max(\mathrm{takeUntil}\ (\geq \beta)\ \ell(xs)(\alpha, \beta)) \\
&\overset{\mathrm{L1}}{=} \max(\mathrm{takeUntil}\ (\geq \beta)\ [\ldots, \mathrm{out}^*_{\alpha_k,\beta}(G_{s\cdot x_k}), \ldots]) \\
&= \mathrm{out}^*_{\alpha_k,\beta}(G_{s\cdot x_k}) \\
&= \mathrm{out}(G_{s\cdot x_k}) \\
&= \phi_{\max}\,(\lambda x.\mathrm{out}(G_{s\cdot x})) \\
&= \mathrm{out}(G_s).
\end{aligned}$$

□

推论 1. *(a)* 若对每个博弈过程 t，都有 $\alpha < q(t) < \beta$，则 $\mathrm{out}^*_{\alpha,\beta}(G) = \mathrm{out}(G)$。*(b)* 若 $\alpha < \mathrm{out}^*_{\alpha,\beta}(G) < \beta$，则 $\mathrm{out}^*_{\alpha,\beta}(G) = \mathrm{out}(G)$。

证明。(a)out(G) 是值 $q(t)$ 之一，因此定理 1(iii) 适用。(b) 根据定理 1(i) 和 (ii)，out(G) $\leq \alpha$ 和 out(G) $\geq \beta$ 不可能成立，因此 (iii) 适用。□

5 计算最优对局

在第 3 节中，我们展示了如何使用以下延续单子的实例

$$(X \to TR) \to TR,$$

其中 R 是博弈结果的类型，T 是读取器单子，用于模拟穷举搜索的阿尔法-贝塔剪枝优化。这使我们能够以更高效的方式计算博弈的最优结果 。

现在我们将说明，只需进行少量修改，就不仅能计算最优结果，还能计算出通向该结果的最优博弈过程，同时仍然利用 alpha-beta 优化。

我们的第一个改动是扩展结果类型 R，使其包含通向该结果的博弈过程：

```
-- 扩展结果类型，使其包含通向该结果的博弈过程类型 R  =(整数，路径)
```

接下来，我们需要对极大极小量词做四处小改动，因为类型 R 不再只是一个整数，而是一对整数和路径。例如，修改后的极大量词将变为（修改处以注释标出）：

```
phiMax::走法-> K (TR) 走法phiMax  = K.
phiMax'其中phiMax' :: 走法 -> (走法 -> T R)-
> T R
```

```
phiMax' xs p = do
  (_, b) <- ask
  rs <- results xs p
  -- changed (>=b) to \(r, ms) -> r >= b
  -- changed maximum to pickMax
  return $ pickMax $ takeUntil (\(r, ms) -> r >= b) rs
```

结果 :: 走法 -> (走法 -> T R) -> T [R]结果 [] _ = 返回 []结果 (y:ys)p = 执行 (a, b) <- 询问 -- 已将 r <- p y 改为 (r, ms) <- p y (r, ms) <-p y rs <- 局部(const (max a r, b)) (results ys p) -- 已将 r 改为 (r, ms) 返回 ((r, ms) : rs)

函数 pickMax 会选择值相同的两个结果中的第一个。这一点很重要：被剪枝的子博弈会返回一个结果，其值只是一个界，因此不能取代先前值相同的结果。修订后的最小量词可以用同样的方法，从第 3.1 节的 phiMin得到，只需将 pickMin替换 minimum即可。由于 phiMax 和phiMin 现在返回K的元素，量词列表变为：

```
phisT ::  [K(T R) 走法]
phisT = [phiMax [0,1,2], phiMin [0,1], phiMax [0,1], phiMin [0,1,2]]
```

接下来，我们需要修改结果函数 q，使其不仅返回博弈结果，还返回导致该结果的博弈过程：

```
-- 将其改为单子化，并添加跟踪，以监控已求值分支及求值顺序

q :: 路径-> T R
q xs  =返回 $ 令 r  = q' xs 在
        追踪（显示 xs ++ " -> " ++ 显示 r）（r  , xs）
```

在这个结果函数上，Arun 的博弈量词不仅会计算最优结果，还会计算一个最优博弈过程（同时仍执行 alpha- beta 剪枝）：

```
最优_结果_和_博弈过程:: R最优_结果_和_博弈过程 =
(runReader $量词 bigPhi q) (-无穷, 无穷)

-- ghci> 最佳_结果_和_博弈过程 --
 [0,0,0,0]-> 5--  [0,0,0,1]-> 6--
 [0,0,0,2]-> 6
```

```
-- [0,0,1,0]-> 7--
 [0,0,1,1]-> 4--
 [0,1,0,0]-> 3--
 [0,1,0,1]-> 3--
 [0,1,0,2]-> 3--
 [0,1,1,0]-> 3--
 [1,0,0,0]-> 6--
 [1,0,0,1]-> 6--
 [1,0,0,2]-> 6--
 [1,0,1,0]-> 6--
 [1,1,0,0]-> 7--
 [1,1,0,1]-> 7--
 [1,1,0,2]-> 7--
 [2,0,0,0]-> 5--
 [2,0,1,0]-> 5--(
6,[1,0,0,0])
```

定义 2。若对于每个 $k < n$，都有$\mathrm{out}(G_{s_k \cdot x_k}) = \mathrm{out}(G_{s_k})$，则博弈过程 $t = x_0 x_1 \cdots x_{n-1}$ 是博弈 G 的最优过程，其中 $s_k = x_0 \cdots x_{k-1}$。

推论 *2*。设 G' 为博弈 G，其结果函数为 $q'(t) = (q(t)\ t)$，并按结果的第一分量进行比较。那么，对于 G 的一个最优博弈过程 t，$out(G') = (out(G)\ t)$。因此，若 $\alpha < out(G) < \beta$，则 $out^*_{\alpha,\beta}(G') = (out(G)\ t)$。

证明。第一个结论可由对博弈树进行归纳法得到，因为非空结果列表的最大值和最小值都属于该列表，并且对于每个部分对局 s，$\mathrm{out}(G'_s)$ 的第一分量都是 $\mathrm{out}(G_s)$。第二个结论由第一个结论以及 G' 的定理 1(iii) 得出。□

6 依赖历史的博弈

到目前为止，我们逐步构建了 α-β搜索。在第 2 节中，对博弈树进行穷举极小极大搜索，就是对各轮的最小值量词和最大值量词取量词乘积。在第 3 节中，我们通过将这些量词替换为读取器单子上的单子化量词，得到了 α-β搜索，同时保持其组合方式不变（仍然使用相同的量词乘积）。第 4 节证明了这两种搜索计算出的最优结果相同，第5节则将两者扩展为也能计算最优对局序列。然而，在所有这些情况下，博弈的量词列表都是固定的。每一轮可用的走法，因此该轮的量词，都不会因之前采取的走法而改变。这类博弈称为无历史依赖博弈。

在本节中，我们更进一步，考虑依赖历史的博弈中的阿尔法-贝塔剪枝；在这类博弈中，每一轮可用的走法取决于此前已经进行的走法。大多数我们感兴趣的博弈都属于这一类型：例如在棋盘游戏中，某个走法通常只有在其涉及的格子或棋子

仍然空闲时才可用。第 3 节和第 5 节中的单子量词无需改变。改变的是它们的组合方式：量词乘积变为依赖历史的量词乘积 [3,5], ，其中每一轮的量词都根据导向该轮的部分对局所构成的函数来选择。第 4 节和第 5 节的正确性结果也适用于这一更一般的情形。

我们首先介绍依赖历史的量词的乘积，并解释正确性证明为何仍然适用（第6.1节）。然后，我们以井字棋为例说明这种方法：每位玩家只能占据仍为空闲的方格（第6.2节）。我们利用它计算最优结果和一个最优博弈过程，并比较 α-β搜索评估的对局序列数量与穷举搜索评估的数量。

6.1 依赖历史的量词的乘积

在目前为止所考虑的博弈中，每一轮的量词都是预先固定的。在许多博弈中，例如井字棋，可用的走法取决于目前为止已经进行的走法。遵循 [3,5],，我们将这类博弈建模为：让每一轮的量词取决于引导至该轮的部分对局，因此一个博弈由量词给出

$$\phi_i \colon X^i \to K_R X \qquad (i < n)$$

以及结果函数 $q\colon X^n \to R$。于是，对于长度为 $i \le n$ 的每个部分对局 s，量词乘积定义为

$$(\bigotimes_s \phi)(q) := \begin{cases} q(s) & \text{if } i = n, \\ \phi_i(s)(\lambda x.(\bigotimes_{s\cdot x} \phi)(q)) & \text{if } i < n, \end{cases}$$

且最优结果为 $\mathrm{out}(G) := (\bigotimes_{[]} \phi)(q)$，由附录 B 中的函数`hsequence` 计算得出。单子量词 $\phi_i^* \colon X^i \to K_{TR} X$ 以相同方式给出单子结果 $\mathrm{out}^*_{\alpha,\beta}(G)$。

对于第 4 节和第 5 节的结果，每个 $\phi_i(s)$ 都是将 $\phi_{\max}$ 或 $\phi_{\min}$ 应用于 s 处可用走法的非空列表，而 $\phi_i^*(s)$ 则是相应的 $\phi^*_{\max}$ 或$\phi^*_{\min}$。对于长度为 i 的部分博弈过程 s 中的量词 $\phi_i(s)$，第 4 节中 $\mathrm{out}(G_s)$ 和 $\mathrm{out}^*_{\alpha,\beta}(G_s)$ 的定义随后应使用量词的依赖积。定理 1 以及推论1和2 的证明通过对博弈树进行归纳法完成；在每个位置，它们都使用该处量词的性质，因此无需修改即可应用于依赖历史的博弈。

在 [5], 中（我们在此请读者参阅详细内容），每一回合的走法类型以及回合数也可能取决于此前已进行的走法。此时，博弈由一个类型树给出，其路径即为对局序列；每个内部节点都带有一个量词，而路径上则定义了一个结果函数。我们的 Agda 形式化工作是在这一一般性框架下展开的：每个内部节点都标记有一名玩家以及该处可选走法的非空列表，并且定理 1 和推论1和2适用于所有此类博弈。

6.2 井字棋的 α-β搜索

在井字棋中，走法是棋盘上的九个格子，按行从左到右编号为 0、……、8；部分对局 s 是迄今为止已占据的格子列表，按占据顺序排列。先手在偶数回合取最大值并走棋，后手在奇数回合取最小值并走棋；双方都可以选择任何尚未占据的格子。因此，对于偶数 i，$\phi_i(s)$ 是 $\phi_{\max}(xs)$；对于奇数 i，$\phi_{\min}(xs)$ 是 xs，其中 xs 是未出现在 s中的格子列表。我们使用第5节中的最小值和最大值量词来计算最优结果及一个最优博弈过程（采用阿尔法-贝塔剪枝）：

```
类型 走法 =Int类型 路径 = [走
法]类型 走法 = [Int]类型 棋盘 =
([走法], [走法])数据 玩家 = X
 | O类型 三 = Int



类型 R =(三, 路径)

类型 T =读取器单子 (Int, Int) -- 读取器单子

-- 在这种情况下 +1 且 -1 可以作为可能的博弈结果的上界和下
界无穷大 :: Int无穷大 = 1


-- 为井字棋定义结果函数 q

获胜::路径 ->布尔值获胜 xs =存在
[[0,1,2],[3,4,5],[6,7,8],[0,3,6],[1,4,7],[2,5,8],[0,4,8],[2,4,6]]
(\ys->ys 包含于 xs)

值 ::棋盘-> 三
值 (x,o) = 如果 x 获胜 则 1 否则如果 o 获胜 则 -1 否则 0

结果::玩家 -> [走法] -> 棋盘 -> 棋盘 结果 无论谁 [] 棋盘
 = 棋盘 结果 X (m : ms) (x, o) =如果 o 获胜, 则 (x, o), 否
则结果 O ms (m : x, o) 结果 O (m : ms) (x, o) =如果 x 获胜,
则 (x, o), 否则结果 X ms (x, m : o)


q' :: 路径-> 三
```

```
q' ms  =值 $ 结果 X ms ([],[])

q :: 路径-> T R
q xs  =返回 $ 设 r  = q' xs 中
  追踪（显示 xs  ++ " -> "  ++ 显示 r）(r  , xs)

-- 定义（依赖历史的）量词

   -- 极大极小玩家的博弈量词phisT:: [路径 ->K (TR) 走法]phisT  =
(phiMax . 减去 [0..8]) : (phiMin . 减去 [0..8]): phisT


   -- 博弈树的全局量词 -- 我们使用序列，因为博弈中任意时刻的可用行
动 -- 取决于之前已经进行的走法bigPhi :: K(T R) 路径bigPhi =序列 $
take 9 phisT


   -- 计算最优结果和最优博弈过程最优_结果_和_博弈过程 :: R最优_结果_和_博弈
过程 = (runReader $ 量词 bigPhi q) (-inf,inf)


-- ghci> 最优_结果_和_博弈过程 --
 [0,1,2,3,4,5,6,7,8]-> 1--
 [0,1,2,3,4,5,6,8,7]-> 1--
 [0,1,2,3,4,6,5,7,8]-> 1-- ... --
 [8,4,7,6,5,2,1,0,3]-> -1 --
 [8,4,7,6,5,2,3,0,1]-> -1 -- (
0,[0,4,1,2,6,3,5,7,8])
```

请注意，我们取 α 和 β 的初始值分别为 -1 和 1。由于界 -1 和 1 也是该博弈的结果，因此推论1(a)不适用。但计算所得的结果 0 严格位于二者之间，所以根据推论1(b)，它就是最优结果。

在这个井字棋示例中，穷举搜索会对博弈树中全部 $9! = 362\,880$ 个完整对局序列计算结果函数；而 α-β搜索只对其中 9335 个计算，大约少 39 倍，并且得出相同的最优结果和最优博弈过程：

```
(0,[0,4,1,2,6,3,5,7,8])
```

参考文献

1. Escardó, M., Oliva, P.: 皮尔士翻译与双重否定移位。载于：Ferreira, F., Löwe, B., Mayordomo, E., Gomes, L.M. (编)，《程序、证明、过程——CiE 2010》，LNCS 6158，第 151–161 页。Springer（2010）

2. Escardó, M., Oliva, P.: 选择函数、条递归与逆向归纳。《计算机科学中的数学结构》**20**(2), 127–168（2010） 3. Escardó, M., Oliva, P.: 序贯博弈、蒂科诺夫定理与双重否定移位的共同之处。载于：第三届 ACM 数学结构化函数式编程研讨会论文集（MSFP 2010 1863605 4 21 32 2010 10 1145 1863597 ）。第 – 页。ACM（）。 https://doi.org/ 。/ 。 Escardó, M., Oliva, P.: 序贯博弈与最优策略。英国皇家学会《会刊 A 》 **467**, 1519–1545 （ 2011 ） 5. Escardó, M., Oliva, P.: 依赖类型的高阶博弈。《理论计算机科学》**974**, 114111（2023）。https://doi.org/10.1016/j.tcs.2023. 114111 6. Escardó, M., Oliva, P.: 通过单子延续实现阿尔法-贝塔剪枝的 Agda 形式化。 https://github.com/martinescardo/TypeTopology/ tree/master/source/Games/AlphaBeta（2026），收录于 TypeTopology 开发项目 7. Hart, T.P., Edwards, D.J.: 树剪枝（TP）算法。技术报告 AIM-030，麻省理工学院人工智能项目，马萨诸塞州剑桥（1961年12月），后以阿尔法-贝塔启发式为题修订 8. Hartmann, J.: 使用选择单子的阿尔法-贝塔剪枝。载于：Gibbons, J. (编)，《函数式编程趋势》。第 378–391 页。施普林格·自然瑞士，Cham（2026） 9. Hartmann, J., Schrijvers, T., Gibbons, J.: 朝着更高效的选择单子迈进。载于：Hemann, J., Chang, S. (编)，《函数式编程趋势》。第 55–74 页。施普林格·自然瑞士，Cham（2025） 10. Knuth, D.E., Moore, R.W.: 阿尔法-贝塔剪枝分析。《人工智能》 **6** 4 293 326 1975 10 1016 0004 3702 75 90019 3 ( ), 11 – ( )。https://doi.org/ 。/ - ( ) - 。阿尔法 - 贝塔剪枝。 = 2 https://en.wikipedia.org/w/index.php?title Alpha%E % 80%93beta_剪枝&oldid =1380302874，维基百科，2026年10月14日版本

A Haskell– 工具

模块工具其中

takeUntil :: (a -> Bool) -> [a] -> [a]takeUntil _ [] = []takeUntil p(x:xs) = x : 若 p x 为真，则 [] 否则 takeUntil p xs pickMax :: (Ord a) => [(a, b)] -> (a, b) pickMax [v] = v pickMax ((v,x) : (w, y) : zs) =若 v >=w，则 pickMax ((v, x) : zs)，否则 pickMax ((w, y) : zs) pickMin :: (Ord a) => [(a, b)] -> (a, b) pickMin [v] = v pickMin ((v,x) : (w, y) : zs) =若 v <=w，则 pickMin ((v, x) : zs)，否则 pickMin ((w, y) : zs) some, every :: [a] -> (a ->Bool) -> Bool some xsp = 对 xs 中存在某个元素使 p 成立

```
每个xs p = 所有 p xs

包含 ::Eq x  => [x] -> [x] -> Bool包含 xsys  =
每个 xs (\x -> 属于 x ys)

减法 :: Eqx  => [x] -> [x] -> [x]减法 xs ys
 = [x  | x<-xs，不属于(elem x ys)]
```

B Haskell– 延续单子

```
模块 K 其中

数据 K rx  = K {量词 :: (x -> r) -> r}实例函子
(Kr) 其中fmap f phi = K (\p -> 量词 phi $ p . f)


实例Applicative (Kr) 其中pure x
 =K (\p -> p x)fs  <*> phis = 执
行f  <- fsx  <- phisreturn (f
x)


实例 单子(K r) 其中return  = purephi  >>= f = K (\p -> 量词 phi (\x -
> 量词 (f x) p))

    -- 针对依赖于迄今所走走法的量词进行序列处理 hsequence ::单子 m
 => [[x] -> m x] -> m  [x]hsequence  =h  []其中 h _ []        =
return  []h s (f :fs)  = 执行 x <- fs xs  <- h (s  ++ [x]) fs return (x :xs)
```