Playing Logic Programs with the Alpha-Beta Algorithm

Jean-Vincent Loddo

31 may 2000

Laboratoire Preuves Programmes et Programmation (PPS) - Université Paris 7 - France

Abstract

Alpha-Beta is a well known optimized algorithm used to compute the values of classical combinatorial games, like chess and go. The known proofs of correction of Alpha-Beta do rely on very specific properties of the values used in the classical context (integers or reals), and on the finiteness of the game tree.

In this paper we prove that Alpha-Beta correctly computes the value of a game tree even when these values are chosen in a much wider set of partially ordered domains, which can be pretty far apart from integer and reals, like in the case of the lattice of idempotent substitutions or ex-equations used in logic programming.

We do so in a more general setting that allows infinite games, and we actually prove that for potentially infinite games Alpha-Beta correctly computes the value of the game *whenever it terminates*.

This correctness proofs allows to apply Alpha-Beta to new domains, like constaint logic programming.

1 Introduction

Game theory has found various applications in the research field of programming languages semantics, so that game theory is a very active research subject in computer science. After the preliminary works of Lamarche [6], Blass [2] and Joyal [5] in the early 90s, the works of Abramsky, Malacaria and Jagadeesan [1] lead to the first fully abstract semantics for functional (PCF) or imperative (Idealized Algol) languages. Then, more recently, specialists of Linear Logics got interested in links between games and the geometry of interaction [7], whereas Curien and Herbelin showed that certain classical abstract machines could be interpreted in terms of games [3].

But these relevant works use more the *vocabulary* of games (player, move, game, strategy) than the results and the techniques of traditional Game Theory: typically, nobody is interested to know, in those games, if there is a winner, and what he wins; the focus there is on the dynamic aspect of player *interaction*, and *game composition*, not on the only interesting notion of classical game theory, *the gain*. This should not be taken as a criticism, but as proof of the richness of Game Theory, which can be useful even when one only takes its vocabulary: the generality of the concepts it manipulates (arenas, multiple and independent agents, strategies of cooperation or of non-cooperation, quantification of the remuneration after each

game) and their intuitive nature, already provide a powerful metalanguage that allows to tackle many of the aspects of modern programming languages.

In a paper written with S. Nicolet [4], the authors showed for the first time that classical notions like payoff, propagation functions and evaluation of a game tree are not sterile in the semantics of programming languages, by introducing a two player combinatorial game whose *value*, defined by means of von Neumann's Mini-Max theorem [8], is the result of the execution of a logic program. In that work, we had to introduce an ad hoc framework to deal with substitutions as game values, and to prove the correctness of this game semantics with respect to the traditional C-semantics of logic programs.

In this paper, we present a general framework for classical two player games, but relaxing many traditional restrictions: infinite plays are allowed, values are no longer required to be totally ordered, and propagation functions can be chose from a wide set of candidates. In this setting, the formal definition of the value of a game is given.

Then, we focus on the problem of *efficiently computing* the value of the game, by using the Alpha-Beta *algorithm*, not just on the existence of the value as formally defined.

Surprisingly enough, we can show that the Alpha-Beta algorithm computes the correct value of the game under a few general assumptions on the domain of values, thus greatly broadening its applicability: it can be, for example, used as a computational engine for logic programming.

But we do not content ourselves with proving correctness of Alpha-Beta on games whose value domain is more abstract than the usual integers and reals; we go much further by introducing the notions necessary to deal with potentially infinite games, and we prove that Alpha-Beta is (partially) correct on potentially infinite games: whenever it terminates, it computes the value of the game. This step takes us out of the usual domain of combinatorial game theory, where game trees can be huge but not infinite, and this is, to our best knowledge, the first correctness proof in this setting.

Then, we present an application to logic programming, and exhibit an example where Alpha-Beta gives us a significant gain in performance.
Finally, we conclude with a selection of future directions for research and application.

2 An abstract theory of two players combinatorics games

In this section, we introduce our formal framework for two player games, together with some fundamental notions, like that of a game-tree and the value of a finite game. We then present the notion of an *approximation* of a game value, as is found in the theory of combinatorial games like chess or go, which are too big to be fully developed, and use it as a key notion to extend the framework to deal with infinite games too.

2.1 The rules of the game

A two player game is the simplest game in combinatorial Game Theory: we find two players, Player and Opponent, each opposed to the other, that play in turn one after the other and that must behave *rationally* (i.e. their move are deterministic, and they both seek to win). The game is assumed to be finite, and once one knows the terminal position in a play, the gain (or loss) of each player is known also; besides, what one looses, the other wins, so there always is precisely one winner.

A game is given once one knows its "syntax", that is the set of all possible plays, and its "semantics", that is the *value* of each of these plays (what Player gains or looses towards Opponent). We will formalize each of these aspects in turn, starting here from the "syntax".

2.2 Basic definitions: syntax

There are two ways of knowing all possible plays, either by giving them extensively as a set, an approach quite unadequate to handle real-world games where this set can be enourmous, even if finite; or by giving a set of "positions" and "rules" that allow to produce all possible plays (like in chess or go).
We will take in what follows this second approach: the following definitions are essentially the traditional ones.

Definition 2.1 (Syntax of a game) *The syntax of a game is formally defined as a tuple*

$$G = (WPOS,\ BPOS,\ IPOS; \mathcal{R})$$

Here $WPOS$ is the set of Player position, while $BPOS$ is the set of opponent positions and we write POS for the disjoint sum $WPOS \oplus BPOS$, and π for a generic position in POS. The third component, $IPOS \subseteq POS$ is the set of initial positions in the game. Finally, $\mathcal{R} \subseteq POS \times POS$ is a *locally finite* (i.e. only a finite number of pairs in the relation can share their firs component) transition relation that represents all the possible moves in a play as transitions betwee positions.
We require that the moves in the game are *alternating*, that is to say that whenever $(\pi, \pi') \in \mathcal{R}$ we have that if π is a player position, then π' is an opponent position and viceversa.
It is often useful to introduce an auxiliary function $succ : POS \to 2^{POS}$, defined as $succ(\pi) = \{\pi' \mid \pi \mathcal{R} \pi'\}$ that explicitly gives the possible moves out of a given position.

Once the syntax of a game is known, we have all the necessary information to determine when a game is finished (we have reached a terminal position, or not).

Definition 2.2 (Terminal, non terminal positions) *Given a game G, we identify the following derived notions*

terminal positions $TPOS = \{\pi \in POS \mid not\, \exists \pi'\ such\ that\ \pi \to \pi'\}$

player terminal positions $WTPOS = TPOS \cap WPOS$

opponent terminal positions $BTPOS = TPOS \cap BPOS$

nonterminal positions $NTPOS = \{\pi \in POS \mid \exists \pi'\ such\ that\ \pi \to \pi'\}$

player terminal positions $WNTPOS = NTPOS \cap WPOS$

opponent terminal positions
$BNTPOS = NTPOS \cap BPOS$

Finally, all the possible plays starting from a given initial position can be represented as a tree, known as a *game tree*.

RIVEDERE LA DEFINIZIONE!

Definition 2.3 (Game tree) *A game tree Γ_π, for an initial position π, is a tree having positions as nodes, and representing all possible plays starting at π. It can be defined formally by induction.*
Since Ris locally finite, the game tree is finitely branching.

A game is *finite* if its game tree is, infinite otherwise.

2.3 Basic definitions: semantics

It is now time to turn to the essential aspect of a game in classical game theory: its *value*. To each terminal position, which is reached when a play is complete, is associated a *gain* for Player, taken out of some domain D (traditionally, the integers), given by an evaluation function h. Of course, this gain for Player is actually a *loss* for opponent, and given a set of possible moves, Player chooses the move that maximizes its gain, while Opponent chooses the move that minimizes its loss; this rational choice can be abstracted by two functions $\uparrow$ and $\downarrow$ on the domain of values. All these elements give us the semantics part of the game.

Definition 2.4 (Evaluation structure) *An* evaluation structure *is a tuple*

$$(D\ ,\ \uparrow, \downarrow\ ,\ h)$$

*where D is the domain of values, $\uparrow$: $D^n \to D$ and $\downarrow$: $D^n \to D$ are functions of all finite arities $n \geq 1$ materializing the rational choices of Player and Opponent, and $h : TPOS \to D$ is the evaluation function giving the value (*gain*) of each terminal position.*

Given an evaluation structure, it is possible to compute the *value* of any finite game (this definition mirrors the traditional one).

Definition 2.5 (Value of a finite game) *The function $Val : POS \to D$ defined as follows asociates to each position in a game its* value*:*

$$Val\ \pi = h(\pi) \qquad if\ \pi \in TPOS$$

$$Val\ \pi = \uparrow_{\pi' \in moves(\pi)} Val\ \pi' \qquad if\ \pi \in WNTPOS$$

$$Val\ \pi = \downarrow_{\pi' \in moves(\pi)} Val\ \pi' \qquad if\ \pi \in BNTPOS$$

2.4 From huge to infinite games

Luckily, many finite games, like chess and go, are so huge that computing their value is not feasible. This is why one needs sometimes to try to compute an approximation of the value of a game from a given position. For that, we simply stop exploring the huge tree at some internal nodes, whose value is arbitrarily provided by some heuristic function. We will see that, while heuristics are simply useful to approximate the value of huge finite games, they are essential to *define* the value of an infinite game.
But to compute approximations, one needs to be able to compare values, that is, in what follows we assume that D is actually equipped with a partial order relation $\leq$.

2.4.1 Heuristics and approximations

An heuristic function is just a function of type $NTPOS \to D$ assigning arbitrary values to nonterminal positions in a game, but only some heuristics are interesting, and one usually distinguish between optimistic and pessimistic heuristic according to the ability of the heuristic to provide approximation gretaer of, or inferior to, the actual value.

Definition 2.6 (Admissible heuristics) *An heuistic function* $\varphi : NTPOS \to D$ *is an* admissible pessimistic heuristic *(resp.* admissible optimistic heuristics *) for a finite game iff* $\forall \pi \in NTPOS. \varphi(\pi) \leq Val\, \pi \qquad (resp.\ \geq)$.

Unfortunately, determining if an heuristic is admissible can be quite hard, and another definition can be more useful in practice: we say an heuristic is monotone if the approximation it provides are coherent among themselves.

Definition 2.7 (Monotone heuristics) *An heuristic function* $\varphi : NTPOS \to D$ *is* monotone pessimistic *(resp.* monotone optimistic*) iff :*

$$\forall \pi \in NTPOS. \quad \begin{cases} \varphi(\pi) \leq h(\pi') & \forall \pi \to \pi' \, s.t. \, \pi' \in TPOS \qquad (resp.\ \geq) \\ \varphi(\pi) \leq \varphi(\pi') & \forall \pi \to \pi' \, s.t. \, \pi' \in NTPOS \qquad (resp.\ \geq) \end{cases}$$

Once we have an heuristic, we can build out of our evaluation structure a structure useful to compute approximations.

Definition 2.8 (Approximation structure) *A* pessimistic approximation structure *(resp.* optimistic*) is a tuple*

$$(D\ ,\ \leq\ ,\ \uparrow\ ,\ \downarrow\ ,\ ,\ h\ ,\ \varphi)$$

where $(D\ ,\ \uparrow\ ,\ \downarrow\ ,\ ,\ h$ *is an evaluation structure, and such that the function* $\varphi' : POS \to D$ *defined as* $\varphi' = \varphi \oplus h$ *satisfies:*

$$\varphi'(\pi) \quad \leq \quad \underset{\pi' \in moves(\pi)}{\uparrow} \quad \varphi'(\pi') \qquad if\ \pi \in WNTPOS \qquad (resp.\ \geq)$$

$$\varphi'(\pi) \quad \leq \quad \underset{\pi' \in moves(\pi)}{\downarrow} \quad \varphi'(\pi') \qquad if\ \pi \in BNTPOS \qquad (resp.\ \geq)$$

If D contains a minimal element $\bot$, the heuristic $\lambda\pi.\bot$ will give us a *canonical* pessimistic approximation structure. Similarly, if D contains a maximal element $\top$, the heuristic $\lambda\pi.top$ will give us a *canonical* optimistic approximation structure.

The monotonicity property is interesting because of the following result.

Proposition 2.9 (Monotonicity and approximations) *If* $(D, \leq)$ *is a* distributive lattice*, and we take* $\uparrow = \vee$ *(the sup) and* $\downarrow = \wedge$ *(the inf) on* D*, then any* monotone pessimistic heuristics *(resp.* optimistic*) gives raise to a* pessimistic approximation structure *(resp.* optimistic*).*

Once we have an approximation structure at hand, we can compute an approximation of the value of a game, by cutting the tree branches at some internal nodes, obtaining another (smaller) tree, and computing the value of this cut tree. Of course, the approximation thus computed depends on where the cut actually take place, so an approximation structure really gives rise to a whole set of approximations. This can be put more formally as follows:

Definition 2.10 (Set of approximations of a game ($SetOfVal_\varphi$)) *Given a game G and an heuristic $\varphi : NTPOS \to D$, we define the approximation function for the game G, relative to φ, written $SetOfVal_\varphi : POS \to 2^D$, as the smallest function $S : POS \to 2^D$ (w.r.t. to the pointwise partial order on $POS \to 2^D$ derived from the partial order $\subseteq$ in 2^D) such that:*

1. *$S(\pi)$ contains $h(\pi)$ if $\pi \in TPOS$*
2. *$S(\pi)$ contains the element $\varphi(\pi)$ if $\pi \in NTPOS$*
3. *$S(\pi)$ contains the element $\overset{n}{\underset{i=1}{\uparrow}} v_i$ if $\pi \in WNTPOS$, where $moves(\pi) = \{\pi_1, \pi_2, \ldots, \pi_n\}$ and $v_i \in S(\pi_i)$*
4. *$S(\pi)$ contains the element $\overset{n}{\underset{i=1}{\downarrow}} v_i$ if $\pi \in BNTPOS$, where $moves(\pi) = \{\pi_1, \pi_2, \ldots, \pi_n\}$ and $v_i \in S(\pi_i)$*

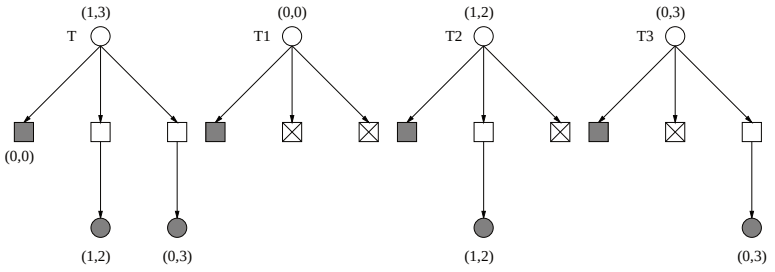
We remark here that $SetOfVal_\varphi(\pi)$ is well defined for all positions π as its value is the least fixed point of a monotone function over the complete lattice 2^D, which always exists due to Knaster-Tarski's fixpoint theorem.
Actually, this really defines a function $SetOfVal : (NTPOS \to D) \to POS \to 2^D$, which we will apply to monotone heuristic functions in order to obtain a set of approximations having a reasonable algebraic structure.

Proposition 2.11 (Algebraic structure of the monotone approximations) *If $(D\ ,\ \leq\ ,\ \uparrow\ ,\ \downarrow\ ,\ ,\ h\ ,\ \varphi)$is a* pessimistic approximation structure *(resp.* optimistic*), then for all position $\pi \in POS$ the set $SetOfVal_\varphi(\pi) \in 2^D$ is an* upper-semi-lattice *(i.e. $\forall x, y \in S. x \vee y \in S$) (resp.* lower-semi-lattice*).*
Moreover, if the game is finite*, then $SetOfVal_\varphi(\pi)$ contains its upper (resp lower) limit $Val\,\pi$.*

To put it in other terms, for finite games, given any monotone pessimistic heuristic h_{pess} and any monotone optimistic approximation h_{opt}, we have that

$$SetOfVal_{h_{pess}}(\pi) \cap SetOfVal_{h_{opt}}(\pi) = Val\,\pi$$

Example 2.12 In the following picture, we give the game tree of a finite game (on the left), and three smaller game trees obtained by cutting the full tree at some positions. The evaluation domain D is $Nat \times Nat$ with $\leq$ the lexicographic order and we use the trivial heuristic $h_{pess} = \lambda\pi.(0,0)$. Each tree is labelled with its valuation (the values of the trees on the right approximations in $SetOfVal_{h_{pess}}(\pi)$: notice that the set of approximations is not totally ordered. Terminal nodes are grey, no terminal are white; player nodes are circles, opponent nodes are squares, and cut nodes are crossed.



□

2.5 Infinite games

The framework set up to approximate the value of huge games can be used as is to handle infinite games too. Indeed the $SetOfVal$ function, defined as we did, is well defined both on finite *and* infinite games. This allows us to formally define two canonical approximations of the value of an infinite game, as partial functions assigning to each position the limit of its approximations, if it exists.

Definition 2.13 (Limit values of a game) *If h_{pess} is a* monotone pessimistic *heuristic, and h_{opt} is a* monotone optimistic *heuristic, then*

$$Val_{h_{pess}}\pi \doteq \sup\, SetOfVal_{h_{pess}}(\pi) \qquad Val_{h_{opt}}\pi \doteq \inf\, SetOfVal_{h_{opt}}(\pi)$$

where we write $x \doteq y$ for "x is equal to y if y defined".
If they coincide, that defines the value $Val\ \pi$ of the infinite game *on* π.

Notice that $Val_{h_{pess}}$ is defined everywhere if the domain is a complete partial order (CPO), and $Val_{h_{opt}}$ is defined everywhere if if the domain is a co-complete partial order (co-CPO). Also, on finite games both approximations coincide with the value of the finite game.

Proposition 2.14 (Relating approximations) *Given an evaluation strucure and two monotone heuristics h_{pess} and $h_{opt}(\pi)$, one pessimistic, the other optimistic, such that $h_{pess}(\pi) \leq h_{opt}(\pi)$ for all non terminal positions π, then we have that*

$$h_{pess}(\pi) \leq y \qquad \forall y \in SetOfVal_{h_{opt}}(\pi) \qquad h_{opt}(\pi) \geq x \qquad \forall x \in SetOfVal_{h_{pess}}(\pi)$$

for all non terminal positions π.

Proof. We give the proof of the first inequation by induction on the definition of $y \in SetOfVal_{h_{opt}}$. The second inequation is proved similarly.

- Base case: $y = h_{opt}(\pi)$ and $h_{pess}(\pi) \leq y$ by hypothesis
- Inductive step: suppose $\pi \in WNTPOS$ (the case $\pi \in BNTPOS$ is similar);

  we have that $y = \overset{n}{\underset{i=1}{\uparrow}}\ y_i$ where $y_i \in SetOfVal_{h_{opt}}(\pi_i)$. Since $(D, \leq, \uparrow, \downarrow$

  $, h_{pess})$ is an approximation structure, we have : $h_{pess}(\pi) \leq \overset{n}{\underset{i=1}{\uparrow}}\ h_{pess}(\pi_i)$,

  and by monotonicity of $\uparrow$ plus the induction hypothesis, we obtain $h_{pess}(\pi) \leq$

  $\overset{n}{\underset{i=1}{\uparrow}}\ y_i = y$.

□

Corollary 2.15 (Finite computations for infinite games) *Under the hypothesis of the previous proposition, we only have two possible cases*

1. $SetOfVal_{h_{pess}}(\pi) \cap SetOfVal_{h_{opt}}(\pi) = \{v\}$ *where* $v = Val\,\pi$
2. $SetOfVal_{h_{pess}}(\pi) \cap SetOfVal_{h_{opt}}(\pi) = \emptyset$

The first case gives us a sufficient condition to stop the computation of the approxiamtions on an infinite game tree: as soon as the intersection is non empty, we know we have the value of the game, without needing to fully compute the set of approximations.

We will use these properties in our analysys of the Alpha-Beta algorithm on infinite games.

3 The Alpha-Beta algorithm

3.1 Sequential version

```
function AlphaBeta(π : Pos ; α, β : D) : D
   begin
   if π ∈ TPOS then return(h(π));
   if π ∈ WNTPOS and moves(π) = {π_1, ..., π_n} then begin
        v := α ↑ h_pess(π);
        i := 1;
        while (not v ≥ β) and (i ≤ n) do begin
                  v := v ↑ AlphaBeta(π_i, v, β);
                  i := i + 1;
                 end;
        end;
   if π ∈ BNTPOS and moves(π) = {π_1, ..., π_n} then begin
        v := β ↓ h_opt(π);
        i := 1;
        while (not v ≤ α) and (i ≤ n) do begin
                  v := v ↓ AlphaBeta(π_i, α, v);
                  i := i + 1;
                 end;
        end;
   return(v);
   end;
```

3.2 Theorem (correctness of Alpha-Beta)

Lemma 1 (Inserimento). Supponiamo che $(D, \leq)$ sia un *distributive lattice* e siano $\uparrow = \vee$ il sup e $\downarrow = \wedge$ l'inf del dominio D. Allora:

$$\forall \alpha, \beta, x. \begin{cases} \alpha \uparrow [\beta \downarrow x] & = & \alpha \uparrow [\beta \downarrow (\alpha \uparrow x)] \\ \beta \downarrow [\alpha \uparrow x] & = & \beta \downarrow [\alpha \uparrow (\beta \downarrow x)] \end{cases}$$

Proof. Per la prima equazione:

$$\alpha \uparrow [\beta \downarrow (\alpha \uparrow x)] \underset{distributive}{=} \alpha \uparrow [(\beta \downarrow \alpha) \uparrow (\beta \downarrow x)] \underset{(\beta \downarrow \alpha) \leq \alpha}{=} \alpha \uparrow (\beta \downarrow x)$$

L'altra tesi per dualit.

Def. (Equality modulo $\alpha\beta$). Let us denote by $AlphaBeta(\pi, \alpha, \beta) =_{\alpha\beta} z$ where $z \in D$, one of the following equations over D, according to the type of the position π:

$$\begin{array}{lcll} \alpha \uparrow [\beta \downarrow AlphaBeta(\pi, \alpha, \beta)] & = & \alpha \uparrow [\beta \downarrow z] & if\ \pi \in WPOS \\ \beta \downarrow [\alpha \uparrow AlphaBeta(\pi, \alpha, \beta)] & = & \beta \downarrow [\alpha \uparrow z] & if\ \pi \in BPOS \end{array}$$

Theorem Supponiamo che $(D, \leq)$ sia un *distributive lattice* e supponiamo di istanziare il metodo astratto *AlphaBeta* con $\uparrow = \vee$ il sup e $\downarrow = \wedge$ l'inf del dominio D e di avere delle euristiche h_{pess} e h_{opt} *monotone* (resp. *pessimista* e *ottimista*) e tali che $h_{pess}(\pi) \leq h_{opt}(\pi)$ pour toute position $\pi \in NTPOS$. Then, for all position $\pi \in POS$, and for all $\alpha, \beta \in D$, if the function $AlphaBeta(\pi, \alpha, \beta)$ terminates, then there exist $x \in SetOfVal_{pess}(\pi)$ and $y \in SetOfVal_{opt}(\pi)$ such that:

$$AlphaBeta(\pi, \alpha, \beta) =_{\alpha\beta} x =_{\alpha\beta} y$$

Proof. Se l'algoritmo termina allora visita una parte finita dell'albero di gioco (eventualmente infinito). Procediamo dunque per induzione strutturale sulla parte visitata. Analizzeremo solo i casi in cui la posizione $\pi \in WPOS$, vale a dire le posizioni del giocatore, essendo facilmente ricostituibili, per dualit, le prove per le posizioni dell'avversario.

Base induttiva. L'algoritmo visita un solo nodo, la radice, e termina restituendo il valore $v \in D$. Ci sono due casi possibili:

- Il nodo terminale

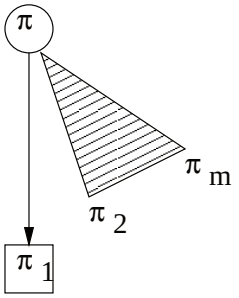
$\Rightarrow v = h(\pi)$ dunque $x = h(\pi) \in SetOfVal_{pess}(\pi)$ and $y = h(\pi) \in SetOfVal_{opt}(\pi)$ verificano banalmente l'uguaglianza modulo $\alpha\beta$ con v.

- Il nodo stato tagliato immediatamente a causa del verificarsi della condizione $\alpha \uparrow h_{pess}(\pi) \geq \beta$

$\Rightarrow v = \alpha \uparrow h_{pess}(\pi)$. Siano $x = h_{pess}(\pi) \in SetOfVal_{pess}(\pi)$ and $y = h_{opt}(\pi) \in SetOfVal_{opt}(\pi)$. Per il lemma 1 si ha che $\alpha \uparrow [\beta \downarrow v] = \alpha \uparrow [\beta \downarrow (\alpha \uparrow h_{pess}(\pi))] = \alpha \uparrow [\beta \downarrow h_{pess}(\pi)] = \alpha \uparrow [\beta \downarrow x] = (*)$ dunque $v =_{\alpha\beta} x$. Inoltre, essendo $\alpha \uparrow h_{pess}(\pi) \geq \beta$ ho che $(*) = \alpha \uparrow \beta$. Ricordando che $h_{pess}(\pi) \leq h_{opt}(\pi)$, ho per monotonia di $\uparrow$ che anche $\alpha \uparrow h_{opt}(\pi) \geq \beta$ per cui $\alpha \uparrow [\beta \downarrow y] = \alpha \uparrow [\beta \downarrow (\alpha \uparrow h_{opt}(\pi))] = \alpha \uparrow \beta$. Vale a dire $v =_{\alpha\beta} y$.

Induzione strutturale. Dans la position π, on suppose que l'algorithme ait analys n coups du joueur parmis les m disponibles ($m \geq n$) avant de terminer et renvoyer son rsultat. On procde par induction sur n.

- Cas $n = 1$



$v = AlphaBeta(\pi, \alpha, \beta) = \alpha \uparrow h_{pess}(\pi) \uparrow AlphaBeta(\pi_1, v_0, \beta) \qquad (*)$ o $v_0 = \alpha \uparrow h_{pess}(\pi)$. Par hypothse d'induction structurelle on sait qu'il existe $x_1 \in SetOfVal_{pess}(\pi_1)$ and $y_1 \in SetOfVal_{opt}(\pi_1)$ such that:

$$AlphaBeta(\pi_1, v_0, \beta) =_{v_0\beta} x_1 =_{v_0\beta} y_1$$

ce qui signifit, etant $\pi_1 \in BPOS$ une position de l'opposant, que

$$\beta \downarrow [v_0 \uparrow AlphaBeta(\pi_1, v_0, \beta)] = \beta \downarrow [v_0 \uparrow x_1] = \beta \downarrow [v_0 \uparrow y_1] \qquad (**)$$

On distingue maintenant entre deux causes de terminaison de l'algorithme: le cas $n = m = 1$ (tous les coups valus) et le cas $n = 1 < m$ (coupure).

- Cas $n = m = 1$

On definit les aproximations: $x =\uparrow x_1 = x_1$ et $y =\uparrow y_1 = y_1$. Par dfinition $x \in SetOfVal_{pess}(\pi)$ et $y \in SetOfVal_{opt}(\pi)$. Donc :

$$\alpha \uparrow [\beta \downarrow v] \underset{(*)}{=} \alpha \uparrow [\beta \downarrow (v_0 \uparrow AlphaBeta(\pi_1, v_0, \beta))] \underset{(**)}{=} \alpha \uparrow [\beta \downarrow (\alpha \uparrow h_{pess}(\pi) \uparrow x_1)]$$

$$\underset{x=x_1 \wedge L1}{=} \alpha \uparrow [\beta \downarrow (h_{pess}(\pi) \uparrow x)] \underset{h_{pess}(\pi) \leq x}{=} \alpha \uparrow [\beta \downarrow x]$$

A noter que la condition $h_{pess}(\pi) \leq x$ est garantie par l'hypothse que h_{pess} soit une heuristique monotone pessimiste. De la mme faon :

$$\alpha \uparrow [\beta \downarrow v] \underset{(*)}{=} \alpha \uparrow [\beta \downarrow (v_0 \uparrow AlphaBeta(\pi_1, v_0, \beta))] \underset{(**)}{=} \alpha \uparrow [\beta \downarrow (\alpha \uparrow h_{pess}(\pi) \uparrow y_1)]$$

$$\underset{y=y_1 \wedge L1}{=} \alpha \uparrow [\beta \downarrow (h_{pess}(\pi) \uparrow y)] \underset{h_{pess}(\pi) \leq y}{=} \alpha \uparrow [\beta \downarrow x]$$

La condition $h_{pess}(\pi) \leq y$ est garantie par l'hypothse $h_{pess}(\pi) \leq h_{opt}(\pi)$ qui fait que n'importe quelle approximation pessimiste sera infrieure ou gale n'importe quelle autre optimiste (prop. ...).

- Cas $n = 1 < m$

Si l'algorithme a coup les $m - 1$ coups restants, c'est que la condition $v_0 \uparrow v_1 \geq \beta$ o $v_1 = AlphaBeta(\pi_1, v_0, \beta)$ tait vrifi, ce qui implique que $\alpha \uparrow [\beta \downarrow v] = \alpha \uparrow [\beta \downarrow (v_0 \uparrow v_1)] = \alpha \uparrow \beta$. On definit les aproximations: $x = x_1 \uparrow x'_1$ o $x'_1 = h_{pess}(\pi_2) \uparrow \ldots \uparrow h_{pess}(\pi_m)$ et $y = y_1 \uparrow y'_1$ o $y'_1 = h_{opt}(\pi_2) \uparrow \ldots \uparrow h_{opt}(\pi_m)$. Par dfinition $x \in SetOfVal_{pess}(\pi)$ et $y \in SetOfVal_{opt}(\pi)$. Pour l'aproximation x, nous avons que :

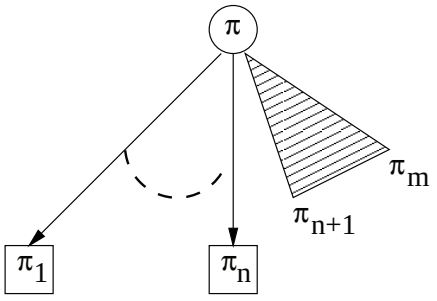
$$\alpha \uparrow [\beta \downarrow x] \underset{Lemma\,1}{=} \alpha \uparrow [\beta \downarrow (\alpha \uparrow x)] \underset{h_{pess}(\pi) \leq x}{=} \alpha \uparrow [\beta \downarrow (\alpha \uparrow h_{pess}(\pi) \uparrow x)]$$

$$= \alpha \uparrow [\beta \downarrow (v_0 \uparrow x_1 \uparrow x'_1)] \underset{distrib.}{=} \alpha \uparrow [\beta \downarrow (v_0 \uparrow x_1)] \uparrow [\beta \downarrow x'_1] \underset{(**)}{=} \alpha \uparrow [\beta \downarrow (v_0 \uparrow v_1)] \uparrow [\beta \downarrow x'_1]$$

$$\underset{v_0 \uparrow v_1 \geq \beta}{=} \alpha \uparrow \beta \uparrow [\beta \downarrow x'_1] \underset{\beta \downarrow x'_1 \leq \beta}{=} \alpha \uparrow \beta$$

En utilisant le fait que $h_{pess}(\pi) \leq y$, etant y une aproximation optimiste, on demontre que $\alpha \uparrow [\beta \downarrow y] = \alpha \uparrow \beta$ avec exactement les mmes passages logiques.

- Cas $n > 1$



Si on considre une position fictive π' ayant tous et seuls les premier $n-1$ coups de π, et un'autre position fictive ayant seulement le n-eme coup, par dfinition de l'algorithme on pourra ecrire que:

$v = AlphaBeta(\pi, \alpha, \beta) = AlphaBeta(\pi', \alpha, \beta) \uparrow AlphaBeta(\pi'', v_1, \beta)$ o $v_1 = AlphaBeta(\pi', \alpha, \beta)$ et en considerant $h_{pess}(\pi') = h_{pess}(\pi'') = h_{pess}(\pi)$. Par hypothse d'induction on sait qu'ils existent $x' \in SetOfVal_{pess}(\pi')$ and $y' \in SetOfVal_{opt}(\pi')$ such that:

$$AlphaBeta(\pi', \alpha, \beta) =_{\alpha\beta} x' =_{\alpha\beta} y'$$

et, aussi, qu'ils existent $x'' \in SetOfVal_{pess}(\pi'')$ and $y'' \in SetOfVal_{opt}(\pi'')$ tels que:

$$AlphaBeta(\pi'', v_1, \beta) =_{v_1\beta} x'' =_{v_1\beta} y''$$

$\alpha \uparrow [\beta \downarrow v] = \alpha \uparrow [\beta \downarrow (v_1 \uparrow AlphaBeta(\pi'', v_1, \beta))] = \alpha \uparrow [\beta \downarrow (v_1 \uparrow AlphaBeta(\pi'', v_1, \beta))] =$

$= \alpha \uparrow [\beta \downarrow (v_1 \uparrow (\beta \downarrow AlphaBeta(\pi'', v_1, \beta)))] = \alpha \uparrow [\beta \downarrow (v_1 \uparrow (\beta \downarrow x''))] = \alpha \uparrow [\beta \downarrow (v_1 \uparrow x'')] =$

$= \alpha \uparrow [\beta \downarrow v_1] \uparrow [\beta \downarrow x''] = \alpha \uparrow [\beta \downarrow x'] \uparrow [\beta \downarrow x''] = \alpha \uparrow [\beta \downarrow (x' \uparrow x'')] = (*)$

Avec les mme arguments on dmontre que $\alpha \uparrow [\beta \downarrow v] = \alpha \uparrow [\beta \downarrow (y' \uparrow y'')] = (**)$

On distingue maintenant entre deux causes de terminaison de l'algorithme appliqu la position π: le cas $n = m$ (tous les coups valus) et le cas $n < m$ (coupure).

- Cas $n = m$

Les coups de π sont exactement la somme des coups de π' et ceux de π''. Donc, en dfinissant les aproximations: $x = x' \uparrow x'$ et $y = y' \uparrow y''$ on aura que $x \in SetOfVal_{pess}(\pi)$ et $y \in SetOfVal_{opt}(\pi)$ et on aura donc la thse par les conditions $(*)$ et $(**)$.

- Cas $n < m$

Si l'algorithme a coup les $m-n$ coups restants, c'est que la condition $v_0 \uparrow v_1 \geq \beta$ o $v_0 = AlphaBeta(\pi', \alpha, \beta)$ tait vrifi, ce qui implique que $\alpha \uparrow [\beta \downarrow v] = \alpha \uparrow [\beta \downarrow (v_0 \uparrow v_1)] = \alpha \uparrow \beta$. On definit les aproximations: $x = x' \uparrow x'' \uparrow x'''$ o $x''' = h_{pess}(\pi_{n+1}) \uparrow \ldots \uparrow h_{pess}(\pi_m)$ et $y = y' \uparrow y'' \uparrow y'''$ o $y''' = h_{opt}(\pi_{n+1}) \uparrow \ldots \uparrow h_{opt}(\pi_m)$. Par dfinition $x \in SetOfVal_{pess}(\pi)$ et $y \in SetOfVal_{opt}(\pi)$. En utilisant la proprit distributive et les conditions $(*)$ on obtient que :

$\alpha \uparrow [\beta \downarrow x] = \alpha \uparrow [\beta \downarrow (x' \uparrow x'' \uparrow x''')] = \alpha \uparrow [\beta \downarrow (x' \uparrow x'')] \uparrow [\beta \downarrow x'''] = \alpha \uparrow [\beta \downarrow v] \uparrow [\beta \downarrow x'''] =$

$= \alpha \uparrow \beta \uparrow [\beta \downarrow x'''] = \alpha \uparrow \beta$. Ce dernier passage parce-que $\beta \downarrow x''' \leq \beta$. Avec les mmes passages, mais en utilisant la $(**)$, on obtient aussi que $\alpha \uparrow [\beta \downarrow x] = \alpha \uparrow \beta$.

4 Dfinition du jeu de la programmation logique

On donne les rgles du jeu pour la programmation logique avec contraintes, qui inclus le cas de la programmation logique classique (contraintes de Herbrand). Avec le langage de contraintes C, on suppose avoir une operation de base, l'intersection de contrainte, que l'on notera avec le symbole $\wedge$. Dans le cas des termes de Herbrand, il s'agirat de l'*mgu* (Most General Identifier). On suppose aussi que la relation d'implication logique $\Rightarrow$ entre contraintes donne au langage C une structure algbrique d'ordre partiel. On supposera aussi l'existance de la contrainte *true* representant la contrainte vide (pas de contraintes).

4.1 Rgles du jeu

Il s'agit d'un jeu affectation statiques des Blancs et des Noirs qui jouent alternativement.

4.1.1 Positions

L'ensemble des positions $\pi \in POS$ est dfinit par la syntaxe :

$$\pi ::= (A, c) \mid [G, c]$$

o G est un but conjonctif (positif), A un atome (positif) et c appartient au langage χ des contraintes.

Il s'agit, dans l'ordre, des positions disjointes des Blancs $WPOS$ et des Noirs $BPOS$. La notation des parenthses rondes et rectangulaires suit la convention, dans la thorie des jeux combinatoire, de reprsenter graphiquement les arbres de jeu avec des noeuds rond pour (le joueur qui prend) les Blancs et des noeuds rectangulaires pour (le joueur qui prend) les Noirs.

Puisque l'affectation des couleurs est statique nous dirons que le "Joueur" est celui qui prend les Blancs et l"'Opposant" est celui qui prend les Noirs.

4.1.2 Positions initiales

Le jeu peut dmarrer dans une quelconque position des Noirs, o la partie contrainte est la contrainte vide (i.e. aucune contrainte). L'ensemble $IPOS$ est donc dfinit par la syntaxe :

$$\pi ::= [G, true]$$

4.1.3 Coups des Blancs

Dans une position (A,c) les Blancs ont autant de coups que de (renommages de) rgles logiques du programme P du type :

$$A \leftarrow c' \mid A_1, A_2, ..., A_n$$

telles que $c \wedge c'$ soit satisfaisable dans l'algbre de contraintes du langage χ. Un coup de la sorte conduit le jeu vers la position (des Noirs) $[(A_1, A_2, ..., A_n), c \wedge c']$.

4.1.4 Coups des Noirs

Dans une position $[(A_1, A_2, ..., A_n), c]$ les Noirs ont autant de coups que de atomes A_i. Chacun de ces coups conduit, respectivement, dans les positions (des Blancs) (A_i, c).

4.2 Smantique du jeu CLP(χ)

Le jeu dfinit ci-dessus admet des arbres de jeu infinis, donc l'valuation doit tre dfinie par la technique des approximations finies, pessimistes et optimistes, de la valeur du jeu. Les dfinitions qui suivent reprsentent une simple instance des dfinitions gnrales pour tout jeu deux joueurs.

4.2.1 Domaine d'valuation

tant donn le langage des contraintes C sur l'algbre χ le domaine d'valuation d'un arbre de jeu est l'ensemble des parties de C :

$$D = 2^C$$

L'ordre partiel $\leq$ sur D est l'extension "covering" [Fages 94] de l'ordre "implication logique" dfinit sur les contraintes :

$$d_1 \leq d_2 \; ssi \; \forall c_1 \in d_1 \exists c_2 \in d_2 \; t.q. \; c_1 \Rightarrow c_2$$

Un ensemble de contraintes correspond la disjonction des contraintes qu'il contains, donc reprsente encore un espace de valeurs dans l'algbre χ. Dans ce sens, un ensemble de contraintes est "meilleur" si l'espace qu'il dnote est plus large ou, en d'autres termes, si l'espace qu'il dnote "couvre" les autres.

4.2.2 valuation des positions terminales

La fonction d'valuation des noeuds terminaux $h : TPOS \to D$ est dfinie par :

$$h\,(A, c) = \emptyset$$
$$h\,[\diamondsuit, c] = c$$

o $\diamondsuit$ est le but vide.

4.2.3 Fonctions des joueurs

Les Blancs fonts l'union ensembliste, tandis que les Noirs fonts la conjonction des contraintes (i.e. l'intersections des espaces dnots) tendue aux ensembles de contraintes.

$$\uparrow, \downarrow : D \to D$$
$$\uparrow = \cup$$
$$\downarrow = \wedge$$

o $\forall d_1, d_2 \in D \qquad d_1 \wedge d_2 = \{c_1 \wedge c_2 \mid c_1 \in d_1,\ c_2 \in d_2\}$.
Les deux fonctions sont videmment monotones pour l'ordre dfinit sur D.

4.2.4 Heuristiques pessimiste et optimiste

Pour toute position non terminale $\pi = (A, c)$ ou $\pi = [G, c]$ les fonctions d'approximations sont dfinies par :

$$h_{pess}(\pi) = \emptyset$$
$$h_{opt}(\pi) = \{c\}$$

Il s'agit bien de heuristiques monotones : soit π une position non terminale quelconque, partir de laquelle le jeu peut voluer dans l'ensemble des positions $\{\pi_1, \pi_2, \pi_3\}$, alors :

- l'union ou la conjonction des $h_{pess}(\pi_i)$ ne peut en aucun cas tre infrieure $h_{pess}(\pi)$ qui est l'ensemble vide
- l'union ou la conjonction des $h_{opt}(\pi_i) = \{c_i\}$ ne peut en aucun cas tre suprieure $h_{opt}(\pi) = \{c\}$ puisque, par dfinition des rgles du jeu, soit $c_i = c$ (π est une position de l'Opposant), soit $c_i = c \wedge c'_i$ (π est une position du Joueur et π_i correspond l'application d'une rgle logique contenant les contraintes c'_i). Dans les deux cas, l'union et la conjonction des $\{c_i\}$ sont couvertes par $\{c\} = h_{opt}(\pi)$

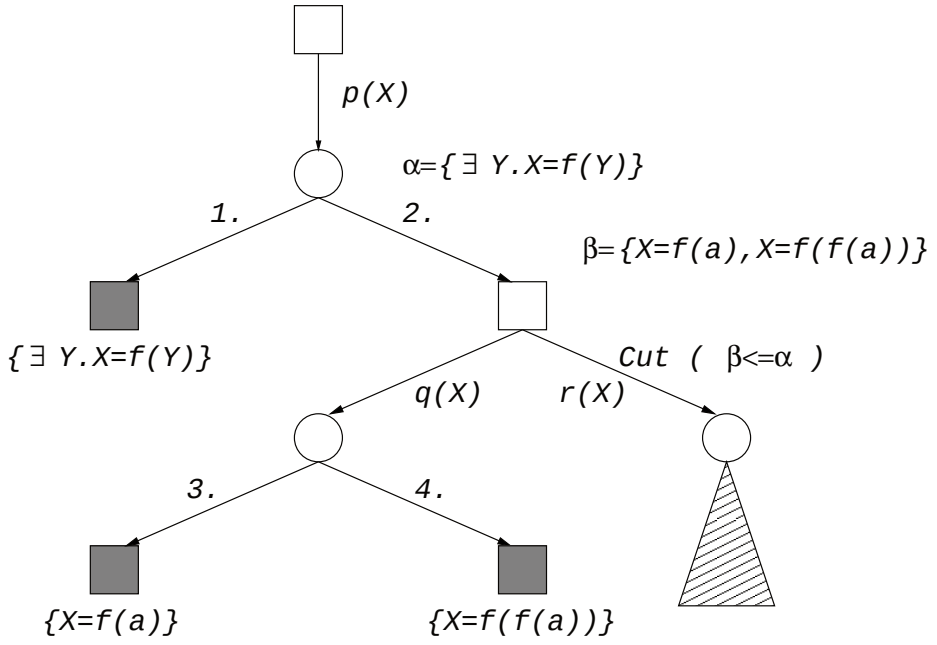
4.3 Exemple

Considerons le programme logique (classique) :

```
1.  p(f(Y)).
2.  p(X) :- q(X),r(X).
3.  q(f(a)).
4.  q(f(f(a))).
5.  r(X) :- ...
```

et, en ce qui concerne les rponses d'un but, considerons de travailler avec les *ex-equations* (voir [Marriot et al.]) au lieu des substitutions idempotentes.

Il est simple de vrifier que les fonctions $\uparrow = \cup$ et $\downarrow = \wedge$ associs aux joueurs vrifient les hypothses du thorme de correction d'Alpha-Bta, qui peut donc tre utilis pour le calcul. Dans l'exemple, avec le but $p(X)$ nous pouvons observer que aprs avoir utilis la rgle 1. et obtenus la rponse $\exists Y.X = f(Y)$ il est inutile de tenter d'avoir mieux en utilisant la rgle 2. Effectivement la rgle 2. oblige trouver des rponses pour $q(X)$, qui seront $\{X = f(a), X = f(f(a))\}$. Puisque l'intersection de contraintes (l'*mgu* entre ex-equations) rend un rsultat qui est infrieur ou gal ses arguments, nous pourrons pas exprer avoir quelque chose de mieux que $\{X = f(a), X = f(f(a))\}$. Et sans avoir calculer les reponses pour $r(X)$ nous pourrions observer que la rgle 2. ne pourra donner mieux que ce qu'on avait deja obtenu per ailleur, avec la rgle 1. Ce raisonement peut expliquer le comportement de la procedure *AlphaBeta* qui coupera le coup correspondant au sous-but $r(X)$ et terminera avec la rponse $\{\exists Y.X = f(Y)\}$:

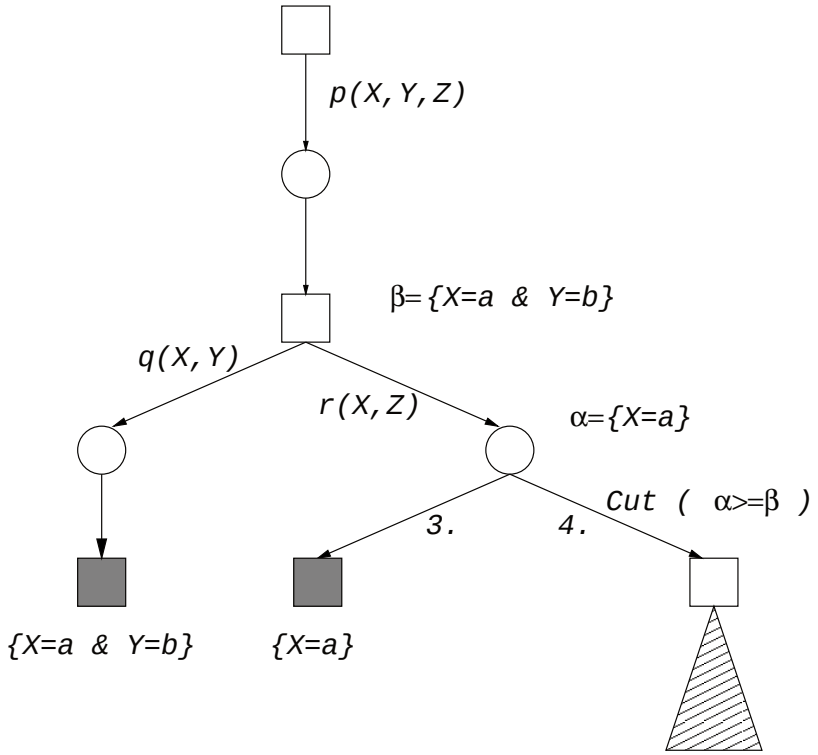


L'exemple qui suit montre le cas des coupures dans les noeuds du joueur :

```
1.  p(X,Y,Z) :- q(X,Y),r(X,Z).
2.  q(a,b).
3.  r(a,Z).
4.  r(X,Z) :- ...
```

Pour le but $p(X, Y, Z)$ on utilisera la rgle 1. Aprs avoir valu le sous-but $q(X, Y)$ qui affecte la variable $X = a$ on utilisera la rgle 3. pour rsoudre $r(X, Z)$. Etand donn la rgle 3. nous pourrons dej assur une reponse compatible avec $X = a$ et, en plus, qui n'affecte pas la variable Z. En d'autre terme nous ne pourrions pas demander plus la rsolution de $r(X, Z)$ etant donn que $X = a$ sera oblig par la

rsolution de $q(X, Y)$. La rsolution de $r(X, Z)$ nous donne deja le maximum compte tenu du contexte et donc on peut arreter la rsolution sans prendre en compte la rgle 4. Encore une fois ceci ce traduit, en considerant l'execution d'*AlphaBeta*, en disant qu'il survient une condition d'elagage $\alpha \geq \beta$:



References

[1] S. Abramsky and R. Jagadeesan. Games and full completeness for multiplicative linear logic. *The Journal of Symbolic Logic*, 59(2):543–574, 1994.

[2] A. Blass. A game semantics for linear logic. *Annals of Pure and Applied Logic*, 56:pages 183–220, 1992.

[3] P. Curien and H. Herbelin. Computing with abstract bohm trees. 1996.

[4] R. Di Cosmo, J.-V. Loddo, and S. Nicolet. A game semantics foundation for logic programming. In C. Palamidessi, H. Glaser, and K. Meinke, editors, *PLILP'98*, volume 1490 of *Lecture Notes in Computer Science*, pages 355–373, 1998.

[5] A. Joyal. Free lattices, communication and money games. *Proceedings of the 10th International Congress of Logic, Methodology, and Philosophy of Science*, 1995.

[6] F. Lamarche. Game semantics for full propositional linear logic. *Proceedings of the 10th Annual IEEE Symposium on Logic in Computer Science*, pages 464–473, 1995.

[7] V. D. P. Baillot and T. Ehrhard. Believe it or not, AJM's games model is a model of classical linear logic. *Proceedings of the 12th Annual IEEE Symposium on Logic in Computer Science*, pages pages 68–75, 1997.

[8] J. von Neumann. Zur Theorie der Gesellschaftsspiele. *Mathaematische Annalen*, (100):195–320, 1928.