A GENERALIZATION OF COMBINATORIAL GAME THEORY AND ITS APPLICATION TO CONSTRAINED LOGIC PROGRAMMING

JEAN-VINCENT LODDO

LABORATOIRE PREUVES PROGRAMMES ET SYSTÈMES (PPS)
UNIVERSITÉ PARIS 7, FRANCE

Abstract. Game theory had, in its origins, a vocation for social and economic sciences, with disparate applications, for example in the processing of medical data. It is perceived today as a very general paradigm of concepts and techniques, whose potential still remains to be discovered in computer science. In this thesis we study a particular branch, combinatorial game theory (with two players), in order to profit from it in the very active field of formal semantics of programming languages.

Within a game, we can separate the syntactic aspect, inherent in the possible outcomes of matches, from the semantic aspect, inherent in the forecasts over the winning player and, possibly, inherent in a quantification of profit, in terms of a stake, like wealth or prestige. To model the concept of profit, the selected structure of evaluation should not necessarily be the structure of booleans (*win* or *loose*), or that of natural or relative numbers. It is enough for this structure to verify some properties, rather weak, guaranteeing the existence of a semantics, even when the play gives rise to infinite matches, as in the game of *bisimulation* between concurrent processes, or as in the game of *logic programming*.

In this work, we study the semantic characterization of a logical language (with or without constraints) in terms of a game with two players. The effect of such an interpretation, beyond the intuitive model whose didactic value would deserve to be examined, makes it possible to reuse *Alpha-Beta,* one of the most celebrated algorithms in combinatorial game theory, as a resolution engine for logical languages. The recent and spectacular results obtained by Chess programs (like the defeat of the world champion Kasparov against the IBM program Deep Blue) testify to a kind of artificial intelligence acquired by these programs that might be transposed and exploited in the resolution of logical languages. The resolution of existential conjunctive goals in a first-order theory of Horn clauses provides an interesting case. Indeed, the ability of Alpha-Beta to simplify calculation or, in other words, to prune the uninteresting paths, is not closely related to a particular type of game or profit, but to a well-chosen set of algebraic properties, satisfied by the game of logical programming. The correction of Alpha-Beta is formally proven for a very wide variety of structures. In this way, the computed values could be natural numbers, as in the case of Chess, or substitutions or constraints, as in the case of logical languages.

1. Introduction

Since many years now, the semantics based on games have found ever-growing applications in computer science and, more specifically, in the study of the semantics of programming languages. Perhaps the most outstanding applications of game theory concern the ubiquitous introduction of chance in algorithms (cf. see [Motwani & Raghavan, 1995] for an eloquent introduction), not to mention that of programs that "play" strategic games. The expressivity of the games-based approach as well as the flexibility of the concepts used give the theory a wider range of applicability.

1.1. Chance and adaptation in computer science.

The basic idea behind random algorithms consists in regarding the programmer as a player having several deterministic algorithms from which to choose (its *pure* strategies), competing against and opponent, the program environment, and whose strategies denote the possible values (or the classes of values, for that matter) of the program inputs. What is on stake here is either the execution time, the quality of the solution, the amount of memory used, or any other measurable property of the performance of the running program. In this sense, an optimal strategy (also called *mixed strategy*), will correspond to defining an algorithm that dynamically chooses, in a random way or in accordance to a well-established law of probabilities, amongst the deterministic methods at its disposal. For instance, all variants of the famous *quicksort* algorithm that choose in a deterministic way the position of the pivot perform quite badly (having a quadratic complexity), under certain permutations of the inputs. On the other hand, the random quicksort "discovered" at the end of the 70s (cf. [Sedgewick, 1978]), is in exact correspondence to a naïve version of the mixed strategy (as it is far from being optimal), assigning the same probability of occurrence to all the deterministic quicksort's, each of which corresponds to a fixed position of the pivot. We may clearly have the impression that this naïve strategy could be optimal if all the permutations of the inputs were equally probable. However, the theory of games allows for the definition (at least in principle) of optimal algorithms adapted to every known probability distribution of the inputs. Moreover, the devising of programs that adapt themselves (during execution) to the whims of the environment, is deemed possible by learning algorithms (cf. for example, see [Owen, 1968]).

The idea of methods that specialize themselves as a result of experience is not new. It reminds us, quite on the contrary, of one of the first goals of the game theory of the beginnings to the twentieth century: the modeling of economics, of notions like *supply* and *demand*; in other words, the modeling of two actors having opposed interests, but nonetheless converging to mutual adaptation. In computer science, this kind of approach, where a given specification is engaged in a 'contest' against the data that will be run by its implementation, applies as well to any collection of equivalent programs that behave, more or less, in accordance to the environment (cf. see, amongst others, [Welsh, 1983], [Karp & Rabin, 1987], [Karp, 1991], [Motwani & Raghavan, 1995]).

1.2. **Player robots.** We may recall the reader at this point of the recent and prominent success obtained in the realm of artificial intelligence by programs that "play" Checkers, Chess, Othello, or some other classic strategic game. The victory of IBM's program Deep Blue against the world champion of Chess, Garry Kasparov, in may 1997, stands out as the landmark achievement of forty years in algorithm research and fine-tuning of specialized architectures. This success has, in some way, revived the hopes raised by artificial intelligence in the past, even if, in our opinion, it has only proved that games are relatively easy to model (because computers excel at it). It does not prove, however, that computers have suddenly become more intelligent.

1.3. **The semantics of programming languages.** The study of the semantics of programming languages has also taken advantage of powerful and intuitive game-theoretic concepts, in particular those of *combinatorial* game theory. Indeed, after the preliminary works of Lamarche (cf. [Lamarche, 1995]), people like Blass (cf. [Blass, 1992]) and Joyal (cf. [Joyal, 1995]), who at the beginning of the 90s showed that a certain category of strategies was cartesian-closed (and so represented a model of λ, amongst others), paved the way to the first completely abstract semantic models of functional languages (PCF) and imperative languages (Idealized Algol). More recently, experts in linear logic, like Baillot, Danos, Ehrhard, Harmer and Regnier (cf. [Danos & Harmer, 2000] and [Baillot et al., 1997]), to cite a few, have worked on the relationship between games and the geometry of interaction; Curien and Herbelin (cf. [Curien & Herbelin, 1998]), on the other hand, have shown that a certain class of classical abstract machines could be interpreted as games.

In the references to the literature cited above, many words are used that belong to a games-based vocabulary (such as *player*, *move*, *match*, *arena*, *strategy*, and so on), although both intuition, or the many results which are proper to the more traditional approach to game theory, are actually not completely exploited. We can find in other people's work, quite on the contrary, key concepts like those of *rational player* and *gain*, adopting a more accomplished, as well as more fundamental, approach to games. For example, in automata theory, focused towards the specification and verification of programs, certain problems related to the modeling of infinite languages are addressed using games (see, for instance, [Perrin & Pin, 2001]). Also, within the framework of concurrent programming, the different *bisimulation* relations between processes, introduced using a special definition technique and needing the notion of the greatest fixed-point, can be reinterpreted, in a more intuitive fashion, as two-player games: if a player wins, the processes are bisimilar; otherwise, they are not (cf. [Stirling, 1997] and [Loddo & Nicolet, 1998]).

Logic programming may as well be dealt with by using combinatorial games (cf. [Loddo & Nicolet, 1998] and [Di Cosmo et al.,1998]). In a sense, it would not be wrong to assert that logic programming can be completely restated in terms of games. Indeed, two players are placed in opposition: each goal to satisfy corresponds to the initial position of a match in which one of the players attempts to prove that the goal can be satisfied in the logic program (a sort of checkers board), while the other does everything to prevent it. We shall see that the answers corresponding to a goal in a logic program are in exact match with the gain of the player in the position defined by the pair formed by the goal and the program.

The main goal of this work is, on the first place, to present in due rigour all the aspects, both *syntactic* and *semantic* of combinatorial games, in order to subsequently apply the formal tools here developed to the paradigm of logic programming. We shall adapt traditional notions, like that of *heuristics* or *value* of a game, to the framework of *infinite* games, with the aim of constructing a theory capable of describing not only the well-known games of Checkers or Chess, but also some unexpected ones, like logic programming. The flexibility of the games-based paradigm as well as the fact that it is fairly simple to model in a computer, encouraged us to devise a general theory, new to our knowledge, with the hope of using it for other applications.

We shall later see how the famous Alpha-Beta algorithm, originally designed for traditional evaluation structures (the natural numbers), can be adapted to other evaluation structures, like that of games in logic programming. We first generalize the Alpha-Beta algorithm, abstracting away from any particular evaluation structure, to then restate it in the more formal style of a big-step operational semantics. A correctness theorem will then give a detailed account of the quality of the answers obtained. With respect to our other work in [Di Cosmo & Loddo, 2000], this results broadens the application spectrum to a more extended set of evaluation structures and, also, to games where players are not forced to play alternate moves (ie., games where the players do not leave, after each move, the turn to their opponents). This shows our intention of applying the algorithm not only to games in logic programming (pure or with constraints), but also to games in logic programming with *negation* which, nonetheless, will not be studied in this article. Searching for the best winning strategy in games like Chess, the Alpha-Beta resolution engine brings its "intelligence" into the picture, much to the benefit of overall performance. Like the famous *branch&bound* algorithms in operational research, it will, in its own, be capable of pruning the uninteresting paths in the tree of all possible outcomes of the game.

2. Background and notations

In the sequel, we shall be using the following conventions. We note $\mathbb{N}$ for the set of natural numbers; $\mathbb{N}_+$ will denote the positive natural numbers. The order relation on natural numbers will be noted $\leq_{\mathbb{N}}$. The *partial functions* from a domain A to values in B will be noted $f : A \rightharpoonup B$. Let $\mathrm{dom}(f) = \{a \in A \mid \exists b. (a, b) \in f\}$ be the *domain* of a partial function. We shall write $\wp^A$ for the *powerset* of a set A, $\wp^A_F$ for the finite subsets, and $\wp^A_\omega$ for the enumerable subsets. The set of *directed* subsets of a partial order $(D, \leq)$ will be noted $\Delta(D, \leq)$. We shall write A^* and A^ω for the set of *finite* and *infinite sequences*, respectively, on on an alphabet A. If $u \in A^*$ is a sequence $u = a_1..a_n$, the notation $x \in u$ will be used to mean $\exists i \in \mathbb{N}. x = a_i$. The *prefix* order between sequences will be noted $\sqsubseteq$. We shall note $\leq^*$ for the *product extension* of any binary relation $\leq$ to sequences, i.e., $x_1..x_n \leq^* y_1..y_m$ iff $n = m \ \wedge \ \forall i{=}1..n.\, x_i \leq y_i$.

3. Syntax of a game

In combinatory game theory, the notion of game we are most familiar with, is a very general one, as found in the context of games of chance. From an initial position, there is a sequence of player moves, where each player chooses amongst many possibilities. At some point in this sequence, chance (or random) moves may occur, as may be the drawing of a dice or the shuffling of a pack of cards. Examples of such are games like Chess, where no random moves take place, Bridge, where chance plays a more prominent role, and Roulette, a game of chance where there is no strategy at all. We shall be dealing with games of the first sort, like Chess, a game that may be explained by the rational choices of its players and by the respective convergence or divergence of their objectives.

We shall distinguish two aspects of these sort of games: a first one that we shall call *syntactic*, characterized by the way in which the game may use all its possible paths or matches; the second one, that we shall call *semantic*, and which concerns the different ways in which the game may be interpreted according to the interests of the players.

3.1. **Arena.** The definition of a game of n players, now widely recognized, places the focus on the tree-like structure of a game: A game is fundamentally the tree given by all the possibilities of playing it; that is to say, the tree of all the possible matches. This presentation is not very efficient in nature, as the size of the trees grows very fast in all real games. That is the reason why, in order to describe a game, we generally give the initial position (or several) and the rules of the game, which allow us to go from one position to the following one. This reminds of the indirect definition of languages in computer science through formal grammar rules, very much like the definition of the structural operational semantics of non-deterministic or concurrent languages. In a process calculus, for instance, the semantics is often expressed through a finite number of logical rules describing the possible state transitions of a system, when the only important thing to us is the global derivation tree of a process, or in other words, the tree of all possible state transitions beginning at its initial state.

Definition 3.1. (SYNTAX OF A TWO-PLAYER COMBINATORY GAME) The *syntax* (or *arena*) *of a two-player combinatory game* (in the sequel, *game syntax* or *game arena*) is a 4-tuple $\mathcal{S} = (\mathbb{P}, \lambda, \rightarrow, \mathbb{P}_{\mathrm{I}})$ where:

- $\mathbb{P}$ is the set of all game positions
- $\lambda : \mathbb{P} \rightarrow \{\text{Player}, \text{Opponent}\}$ is the function that divides the positions into player or opponent positions
- $\rightarrow : (\mathbb{P} \times \mathbb{N}_+) \multimap \mathbb{P})$ is the transition relation representing all the possible moves for every game position. The notation $\pi \xrightarrow{\ell} \pi'$ is used to indicate that the pair $((\pi, \ell), \pi')$ belongs to the relation $\rightarrow$, and that in position π, there exists a possible move, labeled by $\ell \in \mathbb{N}_+$, that drives the game into the new position π'. We require that this relation be locally finite, meaning that only a finite number of possible moves is allowed for every game position:
$$\forall \pi \in \mathbb{P}.\ \exists n \in \mathbb{N}_+.\ \text{s.t. } \pi \xrightarrow{\ell} \pi' \Rightarrow \ell \leq n$$
- $\mathbb{P}_{\mathrm{I}} \subseteq \mathbb{P}$ is the set of initial positions of the game; that is, those from which the game is allowed to start.

Given a game syntax $\mathcal{S}$, the sequence of *moves* beginning at a position π is defined by $\text{Moves}(\pi) = (\ell_1, \pi_1)..(\ell_n, \pi_n)$ iff:

$$\ell_1 <_{\mathbb{N}} \ell_2 <_{\mathbb{N}} .. <_{\mathbb{N}} \ell_n \quad \wedge \quad \forall i{=}1..n.\ \pi \xrightarrow{\ell_i} \pi_i \quad \wedge \quad \pi \xrightarrow{\ell'} \pi' \Rightarrow \exists j.\,(\ell', \pi'){=}(\ell_j, \pi_j)$$

The sequence of *successors* of a position is $\text{Succ}(\pi) = \pi_1..\pi_n$ if $\text{Moves}(\pi) = (\ell_1, \pi_1)..(\ell_n, \pi_n)$. The set of *terminal* positions is $\mathbb{P}_{\mathrm{T}} = \{\pi \in \mathbb{P} \mid \text{Succ}(\pi) = \varepsilon\}$. Its complement set is that of non-terminal positions $\mathbb{P}_{\mathrm{NT}} = \mathbb{P} \setminus \mathbb{P}_{\mathrm{T}}$. The *dual* game syntax is $\mathcal{S}^{\perp} = (\mathbb{P}, \lambda^{\perp}, \rightarrow, \mathbb{P}_{\mathrm{I}})$, where the function that divides the position into classes $\lambda^{\perp}$ is the inverse of λ, i.e. $\lambda^{\perp}(\pi){=}\text{Player} \Leftrightarrow \lambda(\pi){=}\text{Opponent}$.

In a game, it may happen that different moves (denoted by distinct labels), drive the game into the same position. This explains why the transition relation is of type $\rightarrow : (\mathbb{P} \times \mathbb{N}_+) \multimap \mathbb{P}$, and not simply $\rightarrow : \mathbb{P} \times \mathbb{P}$. Another explanation that justifies the existence of labels in the definition above is that, thanks to them, it becomes possible to order the player moves following the standard order for the natural numbers.

3.2. **Game trees.** We shall now formally define the notions of position tree and game tree. In these trees, the nodes are decorated with the positions of the game; the relation $\rightarrow$ gives the connections between nodes. The hypothesis in definition 3.1, requiring that the relation $\rightarrow$ be locally finite, implies that the game trees will be finitely branched.

Definition 3.2. (TREES, CUTS, PREFIXES) Given a set of symbols F, a *tree* t labeled in F is a partial function $t : \mathbb{N}_+^* \rightharpoonup F$, where the domain $\mathrm{dom}(t) = \{u \in \mathbb{N}_+^* \mid \exists e \in F. (u, e) \in t\}$ is not empty and prefix-closed, i.e. $\forall u \sqsubseteq v.\ v \in \mathrm{dom}(t) \Rightarrow u \in \mathrm{dom}(t)$. The elements of $\mathrm{dom}(t)$ are called *addresses*. The set of all trees labeled in F will be noted $\mathrm{IT}(F)$; that of finite trees will be noted $\mathrm{T}(F)$. A tree t' is a *cut* of a second tree t iff $t' \subseteq t$. In a tree t, the *(sequence of) successors* at an address u is the sequence of all pairs of the form (ℓ, e), where $u.\ell \in \mathrm{dom}(t)$ and $t(u.\ell) = e$, ordered using the order relation of the natural numbers, i.e. $\mathrm{Succ}(t, u) = (\ell_1, e)..(\ell_n, e)$ iff $\ell_1 <_{\mathbb{N}} \ell_2 <_{\mathbb{N}} .. <_{\mathbb{N}} \ell_n$, $\forall i.\ t(u.\ell_i) = e_i$, and $\{\ell_1, .., \ell_n\} = \{\ell \mid u.\ell \in \mathrm{dom}(t)\}$. A *leaf* is a pair $(u, e) \in t$ such that $\mathrm{Succ}(t, u) = \varepsilon$. We shall note $\mathrm{Leafs}(t)$ for the set of all leaves of a tree. A tree t' is a *prefix* of another tree t, noted $t' \leq t$, iff $\forall u \in \mathrm{dom}(t').\ u \in \mathrm{dom}(t) \ \wedge\ (\mathrm{Succ}(t', u) = \mathrm{Succ}(t, u) \ \vee\ (u, e) \in \mathrm{Leafs}(t'))$. The *concatenation* of an address u with a tree t, is given by the operation $u.t = \{(u.v, e) \mid (v, e) \in t\}$. The *sub-tree* of t at address $u \in \mathbb{N}_+^*$, is the relation $t|_u \subseteq \mathbb{N}_+^* \times F$, defined by $t|_u = \{(v, e) \mid (u.v, e) \in t\}$. The *childrens* of a tree t are the sequence of subtrees $\mathrm{Childs}(t) = t_1..t_n$ such that $\mathrm{Succ}(t, \varepsilon) = (\ell_1, e_1)..(\ell_n, e_n)$ and $\forall i = 1..n.\ t_i = t|_{\ell_i}$. The *depth* of a tree t is the maximum length of all its addresses, i.e. $\mathrm{depth}(t) = \max_{\leq_{\mathbb{N}}}\{|u| \mid (u, e) \in t\}$, where $|u|$ denotes the length of sequence u. The *regular prefix* of a tree t at level $k \in \mathbb{N}$ is the prefix of t for which all branches are cut at depth k, i.e. $t/^k = \{(u, e) \in t \mid |u| \leq_{\mathbb{N}} k\}$. The *graft* of a tree t_2 at address u onto the tree t_1 is the partial function $t_1 \cdot_u t_2 = \lambda v.\ \mathsf{if}\ (v \sqsubseteq u)\ \mathsf{then}\ t_1(u)\ \mathsf{else}\ t_2(u.v)$.

The choice of representing trees as partial functions (i.e., binary relations, or sets of pairs) allows us to identify the notion of cut with the mathematical notion of subset. In turn, the notion of prefix corresponds to a particular cut discipline: at every address of the tree to prune, the edges are either all retained or altogether cut; there are no half measures. The position tree of a game is a tree where the branches represent the available moves. We shall note $\pi \xrightarrow{u}{}^* \pi'$, where $u \in \mathbb{N}_+^*$, $u = \ell_1..\ell_n$, the reflexive and transitive closure of the relation $\rightarrow$, i.e. :

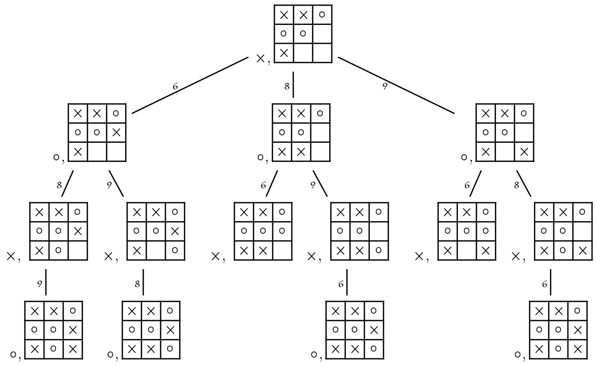
$$\pi \xrightarrow{u}{}^* \pi' \quad \Leftrightarrow \quad (u = \varepsilon \wedge \pi = \pi') \ \vee\ \pi \xrightarrow{\ell_1} \pi_1 \rightarrow .. \rightarrow \pi_{n-1} \xrightarrow{\ell_n} \pi'$$

Definition 3.3. (POSITION TREE, GAME TREE) Given a game arena $\mathcal{S} = (\mathbb{P}, \lambda, \rightarrow, \mathbb{P}_{\mathrm{I}})$, the *position tree* at $\pi \in \mathbb{P}$ is the tree $t_\pi(\mathcal{S}) = \{(u, \pi') \mid \pi \xrightarrow{u}{}^* \pi'\}$. Whenever the arena is implicit in the context, we shall write it more simply as t_π. The *reachable* positions from a position π are given by the image of the function t_π on the set of all possible addresses, i.e. $\mathbb{P}_\pi = t_\pi(\mathbb{N}_+^*)$. If the game has only one initial position ($\mathbb{P}_{\mathrm{I}}$ singleton), the *game tree* is the tree at this unique position. Let $\mathrm{IT}(\mathcal{S}) = \{t_\pi \mid \pi \in \mathbb{P}\}$ denote the set of all the position trees of the game. Amongst these, we shall distinguish the subset of *finite* game trees, that we shall note $\mathrm{T}(\mathcal{S})$. We say that $\pi \in \mathbb{P}$ is a *finite tree position* if $t_\pi \in \mathrm{T}(\mathcal{S})$. We shall write $\mathbb{P}_{\mathrm{FT}}$ for the set of all finite tree positions.

Example 3.4. Tic-Tac-Toe. In this very popular game, the two players alternately put marks on a 3×3 board of cells. Each player has its own mark, generally $\times$ or $\circ$. When the game starts, the board is empty. The goal of the game is to be the first to complete a series of three consecutive marks (either forming a vertical, horizontal or diagonal row). The syntax of the game can be formally described as the 4-tuple $\mathcal{S} = (\mathbb{P}, \lambda, \rightarrow, \mathbb{P}_{\mathrm{I}})$. The set of positions is the cartesian product $\mathbb{P} = \mathrm{Initiative} \times \mathrm{State}$. The positions are, therefore, the pairs where the first component indicates which player's turn it is: $\mathrm{Initiative} = \{\times, \circ\}$; the second component gives the state of the board: $\mathrm{State} = \{1, 2, 3, 4, 5, 6, 7, 8, 9\} \rightarrow \{\emptyset, \times, \circ\}$. We have conventionally numbered the cells of the board as shown below:

1	2	3
4	5	6
7	8	9

FIGURE 3.1. A tree of Tic-Tac-Toe



For this particular game, we respectively assign to player and opponent the marks $\times$ and $\circ$. The position-classifying function $\lambda : \mathbb{P} \to \{\text{Player}, \text{Opponent}\}$ is then defined by: $\lambda(\times, d) = \text{Player}$ and $\lambda(\circ, d) = \text{Opponent}$, for all $d \in \text{State}$. The relation that expresses the possible moves $\to \subseteq \mathbb{P} \times \mathbb{N}_+ \times \mathbb{P}$ is defined as the smallest relation satisfying the rules:

$$\frac{d(i) = \emptyset \quad \neg\text{terminal}(d)}{(\times, d) \xrightarrow{i} (\circ, d[i \mapsto \times])} \qquad \frac{d(i) = \emptyset \quad \neg\text{terminal}(d)}{(\circ, d) \xrightarrow{i} (\times, d[i \mapsto \circ])}$$

The predicate terminal(d) holds whenever three consecutive squares (the *directions*) have been successfully marked by the same player. Formally:

$$\text{Directions} = \{(1,2,3), (4,5,6), (7,8,9), (1,4,7), (2,5,8), (3,6,9), (1,5,9), (3,5,7)\}$$
$$\text{terminal}(d) \quad \Leftrightarrow \quad \exists \vec{x} \in \text{Directions}.\ d(x_1) = d(x_2) = d(x_3) \ \wedge \ d(x_1) \neq \emptyset$$

Finally, the set of initial positions:

$$\mathbb{P}_{\mathrm{I}} = \{(\times, d_v) \in \mathbb{P} \mid d_v = \{(i, \emptyset) \mid 1 \leq i \leq 9\}\}$$

models the situation in which, standing before an empty board, it is the player that owns the $\times$ mark, Player, who must take the initiative to begin the match. Despite the simplicity of the game, we cannot give a picture of the complete Tic-Tac-Toe tree. The count of its nodes lies in the order of 9!. In figure 3.1 we content ourselves with showing the tree at a given position (that is, a branch of the game tree), close to the end of the match, and such that $\pi = (\times, d)$ where $d = \{(1, \times), (2, \times), (3, \circ), (4, \circ), (5, \circ), (6, \emptyset), (7, \times), (8, \emptyset), (9, \emptyset)\}$, i.e. :

×	×	∘
∘	∘	
×		

The possible moves at position π correspond to marking one of the three cells that are still empty: 6, 8 or 9. The tree at position π is equal to the set $\{(\varepsilon, \pi)\} \cup 6.t_{\pi_6} \cup 8.t_{\pi_8} \cup 9.t_{\pi_9}$, where π_6, π_8 and π_9 are the target positions corresponding, respectively, to moves 6, 8 and 9. In other words, if label π decorates the root of t_π, the decoration of the rest of the tree is given in terms of its subtrees: $t_\pi(6.u) = t_{\pi_6}(u)$, $t_\pi(8.u) = t_{\pi_8}(u)$ and $t_\pi(9.u) = t_{\pi_9}(u)$, and so on, recursively.

3.3. **Strategies.** One of the most important notions in the theory of games is certainly that of *strategy*. The intuitive meaning of a strategy is that of a plan for playing game matches. We may think of a player that says to herself: if this happens, I should do that, but if this other thing happens, I should do that, and so on. In combinatory games, a strategy for a player may be represented by a particular cut of the game tree: At each node for which it is her turn to play, she decides upon a move, so there is only one descendant left.

Definition 3.5. (STRATEGIES) Given a game arena $\mathcal{S} = (\mathbb{P}, \lambda, \to, \mathbb{P}_{\mathrm{I}})$, a *global strategy* for the game is a function $\sigma : \mathbb{P}_{\mathrm{NT}} \to \mathbb{P}^*$ such that for every position $\pi \in \mathbb{P}_{\mathrm{NT}}$:

$$\sigma(\pi) = \begin{cases} \pi' & \text{where } \pi' \in \mathrm{Succ}(\pi) \quad \text{if } \lambda(\pi) = \mathrm{Player} \\ \mathrm{Succ}(\pi) & \phantom{\text{where } \pi' \in \mathrm{Succ}(\pi)} \quad \text{if } \lambda(\pi) = \mathrm{Opponent} \end{cases} \qquad (\star)$$

A *local strategy* for a position $\bar{\pi} \in \mathbb{P}$ is a function $\sigma : \mathbb{P}_{\bar{\pi}} \cap \mathbb{P}_{\mathrm{NT}} \to \mathbb{P}^*$ verifying the condition $(\star)$ at (only) every position $\pi \in \mathbb{P}_{\bar{\pi}} \cap \mathbb{P}_{\mathrm{NT}}$. The set of global strategies for the game $\mathcal{S}$ will be noted $\Sigma(\mathcal{S})$. The set of local strategies for a position π will be noted $\Sigma(\mathcal{S}, \pi)$.

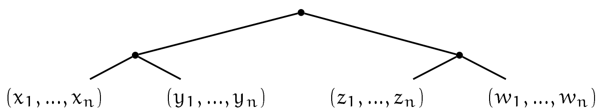
The corresponding notions of counter-strategies, or opponent strategies, are obtained by duality: a *global counter-strategy* (resp. *local*) for a game $\mathcal{S}$, is a global (resp. local) strategy in the dual game arena $\mathcal{S}^{\perp}$.

Definition 3.6. (Game with a strategy) Given a game arena $\mathcal{S} = (\mathbb{P}, \lambda, \to, \mathbb{P}_{\mathrm{I}})$ and a strategy (either global or local) σ, the *game with strategy* σ is defined by the syntax $\mathcal{S}_\sigma = (\mathbb{P}, \lambda, \to_\sigma, \mathbb{P}_{\mathrm{I}})$, where the transition relation is $\to_\sigma = \{((\pi, \ell), \pi') \mid \pi \in \mathrm{dom}(\sigma) \Rightarrow \pi' \in \sigma(\pi)\}$.

A game with a counter-strategy θ, noted $\mathcal{S}^\theta$, is the dual of the game with strategy θ, i.e. $\mathcal{S}^\theta = (\mathcal{S}^\perp)_\theta$. A *match* is a game $\mathcal{S}_\sigma^\theta$ with a strategy σ, a counter-strategy θ and a single initial position π_1. It will therefore be possible to give a representation for it simply in terms of a sequence of game positions: $\pi_1 \to \pi_2 \to \pi_3 \to \pi_4 \to \ldots \to \pi_k \to \ldots$.

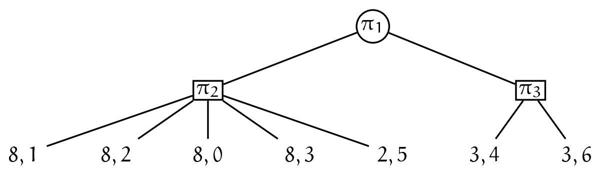
4. Semantics of a game

Normally, at the end of a match, there is either a reward or, rather on the contrary, a stake that players should honor (either in the form of money, prestige, or satisfaction), and that depends on the evolution of the match. We may attempt a formalization of such a situation with a function assigning a cost, traditionally referred to as *payoff*, to each terminal position of the game. For the more general case of n players, the payoff would be an n-tuple with the costs of each participant as components.



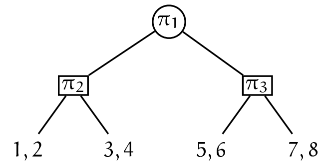
More often the set of values used to represent a single payoff is very simple. For example, the set of booleans $\{0, 1\}$: in words, *won* (1) or *lost* (0). Or, indeed, as is the case in Tic-Tac-Toe, Checkers or Chess, the set $\{-1, 0, 1\}$, with the meaning *won* (1), *lost* (-1) or *draw* (0). For some card games, the set of integers $\mathbb{Z}$ may become necessary to translate the amount of money the players may have won or lost.

Whatever the domains $D_1, .., D_n$ used for the individual payoff of n players may be, the payoff function assigns a cost in the cartesian product $D = D_1 \times .. \times D_n$. Games are always approached on the basis of a fundamental presumption of game theory, the *rationality hypothesis*, according to which the players will choose, at all times during the match, the best move. Under this hypothesis, players are not only rational, but also *selfish*: they incarnate their only interests, which are *independent* from the interests of the other players. To put it differently, each participant will do her *best*, with respect to her payoff, and with the knowledge that her opponents will behave in the same way in their own turns. From a mathematical viewpoint, that corresponds to having the domains equipped with a preference relation $\leq$ and of an operation, that we shall call max, allowing us to select the best cost, and therefore, the best available move. This is why we shall be dealing with structures of type $(D_1, \leq_1, \max_1), \ldots, (D_n, \leq_n, \max_n)$. To illustrate this hypothesis, cornerstone to the theory, let us assume we have a game with two players (where $D_1 = D_2 = \mathbb{N}$), behaving as shown:

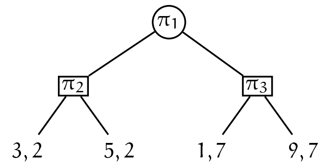


A *player* that is in position π_1, can choose between two moves, either the move that drives the game into the *opponent*'s position π_2, or the move that drives the game into, once again and always, the *opponent*'s position π_3. What is the most profitable choice in position π_1? We might think it is the first one, because most of the moves available in that position for the *opponent* will lead the *player* to her maximal gain, namely 8. However, the *player* cannot allow himself to make such a choice; he must know that the *opponent* does not play at random, trying also to maximize her own gain. If the *opponent* found herself in position π_2, she would not hesitate an instant and would go for the most profitable move, from which she will win 5, leading the *player* to win 2 instead of 8. The rationality hypothesis implies, therefore, that the *player* will choose the second move, allowing him to ensure a gain of 3.

It may accidentally happen that the interests of the players converge, without necessarily implying that the hypothesis requiring that the players be rational and selfish be violated:



In this example, the match ends up with a payoff of $(7, 8)$, a payoff that is maximal for both players. If the accidental convergence of interests is not a cause for trouble, a second aspect may complicate the evaluation of a game. If a certain number of moves are equivalent at some point for the player whose turn is to play (the *active* player), but they are not equivalent for the player that will have to assume the consequences of the other player's play (the *passive* player), we shall have some difficulties to estimate the value of the payoff of the passive player. Let us consider the following example:



The *player* is faced against a difficult choice: If she goes for the left move, the *opponent* will find herself before two available moves that both equal 7. What will the *opponent* do? Will she be *indifferent* and choose at *random*? Will he be *good* enough to help the other player? Will she be *bad* and try to profit at any occasion presented to her to punish her opponent? In other words, if it is clear enough that the *opponent* will max her component part, what will she do to the other player? Will she perform an average (indifferent), a max (good), or a min (bad) ?

This type of game has, therefore, several extreme answers: those having to do with the *worst case* (bad opponent), the *best case* (good opponent), or an intermediate answer, the *average case* (indifferent opponent). We are dealing here with a type of game that may be approached by the theory, far more developed, of *repeated* or *stochastic* games. In these games, the players repeat the matches and, in so doing, are either good or bad (i.e., cooperative, or the other way round), depending on the history of the observed behavioral experience of their opponents.

We shall not be dealing here with this type of game, where the interests may converge. However, in our quest of generalization, we shall found our inspiration in the so-called *zero-sum* games, for which the *sum* of the individual payoffs of the players is *zero*. Indeed, if the sum is zero, one payoff must necessarily be the *inverse* of the other payoff, much like their interests. From a mathematical point of view, we could say that for the case of two players, the two structures, $(D_1, \leq_1, \max_1)$ and $(D_2, \leq_2, \max_2)$ have the same domain $D_1 = D_2$ but with the preference order reversed: $\max_1 = \min_2$ et $\max_2 = \min_1$. For zero-sum games, we can clearly content ourselves with giving the first components of the payoff. This proves particularly useful for two-player $(n = 2)$ games. Indeed, if the second component of the payoffs is always the inverse of the first, we can silently forget about it and concentrate only on the first. We shall therefore give a unique evaluation domain, that we shall call D, with a unique preference order, that of *player*, that we shall note $\leq$, as well as two selection operations: max for *player* and min for *opponent*. For this reason, the zero-sum games with two players are also referred to as *min-max* games.

4.1. **Traditional evaluation of finite games.** In order to evaluate a finite game with syntax $\mathcal{S} = (\mathbb{P}, \lambda, \rightarrow, \mathbb{P}_{\text{I}})$, the technique that is generally put to work consists of decorating the (finite) game tree that unfolds from a position $\pi \in \mathbb{P}$ with the functions min and max for the non-terminal nodes, and with the *payoff* of the terminal position for the tree leaves. Imagine then a sort of backward-propagation mechanism of the values towards the root by performing the operations assigned to the tree nodes. From a mathematical viewpoint, this corresponds precisely to an inductive definition, where the computation of the value of a node is dependent on the computation of the values of its children. We assume that there is always a function $p : \mathbb{P}_{\text{T}} \rightarrow \mathbb{N}$ giving the payoff (i.e., an natural number, for example) at any terminal position. On the basis of this function, we can build the whole edifice of the evaluation of a finite game. Indeed, what we traditionally call the *value* of game is a function $val : \mathbb{P} \rightarrow \mathbb{N}$, defined as follows:

$$val(\pi) = \begin{cases} p(\pi) & \text{if } \pi \in \mathbb{P}_{\text{T}} \\ \max\ val(\pi_1)..val(\pi_n) & \text{if } \mathrm{Succ}(\pi) = \pi_1..\pi_n,\ \lambda(\pi) = \text{Player} \\ \min\ val(\pi_1)..val(\pi_n) & \text{if } \mathrm{Succ}(\pi) = \pi_1..\pi_n,\ \lambda(\pi) = \text{Opponent} \end{cases} .$$

This definition is, of course, restricted to the case of finite games, where the leaves of the tree comprise the base case of the inductive evaluation procedure. Nevertheless, the main motivation for this section is that of giving a meaning or, in other words, a value to all types of game, finite or infinite.

4.2. **Semantics of finite games.** As we proposed ourselves to abstract away from any unnecessary details, to focus only on the required properties needed for the definition of a suitable game semantics, we shall begin by presenting those elements that are necessary for finite games. Let *game algebra* refer to any algebra of the form $\mathcal{D} = (\mathsf{D}; \uparrow, \downarrow : \mathsf{D}^* \rightarrow \mathsf{D})$. The operations $\uparrow : \mathsf{D}^* \rightarrow \mathsf{D}$ and $\downarrow : \mathsf{D}^* \rightarrow \mathsf{D}$, of variable arity, generalizing max and min, are called respectively *player function* and *opponent function*. A game algebra allows, together with a *payoff* function $p : \mathbb{P}_{\text{T}} \rightarrow \mathsf{D}$ for the evaluation of terminal nodes, to assign a value in D to every finite game.

Definition 4.1. (Value of a finite tree position) Given a game arena $\mathcal{S} = (\mathbb{P}, \lambda, \rightarrow, \mathbb{P}_{\text{I}})$, a game algebra $\mathcal{D} = (\mathsf{D}; \uparrow, \downarrow : \mathsf{D}^* \rightarrow \mathsf{D})$ and a payoff function $p : \mathbb{P}_{\text{T}} \rightarrow \mathsf{D}$, the value of finite tree positions is the function $val : \mathbb{P}_{\text{FT}} \rightarrow \mathsf{D}$ defined by:

$$val(\pi) = \begin{cases} p(\pi) & \text{if } \pi \in \mathbb{P}_{\text{T}} \\ \uparrow val(\pi_1)..val(\pi_n) & \text{if } \mathrm{Succ}(\pi) = \pi_1..\pi_n,\ \lambda(\pi) = \text{Player} \\ \downarrow val(\pi_1)..val(\pi_n) & \text{if } \mathrm{Succ}(\pi) = \pi_1..\pi_n,\ \lambda(\pi) = \text{Opponent} \end{cases}$$

Indeed, whenever the game is finite($\mathbb{P}_{\text{FT}} = \mathbb{P}$), the *val* function can be used to evaluate all game positions.

4.3. **Semantics of arbitrary games.** We shall now consider the definition above as a *property* to verify if a function should be regarded as a semantics for positions.

Definition 4.2. (Semantics for positions) A *semantics for positions* of a game syntax $\mathcal{S} = (\mathbb{P}, \lambda, \rightarrow, \mathbb{P}_{\text{I}})$ in a game algebra $\mathcal{D} = (\mathsf{D}; \uparrow, \downarrow)$, is an arbitrary function $[\![.]\!] : \mathbb{P} \rightarrow \mathsf{D}$ such that:

$$\forall \pi \in \mathbb{P}_{\text{NT}}.\quad [\![\pi]\!] = f_\pi [\![\pi_1]\!]..[\![\pi_n]\!]$$

where $\pi_1..\pi_n = \mathrm{Succ}(\pi)$ and $f_\pi = \uparrow$ if $\lambda(\pi) = \text{Player}$; otherwise $f_\pi = \downarrow$.

Whenever a game is finite, the only semantics possible is the *val* function, defined inductively as shown below. However, if the game is not finite ($\mathbb{P} \neq \mathbb{P}_{\text{FT}}$), there are several possible semantics. We shall look at a way of constructing one by using the notion of *game prefix* and *correct heuristics*.

4.4. **Constructing a semantics for positions.** The procedure of inductive evaluation may be generalized to any type of tree labeled by game positions. It suffices to have available two evaluation functions for positions for the tree leaves only. A function, the *payoff*, will be used if the label π of the leaf (u, π) is terminal in the game (i.e. $\pi \in \mathbb{P}_{\text{T}}$); the *heuristics* will be used if the label π is non-terminal (i.e. $\pi \in \mathbb{P}_{\text{NT}}$).

Definition 4.3. (Min-max backward-propagation) Given a game arena $\mathcal{S} = (\mathbb{P}, \lambda, \rightarrow, \mathbb{P}_{\mathrm{I}})$, a game algebra $(\mathsf{D}; \uparrow, \downarrow)$, the *min-max backward-propagation* is a function $v : (\mathbb{P}_{\mathrm{T}} \rightarrow \mathsf{D}) \rightarrow (\mathbb{P}_{\mathrm{NT}} \rightarrow \mathsf{D}) \rightarrow \mathrm{T}(\mathbb{P}) \rightarrow \mathsf{D}$ defined as follows:

$$v_{p,h}(t) = \begin{cases} p(\pi) & \text{if } t = \{(\varepsilon, \pi)\},\ \pi \in \mathbb{P}_{\mathrm{T}} \\ h(\pi) & \text{if } t = \{(\varepsilon, \pi)\},\ \pi \in \mathbb{P}_{\mathrm{NT}} \\ \uparrow v_{p,h}(t_1)..v_{p,h}(t_n) & \text{if } \mathrm{Childs}(t) = t_1..t_n,\ \ \lambda(t(\varepsilon)) = \mathrm{Player} \\ \downarrow v_{p,h}(t_1)..v_{p,h}(t_n) & \text{if } \mathrm{Childs}(t) = t_1..t_n,\ \ \lambda(t(\varepsilon)) = \mathrm{Opponent} \end{cases}$$

The first argument, $p : \mathbb{P}_{\mathrm{T}} \rightarrow \mathsf{D}$, is the payoff function, the second one, $h : \mathbb{P}_{\mathrm{NT}} \rightarrow \mathsf{D}$, is the heuristics function, and the third one, t, is a tree labeled in $\mathbb{P}$.

Whenever the pair (p, h) will be implicit in the context, we shall simplify the notation of the backward-propagation min-max function by noting it v. This inductive procedure allows, in particular, to evaluate not only the finite trees of the game $\mathrm{T}(\mathcal{S}) = \{t \in \mathrm{T}(\mathbb{P}) \mid t = t_\pi\}$, but also its finite prefixes $\widehat{\mathrm{T}}(\mathcal{S}, \pi) = \{t \in \mathrm{T}(\mathbb{P}) \mid t \leq t_\pi\}$ and the finite cuts $\widetilde{\mathrm{T}}(\mathcal{S}, \pi) = \{t \in \mathrm{T}(\mathbb{P}) \mid t \subseteq t_\pi\}$. The *closure* of the set of values $X \subseteq \mathsf{D}$ of a position π, written $\mathrm{clos}_\pi(X)$, is the smallest set Y containing X and such that:

$$\forall y \in Y^*.\ \begin{cases} \uparrow y \in Y & \text{if } \ \lambda(\pi) = \mathrm{Player} \\ \downarrow y \in Y & \text{if } \ \lambda(\pi) = \mathrm{Opponent} \end{cases}$$

Given a pair p and h, we shall note $\widehat{\mathsf{D}}_\pi(\mathcal{S}, p, h)$ and $\widetilde{\mathsf{D}}_\pi(\mathcal{S}, p, h)$ the closure of the image of the min-max backward-propagation, respectively, on the prefixes and on the finite cuts of the tree of position π, i.e. $\widehat{\mathsf{D}}_\pi(\mathcal{S}, p, h) = \mathrm{clos}_\pi\{v_{p,h}(t) \mid t \in \widehat{\mathrm{T}}(\mathcal{S}, \pi)\}$ and $\widetilde{\mathsf{D}}_\pi(\mathcal{S}, p, h) = \mathrm{clos}_\pi\{v_{p,h}(t) \mid t \in \widetilde{\mathrm{T}}(\mathcal{S}, \pi)\}$. To alleviate the notational burden, we shall often omit the functions p and h, sometimes also the syntax $\mathcal{S}$, and we shall use D_π as a synonym of $\widehat{\mathsf{D}}_\pi$. An *extended game algebra* will be a game algebra with an additional relation $\leq : \wp^{\mathsf{D} \times \mathsf{D}}$, suitable to compare the elements of D. We shall write $\leq_\pi$ for the relation $\leq$ restricted to the elements of D_π, i.e. $\leq_\pi = \{(x, y) \in \mathsf{D}_\pi \times \mathsf{D}_\pi \mid x \leq y\}$. Given a non-terminal position $\pi \in \mathbb{P}_{\mathrm{NT}}$, such that $\mathrm{Succ}(\pi) = \pi_1..\pi_n$, every sequence of values $x_1..x_n$ such that $x_i \in \mathsf{D}_{\pi_i}$ is a *generating* sequence of a value of D_π. (To obtain it, it suffices to apply the player function $\lambda(\pi)$ to the sequence).The set of generating sequences of $\pi \in \mathbb{P}_{\mathrm{NT}}$ is $G_\pi = \{u \in \mathsf{D}^* \mid u = x_1..x_n,\ \forall i = 1..n.\ x_i \in \mathsf{D}_{\pi_i}\}$.

Definition 4.4. (Kernel) Given a game $\mathcal{S} = (\mathbb{P}, \lambda, \rightarrow, \mathbb{P}_{\mathrm{I}})$ and an extended game algebra $\mathcal{D} = (\mathsf{D}; \leq, \uparrow, \downarrow)$, a pair (p, h) is a *kernel* (of semantics for positions), where $p : \mathbb{P}_{\mathrm{T}} \rightarrow \mathsf{D}$ is a payoff and $h : \mathbb{P}_{\mathrm{NT}} \rightarrow \mathsf{D}$ is a heuristics, if :

(1) for all positions $\pi \in \mathbb{P}$, the structure $(\mathsf{D}_\pi, \leq_\pi)$ is a *dcpo* (complete directed partial order)

(2) $\uparrow$ and $\downarrow$ are *continuous* with respect to the generating sequences, i.e. for all $\pi \in \mathbb{P}_{\mathrm{NT}}$, we have:
$$\forall X \in \Delta(G_\pi, \leq^*).\ \begin{cases} \sup(\uparrow X) = \uparrow(\sup X) & \text{if } \ \lambda(\pi) = \mathrm{Player} \\ \sup(\downarrow X) = \downarrow(\sup X) & \text{if } \ \lambda(\pi) = \mathrm{Opponent} \end{cases}$$

(3) the heuristics is *correct*; that is, for any position $\pi \in \mathbb{P}_{\mathrm{NT}}$, such that $\mathrm{Succ}(\pi) = \pi_1..\pi_n$ $(n \geq 1)$, the function verifies the following conditions:
$$h(\pi) \ \leq_\pi \begin{cases} \uparrow f(\pi_1)..f(\pi_n) & \text{if } \ \lambda(\pi) = \mathrm{Player} \\ \downarrow f(\pi_1)..f(\pi_n) & \text{if } \ \lambda(\pi) = \mathrm{Opponent} \end{cases}$$
where $f = p \oplus h$ is the union of functions p and h, i.e. $f(\pi) = p(\pi)$ if $\pi \in \mathbb{P}_{\mathrm{T}}$, else $f(\pi) = h(\pi)$.

A kernel is the basis of an homomorphism between the syntactic and the semantic notions. Indeed, the notion of heuristics is crucial for the function v, restricted to the finite prefixes of the game, to be a (monotone) homomorphism. Whenever two trees are in the prefix relation $t \leq t'$, t' can be regarded as a better approximation, with respect to t, of the (possibly infinite) tree at a game position. A still better evaluation of the position will necessarily correspond, if the heuristics is correct, to a bigger effort in the unfolding of the tree. In other words, the value $v(t')$ will be larger, in the sense of the order on the evaluation domain, to the value $v(t)$.

Proposition 4.5. *If* (p,h) *is a kernel for an arena* $\mathcal{S} = (\mathbb{P}, \lambda, \rightarrow, \mathbb{P}_I)$ *and an extended game algebra* $\mathcal{D} = (D; \leq, \uparrow, \downarrow)$*, the backward-propagation function* $v_{p,h}(.)$*, restricted to prefixes of the game* $\widehat{T}(\mathcal{S})$*, is monotonic with respect to the prefix relation, i.e.:*

$$\forall t, t' \in \widehat{T}(\mathcal{S}, \pi).\ \ t' \leq t \ \Rightarrow\ v_{p,h}(t') \leq_\pi v_{p,h}(t)$$

Proof. We can compare trees according to their depths. The relation $\prec = \{(t', t) \mid \mathrm{depth}(t) <_{\mathbb{N}} \mathrm{depth}(t)\}$ is a well-founded relation on the finite prefixes $\prec : \wp^{\widehat{T}(\mathcal{S}) \times \widehat{T}(\mathcal{S})}$, much like its product extension $\prec_2 : \wp^{\widehat{T}(\mathcal{S})^2 \times \widehat{T}(\mathcal{S})^2}$, defined by $\prec_2 = \{((t_1', t_2'), (t_1, t_2) \mid t_1' \prec t_1 \wedge t_2' \prec t_2\}$. We shall prove the proposition using the principle of well-founded induction on $\prec_2$. Let $t', t \in \widehat{T}(\mathcal{S}, \pi)$ such that $t' \leq t$. We would like to show that $v(t') \leq_\pi v(t)$. If $t = \{(\varepsilon, \pi)\}$, then $t' = t$, and hence $v(t') \leq_\pi v(t)$, because $\leq_\pi$ is a reflexive relation. If $\mathrm{Childs}(t) = t_1..t_n$, let us assume $\lambda(t(\varepsilon)) = \mathrm{Player}$. By definition, a prefix is a cut that either contains none or all of the successors. We have then two cases to consider. For the first case, $t' = \{(\varepsilon, \pi)\}$ (no successor at address ε), we have $v(t') = h(\pi)$. Because the heuristics is correct, we have that $h(\pi) \leq_\pi \underset{i=1..n}{\uparrow} p \oplus h(\pi_i)$ where $\mathrm{Succ}(\pi) = \pi_1..\pi_n$. For all positions π_i we have, by definition of v, $p \oplus h(\pi_i) = v(\{(\varepsilon, \pi_i)\})$. By letting $t_i' = \{(\varepsilon, \pi_i)\}$, we can obtain a prefix t_i' for each child t_i of t. Let $t'' = t' \cup \bigcup_{\pi \xrightarrow{\ell} \pi_i} \ell.\{(\varepsilon, \pi_i)\}$. By definition of t'', we have $\mathrm{Childs}(t'') = t_1'..t_n'$. Because the heuristics is correct, we have that $v(t') \leq_\pi v(t'')$. Moreover, by the induction hypothesis, $((t_i', t_i) \prec_2 (t'', t))$, we have $v(t_i') \leq_{\pi_i} v(t_i)$. The continuity of the operations implies monotonicity with respect to the generating sequences, i.e. $\forall u, v \in G_\pi.\ u \leq^* v \ \Rightarrow\ \uparrow u \leq_\pi \uparrow v$. Hence, we have:

$$v(t') \ \leq_\pi \ v(t'') \ = \ \underset{i=1..n}{\uparrow} v(t_i') \ \leq_\pi \ \underset{i=1..n}{\uparrow} v(t_i) \ = \ v(t)$$

Therefore, from the property of transitivity for preorders $(D_\pi, \leq_\pi)$, we obtain $v(t') \leq_\pi v(t)$. The second case is when t' has also sub-trees as its children: $\mathrm{Childs}(t') = t_1'..t_n'$ (all the successors at address ε). Because $t_i' \leq t_i$, by the induction hypothesis $((t_i', t_i) \prec_2 (t', t))$, we have $v(t_i') \leq_{\pi_i} v(t_i)$. We arrive at the desired conclusion, once more, by monotonicity of $\uparrow$. The proof for the case $\lambda(\pi) = \mathrm{Opponent}$ can be obtained dually, and is itself based on the monotonicity of $\downarrow$. □

On the one hand, because the min-max backward-propagation function $v(.)$ on the finite prefixes is monotone, every directed tree prefix will be transformed by this function in a directed value set. On the other hand, whenever the pair (p, h) is a kernel, every directed value set has an upper bound in the dcpo $(D_\pi, \leq_\pi)$.

Proposition 4.6. *Let* (p,h) *be a kernel. The function* $[\![\pi]\!] = \sup\{v_{p,h}(t_\pi/^k) \mid k \in \mathbb{N}\}$ *is a semantics for positions.*

Proof. Let $\pi \in \mathbb{P}$. The set $\{t_\pi/^k \mid k \in \mathbb{N}\}$ is a directed of the structure $(\widehat{T}(\mathcal{S}, \pi), \leq)$. So, the set $\{v_{p,h}(t_\pi/^k) \mid k \in \mathbb{N}\}$ is a directed of the dcpo $(D_\pi, \leq_\pi)$. The latter admits an upper bound, hence the function $[\![.]\!]$ is well-defined. In order to prove that it is a semantics for positions, let us assume $\pi \in \mathbb{P}_{NT}$, with $\mathrm{Succ}(\pi) = \pi_1..\pi_n$ and $\lambda(\pi) = \mathrm{Player}$ (the case for the opponent is dual). We therefore have:

$$\begin{array}{lll}
[\![\pi]\!] & = & \sup\{v(t_\pi/^k) \mid k \in \mathbb{N}\} \\
 & = & \sup\{\underset{i=1..n}{\uparrow} v(t_{\pi_i}/^{k-1}) \mid k \in \mathbb{N}\} \qquad \text{(def. of } v(.)\text{)} \\
 & = & \uparrow \sup_{\leq^*}\{x_1..x_n \mid x_i = v(t_{\pi_i}/^{k-1}),\ k \in \mathbb{N}\} \qquad (\uparrow \text{ continuous}) \\
 & = & \uparrow \sup_{\leq^*}\{x_1..x_n \mid x_i = v(t_{\pi_i}/^{k}),\ k \in \mathbb{N}\} \\
 & = & \uparrow \sup_{\leq}\{x_1 \mid x_1 = v(t_{\pi_1}/^{k}),\ k \in \mathbb{N}\}..\sup_{\leq}\{x_n \mid x_n = v(t_{\pi_n}/^{k}),\ k \in \mathbb{N}\} \\
 & = & \uparrow [\![\pi_1]\!]..[\![\pi_n]\!]
\end{array}$$

□

Let (p, h_1) and (p, h_2) be two kernels for the game algebra $\mathcal{D} = (D; \leq, \uparrow, \downarrow)$ and its *dual* (i.e. $\mathcal{D}^\perp = (D; \geq, \uparrow, \downarrow)$), respectively, giving rise to two semantics $[\![.]\!]_1$, $[\![.]\!]_2$ for the game positions. It is not difficult to verify that if $\forall \pi \in \mathbb{P}_{NT}.\ h_1(\pi) \leq h_2(\pi)$, then $\forall \pi \in \mathbb{P}.\ [\![\pi]\!]_1 \leq [\![\pi]\!]_2$. In this case, we shall refer to the semantics for positions $[\![.]\!]_1$ and $[\![.]\!]_1$ as *pessimistic* and *optimistic*, respectively. In this context, the set of *pessimistic or optimistic values* of finite cuts is defined as follow:

$$\widetilde{\mathsf{D}}_\pi(\mathcal{S},\mathsf{p},\mathsf{h}_1,\mathsf{h}_2) = \widetilde{\mathsf{D}}_\pi(\mathcal{S},\mathsf{p},\mathsf{h}_1) \cup \widetilde{\mathsf{D}}_\pi(\mathcal{S},\mathsf{p},\mathsf{h}_2)$$

The game of logic programming is an example for which both semantics coexist. The *pessimistic* one corresponds to the formulae that are *logically implied* by the program. In turn, the *optimistic* one corresponds to the formulae that are *consistent* (i.e., that are not in contradiction) with the program. In this work, we shall only present the pessimistic semantics for CLP languages.

4.5. **Factoring by strategies.** Under certain conditions, it becomes possible to express the value of a position in terms of the strategies available to the players. To put it differently, the semantics of a position may sometimes be defined in terms of a semantics for the strategies for this position.

Definition 4.7. (SEMANTICS FOR STRATEGIES) Let $\mathcal{S} = (\mathbb{P}, \lambda, \rightarrow, \mathbb{P}_{\mathrm{I}})$ be a game syntax and $\mathcal{D} = (\mathsf{D}; \uparrow, \downarrow)$ a game algebra. A *semantics for strategies* is an arbitrary function $(\!|\,.\,|\!) : \Sigma(\mathcal{S}) \to \mathbb{P} \to \mathsf{D}$, such that for all $\sigma \in \Sigma(\mathcal{S})$, the function $(\!|\,.\,|\!)_\sigma : \mathbb{P} \to \mathsf{D}$ is a semantics for positions of the game $\mathcal{S}_\sigma$.

Intuitively, a strategy defines a semantic value for every position in the game. This value corresponds to the min-max backward-propagation applied to the cut of the game tree implicitly denoted by the strategy. Given a pair (p,h), we shall write $\overline{\mathsf{D}}_\pi(\mathcal{S})$ for the closure of the union of domains $\widehat{\mathsf{D}}_\pi(\mathcal{S}_\sigma)$ for all strategies of the game σ, i.e.,

$$\overline{\mathsf{D}}_\pi(\mathcal{S}) = \mathrm{clos}_\pi\Big(\bigcup_{\sigma \in \Sigma(\mathcal{S},\pi)} \widehat{\mathsf{D}}_\pi(\mathcal{S}_\sigma) \Big)$$

In other words, if we call any finite prefix of a game strategy a *pre-strategy*, therefore, $\overline{\mathsf{D}}_\pi(\mathcal{S})$ denotes the set of (backward-propagation) values of all the possible pre-strategies of the game. Every enumerable subset $\mathsf{Y} \in \wp^{\mathsf{X}}_\omega$ of a set X, can be represented by at least an infinite sequence of X^ω that enumerates Y and that we shall note $\vec{\mathsf{Y}}$. Given an extended game algebra $\mathcal{D} = (\mathsf{D}; \leq, \uparrow, \downarrow)$, the function $\uparrow : \mathsf{D}^* \to \mathsf{D}$ can be extended to the enumerable subsets of $\mathsf{X} \subseteq \mathsf{D}$, if it is commutative, (i.e. $\uparrow x = \uparrow x'$ for each permutation x' of the sequence x) and for every infinite sequence $x \in \mathsf{X}^\omega$, there exists $\sup_{\leq}\{(\uparrow y) \mid y \in \mathsf{X}^*, y \sqsubseteq x\}$. The extension of $\uparrow$ to the enumerable subsets of X, written $\dot{\uparrow} : \wp^{\mathsf{X}}_\omega \to \mathsf{D}$, is therefore defined by $\dot{\uparrow}(\mathsf{Y}) = \sup_{\leq}\{(\uparrow y) \mid y \in \mathsf{X}^*, y \sqsubseteq \vec{\mathsf{Y}}\}$.

Definition 4.8. (REGULAR GAME ALGEBRA) Let $\mathcal{S} = (\mathbb{P}, \lambda, \rightarrow, \mathbb{P}_{\mathrm{I}})$ be a game syntax. An extended game algebra $\mathcal{D} = (\mathsf{D}; \leq, \uparrow, \downarrow)$ is said to be *regular* if (I) $\uparrow$ can be extended to the enumerable subsets of $\overline{\mathsf{D}}_\pi(\mathcal{S})$, (II) $\uparrow$ such that $\forall x \in \overline{\mathsf{D}}_\pi(\mathcal{S}).\ \ x = \uparrow x..x$, (III) the $\uparrow, \downarrow$ operations are associative (i.e. $\uparrow u.(\uparrow v).z = \uparrow u.v.z$ and $\downarrow u.(\downarrow v).z = \downarrow u.v.z$) and distributive (i.e. $\uparrow u.(\downarrow y_1..y_n).v = \downarrow(\uparrow u.y_1.v)..(\uparrow u.y_n.v)$ and the other way round).

With a regular game algebra we can construct two tightly inter-related semantics, one for positions and the other one for strategies.

Definition 4.9. (FACTORING) Let $\mathcal{S} = (\mathbb{P}, \lambda, \rightarrow, \mathbb{P}_{\mathrm{I}})$ be a game syntax and $\mathcal{D} = (\mathsf{D}; \leq, \uparrow, \downarrow)$ a regular game algebra. A semantics for positions $[\![.]\!] : \mathbb{P} \to \mathsf{D}$ can be factored by a semantics for strategies $(\!|\,.\,|\!) : \Sigma(\mathcal{S}) \to \mathbb{P} \to \mathsf{D}$ iff:

$$\forall \pi \in \mathbb{P}. \quad [\![\pi]\!] = \dot{\uparrow} \{(\!|\pi|\!)_\sigma \mid \sigma \in \Sigma(\mathcal{S},\pi)\}$$

Before having a look at how to derive a semantics for strategies by factoring that of positions, we shall first prove that every finite prefix of the game can be approximated by the set of pre-strategies belonging to it.

Lemma 4.10. *Let* $\mathcal{S} = (\mathbb{P}, \lambda, \rightarrow, \mathbb{P}_{\mathrm{I}})$ *be a game syntax and* $\mathcal{D} = (\mathsf{D}; \leq, \uparrow, \downarrow)$ *a regular game algebra. Let* (p,h) *be a kernel and* $\mathsf{t} \in \mathrm{T}(\mathcal{S},\pi)$ *a finite prefix of the tree of position* π. *Then,*

$$v_{\mathsf{p},\mathsf{h}}(\mathsf{t}) = \dot{\uparrow} \{v_{\mathsf{p},\mathsf{h}}(\mathsf{s}) \mid \mathsf{s} \leq \mathsf{t}_\pi(\mathcal{S}_\sigma),\ \mathsf{s} \subseteq \mathsf{t},\ \sigma \in \Sigma(\mathcal{S},\pi)\}$$

Proof. By induction on the depth k of the tree t.

- $k = 0$

  If $t = \{(\varepsilon, \pi)\}$, then the only strategy or pre-strategy that is possible is the tree itself t. Therefore,

$$\begin{array}{rcll} v_{p,h}(t) & = & \dot{\uparrow}\{v_{p,h}(t)\} & \mathcal{D} \text{ regular} \\ & = & \dot{\uparrow}\{v_{p,h}(s) \mid s \leq t_\pi(\mathcal{S}, \sigma),\ s \subseteq t,\ \sigma \in \Sigma(\mathcal{S}, \pi)\} & \end{array}$$

- $k \Rightarrow k+1$ with $\lambda(\pi) = \text{Player}$

  The tree t has $n \geq 1$ children $t_1, .., t_n$ at addresses $\ell_1, .., \ell_n$. A pre-strategy contained in t is derived by choosing a child t_i of t and by constructing a pre-strategy in t_i :

$$\Sigma(\mathcal{S}, \pi) = (\{(\varepsilon, \pi)\} \cdot_{\ell_1} \Sigma(\mathcal{S}, \pi_1)) \ \cup \ .. \ \cup \ (\{(\varepsilon, \pi)\} \cdot_{\ell_n} \Sigma(\mathcal{S}, \pi_n)) \qquad (i)$$

  where $\pi_i = t_i(\varepsilon)$. Then,

$$\begin{array}{rcll} v_{p,h}(t) & = & \dot{\uparrow}\left\{v_{p,h}(t_i)\right\}_{i=1..n} & \text{def. } v_{p,h} \\ & = & \dot{\uparrow}\left\{\dot{\uparrow}\{v_{p,h}(s_i) \mid s_i \leq t_{\pi_i}(\mathcal{S}, \sigma),\ s_i \subseteq t_i,\ \sigma_i \in \Sigma(\mathcal{S}, \pi_i)\}\right\}_{i=1..n} & \text{induction} \\ & = & \dot{\uparrow}\{v_{p,h}(s) \mid s \leq t_\pi(\mathcal{S}, \sigma),\ s \subseteq t,\ \sigma \in \Sigma(\mathcal{S}, \pi)\} & \text{associativity, (i)} \end{array}$$

- $k \Rightarrow k+1$ with $\lambda(\pi) = \text{Opponent}$

  The tree t has $n \geq 1$ children $t_1, .., t_n$ at addresses $\ell_1, .., \ell_n$. A pre-strategy in t is constructing by putting together the n pre-strategies, each relative to a child of t :

$$\Sigma(\mathcal{S}, \pi) = \{(\varepsilon, \pi)\} \cdot_{\ell_1} \Sigma(\mathcal{S}, \pi_1) \cdot \ldots \cdot_{\ell_n} \Sigma(\mathcal{S}, \pi_n) \qquad (ii)$$

  where $\pi_i = t_i(\varepsilon)$. Then,

$$\begin{array}{rcll} v_{p,h}(t) & = & \overset{n}{\underset{i=1}{\downarrow}} v_{p,h}(t_i) & \text{def. } v_{p,h} \\ & = & \overset{n}{\underset{i=1}{\downarrow}} (\dot{\uparrow}\{v_{p,h}(s_i) \mid s_i \leq t_{\pi_i}(\mathcal{S}, \sigma_i),\ s_i \subseteq t_i,\ \sigma_i \in \Sigma(\mathcal{S}, \pi_i)\}) & \text{induction} \\ & = & \dot{\uparrow}\{v_{p,h}(s_1) \downarrow .. \downarrow v_{p,h}(s_n) \mid s_i \leq t_{\pi_i}(\mathcal{S}, \sigma_i),\ s_i \subseteq t_i,\ \sigma_i \in \Sigma(\mathcal{S}, \pi_i)\} & \text{distributivity} \\ & = & \dot{\uparrow}\{v_{p,h}(s) \mid s \leq t_\pi(\mathcal{S}, \sigma),\ s \subseteq t,\ \sigma \in \Sigma(\mathcal{S}, \pi)\} & \text{(ii)} \end{array}$$

□

Proposition 4.11. *Let* $\mathcal{S} = (\mathbb{P}, \lambda, \rightarrow, \mathbb{P}_I)$ *be a game syntax and* $\mathcal{D} = (D; \leq, \uparrow, \downarrow)$ *a regular game algebra. If* $[\![.]\!]$ *is the semantics for positions defined by a kernel* (p, h)*, then* $[\![.]\!]$ *can be factored out by the semantics for strategies defined by:*

$$(\!|\pi|\!)_\sigma = \sup\{v_{p,h}(t/^k) \mid k \in \mathbb{N},\ t \in IT(\mathcal{S}_\sigma, \pi)\}$$

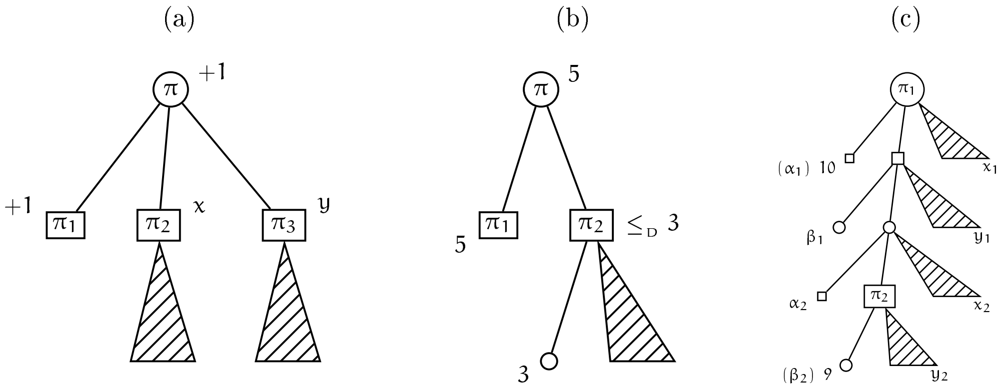
Proof. The function $(\!|.|\!) : \Sigma(\mathcal{S}) \rightarrow \mathbb{P} \rightarrow D$ is a semantics for strategies because it coincides precisely with the semantics for positions in the game $\mathcal{S}_\sigma$, for every given strategy σ. It then remains to show that $[\![\pi]\!] = \dot{\uparrow}\{(\!|\pi|\!)_\sigma \mid \sigma \in \Sigma(\mathcal{S}, \pi)\}$ for every position π:

$$\begin{array}{rcll} [\![\pi]\!] & = & \sup\{v_{p,h}(t_\pi(\mathcal{S})/^k) \mid k \in \mathbb{N}\} & \text{def. } [\![\pi]\!] \\ & = & \sup\left\{\dot{\uparrow}\{v_{p,h}(s) \mid s \leq t_\pi(\mathcal{S}_\sigma),\ s \subseteq t_\pi(\mathcal{S})/^k,\ \sigma \in \Sigma(\mathcal{S}, \pi)\}\right\}_{k \in \mathbb{N}} & \text{lemma} \\ & = & \sup\left\{\dot{\uparrow}\{v_{p,h}(s/^k) \mid s \leq t_\pi(\mathcal{S}_\sigma),\ \sigma \in \Sigma(\mathcal{S}, \pi)\}\right\}_{k \in \mathbb{N}} & \\ & = & \sup\left\{\dot{\uparrow}\ \left\{\dot{\uparrow}\{v_{p,h}(s/^k) \mid s \leq t_\pi(\mathcal{S}_\sigma),\ s \subseteq t_\pi(\mathcal{S})\}\right\}_{\sigma \in \Sigma(\mathcal{S}, \pi)}\right\}_{k \in \mathbb{N}} & \text{associativity} \\ & = & \dot{\uparrow}\ \left\{\sup\left\{\dot{\uparrow}\{v_{p,h}(s) \mid s \leq t_\pi(\mathcal{S}_\sigma),\ s \subseteq t_\pi(\mathcal{S})\}\right\}_{k \in \mathbb{N}}\right\}_{\sigma \in \Sigma(\mathcal{S}, \pi)} & \text{continuity} \\ & = & \dot{\uparrow}\ \{(\!|\pi|\!)_\sigma\}_{\sigma \in \Sigma(\mathcal{S}, \pi)} & \text{def. } (\!|\pi|\!)_\sigma \end{array}$$

□

This result should appear more intuitive if we think of a finite game interpreted by a kernel (p, h) in the structure of the naturals with the functions min and max (which forms a regular game algebra). For a given position π, let us assume we have only the loosing strategies σ, i.e. $(\!|\pi|\!)_\sigma = 0$, excepting for the three strategies σ_1, σ_2 and σ_3, such that $(\!|\pi|\!)_{\sigma_1} = 4$, $(\!|\pi|\!)_{\sigma_2} = 16$ and $(\!|\pi|\!)_{\sigma_3} = 7$. By the proposition above, we know that the value of the position should be $[\![\pi]\!] = \max\{0, 4, 16, 7\} = 16$.

FIGURE 5.1. Examples of pruning



5. ALPHA-BETA PRUNING

The Alpha-Beta algorithm is a method to compute the value of a game, introduced for the first time in [Hart & Edwards, 1961], and since, largely studied in the specialized literature of combinatory games (cf. see, for instance, [Knuth & Moore, 1975] and [Fuller et al., 1973]). The game is assumed to be evaluated in the structure of the integers using the min and max operations. In order to solve this kind of game, or, to put it differently, in order to know the value (loose, win, or whatever), it is often useless to *exhaustively* visit all the nodes of the game tree resulting from an initial position. A significant portion of the game tree may sometimes be enough to determine the *exact* value of the position of interest.

Example 5.1. Figure 5.1.(a) illustrates the case of a player position, evaluated in the structure $D = \{-1, 0, +1\}$ (loose, draw, win), and where the analysis of the first player move allows us to give an answer of $+1$, independently of the unknown values x and y corresponding to the other moves possible. In addition to that, the Alpha-Beta method proposes simplifications that are much less evident. Let us assume that we wish to compute the value, in the structure of the naturals $D = \mathbb{N}$ (using the functions min and max), of a position, given the tree whose shape is shown in figure 5.1.(b). At position π, a first player move will lead to the terminal position π_1, with 5 as payoff. A second move will lead to the non-terminal opponent's position π_2. In this latter position, the opponent will have available, in turn, a first move resulting in a payoff of 3, amongst other possible moves. Despite the incomplete amount of information on the evaluation of each of these matches, we are nevertheless able to know the exact value of the whole tree. Indeed, because the min function always yield a value that is smaller than each of its arguments, the value at position π_2 will be necessarily smaller than 3. So, at the root position, the player will have to choose, in the end, between a gain equal to 5 and a gain that will certainly be smaller than 3 (and hence smaller than 5). We can, therefore, anticipate that the player will choose her first move (having a value of 5), without the need to visit the other moves that are available to the opponent at position π_2. So, let us consider the situation in figure 5.1.(c). At the initial position π_1, the player has an available move with a payoff of 10. We can show right away, independently of the values x_1, β_1, y_1, α_2 and x_2, that if, at position π_2, the opponent has an available move with an associated payoff of 9 (and, hence, smaller than 10), then the value of position π_2 will not in any case be able to affect the value at the root's position. Indeed, the formula used to compute that value π_1 is the following one (where we have written $\uparrow$ for max and $\downarrow$ for min):

$$10 \uparrow \{\beta_1 \downarrow [\alpha_2 \uparrow (9 \downarrow y_2) \uparrow x_2] \downarrow y_1\} \uparrow x_1$$

In order to help the reader, we shall introduce a more familiar notation. We shall replace the symbols $\uparrow$ and $\downarrow$ by $\cdot$ and $+$, respectively, which are commonly used in algebra to denote distributive operations:

$$10 \cdot \{\beta_1 + [\alpha_2 \cdot (9 + y_2) \cdot x_2] + y_1\} \cdot x_1$$

As a matter of fact, applying the distributive property, we realize that the formula above is equivalent to the following:

$$\{10 \cdot \beta_1 \ + \ [10 \cdot \alpha_2 \cdot (9 + y_2) \cdot x_2] \ + \ 10 \cdot y_1\} \cdot x_1$$

We should point out now that $(9 + y2)$ is necessarily smaller than 9, and hence, 10. This allows us to eliminate this argument from the formula (because the function $\cdot = \uparrow =$ max ignores the smaller arguments):

$$\{10 \cdot \beta_1 \ + \ [10 \cdot \alpha_2 \cdot x_2] \ + \ 10 \cdot y_1\} \cdot x_1$$

At last, applying the distributive property, but this time the other way round, we obtain the formula:

$$10 \cdot \{\beta_1 \ + \ [\alpha_2 \ \cdot \ x_2] \ + \ y_1\} \cdot x_1$$

Therefore, the value at position π_2 does not matter at all in the context where we find it: from the player's viewpoint, who studies carefully her possibilities at position π_1, things happen as if she did not really exist. This implies that after having calculated the value 9 of the first opponent's move at position π_2, there will be no reason to justify the evaluation of the other available moves. (The unknown value y_2 is as unimportant as the value 9 already computed.)

5.1. **Traditional Alpha-Beta.** The Alpha-Beta method is traditionally presented as a recursive function written in an imperative pseudo-language (cf. algorithm 1). The `Alpha-Beta` function analyses a position $\pi \in \mathbb{P}$ of the game with two additional parameters, α and β, each one denoting as sort of *threshold* one must not overstep during the incremental computation of the value of π. If one did, the computation would be stopped.

Algorithm 1 *Traditional Alpha-Beta*

```
function Alpha-Beta(π : ℙ, α,β : ℤ) : ℤ

  1. if π ∈ ℙ_T then return p(π)

  2. else let be  π1..πn = Succ(π)

  3. if λ(π) = Player then
        3.1) v ← α
        3.2) for i = 1 to n
              and while (v <_ℤ β) do
              v ← v max Alpha-Beta(πi,α max v,β)

  4. if λ(π) = Opponent then
        4.1) v ← β
        4.2) for i = 1 to n
              and while (α <_ℤ v) do
              v ← v min Alpha-Beta(πi,α,β min v)

  5. return v
```

5.2. **Polymorphic Alpha-Beta.** The correctness of the cuts carried out by the Alpha-Beta method is based on implicit properties of the functions min and max; in particular, those having to do with the distributive law. We are going to isolate these properties and abstract away from the game algebra $\mathcal{D} = (D; \leq, \uparrow, \downarrow)$ used. In the definition of the method, we shall begin by replacing the functions max and min by their generic versions, $\uparrow$ and $\downarrow$, respectively. We must also replace conditions like $(\nu <_{\mathbb{Z}} \beta)$ and $(\nu >_{\mathbb{Z}} \alpha)$, that ensure the evolution of the computation, (`while [..] do [..]`), by, respectively, the conditions $(\nu \not\geq \beta)$ and $(\nu \not\leq \alpha)$. The difference is not trivial if, by abstracting away from the order of the integers, we consider $\leq$ as a *partial* relation, not entirely *total*, like that of the integers. If two values are not comparable, we must not stop the computation.

Next, if we would like the return value ν of the algorithm to be independent of the parameters α and β, computed elsewhere on the game tree, we should initialize ν with a pseudo-value ι, not belonging to the domain D, and such that $\iota \uparrow x = \iota \downarrow x = x$ for all $x \in D$. As a consequence of this, the test associated to the cut will become $(\nu \uparrow \alpha \not\geq \beta)$ for the nodes of the player and $(\nu \downarrow \beta \not\leq \alpha)$ for the nodes of the opponent. We could also perform another simple modification, which may have some important side-effects. Whenever the evaluation domain $(D, \leq)$ allows for a minimum element $\perp$ and a maximum element $\top$, there are always two correct heuristic functions whatever the payoff used: the constant function $h_1(\pi) = \perp$ is trivially correct for $\mathcal{D}$, and the constant function $h_2(\pi) = \top$ is trivially correct for the dual algebra $\mathcal{D}^{\perp} = (D; \geq, \uparrow, \downarrow)$. The presence of heuristics allows us to modify the cut test for a more efficient one. For the player nodes, we shall have a test $(\nu \uparrow \alpha \uparrow h_1(\pi) \not\geq \beta)$; for the opponent nodes, $(\nu \downarrow \beta \downarrow h_2(\pi) \not\leq \alpha)$. This allows us to stop the evaluation at a node without even having started it.

The cut conditions could also be weakened without compromising correctness. For the player nodes, we could use $\alpha \uparrow \beta \neq \alpha \uparrow [\beta \downarrow (v \uparrow h_1(\pi))]$; for the opponent nodes, $\beta \downarrow \alpha \neq \beta \downarrow [\alpha \uparrow (v \downarrow h_2(\pi))]$. The interest of such a change is double. Not only we are dealing here with a weaker condition from a logical viewpoint, and, therefore, more easily verifiable, but also, these new equations liberate us from the relation $\leq$. So, the $\leq$ relation is not required to have any special properties other than those needed as a guarantee to the definition of the semantics of infinite game trees.

Finally, accepting our willingness to enlarge the spectrum of applicability of the algorithm, we could also reconsider entirely the notion of correctness of the algorithm, focused on the *equality* of values. The new tests should be:

- $\alpha \uparrow \beta \not\approx \alpha \uparrow [\beta \downarrow (x \uparrow h_1(\pi))]$ for the player nodes
- $\beta \downarrow \alpha \not\approx \beta \downarrow [\alpha \uparrow (x \downarrow h_2(\pi))]$ for the opponent nodes

and where $\approx$ is an equivalence relation on values. This allows us to obtain a more general correctness result for the algorithm. Indeed, equality is no more than a *special case* of equivalence, generated by the partial orders $(D_\pi, \leq_\pi)$. We can justify this choice by stating that, using the Alpha-Beta algorithm, we are able to compute *abstract* values, very much like it is done through the Abstract Interpretation of programs (cf. [Cousot & Cousot, 1977]). These abstract values are values that are equal modulo an abstraction function. In the case of Alpha-Beta, concerning the techniques proper to Abstract Interpretation, the values operated upon will always be *concrete* values; however, any simplifications carried out (namely, cuts), find their justification in the abstract domain. In other words, the algorithm is correct with respect to the defined equivalence. In the special case where the equivalence is the equality of values, the algorithm computes precisely the value of a position. Otherwise, it computes an equivalent value, namely an equal value in the implicit domain of abstract values.

5.2.1. *SOS formalisation.* As notation for the evaluation with the Alpha-Beta method of a position π we shall use $[\![\, \pi \,]\!]^{\alpha,\beta}_{\approx}$, which makes us think of the notation of a semantics for positions, provided that we drop the parameters α and β, as well as the equivalence $\approx$. The correctness theorem will let us see more clearly the relationship between these two. The main characteristic of the Alpha-Beta algorithm is that of computing the value of certain branches of the game tree using a sort of summary of what is essential, the values α and β, of the branches already inspected. To explain more formally the flow of information from one branch to another, we must introduce the notion of root-cut. A *root-cut* t is a game tree if the position $\pi = t(\varepsilon)$ at the root is terminal, hence $t = \{(\varepsilon, \pi)\}$. For any other case, the root-cut is a cut of the game tree t_π, such that all children $\mathrm{Childs}(t) = t_1..t_n$ are untouched game trees, i.e. $t_i = t_{\pi_i}$ where $\pi_i = t_i(\varepsilon)$. Given an arena $\mathcal{S}$, the set of root-cuts of a position π will be denoted by $\mathrm{IT}(\mathcal{S}, \pi)^{\partial}$. The set of *root-cut finite prefixes* will be $\widehat{\mathrm{T}}(\mathcal{S}, \pi)^{\partial} = \{t' \in \mathrm{T}(\mathbb{P}) \mid t' \leq t,\ t' \in \mathrm{IT}(\mathcal{S}, \pi)^{\partial},\ t' \neq \{(\varepsilon, \pi)\}\}$.

Given a non terminal position $\pi \in \mathbb{P}_{\mathrm{NT}}$, a root-cut $t \in \mathrm{IT}(\mathcal{S},\pi)^{\partial}$ is the *composition* of two root-cuts, denoted by $t = t_1 \bowtie t_2$, if $t = t_1 \cup t_2$ and $t_1 \cap t_2 = \{(\varepsilon,\pi)\}$. This relation is a partial function $\bowtie : \mathrm{IT}(\mathcal{S},\pi)^{\partial} \times \mathrm{IT}(\mathcal{S},\pi)^{\partial} \to \mathrm{IT}(\mathcal{S},\pi)^{\partial}$, trivially associative and commutative. Any game tree with almost two children will be seen as the composition of two root-cuts. Once the two root-cuts t_1 and t_2 composing a tree $t = t_1 \bowtie t_2$ have been chosen, we can start the computation on the first "part" t_1, to resume it on the second "part" t_2 using either α or β (depending on the node type in question) as the information synthesized when visiting t_1.

Definition 5.2. (Alpha-Beta convergence) Let $\mathcal{S} = (\mathbb{P},\lambda,\to,\mathbb{P}_{\mathrm{I}})$ be an arena, and $\mathcal{D} = (\mathrm{D};\leq,\uparrow,\downarrow)$ an extended game algebra. Let (p,h_1) be a kernel for $\mathcal{D}$ and (p,h_2) be a kernel for $\mathcal{D}^{\perp} = (\mathrm{D};\geq,\uparrow,\downarrow)$, such that $h_1 \leq h_2$. Then, we say that the Alpha-Beta evaluation of a position $\pi \in \mathbb{P}$ *converges* to a value $v \in \mathrm{D}$, noted $[\![\, \pi \,]\!]_{\approx}^{\alpha,\beta} \Downarrow v$, iff $[\![\, t_\pi \,]\!]_{\approx}^{\alpha,\beta,\iota} \Downarrow v$, where the relation $[\![\, t \,]\!]_{\approx}^{\alpha,\beta,x} \Downarrow v$, with $t \in \mathrm{IT}(\mathcal{S},\pi)^{\partial}$ and $\alpha,\beta,x \in \mathrm{D}_\iota = \mathrm{D} \cup \{\iota\}$, is defined by the logical rules of algorithm 2.

Algorithm 2 *The polymorphic Alpha-Beta*

$$\frac{t = \{(\varepsilon,\pi)\} \quad \pi \in \mathbb{P}_{\mathrm{T}}}{[\![\, t \,]\!]_{\approx}^{\alpha,\beta,\iota} \Downarrow p(\pi)} \qquad \text{(base)}$$

$$\frac{t = t_1 \bowtie t_2 \qquad [\![\, t_1 \,]\!]_{\approx}^{\alpha,\beta,x} \Downarrow v_1 \qquad [\![\, t_2 \,]\!]_{\approx}^{\alpha,\beta,v_1} \Downarrow v_2 \qquad v = v_2}{[\![\, t \,]\!]_{\approx}^{\alpha,\beta,x} \Downarrow v} \qquad \text{(more)}$$

$$\frac{\begin{array}{l}\lambda(t\,\varepsilon) = \mathrm{Player} \\ \mathrm{Childs}(t) = t_1\end{array} \qquad [\![\, t_1 \,]\!]_{\approx}^{(\alpha\uparrow x),\beta,\iota} \Downarrow v_1 \qquad v = x \uparrow v_1}{[\![\, t \,]\!]_{\approx}^{\alpha,\beta,x} \Downarrow v} \qquad \text{(one)}$$

$$\frac{\begin{array}{l}\lambda(t\,\varepsilon) = \mathrm{Opponent} \\ \mathrm{Childs}(t) = t_1\end{array} \qquad [\![\, t_1 \,]\!]_{\approx}^{\alpha,(\beta\downarrow x),\iota} \Downarrow v_1 \qquad v = x \downarrow v_1}{[\![\, t \,]\!]_{\approx}^{\alpha,\beta,x} \Downarrow v} \qquad \text{(one)}$$

$$\frac{\alpha \uparrow \beta \approx \alpha \uparrow [\beta \downarrow (x \uparrow h_1(\pi))] \qquad \lambda(\pi) = \mathrm{Player}}{[\![\, t \,]\!]_{\approx}^{\alpha,\beta,x} \Downarrow x} \qquad \text{(cut)}$$

$$\frac{\beta \downarrow \alpha \approx \beta \downarrow [\alpha \uparrow (x \downarrow h_2(\pi))] \qquad \lambda(\pi) = \mathrm{Opponent}}{[\![\, t \,]\!]_{\approx}^{\alpha,\beta,x} \Downarrow x} \qquad \text{(cut)}$$

We shall simplify a bit the notation by writing $[\![\, \pi \,]\!]^{\alpha,\beta,x} \Downarrow v$ and $[\![\, t \,]\!]^{\alpha,\beta,x} \Downarrow v$, whenever no ambiguities may arise from the equivalence relation being used in the computation, and in particular, whenever the equivalence will match the equality of the domain. The system of inference rules is composed of four types of rules, namely (base), (more), (one) et (cut):

(base) is the rule for terminal positions, for the *base* of the recursion. If the game tree is composed of a single root, labeled by $\pi \in \mathbb{P}_{\mathrm{T}}$, we can conclude that the evaluation must be equal to the payoff at that position (independently of the parameters α and β).

(more) is that rule that carries out the evaluation loop for a tree t having *more* than one child. Indeed, as long as the tree contains at least two children, it can be decomposed into two root-cuts. Once the evaluation of the first is finished, the return value ν_1 can be used to update the value of the parent ($x' = \nu_1$), and to subsequently start the evaluation of the second. This rule exposes the non-determinism of the algorithm in the choice of the order of inspection of the branches. For n children, there exist $n!$ distinct possible ways of carrying out their sequential inspection.

(one) are the evaluation rules of a tree with only *one* child. Because of duality, we describe here only the rule applicable to the player nodes. The evaluation of this unique child will take place on a window $[\alpha', \beta]$ different from the window $[\alpha, \beta]$, currently being used by the parent. Indeed, since x is the value being evaluated for the parent, and noting that the parent itself will become an ancestor as soon as the evaluation of the child begins, the total number of values currently being evaluated for the ancestors (of player) will be, for the child, $\alpha' = \alpha \uparrow x$. The value currently being evaluated for the child is undefined before entering the evaluation loop, and therefore assigned to ι. Once the evaluation of its unique child is finished, that of the parent is also allowed to finish. The return values of the parent is then equal to $x \uparrow \nu'$.

(cut) are the *cut* rules of an evaluation loop. When the cut condition is verified, the node under evaluation in not interesting from the perspective of an ancestor. Once the computation has been stopped, the result of the evaluation of the node is x, a value that (in any case) does not matter for the current evaluation context.

The following technical lemma intuitively states that the Alpha-Beta algorithm does not "invent" any values. Let $D_\pi(t) = \{v_{p,h}(\tilde{t}) \mid \tilde{t} \subseteq t,\ \tilde{t} \in \widetilde{T}(\mathcal{S}, \pi),\ h = h_1 \lor h = h_2\}$ be the set of values of finite cuts included in the root-cut t.

Lemma 5.3. (Returned values) *Assuming* $[\![\, t \,]\!]_{\approx}^{\alpha,\beta,x} \Downarrow \nu$*, there are two possible consequences. First, if* $x = \iota$ *then* $\nu \in D_\pi(t) \cup \{\iota\}$*; and, second, if* $\exists t'' = t' \bowtie t$ *and* $x \in D_\pi(t')$*, then* $\nu \in D_\pi(t'')$.

Proof. By induction on the depth of the derivation of $[\![\, t \,]\!]_{\approx}^{\alpha,\beta,x} \Downarrow \nu$. □

In table 1, we show the set of axioms (hypotheses) needed by the correctness theorem of the Alpha-Beta algorithm.

Lemma 5.4. (Insertion) *In the table of hypotheses (table 1), we have:*

$$\forall \alpha, \beta, x. \quad \left\{ \begin{array}{lcl} \alpha \uparrow (\beta \downarrow x) & \approx & \alpha \uparrow [\beta \downarrow (\alpha \uparrow x)] \\ \beta \downarrow (\alpha \uparrow x) & \approx & \beta \downarrow [\alpha \uparrow (\beta \downarrow x)] \end{array} \right.$$

Proof. We prove the first equation (the second follows by duality):

$$\begin{array}{rcll} \alpha \uparrow [\beta \downarrow (\alpha \uparrow x)] & \approx & (\alpha \uparrow \beta) \downarrow [\alpha \uparrow (\alpha \uparrow x)] & \text{(A10)} \\ & = & (\alpha \uparrow \beta) \downarrow (\alpha \uparrow x) & \text{(A1), (A3)} \\ & \approx & \alpha \uparrow (\beta \downarrow x) & \text{(A10)} \end{array}$$

□

The *relative* correctness theorem of the Alpha-Beta method states that if a root-cut t evaluates to value ν, then there exists a *finite* prefix t' of t, such that the two values $a = v_{p,h_1}(t')$ and $b = v_{p,h_2}(t')$ are equivalent "modulo α, β and x" to value ν. The meaning of the equality *modulo* the three values α, β and x, is given by the following equations:

$$\begin{array}{lllll} \alpha \uparrow (\beta \downarrow \nu) \approx \alpha \uparrow (\beta \downarrow (x \uparrow a)) \approx \alpha \uparrow (\beta \downarrow (x \uparrow b)) & \text{if } \lambda(t\,\varepsilon) = \text{Player} \\ \beta \downarrow (\alpha \uparrow \nu) \approx \beta \downarrow (\alpha \uparrow (x \downarrow a)) \approx \beta \downarrow (\alpha \uparrow (x \downarrow b)) & \text{if } \lambda(t\,\varepsilon) = \text{Opponent} \end{array}$$

Table 1. Hypotheses for the correctness of Alpha-Beta

(1) The functions of players are s.t.:

(A1)	$x \downarrow (y \downarrow z) = (x \downarrow y) \downarrow z$	$x \uparrow (y \uparrow z) = (x \uparrow y) \uparrow z$	(associativity)
(A2)	$x \downarrow y = y \downarrow x$	$x \uparrow y = y \uparrow x$	(commutativity)
(A3)	$x \downarrow x = x$	$x \uparrow x = x$	(identity)
(A4)	$\downarrow (x) = x \downarrow x$	$\uparrow (x) = x \uparrow x$	(sequence extens.)
(A5)	$\downarrow (x.u) = x \downarrow (\downarrow u)$	$\uparrow (x.u) = x \uparrow (\uparrow u)$	(sequence extens.)

(2) The equivalence relation $\approx$ is a congruence, i.e.:

$$\text{(A6)} \qquad \frac{x \approx x' \qquad x \uparrow y \approx z}{x' \uparrow y \approx z} \qquad \frac{x \approx x' \qquad x \downarrow y \approx z}{x' \downarrow y \approx z}$$

(3) The following cut axioms are verified:

$$\text{(A789p)} \qquad \frac{\alpha \uparrow \beta \approx \alpha \uparrow [\beta \downarrow (x \uparrow h_1(\pi))] \qquad \lambda(\pi) = \text{Player}}{\alpha \uparrow (\beta \downarrow x) \ \approx \ \alpha \uparrow [\beta \downarrow (x \uparrow y)]}$$

$$\text{(A789o)} \qquad \frac{\beta \downarrow \alpha \approx \beta \downarrow [\alpha \uparrow (x \downarrow h_2(\pi))] \qquad \lambda(\pi) = \text{Opponent}}{\beta \downarrow (\alpha \uparrow x) \ \approx \ \beta \downarrow [\alpha \uparrow (x \downarrow y)]}$$

where $\alpha, \beta \in D \cup \{\iota\}$ and $x, y \in \widetilde{D}_\pi(\mathcal{S}, p, h_1, h_2) \cup \{\iota\}$.

(4) The player functions are distributive w.r.t. $\approx$, i.e.:

$$\text{(A10)} \qquad \begin{cases} x \downarrow (y \uparrow z) & \approx & (x \downarrow y) \uparrow (x \downarrow z) \\ x \uparrow (y \downarrow z) & \approx & (x \uparrow y) \downarrow (x \uparrow z) \end{cases}$$

Notation 5.5. The notation $\ddot{y}$, used for intervals of the form $[y, y]$, will help us put together the equations above in a more compact way:

$$\begin{array}{lll} \ddot{\alpha} \uparrow (\ddot{\beta} \downarrow \ddot{v}) \ \approx \ \ddot{\alpha} \uparrow (\ddot{\beta} \downarrow (\ddot{x} \uparrow [a, b])) & & \text{if } \lambda(t\,\varepsilon) = \text{Player} \\ \ddot{\beta} \downarrow (\ddot{\alpha} \uparrow \ddot{v}) \ \approx \ \ddot{\beta} \downarrow (\ddot{\alpha} \uparrow (\ddot{x} \downarrow [a, b])) & & \text{if } \lambda(t\,\varepsilon) = \text{Opponent} \end{array}$$

where $[x, y] \uparrow [z, t]$ stands for $[x \uparrow z, y \uparrow t]$, and $[x, y] \downarrow [z, t]$ for $[x \downarrow z, y \downarrow t]$. We also write $\ddot{v}(t)$ to denote the interval $[v_{p,h_1}(t), v_{p,h_2}(t)]$.

Theorem 5.6. (Relative correctness) *In the table of hypotheses (table 1), let* $\alpha, \beta, x \in D \cup \{\iota\}$ *and* $t \in IT(\mathcal{S}, \pi)^\partial$*, such that:*

$$\text{(Hp)} \qquad (x = \iota) \quad \textit{or} \quad \exists t'' = t' \bowtie t. \ \ x \in D_\pi(t').$$

If the method converges, i.e. $[\![t]\!]_{\approx}^{\alpha,\beta,x} \Downarrow v$*, then there exists a finite prefix* $t' \in \widehat{T}(\mathcal{S}, \pi)^\partial$*,* $t' \leq t$*, such that* $t \neq \{(\varepsilon, \pi)\}$ *if* π *is non-terminal, and*

- *if* $\lambda(\pi) = \text{Player}$ *:* $\qquad \ddot{\alpha} \uparrow (\ddot{\beta} \downarrow \ddot{v}) \ \approx \ \ddot{\alpha} \uparrow (\ddot{\beta} \downarrow (\ddot{x} \uparrow \ddot{v}(t')))$
- *if* $\lambda(\pi) = \text{Opponent}$ *:* $\qquad \ddot{\beta} \downarrow (\ddot{\alpha} \uparrow \ddot{v}) \ \approx \ \ddot{\beta} \downarrow (\ddot{\alpha} \uparrow (\ddot{x} \downarrow \ddot{v}(t')))$

Proof. By induction on the length of the derivation of $[\![t]\!]_{\approx}^{\alpha,\beta,x} \Downarrow v$. We proceed by cases on the last rule used, always assuming that $\lambda(\pi) = \text{Player}$. (The case for the opponent's position is dual by the symmetry of the axioms of the hypothesis of Alpha-Beta correctness).

- ***(base)***

$$\frac{t = \{(\varepsilon, \pi)\} \quad \pi \in \mathbb{P}_T}{[\![\, t \,]\!]_{\approx}^{\alpha,\beta,\iota} \Downarrow p(\pi)}$$

In this case, $x = \iota$. The tree t is itself the prefix we were looking for. It is, indeed, a finite tree, such that:

$$\begin{array}{rcll} \ddot{\alpha} \uparrow (\ddot{\beta} \downarrow \ddot{v}) & = & \ddot{\alpha} \uparrow (\ddot{\beta} \downarrow \ddot{x} \uparrow [p(\pi), p(\pi)]) & v = p(\pi),\ x = \iota \\ & = & \ddot{\alpha} \uparrow \{\ddot{\beta} \downarrow [\ddot{x} \uparrow \ddot{v}(t)]\} & (\ddot{v}(t) = [p(\pi), p(\pi)]) \end{array}$$

Since $\approx$ is an equivalence relation, it verifies the reflexive property, thus containing the equality relation.

- ***(more)***

In this case, the tree t has been divided into two subtrees, that have been evaluated sequentially: $t = t_1 \bowtie t_2$.

$$\frac{\begin{array}{l} t = t_1 \bowtie t_2 \\ \lambda(t\,\varepsilon) = \text{Player} \end{array} \quad \dfrac{\vdots}{[\![\, t_1 \,]\!]_{\approx}^{\alpha,\beta,x} \Downarrow v_1} \quad \dfrac{\vdots}{[\![\, t_2 \,]\!]_{\approx}^{\alpha,\beta,(x\uparrow v_1)} \Downarrow v}}{[\![\, t \,]\!]_{\approx}^{\alpha,\beta,x} \Downarrow v}$$

The evaluation of t_1 is carried out on a value x under evaluation. The pair x, t_1 verifies the hypothesis (Hp), because there exists t_1'' such that $t_1'' = t' \bowtie t_1$ and $x \in D_\pi(t')$. Therefore, by the induction hypothesis on t_1, there is a finite prefix $t_1' \le t_1$, such that:

$$\exists j. \quad \ddot{\alpha} \uparrow (\ddot{\beta} \downarrow \ddot{v}_1) \;\approx\; \ddot{\alpha} \uparrow (\ddot{\beta} \downarrow (\ddot{x} \uparrow \ddot{v}(t_1'))) \qquad (t_1.E)$$

By lemma 5.3, $v_1 \in D_\pi(t_1'')$. Hence, the initial value $x' = v_1$, used for t_2, is such that there exists $t'' = t' \bowtie t = t' \bowtie (t_1 \bowtie t_2) = (t' \bowtie t_1) \bowtie t_2 = t_1'' \bowtie t_2$ and $x' \in D_\pi(t_1'')$. Therefore, by the induction hypothesis on t_2, there is a finite prefix $t_2' \le t_2$, such that:

$$\exists i. \quad \ddot{\alpha} \uparrow (\ddot{\beta} \downarrow \ddot{v}) \;\approx\; \ddot{\alpha} \uparrow (\ddot{\beta} \downarrow (\ddot{v}_1 \uparrow \ddot{v}(t_2'))) \qquad (t_2.E)$$

Because t_1' and t_2' are, respectively, prefixes of t_1 and t_2 (not equal to $\{(\varepsilon, \pi)\}$, since π is non-terminal), the tree $t' = t_1' \cup t_2'$ is a prefix of t (not equal to $\{(\varepsilon, \pi)\}$). The co-evaluation of this prefix, $\ddot{v}(t') = \ddot{v}(t_1') \uparrow \ddot{v}(t_2')$, verifies the statement of the theorem:

$$\begin{array}{rcll} \ddot{\alpha} \uparrow (\ddot{\beta} \downarrow \ddot{v}) & \approx & \ddot{\alpha} \uparrow \ \{\ddot{\beta} \downarrow [\ddot{v}_1 \uparrow \ddot{v}(t_2')]\} & (t_2.E) \\ & \approx & \ddot{\alpha} \uparrow \ (\ddot{\beta} \downarrow \ddot{v}_1) \uparrow [\ddot{\beta} \downarrow \ddot{v}(t_2')] & (A10) \\ & \approx & \ddot{\alpha} \uparrow \ \{\ddot{\beta} \downarrow [\ddot{x} \uparrow \ddot{v}(t_1')]\} \ \uparrow \ [\ddot{\beta} \downarrow \ddot{v}(t_2')] & (A1),\ (A6),\ (t_1.E) \\ & \approx & \ddot{\alpha} \uparrow \ \{\ddot{\beta} \downarrow [\ddot{x} \uparrow \ddot{v}(t_1') \uparrow \ddot{v}(t_2')]\} & (A10) \\ & = & \ddot{\alpha} \uparrow \ \{\ddot{\beta} \downarrow [\ddot{x} \uparrow \ddot{v}(t')]\} & \text{def. de}\, t' \end{array}$$

- ***(one)***

$$\frac{\begin{array}{l} \lambda(t\,\varepsilon) = \text{Player} \\ \text{Childs}(t) = t_1 \end{array} \quad [\![\, t_1 \,]\!]_{\approx}^{(\alpha\uparrow x),\beta,\iota} \Downarrow v_1 \qquad v = x \uparrow v_1}{[\![\, t \,]\!]_{\approx}^{\alpha,\beta,x} \Downarrow v}$$

There are two sub-cases to consider, depending on the position of the child t_1, at either player or opponent.

  - $\lambda(t_1\,\varepsilon) = $ Player. In this case, the player has two subsequent moves (the game is not strictly alternating). The value currently being computed for t_1 has been set to ι. Hence, we can apply the induction hypothesis on t_1, since the condition of the theorem (Hp) holds. Therefore, there exists a finite prefix $t_1' \le t_1$ such that:

$$\exists i. \quad (\ddot{\alpha} \uparrow \ddot{x}) \uparrow (\ddot{\beta} \downarrow \ddot{v}_1) \approx (\ddot{\alpha} \uparrow \ddot{x}) \uparrow [\ddot{\beta} \downarrow (\ddot{\iota} \uparrow \ddot{v}(t_1'))] \qquad (t_1.E)$$

  And so, there exists a tree $t' = \{(\varepsilon, \pi)\} \cup \ell_1.t_1'$, that is a finite prefix of t (different from $\{(\varepsilon, \pi)\}$) verifying the statement of the theorem:

$$\begin{array}{rcll}
\ddot{\alpha}\uparrow(\ddot{\beta}\downarrow\ddot{v}) & = & \ddot{\alpha}\uparrow\{\ddot{\beta}\downarrow(\ddot{x}\uparrow\ddot{v}_1)\} & \ddot{v}=\ddot{x}\uparrow\ddot{v}_1\\
 & \approx & \ddot{\alpha}\uparrow\{\ddot{\beta}\downarrow[(\ddot{\alpha}\uparrow\ddot{x})\uparrow\ddot{v}_1]\} & \text{(Insertion), (A1)}\\
 & \approx & \ddot{\alpha}\uparrow\{\ddot{\beta}\downarrow[(\ddot{\alpha}\uparrow\ddot{x})\uparrow(\ddot{\beta}\downarrow\ddot{v}_1)]\} & \text{(Insertion)}\\
 & \approx & \ddot{\alpha}\uparrow\{\ddot{\beta}\downarrow[(\ddot{\alpha}\uparrow\ddot{x})\uparrow[\ddot{\beta}\downarrow(\ddot{\iota}\uparrow\ddot{v}(t_1'))]]\} & (t_1.\mathrm{E}),\ (\mathrm{A6})\\
 & = & \ddot{\alpha}\uparrow\{\ddot{\beta}\downarrow[(\ddot{\alpha}\uparrow\ddot{x})\uparrow[\ddot{\beta}\downarrow\ddot{v}(t_1')]]\} & \iota\uparrow y=y\\
 & \approx & \ddot{\alpha}\uparrow\{\ddot{\beta}\downarrow[(\ddot{\alpha}\uparrow\ddot{x})\uparrow\ddot{v}(t_1')]\} & \text{(Insertion)}\\
 & \approx & \ddot{\alpha}\uparrow\{\ddot{\beta}\downarrow[\ddot{x}\uparrow\ddot{v}(t_1')]\} & \text{(Insertion)}\\
 & = & \ddot{\alpha}\uparrow\{\ddot{\beta}\downarrow[\ddot{x}\uparrow\ddot{v}(t')]\} & \ddot{v}(t')=(\uparrow\ddot{v}(t_1'))=\ddot{v}(t_1')
\end{array}$$

– $\lambda(t_1\,\varepsilon)=$ Opponent. Because the value of t_1 has been set to ι, we can apply the induction hypothesis on t_1. Therefore, there exists a finite prefix $t_1'\leq t_1$ such that:

$$\exists i.\quad (\ddot{\beta}\downarrow\ddot{x})\downarrow[\ddot{\alpha}\uparrow\ddot{v}_1]\approx_i(\ddot{\beta}\downarrow\ddot{x})\downarrow[\ddot{\alpha}\uparrow(\ddot{\iota}\downarrow\ddot{v}(t_1'))]\qquad (t_1.\mathrm{E})$$

And so, there exists a tree $t'=\{(\varepsilon,\pi)\}\cup\ell_1.t_1'$, that is a finite prefix of t (different from $\{(\varepsilon,\pi)\}$) de t, verifying the statement of the theorem:

$$\begin{array}{rcll}
\ddot{\beta}\downarrow[\ddot{\alpha}\uparrow\ddot{v}] & = & \ddot{\beta}\downarrow[\ddot{\alpha}\uparrow(\ddot{x}\downarrow\ddot{v}_1)] & \ddot{v}=\ddot{x}\downarrow\ddot{v}_1\\
 & \approx & \ddot{\beta}\downarrow[\ddot{\alpha}\uparrow((\ddot{\beta}\downarrow\ddot{x})\downarrow\ddot{v}_1)] & \text{(Insertion), (A1)}\\
 & \approx & \ddot{\beta}\downarrow[\ddot{\alpha}\uparrow((\ddot{\beta}\downarrow\ddot{x})\downarrow(\ddot{\alpha}\uparrow\ddot{v}_1))] & \text{(Insertion)}\\
 & \approx & \ddot{\beta}\downarrow[\ddot{\alpha}\uparrow((\ddot{\beta}\downarrow\ddot{x})\downarrow[\ddot{\alpha}\uparrow(\ddot{\iota}\downarrow\ddot{v}(t_1'))])] & (t_1.\mathrm{E}),\ (\mathrm{A6})\\
 & = & \ddot{\beta}\downarrow[\ddot{\alpha}\uparrow((\ddot{\beta}\downarrow\ddot{x})\downarrow[\ddot{\alpha}\uparrow\ddot{v}(t_1')])] & \iota\downarrow y=y\\
 & \approx & \ddot{\beta}\downarrow[\ddot{\alpha}\uparrow((\ddot{\beta}\downarrow\ddot{x})\downarrow\ddot{v}(t_1'))] & \text{(Insertion)}\\
 & \approx & \ddot{\beta}\downarrow[\ddot{\alpha}\uparrow(\ddot{x}\downarrow\ddot{v}(t_1'))] & \text{(Insertion), (A1)}\\
 & = & \ddot{\beta}\downarrow[\ddot{\alpha}\uparrow(\ddot{x}\downarrow\ddot{v}(t'))] & \ddot{v}(t')=\ddot{v}(t_1')
\end{array}$$

- ***(cut)***

$$\frac{\lambda(\pi)=\text{Player}\qquad \alpha\uparrow\beta\ \approx\ \alpha\uparrow[\beta\downarrow(x\uparrow h_1(\pi))]}{[\![\, t\,]\!]_{\approx}^{\alpha,\beta,x}\Downarrow x}$$

Since any root-cut has at least a child, the tree $t'=t/^1$ must be a prefix of t different from $\{(\varepsilon,\pi)\}$. Let us consider the co-evaluation $\ddot{v}(t')=[y_1,y_2]$ of such a prefix. By hypothesis (Hp), there are two possibilities: either $x=\iota$, or there exists t'' such that $t''=t_x\bowtie t$ and $x\in D_\pi(t_x)$. Hence, $x\in D_\pi(\mathcal{S},p,h_1,h_2)$. We also have that $y_1,y_2\in D_\pi(t)\subseteq D_\pi(\mathcal{S},p,h_1,h_2)$. Therefore, we can apply the cut correctness axiom (A789p) in order to prove the statement of the theorem:

$$\begin{array}{rcll}
\ddot{\alpha}\uparrow(\ddot{\beta}\downarrow\ddot{v}) & = & \ddot{\alpha}\uparrow(\ddot{\beta}\downarrow\ddot{x}) & \ddot{v}=\ddot{x}\\
 & \approx & \ddot{\alpha}\uparrow(\ddot{\beta}\downarrow(\ddot{x}\uparrow[y_1,y_2])) & x,y_1,y_2\in D_\pi(\mathcal{S},p,h_1,h_2)\ ,\ (\mathrm{A789p})\\
 & = & \ddot{\alpha}\uparrow(\ddot{\beta}\downarrow[\ddot{x}\uparrow\ddot{v}(t')]) & \ddot{v}(t')=[y_1,y_2]
\end{array}$$

□

The relative correctness theorem applied, in particular, to game trees t_π, may explain the relationship existing between the Alpha-Beta algorithm and the semantics for game positions $[\![.]\!]_1$ and $[\![.]\!]_2$, obtained from the kernels (p,h_1) and (p,h_2), respectively.

Corollary 5.7. (Correctness of Alpha-Beta) *In the table of hypotheses (table 1), let* $\pi\in\mathbb{P}$*. Then:*

$$[\![\,\pi\,]\!]_{\approx}^{\iota,\iota}\Downarrow v\qquad\Rightarrow\qquad [\![\pi]\!]_1\approx[\![\pi]\!]_2\approx v$$

Proof. Condition $[\![\,\pi\,]\!]^{\iota,\iota}\Downarrow v$ implies $[\![\,t_\pi\,]\!]^{\iota,\iota,\iota}\Downarrow v$. By the relative correctness theorem, we know that there must exist a prefix t' of the game tree t_π, and such that if, on the one hand, the position is a player position:

$$\ddot{\iota}\uparrow(\ddot{\iota}\downarrow\ddot{v})\approx\ddot{\iota}\uparrow(\ddot{\iota}\downarrow(\ddot{\iota}\uparrow\ddot{v}(t')))$$

TABLE 2. CSLD resolution

$$\frac{\begin{array}{ll} \gamma \in \mathcal{P} & a{=}p(t_1,..,t_n) \in b\,,\ a'{=}p(t'_1,..,t'_n) \\ \sigma \text{ renaming} & c'' = c{\wedge}c'{\wedge}(t_1{=}t'_1){\wedge}..{\wedge}(t_n{=}t'_n) \\ \sigma \text{ s.t. } FV(?c;b){\cap}FV(\breve{\gamma}\sigma){=}\emptyset & c'' \text{ consistent in } \chi \\ \breve{\gamma}\sigma = (a' : -c'\,;\, b') & b'' = (b \setminus \{a\}) \cup b' \end{array}}{?c;b \xrightarrow[\mathcal{P}]{\chi} ?c'';b''}$$

On the other hand, if the position is an opponent position:

$$\ddot{\iota} \downarrow (\ddot{\iota} \uparrow \ddot{v}) \ \approx\ \ddot{\iota} \downarrow (\ddot{\iota} \uparrow (\ddot{\iota} \downarrow \ddot{v}(t')))$$

In both cases, the equation simplifies to $\ddot{v} \approx \ddot{v}(t')$ (because $a = \iota \uparrow a = a \uparrow \iota = \iota \downarrow a = a \downarrow \iota$) Since the pessimistic evaluation of a position can never overrun the optimistic one (because $h_1 \leq h_2$ and by monotonicity of min-max backwards-propagation), the value at position π is necessarily equivalent to ν. □

The algorithm just defined therefore computes the value of those positions that are independent on the approach chosen for their evaluation, which can either be pessimistic (with h_1) or optimistic (with h_2). In other words, the algorithm is unbiased on the approach used; it just restricts itself to finding the exact value of a position, provided one exists.

6. The Game of Constrained Logic Programming

6.1. **Background.** The basic idea of Constrained Logic Programming (CLP), with respect to Logic Programming (LP), is to provide an interpretation for terms and a subset of the predicates, by giving an *algebra of constraints* χ. The predefined predicates are usually referred to as χ-*primitives*. In addition, the (new) predicates defined by the program are usually called *logic predicates*. The set of *Horn clauses*, comprising a subset of first-order clauses, is defined by the syntax:

$$\begin{array}{rcllll} \gamma & ::= & \forall X.\gamma \ \mid\ \breve{\gamma} & & \\ \breve{\gamma} & ::= & a \vee \neg a_1 \vee .. \vee \neg a_n & (n \geq 0) & \text{(program clause)} \\ & \mid & \neg a_1 \vee .. \vee \neg a_n & (n \geq 1) & \text{(goal)} \\ & \mid & \text{false} & & \text{(empty clause)} \end{array}$$

where V is a set of variables, $X \in V$, and $a = p(t_1,..,t_n)$ is an atom where each t_i denotes a term with variables. In a CLP(χ) language, a program clause $a \vee \neg a_1 \vee .. \vee \neg a_n$ is such that the atom $a = p(t_1,..,t_n)$, called the *head* of the clause, represents a logical predicate ($p \notin \chi$), and is traditionally written $a : -a_1,..,a_n$. Also, inside the body $a_1,..,a_n$ of a clause, we group together all predefined atoms on the left-hand side, and all the logical atoms on the right-hand side. This means that program clauses have the form $a : -c; b$, where c is a conjunction of χ-primitives (called *constraints*), and b is a conjunction of logical atoms (called *body*). Both may be empty (written $\emptyset$). Likewise, a goal clause $\neg a_1 \vee .. \vee \neg a_n$ is written $?c; b$. A *program* $\mathcal{P} = \gamma_1..\gamma_n$ is a finite sequence of program clauses. Given a program $\mathcal{P}$, a goal $?c; b$ denotes a question, that of knowing if the conjunction $c \wedge b$ is a logical consequence of the program.

More precisely, what we would like to find out is the set of additional constraints c' which, added to the goal, would make $c \wedge c' \wedge b$ a logical consequence of $\mathcal{P}$. A constraint c' having this property is called an *answer* to the goal. The algebra of constraints χ has two fundamental properties: it includes the equality predicate $=$, and the satisfaction of constraints is decidable. This is the hypothesis on the basis of the CSLD resolution, that is a transition relation between goals, defined by the logical rule in the table 2. In this table, a renaming is a bijection $\sigma : V \to V$ and $FV(\gamma)$ is the set of free variables of clause γ, defined in the standard way. A *computed answer* is a constraint c' such that $?c; b \xrightarrow[\mathcal{P}]{\chi} ?c'; \emptyset$. The CSLD resolution capture the sens of logical consequence. Indeed, the semantics of computed answers is proved sound and complete with respect to the set of answers to goals. However, the most interesting result concerning our games-based interpretation is the *compositionality* property of CSLD resolution (see for example [Gabbrielli et al., 1995]).

Theorem 6.1. (Compositionality of CSLD resolution)

$$?\,d\,;\,a_1,..,a_n \overset{\chi}{\underset{\mathcal{P}}{\longmapsto}}{}^{*}\, c\,;\,\emptyset \qquad \textit{iff} \qquad \begin{cases} ?\,\mathrm{true}\,;\,a_1 \overset{\chi}{\underset{\mathcal{P}}{\longmapsto}}{}^{*}\, c_1\,;\,\emptyset \\ \vdots \\ ?\,\mathrm{true}\,;\,a_n \overset{\chi}{\underset{\mathcal{P}}{\longmapsto}}{}^{*}\, c_n\,;\,\emptyset \\ c = d \wedge c_1 \wedge .. \wedge c_n \end{cases}$$

where $n \geq 1$.

The algebra of constraints χ always comes equipped with two operations, $\oslash$ and $\owedge$, denoting, respectively, the disjunction and conjunction of constraints. We shall assume that they are mutually distributive. This hypothesis will be the basis of the definition of the game semantics for the CLP(χ) schema.

6.2. **CLP(χ) game rules.** In order to have a picture of a CLP(χ) game, think of two imaginary players engaged in a contest over a specific logic program, with the latter taking the role of the board. Given an atomic goal (that is, an initial position), the *player* must attempt to prove it using a compatible (unifiable) logical rule of the program. Once decided upon a move, the turn comes to the other player, the *opponent*, who must try to find out a sub-goal of the rule that the player will not be able to prove.

Definition 6.2. (Arena of a CLP(χ) game) The syntax of a CLP(χ) game is a 4-tuple $\mathcal{S} = (\mathbb{P}, \lambda, \rightarrow, \mathbb{P}_{\mathrm{I}})$, where:

- the set of positions $\mathbb{P}$ is a set of 4-tuples of the form $\pi =< \mathcal{P}, g, c, h >$, where $\mathcal{P}$ is the program; g is either a goal without constraints $g = ?\emptyset\,;\,b$, written $g = ?b$, or a simple atom $g = a$, where $a = p(t_1, .., t_n)$ and $p \notin \chi$; c is a constraint; and $h \in (\wp_{\mathrm{F}}^{V})^{*}$, called the *history* (of introduced variables), is a sequence of finite sets of variables;
- the function $\lambda(.)$ is defined as follows: $\lambda(\mathcal{P}, g, c, h) =$ Player if $g = a$, else (if $g = ?b$) $\lambda(\mathcal{P}, g, c, h) =$ Opponent;
- the transition relation $\rightarrow$ on positions is defined in table 3, where we suppose we have a function $\xi_V : (\wp_{\mathrm{F}}^{V})^{*} \times \mathbb{N} \rightarrow \wp^{V}$ such that $(h, i) \neq (h', i') \;\Rightarrow\; \xi_V(h, i) \cap \xi_V(h', i') = \emptyset$;
- the set of initial positions $\mathbb{P}_{\mathrm{I}}$ is the set of opponent positions with the h component equal to the set of free variables in the goal, i.e. $\mathbb{P}_{\mathrm{I}} = \{< \mathcal{P}, ?b, c, h > \mid h = FV(?\,b\,;\,c)\}$.

The function $\xi_V : (\wp_{\mathrm{F}}^{V})^{*} \times \mathbb{N} \rightarrow \wp^{V}$ allows us to partition the set of variables V in as many subsets, still infinite, as there are pairs (h, i), where h is a history and i a natural number. For any position $\pi =< \mathcal{P}, g, c, h >$, the program $\mathcal{P}$ denotes the board where the game takes place, the constraint component c represent the current payoff for the player (that can only augment), and the last component h is the sequence of finite sets of *fresh* variables, introduced during the game by a mechanism of renaming of program clauses. Programs are considered here as sequences of clauses, and not as simple sets. The notation $\gamma = \mathcal{P}_i$, not only indicates that γ is a program clause, but also that it is the i-th clause (with $i \geq 1$). This allows us to assign numbers to program clauses and, as a consequence of it, to player moves. By rule (Oppon. Moves), and in a position $< \mathcal{P}, (?a_1, .., a_n), c, h >$, there are as many possible moves of opponent as there are atoms a_i in the body of the goal. With this move, the games moves forward into the new player position $< \mathcal{P}, a_i, c, h >$, without adding any constraints or sets of variables to the existing. Intuitively speaking, the opponent has as many possibilities available as there are atoms whose conjunction must be proved false. Because of this, he does not introduce any new constraints or variables. On the other hand, by the rule (Player Moves), and in a position $< \mathcal{P}, a, c, h >$, the player has as many moves available as there are program clauses compatible with goal a and constraint c. (The meaning of the word ‘compatible’ is that of the CSLD resolution of the goal $?\,c;\,a$.) To effectively

TABLE 3. Moves in a CLP(χ) game

$$\frac{}{<\mathcal{P},(?a_1,..,a_n),c,h> \xrightarrow{i} <\mathcal{P},a_i,c,h>} \qquad \binom{\text{Oppon.}}{\text{moves}}$$

$$\frac{\begin{array}{ll} \gamma = \mathcal{P}_i & a = p(t_1,..,t_n) \\ \sigma \text{ renaming s.t. } v = FV(\breve{\gamma}\sigma) \subseteq \xi_V(h,i) & a' = p(t'_1,..,t'_n) \\ \quad \text{and } FV(?c;a) \cap FV(v) = \emptyset & c'' = c \wedge c' \wedge (t_1 = t'_1) \wedge .. \wedge (t_n = t'_n) \\ \breve{\gamma}\sigma = (a' :- c' ; b) & c'' \text{ consistent in } \chi \end{array}}{<\mathcal{P},a,c,h> \xrightarrow{i} <\mathcal{P},?b,c'',h.v>} \qquad \binom{\text{Player}}{\text{moves}}$$

$$\frac{\exists i \geq 1.\ <\mathcal{P},a,c,h> \xrightarrow{i} <\mathcal{P},?b,c',h'>}{<\mathcal{P},a,c,h> \xrightarrow{0} <\mathcal{P},?\emptyset,\text{false},h>} \qquad \binom{\text{Player}}{\text{suicide}}$$

perform her move, player will have to introduce new constraints, that will add on top of the existing ones, forming a new whole satisfiable in χ. The new constraints will depend of new variables, also added to fourth component of the position. The rule (Player suicide), applicable (only) if player has at least another move available (by rule (Player Moves)), does not really stand as a second chance for player, as it systematically drives the game into a position where the score (false) is the worst possible. Actually, this is a sort of dummy move, included here only for technical reasons; it will be useful in the backwards-propagation of tree values to recover the history component of the position, a piece of information that, otherwise, would be lost. With this game rules, therefore, the player will never have a single and unique move: There will be none, or at least two.

For this game, it is not difficult to characterize the set of terminal positions. On one side, we have the terminal positions of the opponent, that we shall note $<\mathcal{P},(?\emptyset),c,h>$, are those positions $<\mathcal{P},(?a_1,..,a_n),c,h>$ for which there are no atoms in the body of the goal ($n=0$). And on the other side, we have the terminal positions of player $<\mathcal{P},a,c,h>$ are those for which there are no compatible program clauses.

6.3. **Regular game algebra and kernels for a CLP(χ) game.** We draw inspiration for the choice of values for CLP(χ) games mainly from the work on *EX-equations* (introduced in [Marriot et al., 1994]), and, more generally, from the theory of abstract interpretation (cf. the milestone works [Cousot & Cousot, 1977] and [Cousot & Cousot, 1979], and the more specific studies on the paradigm of logic programming [Cousot & Cousot, 1992]).

The values of game will be pairs (ρ,h), composed by a logic formula ρ, called *answer*, and a history h of introduced variables. If we split the history component into two parts, $h=p.s$, a prefix p and a suffix s, we shall be able to derive, by taking the existential closure of the variables of the suffix, a new formula $\exists X_1..\exists X_n.\rho$, that we shall refer to as *EX-answer*, where $\{X_1,..,X_n\} = \bigcup_j s_j$. The reason for the existence of the history component is that later allowing the comparison of values of the greatest prefix common to their history, carried out by comparing (using the order of logical implication) the EX-answers resulting from the existential closure of the suffix variables.

Notation 6.3. In the sequel, we shall write $\exists(h).\rho$ to mean $\exists X_1..\exists X_n.\rho$, où $\{X_1,..,X_n\} = \bigcup_j h_j$.

Definition 6.4. (VALUES OF A CLP(χ) GAME) The set of *values* of a CLP(χ) game is:

$$D \;=\; \{(\rho,h) \mid \rho \in \text{Ans}(\chi,V),\; h \in (\wp_F^V)^*,\; FV(\rho) \subseteq \cup_j h_j\}$$

where the language of answers Ans(χ,V) is defined by the syntax:

$$
\begin{array}{lcl@{\qquad}l}
\rho & ::= & \bigvee_{i \in I} c_i & \text{(answer)} \\
c & ::= & a_1 \wedge .. \wedge a_n \quad (n \geq 0) & \text{(constraint)} \\
 & | & \text{false} &
\end{array}
$$

where a_i are atoms with variables in V. The formula true will be a synonym of the constraint $a_1 \wedge .. \wedge a_n$ with $n = 0$. Two game values $x = (\rho, h)$ and $y = (\rho', h')$ are said to be *strongly equivalent*, noted $x \equiv y$, if the history components are the same ($h = h'$) and the answers are logically equivalent ($\rho \equiv \rho'$) in the χ algebra, i.e. $\chi \models \forall(\rho \Leftrightarrow \rho')$.

Logically speaking, the notion of equivalence $\equiv$ on answers (logic formulae), allows us to abstract away from the inessential differences between the values of the game. For instance, the notion of strong equivalence equates the following values:

$$(\text{true} \wedge X{=}1 \wedge Y{=}3, \{X, Y\}) \;\equiv\; (X{=}1 \wedge Y{=}3, \{X, Y\}) \;\equiv\; (Y{=}3 \wedge X{=}1, \{X, Y\})$$

Definition 6.5. (Operations of a CLP(χ) game algebra) The relation $\leq$ between values of a game CLP(χ) is defined by the following rule:

$$\frac{p = \inf_{\sqsubseteq}\{h, h'\} \qquad s = FV(\rho) \backslash p \qquad r = FV(\rho') \backslash p \qquad \chi \models \forall(\exists(s).\rho \Rightarrow \exists(r).\rho')}{(\rho, h) \leq (\rho', h')}$$

The player operations are defined by $(\rho, h) \uparrow (\rho', h') = (\rho \varovee \rho', p)$ and $(\rho, h) \downarrow (\rho', h') = (\rho \varowedge \rho', p)$, where $\rho = c_1 \vee .. \vee c_n$, $\rho' = d_1 \vee .. \vee d_m$. The operations $\varovee$ and $\varowedge$ respectively correspond to the disjunction and the conjunction of constraints in algebra χ, extended to disjunctions in the standard way, i.e. $\rho \varovee \rho' = c_1 \vee .. \vee c_n \vee d_1 \vee .. \vee d_m$ and $\rho \varowedge \rho' = \bigvee_{i,j} c_i \wedge d_j$; and p is the greatest common prefix of the sequences $h = p.s$ and $h' = p.s'$, i.e. $p = \inf_{\sqsubseteq}\{h, h'\}$.

Intuitively, the player functions, of type $\uparrow: D^* \to D$ and $\downarrow: D^* \to D$, correspond, respectively, to the logical *disjunction* and *conjunction* of constraints. To put it differently, the player collects (i.e., applies the logical *or*) to all answers arising from any available moves, while the opponent verifies the coherence (i.e., performs a logical *and*) of the values of any available moves. Both functions 'develop' the history component the same way: they always restore, on the result, the common part; in other words, the greatest common prefix of the history arguments. This allows, thanks to the mechanism of backward-propagation of values, the restoration of only those variables that are relevant to the goals constituting the positions of the game.

The cost, or payoff, of terminal positions depend on the player whose turn is to play, and who did not have any moves available. If the player finds herself stuck (no program clause allows the predicate a of position $< \mathcal{P}, a, c, h >$ to be solved), then her payoff would be as expensive as she could get, namely the answer false (with history h). Conversely, if the opponent is stuck (there are no predicates in the goal of her position $< \mathcal{P}, ?\emptyset, c, h >$), then the payoff is the answer (c, h). Once again, we refer to the worst score possible at the current position, for the player whose turn is to play. Indeed, the opponent cannot force her rival to pay an even more important price than that collected up during the match. She should then "content" herself with the constraint accumulated up to that point. We are dealing here with a sort of game where getting stuck is not an option.

Besides, the pessimistic or optimistic character of the heuristics depends on the adopted viewpoint, which is that of the player. Since the player accumulates constraints through the application of program clauses, he cannot expect anything better than paying the price denoted by the constraint c associated to a position $< \mathcal{P}, x, c, h >$. This would be the optimistic heuristic. Otherwise, is she is really pessimistic about it, she will agree that the path taken will lead her to a satisfactory solution. Her pessimistic heuristic is therefore the answer denoted by the constraint false. The opponent's viewpoint is, of course, dual. For her, a constraint c cannot be more pessimistic (because she can hope to continue playing in order to oblige her rival to accumulate more constraints); the constraint false cannot, inversely, be more optimistic (as it would mean that she got away with her objective to prevent the player from proving the goal).

Definition 6.6. (Payoff and heuristic for a CLP(χ) game) The payoff functions $p : \mathbb{P}_T \to D$ and heuristics $h_1, h_2 : \mathbb{P}_{NT} \to D$ for a CLP(χ) game are defined by:

$$p(\pi) = \begin{cases} (\text{false}, h) & \text{if} \quad \pi =< \mathcal{P}, a, c, h > \\ \\ (c, h) & \text{if} \quad \pi =< \mathcal{P}, ?\emptyset, c, h > \end{cases} \qquad \begin{array}{rcl} h_1(< \mathcal{P}, x, c, h >) & = & (\text{false}, h) \\ \\ h_2(< \mathcal{P}, x, c, h >) & = & (c, h) \end{array}$$

Much like the payoff function, both heuristics return in all cases a formula equipped with the history of the position given to them as argument. The function history : $\mathbb{P} \to D$ is the projection of the history component of the game positions, i.e. history$(< \mathcal{P}, x, c, h >) = h$. The following lemma is a key result. It states that the values of a game position, which can either be pessimistic (obtained from p and h_1) or optimistic (obtained from p and h_2), all share the same history.

Lemma 6.7. $\forall \pi \in \mathbb{P}. \quad \forall (\rho, h) \in D_\pi(\mathcal{S}, p, h_1, h_2). \quad h = \text{history}(\pi)$

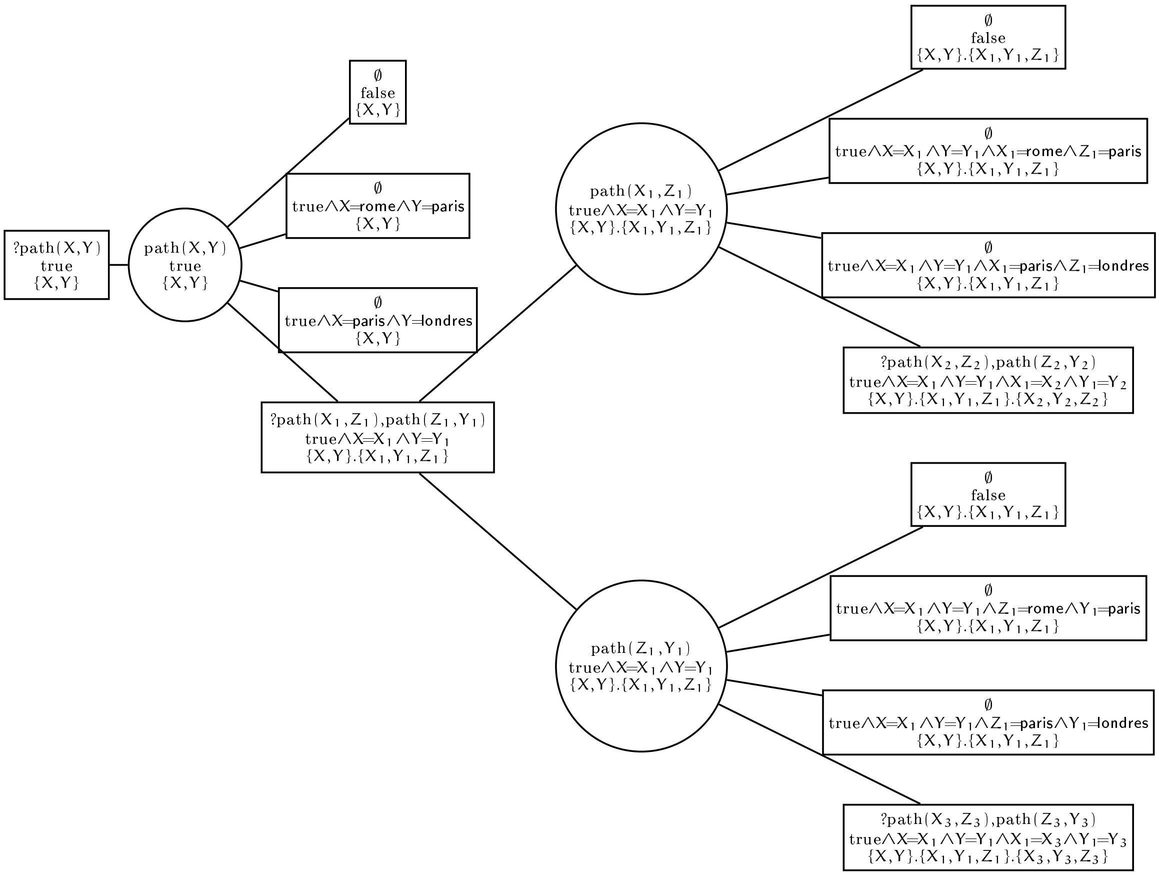
Proof. The set $D_\pi(\mathcal{S}, p, h_1, h_2) = \{(\rho, h) \in D \mid (\rho, h) = v_{p,h_1}(t) \vee (\rho, h) = v_{p,h_2}(t) \ , \ t \in \widehat{T}(\mathcal{S}, \pi)\}$ is the image corresponding to the backward-propagation of values (using h_1 or h_2) for all the finite prefixes of position π. Let $(\rho, h) \in D_\pi(\mathcal{S}, p, h_1, h_2)$. Then, there exists $t \in \widehat{T}(\mathcal{S}, \pi)$ such that $(\rho, h) = v_{p,h_1}(t)$ or $(\rho, h) = v_{p,h_2}(t)$. Let us assume the first case (the other being symmetric). We shall prove the statement of the lemma by induction on the depth (maximum address length) k of the prefix.

- $k = 0$. The prefix t is either a complete game tree, in which case π is terminal, so $v_{p,h_1}(t) = p(\pi) = (\rho, h)$ and, hence, $h = \text{history}(\pi)$; or, it is a leaf prefix at address ε. In the latter case, $v_{p,h_1}(t) = h_1(\pi) = (\rho, h)$, and hence $h = \text{history}(\pi)$.
- $k \Rightarrow k + 1$. If t is a prefix of length $k + 1$, all its children t_i are prefixes of the position trees π_i that follow π in the game. By the induction hypothesis, their values must share the same history h_i. Suppose $\lambda(\pi) =$ Player. By the rule (Player moves), the history h_i of positions π_i corresponds to that of position π, extended with the set of new variables introduced as a result of using the i-th program clause allowing the game to move forward. Therefore, $h_i = h.\nu_i$. In the game rules, we have assumed that the function ξ_V used for the generation of new variables, is such that all ν_i, as well as all history parts that will accumulate from positions π_i, are mutually disjoint. Thanks to the (Player suicide) rule, we know there must be at least a second move with h as its history component. Therefore, since $v_{p,h_1}(t) = \uparrow_{i=1}^{n} v_{p,h_1}(t_i)$, then the function of the player will restore, in the history component, the greatest common prefix, namely $h = \text{history}(\pi)$. Suppose now that $\lambda(\pi) =$ Opponent. By the (Oppon. moves) rule, the positions π_i all have the same history component as their antecedent π. The value of the prefix must then be $v_{p,h_1}(t) = \downarrow_{i=1}^{n} v_{p,h_1}(t_i)$. The opponent's function will restore, as for the player, the greatest prefix common to the histories h_i, which are all equal to $h = \text{history}(\pi)$.

□

The implications of the above lemma are manyfold. First of all, the structures $(D_\pi(\mathcal{S}, p, h_1), \leq_\pi)$ and $(D_\pi(\mathcal{S}, p, h_2), \geq_\pi)$ are both preorders. We can, therefore, use the *completion of ideals* to 'complete' them in such as way as to obtain two dcpo's (cf. [Amadio & Curien, 1998]). Second of all, the $\uparrow$ and $\downarrow$ operations are both trivially distributive (because, they both correspond to the $\otimes$ and $\oslash$ operations for the same history; and recall that they are distributive by assumption), commutative and continuous. Third, and last of all, the heuristics are both correct, h_1 for the algebra $\mathcal{D} = (D; \leq, \uparrow, \downarrow)$ and h_1 for $\mathcal{D}^\perp$. So, (p, h_1) and (p, h_2) are both kernels, giving a pessimistic semantics $[\![.]\!]_1$ and an optimistic semantics $[\![.]\!]_2$ of the CLP(χ) game where the heuristics are in the order relation $h_1(\pi) \leq_\pi h_2(\pi)$. Moreover, because the algebra $\mathcal{D}$ is regular with respect to h_1, we may write the pessimistic semantics $[\![.]\!]_1$ (denoting logical consequences) in terms of game strategies. After the example, we shall prove the adequacy of $[\![.]\!]_1$ to the traditional operational semantics, based on the CSLD resolution.

FIGURE 6.1. Example program path



Example 6.8. Consider the following logic program ((χ is the Herbrand term algebra with the equality predicate):

```
path(rome,paris).
path(paris,londres).
path(X,Y):- path(X,Z),path(Z,Y).
```

and suppose we want to solve the goal `path(X,Y)`. Using the terminology of games, this amounts to computing the semantics of the position $\pi = (\mathcal{P}, \text{path}(\mathsf{X}, \mathsf{Y}), \text{true}, \{\mathsf{X}, \mathsf{Y}\})$, where $\mathcal{P}$ is the program containing the three rules shown above. In the sequel, as in the figure, we shall omit the first components of the 4-tuple, always containing the program. The tree of position π, which is also an infinite game tree, is illustrated in figure 6.1 up to depth 5. Already in depth 3, prefix $t_\pi/^3$ provides all interesting information about the pessimistic semantics of the position. Indeed, in this depth, the min-max backwards-propagation returns (for the pessimistic heuristic), the disjunction of the four χ-constraints:

$$(\text{false} \vee (\text{true} \wedge \mathsf{X} = \mathsf{rome} \wedge \mathsf{Y} = \mathsf{paris}) \vee (\text{true} \wedge \mathsf{X} = \mathsf{paris} \wedge \mathsf{Y} = \mathsf{londres}) \vee \text{false}, \{\mathsf{X}, \mathsf{Y}\})$$

The first constraint, (false), corresponds to the suicidal move of the player (from which nothing can be obtained); the second constraint ($\text{true} \wedge \mathsf{X} = \mathsf{rome} \wedge \mathsf{Y} = \mathsf{paris}$) corresponds to the application of the first program clause `path(rome,paris)`; the third constraint ($\text{true} \wedge \mathsf{X} = \mathsf{paris} \wedge \mathsf{Y} = \mathsf{londres}$) corresponds to the application of the second program clause `path(paris,londres)`; the fourth, and last, constraint corresponds to the pessimistic value of the position reached after application of the third clause `path(X,Y):- path(X,Z),path(Z,Y)`. The resulting disjunction is then equivalent to the value:

$$((\mathsf{X}{=}\mathsf{rome} \wedge \mathsf{Y}{=}\mathsf{paris}) \vee (\mathsf{X}{=}\mathsf{paris} \wedge \mathsf{Y}{=}\mathsf{londres}), \{\mathsf{X}, \mathsf{Y}\})$$

In turn, in depth 5, the pessimistic approximation results in a value equivalent to:

$$((\mathsf{X}{=}\mathsf{rome} \wedge \mathsf{Y}{=}\mathsf{paris}) \vee (\mathsf{X}{=}\mathsf{paris} \wedge \mathsf{Y}{=}\mathsf{londres}) \vee (\mathsf{X}{=}\mathsf{rome} \wedge \mathsf{Y}{=}\mathsf{londres}), \{\mathsf{X}, \mathsf{Y}\})$$

This value denotes (once applied to the initial goal) the set of all logical consequences of the program. Besides, it would not be difficult to verify that the optimistic approximations are in all cases equal to a value equivalent to $(\mathsf{true}, \{\mathsf{X}, \mathsf{Y}\})$. This means that if we add to program $\mathcal{P}$ the logic formula $\forall \mathsf{X}, \mathsf{Y}.\, \mathsf{path}(\mathsf{X}, \mathsf{Y})$, the set of (closed) formulae obtained is *consistent*. In the next section, we are going to prove formally the adequacy of the pessimistic game semantics to the traditional CLP(χ) semantics.

6.4. **Adequacy to the operational semantics of CLP(χ).** The winning strategies for the game of CLP(χ) are those finite strategies for which the logical contents is not equivalent to the constraint false.

Definition 6.9. (Winning strategies for the game of CLP(χ)) A player strategy $\sigma \in \Sigma(\mathcal{S}, \pi)$ for the position π is a *winning* strategy if it is finite, hence $v_{p,h_1}(t_\pi(\mathcal{S}_\sigma)) = v_{p,h_2}(t_\pi(\mathcal{S}_\sigma)) = (\rho, h)$, and if it does not denote the worst of scores: $\rho \not\equiv \mathsf{false}$. We shall write $v_p(\sigma)$ for the logical contents ρ of the value of the winning strategy, and WS(π) for the set of all winning strategies of position π.

There are two types of finite strategies: those not including the suicidal move, and those that do. The payoff of any position reached from a suicidal move is, by definition of $p(.)$ in the game of CLP(χ), equal to the constraint false. Hence, no strategy containing a suicidal move can be a winning strategy.

The theorem of compositionality of the CSLD resolution (cf. theorem 6.1) in particular states (for $n = 1$) that the derivation $?\,d\,;\pi \overset{\chi}{\underset{\mathcal{P}}{\longmapsto}}{}^{*}\, c\,;\emptyset$ is equivalent to the condition $?\,\mathsf{true}\,;\pi \overset{\chi}{\underset{\mathcal{P}}{\longmapsto}}{}^{*}\, c'\,;\emptyset$, where $d \wedge c' = c$. This remarks leads us to a fact useful for the adequacy result of the game semantics for CLP(χ). Indeed, we can deduce a corollary that generalizes the results presented in [Di Cosmo et al.,1998] (lemma *goal fusion* and lemma *goal split*) about the games-based interpretation of pure logic programming (LP), to the case of CLP.

Lemma 6.10. (Goal split and fusion) CSLD resolution is such that:

$$?\,d\,;\pi_1,..,\pi_n \overset{\chi}{\underset{\mathcal{P}}{\longmapsto}}{}^{*}\, c\,;\emptyset \qquad \textit{iff} \qquad \begin{cases} ?\,d\,;\pi_1 \overset{\chi}{\underset{\mathcal{P}}{\longmapsto}}{}^{*}\, c_1\,;\emptyset \\ \vdots \\ ?\,d\,;\pi_n \overset{\chi}{\underset{\mathcal{P}}{\longmapsto}}{}^{*}\, c_n\,;\emptyset \\ c \equiv c_1 \wedge .. \wedge c_n \end{cases}$$

where $n \geq 1$.

Proof. ($\Rightarrow$) If $?\,d\,;\pi_1,..,\pi_n \overset{\chi}{\underset{\mathcal{P}}{\longmapsto}}{}^{*}\, c\,;\emptyset$, by the theorem of compositionality, for all $i = 1..n$, $?\,\mathsf{true}\,;\pi_i \overset{\chi}{\underset{\mathcal{P}}{\longmapsto}}{}^{*}\, c'_i\,;\emptyset$ and $c = d \wedge c'_1 \wedge .. \wedge c'_n$. Once again, by the same theorem (for $n = 1$), the condition $?\,\mathsf{true}\,;\pi_i \overset{\chi}{\underset{\mathcal{P}}{\longmapsto}}{}^{*}\, c'_i\,;\emptyset$ implies $?\,d\,;\pi_i \overset{\chi}{\underset{\mathcal{P}}{\longmapsto}}{}^{*}\, c_i\,;\emptyset$, where $c_i = d \wedge c'_i$. Therefore $c = d \wedge c'_1 \wedge .. \wedge c'_n \equiv (d \wedge c'_1) \wedge .. \wedge (d \wedge c'_n) = c_1 \wedge .. \wedge c_n$. ($\Leftarrow$) Suppose that for all $i = 1..n$, $?\,d\,;\pi_i \overset{\chi}{\underset{\mathcal{P}}{\longmapsto}}{}^{*}\, c_i\,;\emptyset$ and $c = c_1 \wedge .. \wedge c_n$. By the theorem of compositionality, for every $i = 1..n$ we have $?\,\mathsf{true}\,;\pi_i \overset{\chi}{\underset{\mathcal{P}}{\longmapsto}}{}^{*}\, c'_i\,;\emptyset$ where $c_i = d \wedge c'_i$. And, again, by the same theorem, $?\,d\,;\pi_1,..,\pi_n \overset{\chi}{\underset{\mathcal{P}}{\longmapsto}}{}^{*}\, c\,;\emptyset$ where $c = d \wedge c'_1 \wedge .. \wedge c'_n \equiv (d \wedge c'_1) \wedge .. \wedge (d \wedge c'_n) = c_1 \wedge .. \wedge c_n$. $\square$

The existence of a computed answer c for a given goal $?\,d; b$, that is, the existence of a CSLD derivation of the form $?\,d; b \overset{\chi}{\underset{\mathcal{P}}{\longmapsto}} ?\,c; \emptyset$, translated, in terms of games, into the existence of a winning strategy whose value is logically equivalent to c.

Theorem 6.11. (Adequacy) *The set of answers computed by CSLD resolution correspond to the set of winning strategies in the game of CLP(χ):*

$$?\,d; b \overset{\chi}{\underset{\mathcal{P}}{\longmapsto}}{}^{*}\, ?\,c; \emptyset \qquad \textit{iff} \qquad \exists \sigma \in \mathrm{WS}(\pi).\ \ v_p(\sigma) \equiv c$$

where $\pi =< \mathcal{P}, ?b, d, h >$ *and* $h = \mathsf{FV}(?d; b)$.

Proof. The proof of this result for pure logic programming (LP) can be found in [Di Cosmo et al.,1998], and is mainly based upon the compositionality lemma on computed answers. To extend it to the case of a CLP schema, the version for CLP(χ) of the compositionality lemma must be used (cf. corollary 6.10).

($\Rightarrow$) By induction on the length k of the CSLD derivation

- $k = 1$. The body of the goal must necessarily be atomic $b = A_1$, and there is a program clause (namely, the i-th), that unifies A_1, where the constraint c' is such that $c = d \wedge c'$. Hence, in the game position $\pi =< \mathcal{P}, ?A_1, d, h >$, the opponent has a single move available, leading into a player position $\pi' =< \mathcal{P}, A_1, d, h >$, where the player is allowed to use the same clause to reach position $\pi'' =< \mathcal{P}, \emptyset, c, h'' >$. We can, therefore, construct the strategy $\sigma = \{(\varepsilon, \pi)\} \cdot_1 \{(\varepsilon, \pi')\} \cdot_i \{(\varepsilon, \pi'')\}$ with a c as value.
- $k \Rightarrow k + 1$. Suppose $?\,d; b \overset{\chi}{\underset{\mathcal{P}}{\longmapsto}}{}^{k+1} ?\,c; \emptyset$. If $b = A_1, .., A_n$, by corollary 6.10, we have:

  $$\forall i = 1..n. \quad ?\,d; b \overset{\chi}{\underset{\mathcal{P}}{\longmapsto}}{}^{*}\, ?\,c_i; \emptyset$$

  where $c_1 \wedge .. \wedge c_n \equiv c$. The individual derivations must have a length not greater than $k + 1$. We have:

  $$?\,d; A_i \ \overset{\chi}{\underset{\mathcal{P}}{\longmapsto}}\ ?\,d_i; b_i \ \overset{\chi}{\underset{\mathcal{P}}{\longmapsto}}{}^{k}\ ?\,c_i; \emptyset$$

  In position $\pi =< \mathcal{P}, (?A_1, .., A_n), d, h >$, for any move A_i chosen by the opponent, in the new position $\pi'_i =< \mathcal{P}, A_i, d, h >$, the player will have a move (by a rule ℓ_i) leading to position $\pi''_i =< \mathcal{P}, b_i, d_i, h_i >$ from which there must be, by the induction hypothesis, a winning strategy σ_i whose value will be c_i. This is precisely the definition of a winning strategy for position π. Indeed, we can construct the strategy:

  $$\sigma = \{(\varepsilon, \pi)\} \ \cdot_1 \{(\varepsilon, \pi'_1)\} \cdot_{1.\ell_1} \sigma_1 \ \ldots \ \cdot_n \{(\varepsilon, \pi'_n)\} \cdot_{n.\ell_n} \sigma_n$$

  with value $c_1 \wedge .. \wedge c_n \equiv c$.

($\Leftarrow$) By induction on the depth k of the winning strategy s.

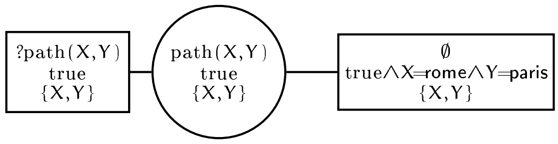
- $k = 0$. If $\sigma = \{(\varepsilon, \pi)\}$, then the position π belongs to the opponent, so $\pi =< \mathcal{P}, ?b, d, h >$, terminal, hence $b = \emptyset$, and such that $d \not\equiv$ false and $d = c$. The relation $\overset{\chi}{\underset{\mathcal{P}}{\longmapsto}}{}^{*}$ is the reflexive and transitive closure of $\overset{\chi}{\underset{\mathcal{P}}{\longmapsto}}$, and so we obtain the statement of the theorem $?\,d; b \overset{\chi}{\underset{\mathcal{P}}{\longmapsto}}{}^{*} ?\,d; b \ = \ ?\,c; \emptyset$.
- $k \Rightarrow k+1$. The strategy s has $n \geq 1$ children located at addresses $\ell_1, .., \ell_n$. Each child is, by definition, itself a strategy s_i, corresponding to a move of the opponent $A_i \in b$, with value c_i. Therefore, by the induction hypothesis, $?\,d; A_i \overset{\chi}{\underset{\mathcal{P}}{\longmapsto}}{}^{*} ?\,c_i; \emptyset$, and by corollary 6.10, we obtain the statement of the theorem $?\,d; b \overset{\chi}{\underset{\mathcal{P}}{\longmapsto}}{}^{*} ?\,c; \emptyset$, with $c_1 \wedge .. \wedge c_n \equiv c$.

$\square$

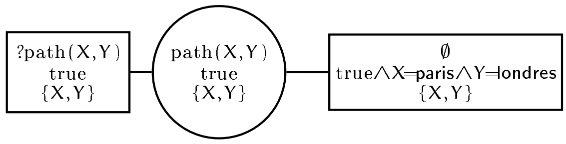
Because the operations of the players are associative and distributive, because of proposition 4.11, we are able to characterize the pessimistic semantics for a position through the set of its strategies.

FIGURE 6.2. Strategies for the *path* program

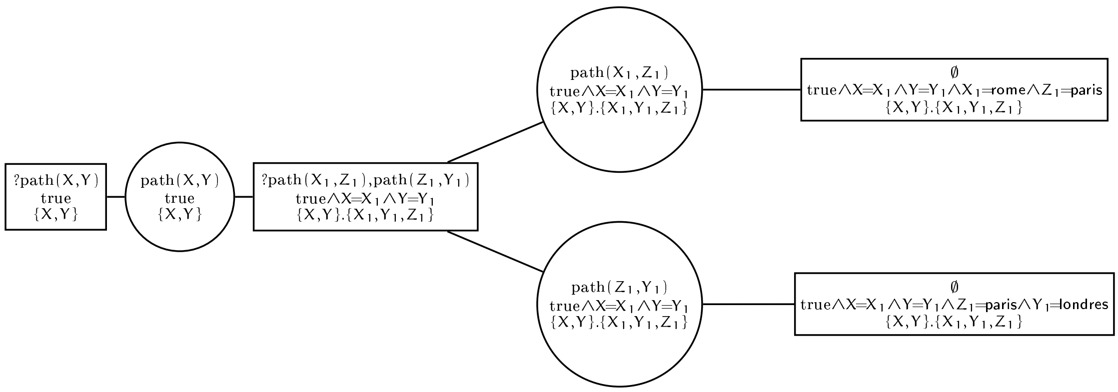
Winning strategy σ_1, logical contents $\rho_1 \equiv (\mathsf{X}{=}\mathsf{rome} \wedge \mathsf{Y}{=}\mathsf{paris})$:



Winning strategy σ_2, logical contents $\rho_2 \equiv (\mathsf{X}{=}\mathsf{paris} \wedge \mathsf{Y}{=}\mathsf{londres})$:



Winning strategy σ_3, logical contents $\rho_3 \equiv (\mathsf{X}{=}\mathsf{rome} \wedge \mathsf{Y}{=}\mathsf{londres})$:



Corollary 6.12. *The pessimistic semantics for positions coincides with the disjunction of all the winning strategies of the game:*

$$[\![\pi]\!]_1 = \Big(\bigvee_{\sigma \in \mathrm{WS}(\pi)} v_p(\sigma) \;,\; \mathrm{history}(\pi) \Big)$$

Example 6.13. In general, an upper bound may not exist for the structure of values $(D_\pi, \leq_{D_\pi})$ of a given position π, but it always exists for its completion of ideals. Let us consider, for example, the following program in the schema CLP($\mathbb{N}$):

```
p(0).
p(X+1):- p(X).
```

For goal `p(X)`, the set of answers is written as the infinite disjunction

$$(\mathsf{X} = 0) \vee (\mathsf{X} = 1) \vee (\mathsf{X} = 2) \vee \ldots \vee (\mathsf{X} = k) \vee \ldots$$

that may only be represented in the ideal completion of the preorder $(D_\pi, \leq_{D_\pi})$. In contrast to this, position $\pi = (\mathcal{P}, \mathrm{path}(\mathsf{X}, \mathsf{Y}), \mathrm{true}, \{\mathsf{X}, \mathsf{Y}\})$ analyzed in figure 6.1 (page 27), is an example of a position having a pessimistic semantics directly representable in D_π. Indeed, we can prove that it is the disjunction of the values of the three winning strategies illustrated in figure 6.2, i.e. $[\![\pi]\!]_1 = (\rho_1 \vee \rho_2 \vee \rho_3, \{\mathsf{X}, \mathsf{Y}\})$.

6.5. **Application of the Alpha-beta algorithm to CLP(χ).** The relation of strong equivalence $\equiv$ on game values, is perhaps too strict a notion (or too concrete) that makes impossible any simplifications of the computation such as the cuts that are familiar to the Alpha-Beta method. It must therefore be weakened, taking into consideration that the variables generated in the game of CLP(χ) (due to the renaming of the program clauses) are existentially closed. In other words, the abstract meaning attributed to a value (ρ, h) is that of the formula ρ where all the variables not appearing in h are existentially closed. For example, the value $(X = Y + 1, \{X\})$ gets interpreted as the formula $\exists Y. X = Y + 1$ (in itself, implicitly universally quantified).

Definition 6.14. (Abstract values, weak equivalence) The set of *abstract values* of a CLP(χ) game is defined as:

$$D_{\equiv}^{\text{EX}} = \{[\rho]_{\equiv}, h) \mid \rho \text{ is an EX–answer}, \ h \in \wp_F^{(V^*)}\}$$

The abstraction function $\delta : D \to D_{\equiv}^{\text{EX}}$, defined by $\delta(\rho, h) = ([\exists(\bar{h}).\rho]_{\equiv}, h)$, where $\bar{h} = FV(\rho) \setminus h$, transforms a game value (ρ, h) of D into a pair $([\exists(\bar{h}).\rho]_{\equiv}, h)$, where $\bar{h}$ is the set of free variables of ρ that do not occur in h. The equivalence suggested by the abstraction is the *weak equivalence* on values, written $\approx$, and defined by $x \approx y \ \Leftrightarrow \ \delta(x) = \delta(y)$.

Theorem 6.15. *The polymorphic Alpha-Beta, with the kernels* (p, h_1) *and* (p, h_2)*, as defined for a CLP(χ) game, is sound and complete modulo weak equivalence* $\approx$ *with respect to CSLD resolution. Formally:*

$$[\![< \mathcal{P}, (?b), d, h >]\!]_{\approx}^{\iota,\iota} \Downarrow (\rho, h) \qquad \Rightarrow \qquad (\bigvee_{?d;b \xrightarrow[\mathcal{P}]{\chi} ?c_i;\emptyset} c_i \ , h) \ \approx \ (\rho, h)$$

where $h = FV(?\,d; b)$.

Proof. We need to prove the set of axioms of table 1 (page 19). The player functions are trivially associative, commutative, and such that $x \downarrow x = x \uparrow x = x$. Axioms (A1), (A2), (A3), (A4), (A5) are therefore verified. Besides, if $x = (\rho, h) \approx (\rho', h') = y$, then $h = h'$. Hence, $x \uparrow z$, with $z = (\rho'', h'')$, will necessarily have the same history component as $y \uparrow z$, and it will also be shorter in length (by virtue of the application of $\uparrow$) than that of x. So, if the two formulas ρ and ρ' are equivalent in a longer history (having less existential quantifications), they will also be equivalent in a shorter history h'' (having more existential quantifications). Therefore, in as much as the logical contents is concerned, the formula $\exists(\bar{h}''). \rho \oslash \rho''$ will prove equivalent to the formula $\exists(\bar{h}''). \rho' \oslash \rho''$. The congruence proof for $\downarrow$ is dual. Axiom (A6) is thus verified. Concerning axioms (A789p) ((A789o) is proved dually), we should verify the validity of the following rule:

$$\frac{\alpha \uparrow \beta \approx \alpha \uparrow [\beta \downarrow (x \uparrow h_1(\pi))] \qquad \lambda(\pi) = \text{Player}}{\alpha \uparrow (\beta \downarrow x) \ \approx \ \alpha \uparrow [\beta \downarrow (x \uparrow y)]}$$

Assume that the hypotheses of the rule hold in the game of CLP(χ). The history component of $\alpha \uparrow \beta$ is then equal to that of $\alpha \uparrow [\beta \downarrow (x \uparrow h_1(\pi))]$, and so the history of x cannot be shorter than that of $\alpha \uparrow \beta$. Hence, the history of $\alpha \uparrow (\beta \downarrow x)$ is the same as the history of $\alpha \uparrow \beta$. Likewise, since $x, y \in D_\pi(\mathcal{S}, p, h_1, h_2) \oplus \{\iota\}$, either $y = \iota$ and the theorem is true, or the history of y cannot be shorter than the history of $h_1(\pi)$. Therefore, $\alpha \uparrow [\beta \downarrow (x \uparrow y)]$ has, too, the same history component as $\alpha \uparrow \beta$ and as $\alpha \uparrow (\beta \downarrow x)$, that we shall call h. Once the history component h has been fixed, proving the equivalence of values amounts to verifying that the logical components are equivalent, after performing the existential closure of those variables $\bar{h}$ not occurring in h. It remains then to prove that $\exists(\bar{h}). \alpha_1 \vee (\beta_1 \wedge x_1)$ is logically equivalent to $\exists(\bar{h}). \alpha_1 \vee [\beta_1 \wedge (x_1 \vee y_1)]$, where, by writing ν_1, we denote the first projection (i.e., the ρ component of $\nu = (\rho, h)$). On the one hand, we have:

$$\exists(\bar{h}). \alpha_1 \vee [\beta_1 \wedge (x_1 \vee y_1)] \ \Rightarrow \ \exists(\bar{h}). \alpha_1 \vee [\beta_1 \wedge \text{true}] \ \equiv \ \exists(\bar{h}). \alpha_1 \vee \beta_1$$

On the other hand, by the hypothesis of axiom (A789p) and by the fact that the heuristic is pessimistic, we have:

$$\exists(\bar{h}).\,\alpha_1 \vee [\beta_1 \wedge (x_1 \vee y_1)] \quad \Leftarrow \quad \exists(\bar{h}).\,\alpha_1 \vee [\beta_1 \wedge (x_1 \vee (h_1(\pi))_1)] \quad \equiv \quad \exists(\bar{h}).\,\alpha_1 \vee \beta_1$$

The logical components of $\alpha \uparrow [\beta \downarrow (x \uparrow y)]$, $\alpha \uparrow (\beta \downarrow x)$ and $\alpha \uparrow \beta$, are therefore equivalent. Finally, as far as axiom (A10) is concerned, since the player operations are distributive with respect to the strong equivalence on values, it follows that this fact should true for any other relation containing strong equivalence, such as weak equivalence $\approx$. The correctness theorem of Alpha-Beta can then be applied. We thus obtain that $[\![< \mathcal{P}, (?b), c, h >]\!]_{\approx}^{\iota,\iota} \Downarrow (\rho, h)$ implies $[\![\pi]\!]_1 \approx (\rho, h)$. The proposition is, therefore, a direct consequence of theorem 6.11. $\square$

Example 6.16. Consider the CLP($\mathbb{N}$) program composed of the following clauses:

```
1. p(X):- X>=0,X<=4.          5. r(X):- s(X),t(X).
2. p(X):- q(X),r(X).          6. s(X):- X>=2.
3. q(X):- X<=5.               7. t(X):- ...
4. r(X):- X>=3,X<=6.
```

By looking at goal `p(X)`, we realize that rule 1 of the program yields a result, the interval $[0,4]$. The latter can be partially refined by the result of the application of rule 2. On the one hand, we have that the resolution of predicate `q` restricts variable `X` to the interval $[-\infty, 5]$. On the other hand, the resolution of predicate `r`, leads to a first solution $[3,6]$, extending the solutions of the initial goal `p(X)` to the interval $[0,5]$. Any other refinement is impossible. Indeed, rule 5 restricts variable `X` to be greater than 2, but recalling that it should at the same time be smaller than 5. It is therefore useless to continue searching for any solutions as may result from rule 7, because we will not be able to incorporate other solutions besides those already obtained for $[0,5]$. In figure 6.3 we can trace the computation leading to such a simplification. At address 1.2.2.5, the position of the opponent is evaluated with α as parameter, equal to the $\uparrow$ of the incomplete values of the preceding positions of the player, namely $\alpha = \alpha_1 \uparrow \alpha_2$, where α_1 is the value currently being computed at address 1 and α_2 is the value currently being computed at 1.2.2. Hence:

$$\alpha \equiv < (X \geq 0 \wedge X \leq 4) \vee (X = Y \wedge Y \geq 3 \wedge Y \leq 6), \{X\} >$$

The β parameter, used for the evaluation of position π at address 1.2.2.5 is the value currently being computed of position 1.2, that is $\beta \equiv < (X = Y \wedge Y \leq 5), \{X\}.\{Y\} >$. After the evaluation of the first move (sub-goal `s(Z)`), the value currently being computed of π is $x \equiv < (X = Y \wedge Y = Z \wedge Z \geq 2), \{X\}.\{Y\}.\{Z\} >$. We also know that:

$$\beta \downarrow \alpha \equiv < (X = Y \wedge Y \leq 5 \wedge X \geq 0 \wedge X \leq 4) \vee (X = Y \wedge Y \leq 5 \wedge Y \geq 3), \{X\} >$$

and that:

$$\begin{aligned} \beta \downarrow (\alpha \uparrow (x \downarrow h_2(\pi))) &\equiv \beta \downarrow < (X \geq 0 \wedge X \leq 4) \vee (X = Y \wedge Y \geq 3 \wedge Y \leq 6) \\ &\qquad \vee (X = Y \wedge Y = Z \wedge Z \geq 2), \{X\} > \\ &\equiv < (X \geq 0 \wedge X \leq 4 \wedge X = Y) \vee (X = Y \wedge Y \geq 3 \wedge Y \leq 5) \\ &\qquad \vee (X = Y \wedge Y = Z \wedge Z \geq 2 \wedge Y \leq 5), \{X\} > \end{aligned}$$

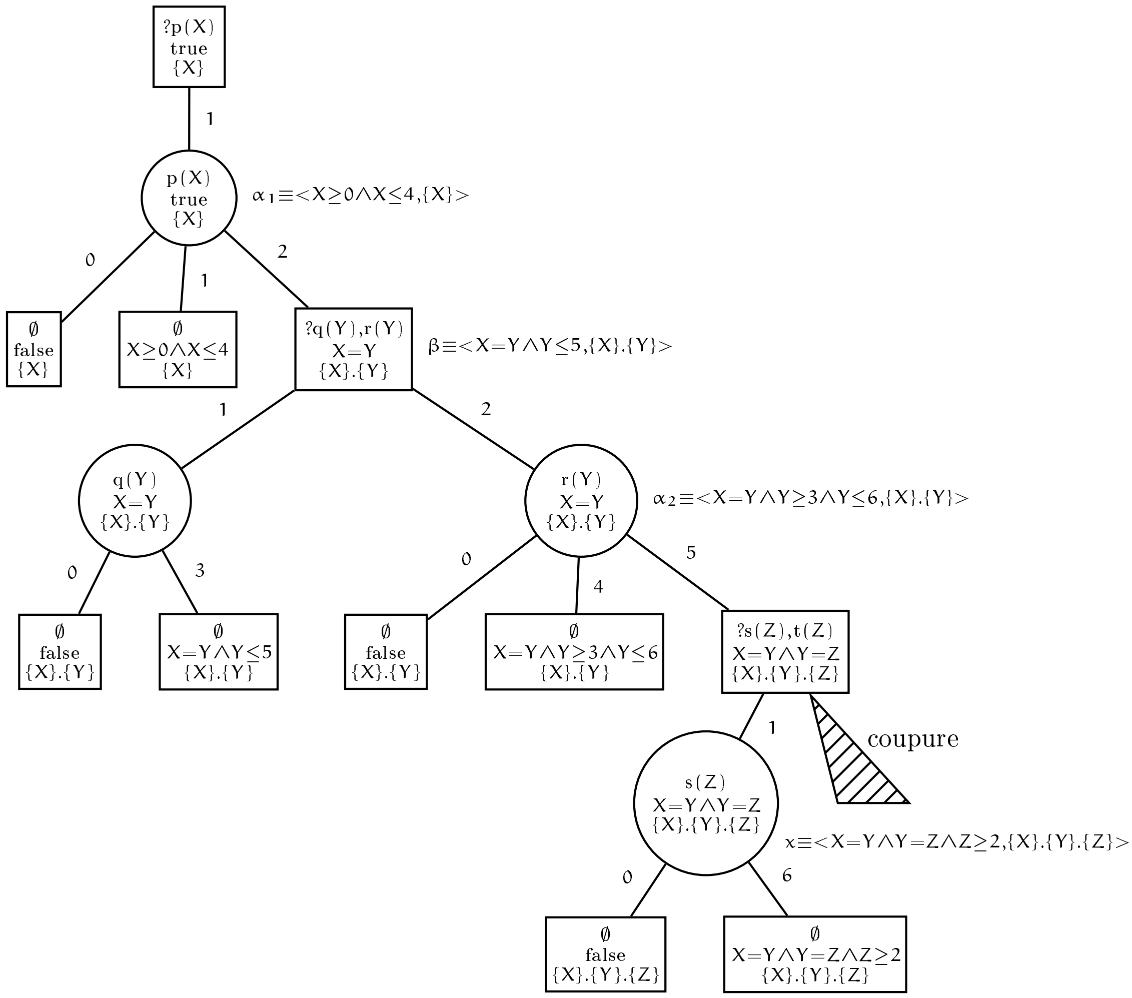
In order to verify the weak equivalence on both sides, we have to compare the existential closures of the variables not occurring in history $\{X\}$. We have then to compare the formula of $\beta \downarrow \alpha$, closed with respect to Y,

$$\exists Y.\ (X = Y \wedge Y \leq 5 \wedge X \geq 0 \wedge X \leq 4) \vee (X = Y \wedge Y \leq 5 \wedge Y \geq 3)$$
$$\equiv$$
$$\exists Y.\ (X = Y \wedge X \geq 0 \wedge X \leq 4) \vee (X = Y \wedge X \leq 5 \wedge X \geq 3)$$
$$\equiv$$
$$X \geq 0 \wedge X \leq 5$$

with the formula of $\beta \downarrow (\alpha \uparrow (x \downarrow h_2(\pi)))$, closed with respect to Y and Z:

$$\exists Y.\exists Z.\ (X \geq 0 \wedge X \leq 4 \wedge X = Y) \vee (X = Y \wedge Y \geq 3 \wedge Y \leq 5) \vee (X = Y \wedge Y = Z \wedge Z \geq 2 \wedge Y \leq 5)$$
$$\equiv$$

FIGURE 6.3. Example of Alpha-Beta pruning in a CLP(ℕ) game



$$\exists Y.\exists Z.\ (X\geq 0\wedge X\leq 4\wedge X=Y)\vee(X=Y\wedge X\geq 3\wedge X\leq 5)\vee(X=Y\wedge Y=Z\wedge X\geq 2\wedge X\leq 5)$$

$$\equiv$$

$$(X\geq 0\wedge X\leq 4)\vee(X\geq 3\wedge X\leq 5)\vee(X\geq 2\wedge X\leq 5)$$

$$\equiv$$

$$X\geq 0\wedge X\leq 5$$

By the distributivity property of the logical connectives, we know that the formulae must be logically equivalent, thus justifying the cut performed.

7. CONCLUSION

The first contribution of this work, concerns the precise identification of those conditions necessary to the application of traditional notions, such as the *value* of a position, game, or strategy. Rather surprisingly, we were able to show that the total order structure of the domains generally employed is not really essential. Quite on the contrary, we can relax this assumption and require that the relation used be simply a dcpo (or a preorder that will be completed by an ideal completion), but only locally for every position of the game, instead of globally, with respect to all other values. Likewise, amongst all the common characteristic that are specific to functions, like *max* and *min*, we have argued that *monotonicity* (implied by *continuity*) is the only crucial property.

This permits us, provided we do not violate a few weak hypotheses, to give a "game" semantics, using an arbitrarily complex value structure.

The generalization performed also captures the case of infinite games, for which the way of interpreting the result of a match is not unique. We will be able, for instance, to use two semantics, one *pessimistic*, and one *optimistic*, to give a meaning to the game by an interval. This gives a way of characterizing, by means of games, certain problems in theoretical computer science, where notion of infinite activity is always present, and natural. For example, the notion of bisimulation for concurrent processes, defined as the greatest fixed-point of an operation on processes, may be reformulated in terms of games. The interesting boolean answer, *bisimilar* or *not bisimilar*, may be given by the optimistic semantics of a game. This is not a result of chance. The approach of a definition in terms of the greatest fixed-point is, in a certain sense, that of starting up the evaluation, in a very optimistic mood, with all possible values, in order to progressively eliminate them, through the iterative application of the operation under concern.

Having defined a (finite or infinite) game semantics, we wondered if the algorithms that compute the semantic values could possibly be adapted and generalized on their own. Some, like *SSS**, seem very difficult to adapt (as the definition of strategy does not seem applicable in all the cases). Others, like *Alpha-Beta* let themselves to generalization, beyond the structure of relative integers with max and min. We have therefore given a correctness proof for an Alpha-Beta algorithm adapted to more general structures, as is the case of distributive lattices. The algorithm is presented in an original way, in the style of the *natural* operational semantics of *Kahn*, thus simplifying the correctness proof.

As far as pure logic programming (LP) is concerned or, more generally, constrained logic programming (CLP), this work is an effort in the reunification of two independent ideas. The first one, published in [Di Cosmo et al.,1998], provided an explanation of the LP paradigm in terms of two-player combinatory games, generalizing the notions of *gain* (using *substitutions* instead of *integers*) and *operation*, the latter associated to a player (using the *mgu* of substitutions instead of *min* and *max*). The second idea, published in [Di Cosmo & Loddo, 2000], showed that the *Alpha-Beta* algorithms defined on the integers could be generalized, allowing to compute the value of a game, even when we replace the specific *min* and *max* operators of real games by the sup and inf of lattices. The application we were interested in was that of computing efficiently the set of answers of a logical goal of CLP(χ) program that were *most general* (i.e., the best from a logical viewpoint). Unfortunately, the game so defined for LP in [Di Cosmo et al.,1998] did not verify the conditions required in [Di Cosmo & Loddo, 2000]. Conversely, no formal adequacy result of the game semantics was given in [Di Cosmo & Loddo, 2000].

In this article, we have shown in a formal way that it is concretely possible to put together these two ideas to obtain the desired result: (I) have a semantics for CLP games proven correct with respect to the traditional semantics; and (II) compute in a correct and efficient way the of most general answers in a distributive CLP(χ) schema. We could obtain such a result by an additional generalization of the notion of game and its associated proofs, as presented in [Di Cosmo et al.,1998], by an additional extension of Alpha-Beta with respect to the algorithm presented in [Di Cosmo & Loddo, 2000], and, finally, by defining this algorithm in the style of structural operational semantics (SOS), thus simplifying the correctness proof and, at the same time, making it more mathematically pleasing.

This interpretation in terms of games of logic programming allows the introduction of shortcuts into the process of computation, which are novel in the resolution of a logical goal. Of course, this approach leads us to consider more efficient interpreters and compilers than those currently used in the search of the set of (most general) answers of a goal. It is important that the cut test be effective and efficient (i.e., the equivalence condition on the logically equivalence of existentially closed formulas). This calls for a constraints solver, or some auxiliary algorithm, to be able to answer efficiently to this sort of question. Concerning efficient, it is important to remark that the cuts carried out eliminate from the search space, not only the finite, but also the infinite branches of the game tree. It may happen that the cost of performing the cut was worth the pruning of an important (and possibly infinite) branch of the tree. It may also happen that Alpha-Beta stops after a finite amount of time, while the other traditional interpreters loop indefinitely. It seems we are dealing here with an algorithm that offers unprecedented advantages, at least in comparison with the classical ones based on SLD or CSLD resolution.

References

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

[Abramsky & Jung, 1994] S. Abramsky and A. Jung. *Domain Theory.* In the Handbook of Logic in Computer Science. Vol. 3, pp. 1-168. Oxford University Press. 1994.

[Amadio & Curien, 1998] R. Amadio and P.-L. Curien. *Domains and Lambda-Calculi.* Cambridge Tracts in Theoretical Computer Science. Vol. 46. Cambridge University Press. 1998.

[Asperti & Ciabattoni, 1997] A. Asperti and A. Ciabattoni. *Logica a Informatica.* McGraw-Hill. Milano. 1997.

[Baillot et al., 1997] P. Baillot, V. Danos, T. Ehrhard and L. Régnier. *Believe it or not, AJM's games model is a model of classical linear logic.* Proceedings of the twelfth Symposium on Logic in Computer Science. Varsovie. 1997.

[Barendregt, 1984] H. Barendregt. *The Lambda Calculus, Its Syntax and Semantics.* Studies in Logic and the Foundations of Mathematics. Vol. 103. North-Holland Publishing Company. Amsterdam. 1984.

[Blass, 1992] A. Blass. *A game semantics for linear logic.* Annals of Pure and Applied Logic, 56(1-3), pp. 183-220. 1992.

[Bossi et al., 1994] A. Bossi, M. Gabbrielli, G. Levi and M. Martelli. *The s-semantics approach: Theory and applications.* Journal of Logic Programming, 19-20, pp. 149-197. 1994.

[Burris & Sankap., 1981] S. Burris and H.P. Sankappanavar. *A Course in Universal Algebra.* Springer Verlag. Berlin. 1981.

[Courcelle, 1983] B. Courcelle. *Fundamental properties of infinite trees.* Theoretical computer science 25 (1983), pp. 95-169. 1983.

[Cousot & Cousot, 1977] P. Cousot and R. Cousot. *Abstract Interpretation: A Unified Lattice Model for Static Analysis of Programs by Construction or Approximations of Fixpoints.* In Proceedings of the 4$^{\text{th}}$ Annual ACM Symposium on Principles of Programming Languages. ACM Press. 1977.

[Cousot & Cousot, 1979] P. Cousot and R. Cousot. *Systematic design of program analysis frameworks.* In Proceedings of the 6$^{\text{th}}$ Annual ACM Symposium on Principles of Programming Languages. ACM Press. 1979.

[Cousot & Cousot, 1992] P. Cousot and R. Cousot. *Abstract interpretation and application to logic programs.* Journal of Logic Programming. Vo.13, n.2-3, pp.103-179. 1992.

[Curien & Herbelin, 1998] P.-L. Curien and H. Herbelin. *Computing with Abstract Bohm Trees.* In Proceedings of the 3$^{\text{th}}$ Fuji International Symposium on Functional and Logic Programming. Eds M. Sato & Y. Toyama, World Scientific, 20-3. 1998.

[Danos & Harmer, 2000] V. Danos and R. Harmer. *Probabilistic Game Semantics.* Proceedings of the 15$^{\text{th}}$ Symposium on Logic in Computer Science, pp. 204-213. 2000.

[Davey & Priestley, 1990] B. A. Davey and H. A. Priestley. *Introduction to Lattices and Order.* Cambridge University Press. 1990.

[Di Cosmo & Loddo, 2000] R. Di Cosmo and J.-V. Loddo. *Playing Logic Programs with the Alpha-Beta Algorithm.* In proceedings of the 7$^{\text{th}}$ International Conference on Logic for Programming and Automated Reasoning (LPAR'2000). St. Gilles, Ile de la Réunion, France. 2000.

[Di Cosmo et al.,1998] R. Di Cosmo, J.-V. Loddo and S. Nicolet. *A Game Semantics Foundation for Logic Programming.* In proceedings of the 6$^{\text{th}}$ International Conference on Algebraic and Logic Programming (PLIP'98). LNCS, Springer-Verlag. Pisa, Italy. 1998

[Diderich & Gengler, 1995] C. G. Diderich and M. Gengler. *Minimax and Its Applications.* Kluwer Academic Publishers, The Netherlands. Chapter: *A Survey on Minimax Trees and Associated Algorithms*, pp. 25-54. 1995.

[Eder, 1985] E. Eder. *Properties of substitutions and unifications.* Journal of symbolic Computation, Vol. 1, pp. 31-46. 1985.

[Fages, 1996] F. Fages. *Programmation logique par contraintes.* Elipse, École Polytechnique. 1996.

[Fages, 1997] F. Fages. *Constructive negation by pruning.* Journal of Logic Programming, 32(2), pp. 85-118. 1997.

[Falaschi et al., 1993] M. Falaschi, G. Levi, M. Martelli, and C. Palamidessi. *A Model-Theoretic Reconstruction of the Operational Semantics of Logic Programs.* Information and Computation, 102(1), pp 86-113. 1993.

[Fuller et al., 1973] S. H. Fuller, J. G. Gaschnig, and J. J. Gillogly. *Analysis of the alpha-beta pruning algorithm.* Technical report. Department of Computer Science, Carnegie-Mellon University, Pittsburgh, Pennsylvania. 1973.

[Gabbrielli et al., 1995] M. Gabrielli, G. M. Dore, and G. Levi. *Observable semantics for constraint logic programs.* Journal of Logic and Computation, 5(2), pp. 133-171. 1995.

[Gécseg & Steinby, 1997] F. Gécseg and M. Steinby. *Tree Languages.* In the Handbook of Formal Languages. Vol. 3. Springer-Verlag. 1997.

[Gericke, 1966] H. Gericke. *Lattice Theory.* Frederick Ungar Publishing Co. New York. 1966.

[Hart & Edwards, 1961] T. P. Hart and D. J. Edwards. *The Tree Prune (TP) algorithm.* Artificial Intelligence Project Memo 30. MIT Research Laboratory of Electronics and Computation Center, Cambridge, Massachusetts. 1961.

[Jaffar & Maher, 1994] J. Jaffar and M. J. Maher. *Constraint logic programming: A survey.* Journal of Logic Programming, 19/20, pp. 503-581. 1994.

[Joyal, 1995] A. Joyal. *Free Lattices, Communication and Money Games.* In Proceedings of the International Congress of Methodology and Philosophy of Sciences. Springer Verlag. Firenze, Italy. 1995.

[Karp, 1991] R. M. Karp. *An introduction to randomized algorithms.* Discrete Applied Mathematics 34, pp. 165-201. 1991.

[Karp & Rabin, 1987] R. M. Karp and M. O. Rabin. *Efficient randomized pattern-matching algorithms.* IBM Journal of Research and Development, Vol. 31, No. 2, pp. 249-260. 1987.

[Knuth & Moore, 1975] D. E. Knuth and R. W. Moore. *An analysis of alpha-beta pruning.* Artificial Intelligence, Vol. 6, pp. 293-326. 1975.

[Lamarche, 1995] F. Lamarche. *Games semantics for full propositional linear logic.* In Proceedings of 10^{th} Annual IEEE Symposium on Logic in Computer Science, IEEE Computer Society Press, pp. 464-473. San Diego, California. 1995.

[Lassaigne & Rougemont, 1993] R. Lassaigne and M. de Rougemont. *Logique et fondements de l'informatique.* Hermès. 1993

[Lipson, 1981] J. D. Lipson. *Elements of Algebra and Algebraic Computing.* AddisonWesley. 1981

[Loddo & Nicolet, 1998] J.-V. Loddo and S. Nicolet. *Théorie des jeux et langages de programmations.* Technical report. École normale supérieure (ENS), 45 Rue d'Ulm, Paris, France. May 1998.

[Loeckx et al., 2000] J. Loeckx, H.-D. Ehrich and M. Wolf. *Algebraic Specification of Abstract Data Types.* Handbook of Logic in Computer Science, Vol. 5, pp. 217-316. Oxford University Press. 2000.

[Lloyd, 1987] J.W. Lloyd. *Foundation of Logic Programming.* Springer Verlag. 1987.

[Mal'cev, 1973] A. I. Mal'cev. *Algebraic Systems.* Springer Verlag. 1973.

[Marriot et al., 1994] K Marriot, H Søndergaard, and N D Jones. *Denotational Abstract Interpretation of Logic Programs.* ACM Transactions on Programming Languages and Systems. 1994

[Mitchell, 1996] J. Mitchell. *Foundations for Programming Languages.* MIT Press. 1996.

[Motwani & Raghavan, 1995] R. Motwani, P. Raghavan. *Randomized Algorithms.* Cambrigde University Press. 1995.

[von Neumann, 1928] J. von Neumann. *Zur Theorie der Gesellschaftsspiele.* In Mathaematische Annalen 100, pp. 195-320. 1928.

[Owen, 1968] G. Owen. *Game Theory.* Saunders. Philadelphia. 1968.

[Palamidessi, 1990] C. Palamidessi. *Algebraic Properties of Idempotent Substitutions.* In Proceedings of the 17^{th} International Colloquium on Automata, Languages and Programming (ICALP). LNCS 443, pp. 386-399. Springer Verlag. Warwick, England. 1990

[Pearl, 1990] J. Pearl. *Heuristique. Stratégie de Recherche Intelligente pour la Résolution de Problème par Ordinateur.* Coll. Intelligence Artificielle. Cepaduès. 1990.

[Perrin & Pin, 2001] D. Perrin and J-E. Pin. *Infinite Words.* To publish; available from `http://www.liafa.jussieu.fr/~jep/Resumes/InfiniteWords.html`. February 2001.

[Plotkin, 1993] B. Plotkin. *Universal algebra, Algebraic logic and Databases.* Kluwer Publ.Co. 1993.

[Sedgewick, 1978] R. Sedgewick. *Implementing Quicksort Programs.* CACM 21(10), pp. 847-857. 1978

[Sterling & Shapiro, 1986] L. Sterling and E. Shapiro. *The Art of Prolog.* MIT Press. 1986.

[Stirling, 1997] C. Stirling. *Bisimulation, model checking and other games.* Notes for a Mathfit instructional meeting on games and computation, held in Edinburgh, Scotland. June 1997.

[Taylor, 1999] Paul Taylor. *Practical Foundations of Mathematics.* Cambridge University Press. 1999.

[Tucker & Zucker, 2000] J. V. Tucker and J. I. Zucker. *Computable Functions and Semicomputable Sets on Many-Sorted Algebras.* Handbook of Logic in Computer Science, Vol. 5, pp. 317-523, Oxford University Press. 2000.

[Welsh, 1983] D. J. A. Welsh. *Randomized algorithms.* Discrete Applied Mathematics 5, pp. 133-145. 1983.