A Game Semantics Foundation for Logic Programming

(Extended Abstract)

Roberto Di Cosmo, Jean-Vincent Loddo, and Stephane Nicolet

DMI-LIENS Ecole Normale Supérieure, 45, Rue d'Ulm, 75005 Paris, France

Abstract. We introduce a semantics of Logic Programming based on an classical Game Theory, which is proven to be sound and complete w.r.t. the traditional operational semantics and Negation as Failure. This game semantics is based on an abstract reformulation of classical results about two player games, and allows a very simple characterization of the solution set of a logic program in terms of approximations of the value of the game associated to it, which can also be used to capture in a very simple way the traditional "negation as failure" extensions. This approach to semantics also opens the way to a better understanding of the mechanisms at work in parallel implementations of logic programs and in the operational semantics of logic programs with negative goals. [1]

1 Introduction

Game theory has found, in recent years, various applications in the research field of programming languages semantics, so that game theory is a very active research subject in computer science. After the preliminary works of Lamarche [Lam95], Blass [Bla92] and Joyal [Joy95] in the early 90s, the works of Abramsky, Malacaria and Jagadeesan [AJ94] lead to the first fully abstract semantics for functional (PCF) or imperative (Idealized Algol) languages. Then, more recently, specialists of Linear Logics got interested in links between games and the geometry of interaction [PBE97], whereas Curien and Herbelin showed that certain classical abstract machines could be interpreted in terms of games [CH96].

On the down side, all these relevant works seem to use the *vocabulary* of games (player, move, game, strategy) more than the results and the techniques of traditional Game Theory: typically, nobody is interested to know, in those games, if there is a winner, and what he wins; the focus there is on the dynamic aspect of player *interaction*, and *game composition*, not on the *gain*.

This should not be taken as a criticism, but as proof of the richness of Game Theory, which can be useful even when one only takes its vocabulary: the generality of the concepts it manipulates (arenas, multiple and independent agents, strategies of cooperation or of non-cooperation, quantification of the remuneration after each game) and their intuitive nature, already provide a powerful metalanguage that allows many of the aspects of modern programming languages to be tackled.

Nevertheless, it seems to us of the highest interest to be able to show, for the first time, that classical notions like payoff, propagation functions and evaluation of a game tree are not sterile in the semantics of programming languages. In this paper, the *gain* will be the central notion, while the interaction part, like in classical Game Theory, will be rather trivial.

We will first present in this paper a simple framework for classical two player games, in which well known results from Game Theory, like the monotonicity of evaluation by subgames, can be reformulated in an abstract way, via the notion of *evaluation structure* and then take advantage of this abstract reformulation to apply these results no longer to real numbers representing monetary payoffs, but to algebraic structures built out of a notion which is at the core of logic programming: idempotent substitutions.

We will see how a logic program can be interpreted as a set of rules for a two player game and how traditional semantics like the minimum Herbrand model, the computed answers, and even negation as failure can all be recovered by means of optimistic or pessimistic approximation of the value of the game tree over a suitable domain of values (typically a set of sets of idempotent substitutions). We will also hint at some new possible semantics suggested by taking as value domain the complete lattice of idempotent substitutions described in [Pal90].

Let us say first of all that our goal is not to prove again some well established theorems in Logic

[1] A full version of the paper is available as http://www.dmi.ens.fr/ dicosmo/plgames.ps.gz

Programming, but to show that a simple, natural, intuitive framework based on the classical notion of game can be used to recover them, in an unified framework.
The slogan *"each logic program is a game"* arose by a sort of a side effect: while working on bisimulation games for concurrency theory [LN97], we implemented a Prolog prototype which turned out to be very simple (only six lines of code). Actually, it was *too* simple: it was impossible not to have the clear feeling that we had come across, in one way or the other, a structural shift of the problem, so we looked more closely at the mechanisms at work, and the outcome of this investigation was the foundation of a game semantics for logic programming that we present in this paper.

Some basic notions of Game Theory

Game theory is the branch of mathematics which tries to model and to understand the behavior that rational agents should choose in a competitive arena. We want to informally recall here some basic intuitive notions of game theory which form the basis of the next section and corresponds, mainly, to the intuitive idea of randomized strategic games and refer the interested reader to the comprehensive handbook [Ae92] for a recent introduction to the topic.

Classically, a *game* is given extensionally as a tree in which the root represents the *initial position* of a play, the nodes are *positions* and edges are *moves*. In a general n-players game, there are $n + 1$ players (n human players plus an (optional) special one, the *chance* player, used to model random processes, like throwing a dice), and nodes are labeled with the name of the player whose turn is to play in that position. The *payoff function* is a function giving terminal positions p an n-ary vector of *values* (usually reals), in which the j^{th} component represents the amount paid (or received) by player j when the game ends in position p. A *strategy* for player j is a function which associates to each of the (human) player's node exactly one of his moves in that player's position: it describes the way he will behave in a play. The most studied games in Game Theory have been the two players, zero-sum games: for them, the *Minimax Theorem* (von Neumann, 1928 [vN28]) states that the dual approaches of *maximizing* one's gains or *minimizing* one's losses lead to the same expected value for each player, so each game has *one* optimal value, the *minimax* value. This can be rephrased in terms of game trees by saying that one can compute the value of a two-player, zero-sum game by assigning the payoff values to the terminal nodes of the game tree, and propagating them bottom-up applying the *Max* (resp. *Min*) *propagation functions* to the values of the player (resp. opponent) subtrees.

It is important to note that the choice of the *Max* and *Min* functions depends on the game: usually they are the maximum and minimum over a set of reals, but there are well-known cases where a different function is needed. For example, in the presence of random events (dices etc.), the mean value is used instead of the maximum. Similarly, the game presented in this paper uses substitutions as values and intersection of substitutions as the *Max* function.

In the next section, we will rephrase and formalize these notions for two player games in an abstract setting, and clearly separate the notion of game rules, and game tree from the notion of evaluation.

Structure of the paper

The paper is organized as follows: in Section 2 we set up the formal framework for abstract two player games, and reformulate and prove in this framework some classical results related to approximations and strategies. Then we introduce the game associated to a logic program in Section 3, and turn to show how, with the proper choice of evaluation structures defined in Section 4, several classical semantics can be recovered (Sections 5 and 6). Finally, we hint at some new semantics and various applications in the Conclusions 7.

2 An abstract framework for two player games

In this section we introduce a general framework which is quite convenient to describe the large class of two-player games. We will introduce in an abstract way the general notions of a *game tree*, *evaluation of* *a game tree*, *strategy* and *winning strategy*, and a few key lemmas that are at the core of the approximation procedures of infinite (or very large) game trees.

This will allow us later on to easily deal with various variants of games and valuations specialized for Logic Programming.

2.1 Game tree

The intuitive notion of a game is best described as a tree, which is a convenient way, given the rules of the game, to describe the set of *all* possible plays starting from a given initial position, so that a particular play is just a branch of the tree.

Informally, a (possibly infinite) Game Tree is a tree whose nodes are divided into two sets of *Player* and *Opponent* nodes, representing Player and Opponent positions, and whose edges are labeled by moves.

This tree is usually given implicitly by defining the "rules of the game", which is just a function P assigning to each Player or Opponent position the set of legal moves, and an initial position G. From these elements, it is easy to build the whole tree $\Gamma(P, G)$. We will denote game trees using (possibly) infinite *terms* built via the constructors `PlayerLeaf`(G) and `OppLeaf`(G) (where G is a terminal position) , the n-ary constructors `Player` and `Opp` , and the binary constructor `Label` used to label an edge in the tree with the corresponding move of the game (we will use often a more convenient notation for labeling egdges, writing for example `Player`$(\frac{m_1}{\Gamma_1}, .., \frac{m_n}{\Gamma_n})$ instead of `Player(Label(`m_1`,`Γ_1`)`$,..,$`Label(`m_n`,`Γ_n`))`. Typically, when drawing such trees, a Player node is depicted by a circle, while an Opponent node is depicted by a square.

2.2 Game evaluation

The major aim of game theory is to study the *value* associated to a game, traditionally representing a sort of "best" gain Player can expect to get given an initial position. This *value of the game* is obtained by combining in some way all possible gains Player can obtain in all possible plays starting from the given initial position.

Traditionally, this is done by assigning a gain (or **payoff**) from a given domain D, usually the real numbers, only to the final positions, by means of a function $EvaluateTerminal$, and these values are then propagated bottom up to the root by means of two functions $\sqcup$ and $\sqcap$ which give the value of an internal Player or Opponent node, knowing the values of its sons. We require that the domain of values be a partial order $(D, \leq)$ with a maximum and a minimum element $\top, \bot$. We will additionally require $(D, \leq)$ to be ω-chain complete (i.e. any ascendent or descendent chain has *lub* and *glb* respectively) to deal with infinite game trees, but for finite game trees this is not necessary. The propagations functions $\sqcup$ and $\sqcap$ are associative and commutative variable n-ary operations ($n \geq 1$) that can be seen as generated by their binary version, so we will often give only the binary version in what follows.

In general it is important to take into account not only the strength of the terminal position, but also the way by which this position has been reached: for this, we add to the payoff of a terminal position a component which depends on the path to that node, *i.e* the sequence of moves of the play. This can be stated in an elegant way by saying that, at each turn, a player has to pay something to stay in the game, so that each move of the game has a local payoff (given by a function $Price$), which gets accumulated as the play proceeds by means of a function $+$.

More formally, an *evaluation structure* for a game is a 9-uple collecting the above elements:

$$(D,\ \top,\ \bot,\ \leq,\ \sqcup,\ \sqcap,\ +,\ EvaluateTerminal,\ Price)$$

Notation 1 *When interested in the value of a game tree, we will often need to label edges not only with the moves, but also with their associated local payoffs. Since we are more interested in payoffs than move names, we will just annotate the edges with their local payoffs, and, with a slight abuse of notation, write* `Player`$(\frac{Price_1}{\Gamma_1}, .., \frac{Price_n}{\Gamma_n})$ *instead of* `Player`$(\frac{(m_1, Price_1)}{\Gamma_1}, .., \frac{(m_n, Price_n)}{\Gamma_n})$*.*

Given an evaluation structure, it is possible to compute the value of a finite game tree as follows:

Definition 1 (Value of a finite game tree).
The value of a finite game tree is defined in a bottom-up fashion from the leaves to the root, by first giving values to the terminal nodes and then combining the values of the subtrees as follows:

- **val** *(t)* = $EvaluateTerminal(t)$ *if t is a leaf.*
- **val** *(t)* = $\sqcup(x_1 + \mathbf{val}\,(t_1), \cdots, x_n + \mathbf{val}\,(t_n))$
  if the root of t *is a* `Player` *node and* x_i = $Price(m_i)$ *are the prices of the moves* m_i *leading to opponent positions* t_i.
- **val** *(t)* = $\sqcap(x_1 + \mathbf{val}\,(t_1), \cdots, x_n + \mathbf{val}\,(t_n))$
  if the root of t *is an* `Opp` *node and* x_i = $Price(m_i)$ *are the prices of the moves* m_i *leading to player positions* t_i.

For example, the famous Minimax algorithm simply computes the value obtained by choosing as evaluation structure $(R, +\infty, -\infty, \leq, max, min, +, h, zero)$, where $zero$ is the constant zero function and h is the function, often named *heuristic* for historical reasons, that evaluates the leaves for the given game.

The game presented in this paper (see section 3 for details) will also fit into this general framework, with the construction of the game tree imposed by the rules of the logic program (this can been seen as a syntaxic analysis of the logic program) and the evaluation of the game giving the semantics of the program. In fact, we will show that different choices for the evaluations structure allow us to recover different traditional semantics (see section 4 for two usual semantics, namely the denotation of computed answers and the SLDNF extension), and suggest new ones (see 7).

2.3 Infinite game trees and approximations of a game value

If the propagation functions used to evaluate a game tree have good regularity properties (if they are all monotone, for instance), it is possible to get *approximations* of the true value of a game tree by giving approximate values to some of the leaves or some of the internal nodes, instead of fully evaluating the tree. This is the key idea for defining the value of infinite game trees as the limit of a sequence of approximations, and comes directly from traditional practice of games : for instance, the size of the chess game tree is so big that it can be considered infinite for all practical purpose, so real chess playing programs cut it at a certain level and provide an approximation evaluation of the tree using appropriate heuristics for the values of cut nodes. In general, if we do not know anything about the heuristic function used, we cannot have any idea of the quality of the approximation. However, if we know that the bias it introduces has always the same sign (that is, if we know the heuristic function always overestimate or always underestimate the true value of the node), then we can know the bias of the approximation of the root value with respect to the true (unknown) value, by using the following key lemma :

Lemma 1 (monotonicity of sub-games).
If the functions $+$, $\sqcup$ *and* $\sqcap$ *are monotone (non decreasing) w.r.t.* $\leq$, *then for any context tree* $\Gamma[.]$ *and any game trees* Γ_1 *and* Γ_2,

$$\mathbf{val}\,\Gamma_1 \leq \mathbf{val}\,\Gamma_2 \Longrightarrow \mathbf{val}\,\Gamma[\Gamma_1] \leq \mathbf{val}\,\Gamma[\Gamma_2]$$

Proof. The composition of non decreasing function is still a non decreasing function.

Using the above lemma, it is possible to approximate the value of an infinite or very large game tree with a sequence of finite approximations. An approximation is built in two steps: first one applies a function $Cut(k, \_)$ (which is formally defined in the long version of the paper) to the tree to cut it at the k^{th} Opponent level (that is, at depth $2k$), replacing all non-leaf subtrees with a special constant Ω, which represents the interruption of the tree development process.

Then one approximates the values of the Ω nodes using values with known bias and propagate them to the root to obtain an approximation of the tree value. For example, using $\top$ (resp. $\perp$) one gets the greatest (resp. lowest) approximation, that we call here optimistic (resp. pessimistic).

Definition 2 (Pessimistic and optimistic approximations of the game value).
The function $\mathbf{val}_{pess}$ *is an extension of the function* $\mathbf{val}$ *on the game trees containing the constant* Ω*, which is evaluated by the worst possible value, that is by setting* $\mathbf{val}_{pess}\Omega = \bot$*. The dual optimistic version* $\mathbf{val}_{opt}$ *evaluates* Ω *with the best possible value, that is by setting* $\mathbf{val}_{opt}\Omega = \top$*. Given a game tree* Γ*, we denote* $\mathbf{val}^k_{pess}\Gamma$ *the approximation* $\mathbf{val}_{pess}Cut(k,\Gamma)$ *and* $\mathbf{val}^k_{opt}\Gamma$ *the dual approximation* $\mathbf{val}_{opt}Cut(k,\Gamma)$*.*

Two fundamental properties of the approximations are the following

Lemma 2 (Monotonicity of progressive approximations).
If the functions $+$ *,* $\sqcup$ *and* $\sqcap$ *are non decreasing, then the function* $\mathbf{val}^k_{pess}$ *of a game tree is increasing (monotone) with respect to the cut level* k*, whereas the function* $\mathbf{val}^k_{opt}$ *is decreasing.*

Proof. Using lemma 1.

When a game tree Γ is finite then exists a minimum natural number h such that $\Gamma = Cut(h,\Gamma)$, and we will call this integer the *depth* of Γ.

Evaluation of an infinite game tree We can now consider the problem of defining the value of an infinite game-tree. We have established the monotonicity of finite approximations, so we could define the value of an infinite tree as the limit of the approximate values, but then we may get different results depending on the choice of the value for the Ω nodes. Indeed, we will define two approximate values, one pessimistic, and the other optimistic, that may or may not coincide. Notice that to ensure the existence of such limits, we need to require our value domain $(D,\leq)$ to be ω-chain complete (this is the case for R and for the evaluation structures we will use for Logic Programming).

Lemma 3. *If the functions* $+$ *,* $\sqcup$ *and* $\sqcap$ *are non decreasing, then*

– *if the game tree* Γ *is finite and has depth* h*, then*

$$\mathbf{val}^k_{pess}\Gamma \leq \mathbf{val}\,\Gamma \leq \mathbf{val}^k_{opt}\Gamma \qquad \forall k \geq 0$$

$$\mathbf{val}^k_{pess}\Gamma = \mathbf{val}\,\Gamma = \mathbf{val}^k_{opt}\Gamma \qquad \forall k \geq h$$

– *if the game tree* Γ *is infinite, then*

$$\mathbf{val}^k_{pess}\Gamma \leq \mathbf{val}^k_{opt}\Gamma \qquad \forall k \geq 0$$

so, assuming $(D,\leq)$ *is an* ω*-chain complete domain, if we define* $\mathbf{val}^\infty_{pess}\Gamma$ *and* $\mathbf{val}^\infty_{opt}\Gamma$ *as the limits of corresponding chains, we obtain*

$$\mathbf{val}^\infty_{pess}\Gamma \leq \mathbf{val}^\infty_{opt}\Gamma$$

2.4 Strategies

Finally, we recall another central notion of game theory, which relates game trees to rational behaviour: that of a *strategy* for Player or for Opponent. Intuitively, a strategy for one of the players determines uniquely a move to play in any given position. Formally, a Player (Opponent) strategy is a subtree of a game tree which is deterministic on the Player (Opponent) levels. We write $\Phi(t)$ for the Player strategies in a game tree t and $\Psi(t)$ for the Opponent strategies in a game tree t.

Remark 1 (Depth and value of a finite strategy). Finite strategies are just finite game trees of a special shape, so the definition of depth and of the value of a game tree extends naturally to strategies.

Now, we introduce the notion of *winning* strategy for the Player and the dual notion of *winning* strategy for the Opponent.

Definition 3 (Winning Strategies).

- *A winning strategy for Player in a (possibly infinite) game tree Γ is a finite player strategy $\varphi \in \Phi(\Gamma)$ such that $\mathbf{val}\,\varphi \neq \bot$. The set of all winning strategies of depth k will be denoted by W_k^Γ, and $W^\Gamma = \bigcup\limits_k W_k^\Gamma$.*
- *A winning strategy for Opponent in a (possibly infinite) game tree Γ is a finite opponent strategy $\psi \in \Psi(\Gamma)$ such that $\mathbf{val}\,\psi \neq \top$. The set of all winning strategies of depth k for the Opponent will be denoted by L_k^Γ, and $L^\Gamma = \bigcup\limits_k L_k^\Gamma$.*

A strategy fixes the behaviour of one of the players, so the value of a strategy provides an approximation from below of the value of the game for that player, and under some additional assumptions on the propagation functions (weak or strong distributivity), this approximated value can be tightly related to the approximation obtained by cutting the trees at a fixed depth.

Definition 4 (Distributive properties of propagation functions). *We will say that a binary operator g* $\leq$-distributes *(resp.* $\geq$-distributes*) with respect to another binary operator f if* (x f y) g z $\leq$ (x g z) f (y g z) *(resp.* (x f y) g z $\geq$ (x g z) f (y g z)*), and we will say that g* strongly distributes *(or distribute, for short) with respect to f if it both $\leq$-distributes and $\geq$-distributes w.r.t. g.*

For example, *min* and *max* are functions that mutually strongly distribute.

Theorem 2 (Distributivity and strategies).

- *if $\bot$ is the neutral element of $\sqcup$ and $\sqcap$* $\geq$-distributes *w.r.t. $\sqcup$, then* $\quad \bigsqcup\limits_{\varphi \in W_k^\Gamma} \mathbf{val}\,\varphi \leq \mathbf{val}_{pess}^k \Gamma$
- *if $\top$ is the neutral element of $\sqcap$ and $\sqcup$* $\leq$-distributes *w.r.t. $\sqcap$, then* $\quad \mathbf{val}_{opt}^k \Gamma \leq \bigsqcap\limits_{\psi \in L_k^\Gamma} \mathbf{val}\,\psi$

Moreover, if the operations strongly distribute, then we can write equality in place of inequality.

Proof. By induction on k using the distributivity properties of $\sqcup$ and $\sqcap$.

3 The Rules of the Game

There are traditionally two classical views of a logic program, depending on whether each clause $B \leftarrow A_1, \ldots, A_n$ in it is interpreted from right to left or from left to right:

- the logical vision: if $A_1 \wedge \cdots \wedge A_n$ then B
- the effective vision: to compute B, one must first compute $A_1, \cdots, A_n$. Another way of saying it is that B is a procedure whose body is $A_1, \cdots, A_n$

Our game theoretic alternative will be the following: to each logic program we associate a simple two player game, in which one of the players tries to prove the goal and his opponent tries to disprove it: the beauty of the approach is that the logical rules written by the programmer are *just* the rules of the game, and this allows us to get a simpler understanding of the mechanisms involved in logic programming.

Nevertheless, the game we define here give rise, for a given goal G, to a game tree which is *not* the naïve one obtained by taking the tree of all possible SLD derivations starting from G; we want to clearly separate in our framework the different features making up an SLD derivation: the construction of the tree, the visit of the tree and its evaluation. For this, we use player positions consisting of just one atom, and check the consistency of solutions using the propagation function in the opponent nodes of the game tree.

We can now give a formal definition for the game $\Gamma(P)$ associated to a logic program P and the game tree $\Gamma(P, G)$ for the game $\Gamma(P)$ starting at position G.

Definition 5 (Game of a logic program).
The two-player game associated to a logic program P has the following components:

Positions : *a* player position *is an* atomic formula A*, while an* opponent position *is a* sequence *of atomic formulae.*
Initial position : *only opponent positions are legal starting positions of a game, and Opponent begins.*
Player moves : *Player moves by choosing a (variant of a) rule $H \leftarrow A_1, .., A_n$ in P whose head H unifies with the current position A. This leads to an opponent position $(A_1\theta, .., A_n\theta)$, where θ is the mgu of A and H. If no rule unifies with the current position, the game stops in a Player leaf.*
Opponent moves : *Opponent moves by selecting one of the subgoals A_i of an opponent position $G = A_1, .., A_n$, and the game continues in the player position A_i. If the current opponent position is the empty sequence, the game stops in a Opponent leaf.*

Now that we know what are the Player and Opponent nodes, and we know how to build the sons of a Player and an Opponent position, we immediately know what is the game tree associated to a given program P and starting position G, we will write $\Gamma(G, P)$ with a little abuse of notation (confusing the logic program P with the "game rules" function P of the previous section).

Notice that the game tree built this way is *not* a complete SLD-tree, which can be obtained as the game tree of a *different* game where player and opponent positions are *both* sequences of formulae and the rules are resolution steps over sequence of atomic formulae (this different game is hinted at in [Doe93]). Our game tree is much more compact, as shown in 5.3: in fact, each SLD-tree describes a particular *visit* of the game tree, given the chosen selection rule which imposes the order of the visit of the sub-game trees at Opponent's nodes.

Notice that, as a consequence of this compactness of representation, it is not enough, for a goal to be provable, that Player always arrives to a terminal winning position: he also needs to have used compatible payoffs. In other terms, it will be the "value" of the game tree, for a given evaluation structure, that is related to provability.

We can see all this in the following example, where we use informally the evaluation structure which will be formally defined in 4.

Example 1 (A simple example). Consider the logic program P consisting of the following three rules

```
1.  path(X,Z) :- arc(X,Y), path(Y,Z)
2.  path(X,X)
3.  arc(a,b)
```

This program P defines the game $\Gamma(P)$. If we try to satisfy the goal path(X,b) in P, we use $\Gamma(P)$ with the initial position $path(X, b)$ and get the game tree $\Gamma(P, path(X, b))$, which will be the regular infinite tree of figure 1, where the gray subtree is equal to $\Gamma(P, path(X, b))$ up to the renaming of X to Y.

Remark 2. As usual, we use variants of program rules obtained by replacing the free variables of the rule with fresh variables. Here, we will also require that each application of a node formation rule uses a different supply of fresh variables.

4 Evaluation Structures for Traditional Semantics

It is now time to introduce the evaluation structures that can be used to recover the different traditional semantics. To recover the computed answer semantics and SLDNF, we choose as values sets of idempotent substitutions, ordered by inclusion

It is worth noticing that a slightly different evaluation structure, where values are multisets of idempotent substitutions can be used to give the semantics of actual prolog machines.

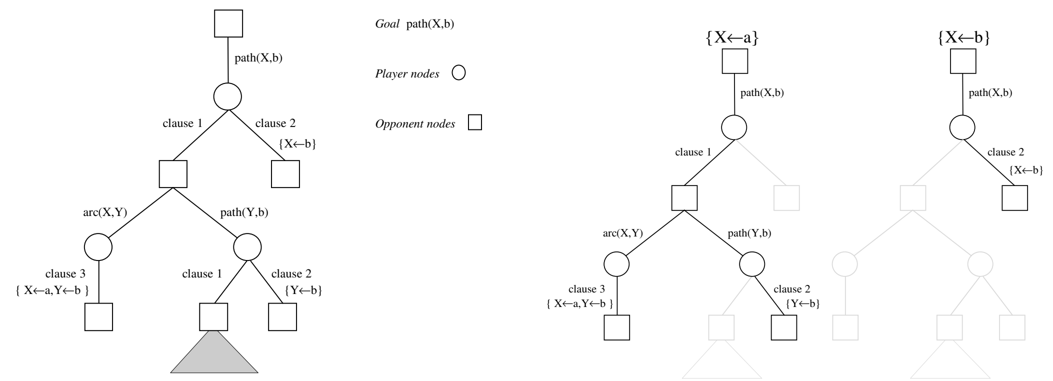

Fig. 1. An example game tree and two winning strategies.

4.1 An evaluation structure for computed answers semantics and SLDNF

We are interested here in handling the calculated substitutions of a query of a logic program, so we will construct our domain using the *complete lattice of idempotent substitutions* (introduced in [Ede85] and [Pal90]), in which each point of the lattice is a class of equivalent (up to renaming) idempotent substitutions.

domain of values : $\top$ will be the (infinite) set of all idempotent substitutions, $\bot$ will be the empty set $\emptyset$, and D will be just the powerset $(2^{\top}, \subseteq)$ of subsets of $\top$ ordered by inclusion.

payoffs : In the game, the first player unable to move will *lose*, so the function $EvaluateTerminal$ is defined as $EvaluateTerminal(\texttt{PlayerLeaf}(A)) = \bot = \emptyset$ independently from the player terminal position A, and $EvaluateTerminal(\texttt{OppLeaf}(\Box)) = \top$. The local payoff of the moves are defined as follow :
- the Player has to perform an unification step to choose a rule of the logic program, so the *payoff* of his move will just be the mgu θ used in this unification.
- the Opponent has no price to pay, so his local payoff is always ϵ, the identity substitution, and we will omit the labeling of Opponent moves altogether in what follows

accumulation of payoffs : The accumulation function will be the usual composition of substitution, extended to sets of substitions, that is, $S_1 \cdot S_2 = \{\sigma_1 \cdot \sigma_2) \mid \sigma_1 \in S_1, \sigma_2 \in S_2\}$, so that $\top = \{\epsilon\} \cdot \top$ and $\emptyset = \emptyset \cdot \top$.

propagation functions : The propagation functions $\sqcup$ and $\sqcap$ are defined as $\sqcup = \cup$ (the usual union of sets) and $\sqcap = \uparrow$, the extension to sets of substitutions of the operator of parallel composition of substitions defined in [Pal90].

We say that two substitutions θ_1 and θ_2 are *compatible* if $\theta_1 \uparrow \theta_2 \neq \emptyset$. As usual when working with logic programing, we will assume that we only keep the relevant parts of the game values.

Remark 3. It can been seen that the value of a finite game (and also the finite approximations of an infinite game) is either $\emptyset$ or can be put in the form $\{\theta_1, \ldots, \theta_n\} \cdot \top$ with $n \geq 1$. In particular, the value of a winning strategy for Player is always $\{\theta\} \cdot \top$, and we will allow us a slight abuse of notation by saying shortly that its value is θ. Since the $EvaluateTerminal$fonction is independent from terminal positions, we can forget they and denote leaves only with the constructors `PlayerLeaf`and `OppLeaf`.

Using theorem 2, we immediately obtain the following

Proposition 1 (Approximations by cut vs. strategies).

- $\mathbf{val}^k_{pess}\mathit{\Gamma} = \bigcup_{\varphi \in W^{\Gamma}_k} \mathbf{val}\,\varphi$
- $\mathbf{val}^k_{opt}\mathit{\Gamma} \subseteq \mathop{\uparrow}_{\psi \in L^{\Gamma}_k} \mathbf{val}\,\psi$

Proof. Because $\uparrow$ distributes strongly w.r.t. $\cup$, and $\cup$ $\subseteq$-distributes w.r.t. $\uparrow$.

5 Recovering the Least Herbrand Model through the pessimistic approximations

It is well known that some of the traditional semantics of logic programming, like the least Herbrand model, have operational characterizations in terms of SLD-derivations. We show here that it is possible to give an equivalent definition of these denotations in terms of winning strategies. Since the initial positions of the game are just atomic goals, we can freely use in the following the terms "atomic, ground, Herbrand, conjunctive, etc." to characterize them.

The least Herbrand model denotation (success set) of a program P becomes now the set of winning Herbrand positions for Player, that is the Herbrand positions for which he has a winning strategy:

$$W_P = \{A \mid A \text{ Herbrand position such that } \exists \text{ a winning player strategy in } \Gamma(P, A)\}$$

For the denotation of computed answers (S-semantics in [BGLM94]), we have the most general atomic positions (*i.e.* positions with the form $A(X_1, .., X_n)$), instantiated by the computed answers:

$$W_P^S = \{A\theta \mid A \text{ most general position such that } \exists \text{ a winning player strategy in } \Gamma(P, A) \text{ with value } \theta\}$$

We say that W_P and W_P^S are the two *winning position denotations* of a program P, and prove the soundness and the completeness of these game-based denotations with respect to the least Herbrand model O_P and the s-semantics O_P^S, by relating them to the well-known operational characterizations: $O_P = \{A \mid A \text{ Herbrand atom and } \exists\ A \underset{P}{\Longrightarrow}^* \square\}$ and $O_P^S = \{A\theta \mid A \text{ most general atom and } \exists\ A \overset{\theta}{\underset{P}{\Longrightarrow}}{}^* \square\}$.

5.1 Fusion and splitting of SLD derivations

In order to relate strategies to SLD-derivations, we need two key results about SLD-derivations, namely that we can fuse two successful derivations and that we can split a successful derivation starting from a conjunctive goal into several successful derivations starting from the components of the conjunction. These are stated in the following lemmas, which are immediate consequences of work in [Pal90] and [CM95], but, for completeness, we provide also a self-contained full proof of a more general result in the appendix of the full paper.

Lemma 4 (Goal fusion for successful SLD-derivations).
Suppose that θ_1 and θ_2 are compatible, that $G_1 \overset{\theta_1}{\Longrightarrow}{}^* \square$ and that $G_2 \overset{\theta_2}{\Longrightarrow}{}^* \square$, then $(G_1, G_2) \overset{\theta_1 \uparrow \theta_2}{\Longrightarrow}{}^* \square$.

Lemma 5 (Goal split for successful SLD-derivation).
Conversely, if $(G_1, G_2) \overset{\theta}{\underset{P}{\Longrightarrow}}{}^* \square$ then there exist two shorter SLD-derivations $G_1 \overset{\theta_1}{\underset{P}{\Longrightarrow}}{}^* \square$, $G_2 \overset{\theta_2}{\underset{P}{\Longrightarrow}}{}^* \square$ such that $\theta = \theta_1 \uparrow \theta_2$.

5.2 Soundness and completeness of the Winning Positions Denotations

We show here first that our game based denotations are sound, that is, whenever a winning strategy for Player exists, we can find a successful SLD-derivation. We then show that they are also complete, that is, that whenever a successful SLD-derivation exists, we can find a winning strategy for Player. What is more, the computed answer and the value of the strategy are the same.

Theorem 3 (From winning strategies to successful SLD-derivations (soundness)).
If there exists a winning strategy for Player with value θ in $\Gamma(P, G)$ then there exists a successful SLD-derivation of G in P with computed answer θ.

Proof. The idea of the proof is to construct the SLD-derivation piece by piece, while performing a visit of the winning strategy. Formally, we need an induction on the structure of the strategy and, in fact, since a winning strategy φ is a finite Player-Opponent tree that has no player leaf, we can prove the thesis by induction on the number k of opponent levels of the strategy.

- **k = 1**. If $\varphi = \texttt{OppLeaf}$ then $\Gamma(P, G) = \texttt{OppLeaf}$ then G is the empty goal by construction and we have a successful SLD-derivation of G with computed answer $\top$, which is the value of the strategy φ, as claimed.
- **Induction step** Suppose $G = (A_1, \ldots, A_n)$, so $\varphi = \texttt{Opp}(\texttt{Player}(\frac{\theta_1}{\varphi_1}), \ldots, \texttt{Player}(\frac{\theta_n}{\varphi_n}))$ with each φ_i a strategy in a sub-game γ_i of $\Gamma(P, G)$. The rules of the game imply that each of these γ_i has been obtained by using a program clause $H_i \leftarrow G_i$ in an SLD step $A_i \underset{H_i \leftarrow G_i}{\overset{\theta_i}{\Longrightarrow}} G_i\theta_i$ where φ is finite and its value is strictly better than $\emptyset$, the sub-strategies φ_i are finite and their values v_i are strictly better than $\emptyset$. Since the φ_i's are winning strategies in the γ_i's, we can apply the induction hypothesis n times to get n successful SLD-derivations $G_i\theta_i \underset{P}{\overset{v_i}{\Longrightarrow}}{}^* \square$ of $G_i\theta_i$ in P with computed answers v_i So we can get a successful SLD-derivation of A_i in P, with computed answer $\theta_i v_i$. Since $v = \theta_1 v_1 \uparrow \cdots \uparrow \theta_n v_n$ is the value of φ, a winning strategy, it is strictly better than $\emptyset$, so we can apply proposition 4 to construct a successful SLD-derivation of $(A_1, .., A_n)$ with computed answer v, and we are done.

Theorem 4 (From successful SLD-derivations to winning strategies (completeness)). *If $G \underset{P}{\overset{\theta}{\Longrightarrow}}{}^* \square$ then there exists a winning strategy for Player with value θ in the game $\Gamma(P, G)$.*

Proof. Let ξ be a SLD-derivation $G \underset{P}{\overset{\theta}{\Longrightarrow}}{}^* \square$. We shall construct a winning strategy in $\Gamma(P, G)$ The proof is by induction on the length l of ξ.

1. If G consists of a unique atom A, we consider explicitly the head and the tail of the SLD-derivation $A \overset{\theta'}{\Longrightarrow} G' \overset{\theta''}{\Longrightarrow}{}^* \square$. If $G' = \square$, the result is trivial, otherwise we apply the induction hypothesis on the tail, which gives us a winning strategy φ' in the game $\Gamma(P, G')$ with value θ''. Then the winning strategy for Player is obvious: answer the first opponent move A with the same rule that was used to make the first step of the SLD-derivation, reaching the position G', then play like φ' in $\Gamma(P, G')$. Formally, $\varphi = \texttt{Opp}(\texttt{Player}(\frac{\theta'}{\varphi'}))$ is a strategy in $\Gamma(P, G)$, with value $\mathbf{val}\,\varphi = (\theta' \cdot \mathbf{val}\,\varphi') = \theta'\theta'' = \theta$
2. If the goal G has the form $A_1, A_2, \ldots, A_n$ with $n \geq 2$ then we can apply the proposition 5 to get n successful SLD-derivations $\xi_i = A_i \overset{\theta_i}{\Longrightarrow} \square$, with the additional property $\theta = \theta_1 \uparrow \theta_2 \uparrow \cdots \uparrow \theta_n$. Applying the induction hypothesis on the ξ_i, we get n winning strategies φ_i in the games $\Gamma(P, A_i)$ with values θ_i. A winning strategy for Player follows easily: use the strategy φ_1 when the opponent plays A_1, use the strategy φ_2 when opponent plays A_2, and so on... Formally, if $\varphi_i = \texttt{Opp}(\texttt{Player}(\frac{\theta'_i}{\varphi'_i}))$ with value θ_i , then we can form the strategy $\varphi = \texttt{Opp}(\texttt{Player}(\frac{\theta'_1}{\varphi'_1}), \ldots, \texttt{Player}(\frac{\theta'_n}{\varphi'_n}))$ which has the value $\theta_1 \uparrow \theta_2 \uparrow \cdots \uparrow \theta_n = \theta$

Since we know, by Proposition 1, that $\mathbf{val}^k_{pess}\Gamma = \bigcup_{\varphi \in W^\Gamma_k} \mathbf{val}\,\varphi$, we immediately get the main result of this section:

Theorem 5 (Equivalence of computed answer semantics and the pessimistic value). *The pessimistic approximation of the value of the game associated to a logic program P on an initial position G is exactly the computed answer semantics of G in P.*

One can now trivially recover the least Herbrand model as usual.

5.3 Expressive power of winning strategies

Before turning to Negation as Failure, we pause for a moment to consider the amount of SLD-derivations which are subsumed in a game tree. It turns out that a strategy represents exactly a natural class of equivalent SLD derivations, derived from the notion of *switching* introduced in a classical technical lemma on SLD-derivations (see [Apt97]), which says, informally, that two SLD-derivation steps can be exchanged provided that in the second step an instance of an "old" atom is selected.

Definition 6 (Equivalence of SLD-derivation). *If ξ and ξ' are two SLD-derivations which satisfy the conditions of the switching lemma, then we say that ξ' is a* switch *of ξ and we write $\xi' \leftrightarrow \xi$. By definition $\leftrightarrow$ is a symmetric relation, so we define the equivalence $\approx$ as the reflexive and transitive closure of $\leftrightarrow$.*

Looking better at the proofs of the soundness and completeness results, we see that a winning strategy represents precisely the class of switch equivalent successful SLD-derivations. For a formal proof, we should reconstruct the strategy implicitly associated to an SLD-derivation and prove that any equivalent SLD-derivation corresponds to the same strategy, but here we only want to remark that equivalent SLD-derivations have the same skeleton and the strategy is indeed this skeleton.

It is possible to exactly determine the number of different (but switch-equivalent) SLD-derivations denoted by a strategy. Observe that if an SLD-derivation is the switch of another then they consist of the same number of SLD-steps (the same holds for $\approx$), so all derivations in an equivalence class have the same length. We can then compute the length $\ell(\varphi)$ and the number $\sharp(\varphi)$ of successful SLD-derivations that a strategy denotes.

$$\ell(\varphi) = \begin{cases} 0 & \text{if } \varphi = \texttt{OppLeaf}, \\ \sum\limits_{j=1}^{n} (\ell(\varphi_j) + 1) & \text{if } \varphi = \texttt{Opp}(\texttt{Player}(\frac{\theta_1}{\varphi_1}), .., \texttt{Player}(\frac{\theta_n}{\varphi_n})) \end{cases}$$

$$\sharp(\varphi) = \begin{cases} 1 & \text{if } \varphi = \texttt{OppLeaf} \\ \prod\limits_{j=1}^{n} \sharp(\varphi_j) \cdot \dfrac{\ell(\varphi)!}{\prod\limits_{j=1}^{n} (\ell(\varphi_j)+1)!} & \text{if } \varphi = \texttt{Opp}(\texttt{Player}(\frac{\theta_1}{\varphi_1}), .., \texttt{Player}(\frac{\theta_n}{\varphi_n})) \end{cases}$$

Example 2. Consider a strategy with the structure shown in the following picture (figure 2). We can calculate that this strategy alone represents 13440 different (swith-equivalent) successful SLD-derivations.

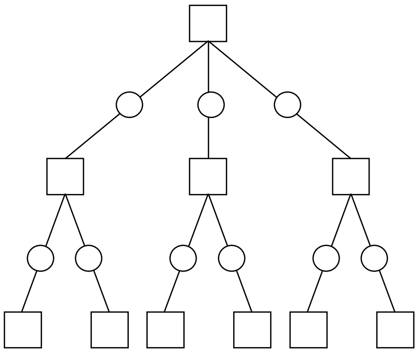

Fig. 2. A strategy with three Opponent levels

SLD trees, AND/OR trees game trees

Let us use a few words to clarify the differences between our game trees, with their winning strategies, and traditional objects in logic programming like the very old AND/OR trees which were abandoned in favor of SLD trees. First of all, it is true that a *specific* SLD tree, with a fixed selection rule, is as concise as our game trees in representing the full search space, due to the well known independence on the selection rules. But this conciseness comes, for SLD trees, at the price of the introduction of an arbitrary element: the selection rule, precisely. There is no "canonical" SLD tree, while the game tree *is* a canonical object.

On the other hand, one could hold as a criticism to game trees, like to the old AND/OR trees, the fact that there are cases where the game tree is infinite while some specific SLD tree is not, for example, the goal `p(X),q(X)` produces a finite SLD tree with the leftmost selection rule in the program with clauses `p(a)` and `q(b) :- q(b)`, while the game tree is infinite, as it contains a branch for the loop generated by the second clause that the leftmost rule avoids (while a rightmost rule would not).

But the game tree is just a mathematical object that we are interested in evaluating, and an infinite tree *can* sometimes be evaluated in finite time: in this example, the optimistic approximation yields $\emptyset$ already at the first opponent level, so we know the answer is $\emptyset$ *without* building the whole tree.

The advantage of separating the tree as a canonical object from the evaluation, which is the real semantics, is to allow a great deal of flexibility in studying the semantic properties, but one must keep in mind that, in practice, nobody will build the whole tree before evaluating it, but rather incrementally refine an initial evaluation while incrementally building the part of the tree needed for the evaluation: SLD trees are just an example of such exploration procedures.

6 Recovering SLDNF through the optimistic approximations

We have seen in the previous section that the pessimistic approximations of the game value (using its characterisation in terms of winning strategies for Player of Theorem 2) allows us to recover traditional semantics of logic programming. Turning our attention to the treatment of negation, we can now show that the sequence of *optimistic* values can be used to describe successive approximations of the greatest fix-point of the logic consequences operator T_P, thus allowing us to use the *very same* game and evaluation structure as above to recover the semantics of negation as failure. More precisely, we will prove that the process of going from an approximation level k to $k+1$ has the same pruning power as an application of T_P to $T_P^k(H)$, starting from the Herbrand base H (or, put in another way, that an application of T_P corresponds exactly to an alternation Player-Opponent in the game process).

In the following, we will say that a goal G is *game-refuted at depth* k if $\mathbf{val}^k_{opt}\Gamma(P,G)$=$\emptyset$. We will use $T_P \downarrow k$ to denote the k^{th} application of T_P, starting from the Herbrand base H.

Let's start by proving soundness, by showing that game refutations at depth k are included in finite failures exhibited after k applications of T_P.

Lemma 6 (Monotonicity by generalisation).
If $\mathbf{val}^k_{opt}\Gamma(P,G)=\emptyset$*, then* $\mathbf{val}^k_{opt}\Gamma(P,G\theta)=\emptyset$ *for all substitution* θ*.*

Lemma 7 (Game value of ground positions as binary answers).
If G *is a ground position, then* $\mathbf{val}^k_{opt}\Gamma(P,G) \in \{\emptyset, \top\}$*.*

Proof. By induction on k.

The previous lemma allows us to say that, if $A_1,..,A_n$ are ground positions such that $\mathbf{val}^k_{opt}\Gamma(P,(A_1,..,A_n))=\emptyset$, then there exists at least one A_i such that $\mathbf{val}^k_{opt}\Gamma(P,A_i)=\emptyset$.

Theorem 6 (Soundness of optimistic semantics with respect to $T_P \downarrow \omega$).
If A *is an Herbrand atom game-refutated at depth* k*, then* $A \notin T_P \downarrow k$

Proof. The initial position of the game is formed of exactely one atom, so the root has only one son (that is, the shape of the game is `Opp(PlayerLeaf)` or `Opp(Player(`$\frac{\theta_1}{\Gamma_1},\ldots,\frac{\theta_n}{\Gamma_n}$`))`. The proof proceeds by induction on the approximation level $k \geq 1$ of the game tree.

– $\mathbf{k} = \mathbf{1}$. The form of a game tree cannot be `Opp(Player(`$\frac{\theta_1}{\Gamma_1}, \ldots, \frac{\theta_n}{\Gamma_n}$`))`, otherwise its optimistic aproximation at level 1 would be different from $\emptyset$. So its form must be `Opp(PlayerLeaf)`, which means that no program clause head unifies with A, and so A cannot be in $T_P \downarrow 1$.
– **Induction step** The proof is by contradiction. Suppose that $A \in T_P \downarrow (k+1)$. By definition of T_P, there exists a program clause $B \leftarrow B_1 \cdots B_n$ and a substitution σ such that $A = B\sigma$ and $\{B_1\sigma \cdots B_n\sigma\} \subseteq T_P \downarrow k$. Then, B unifies with the atom A with a mgu θ, more general than σ The form of the game tree is `Opp(Player(`$\frac{\theta_1}{\Gamma_1}, \ldots, \frac{\theta_n}{\Gamma_n}$`))`, where one of θ_j is θ and $\Gamma_j = \Gamma(P, (B_1 ... B_n)\theta)$.
By induction hypothesis $\mathbf{val}_{opt}^{k+1}\Gamma(P, A) = \emptyset$, which implies that $\mathbf{val}_{opt}^{k}\Gamma(P, (B_1 ... B_n)\theta) = \emptyset$. Lemma 6 says that $\mathbf{val}_{opt}^{k}\Gamma(P, (B_1 ... B_n)\sigma) = \emptyset$ because θ is more general than σ. Lemma7 implies that $\exists\ i$ such that $\mathbf{val}_{opt}^{k}\Gamma(P, B_i\sigma) = \emptyset$ because the conjunctive goal $(B_1 ... B_n)\sigma$ is ground. Then, by induction hypothesis we have $B_i\sigma \notin T_P \downarrow k$ which contradicts the above result $\{B_1\sigma, \ldots, B_n\sigma\} \subseteq T_P \downarrow k$.

Having established the soundness, let us turn now to prove completeness. In the following, if S is a set of atoms, we will use the notation $[S]$ to indicate the union of the *ground instances* of the atoms of S. The following lemma is classical :

Lemma 8.
If A unifies with the head of the program clause $B \leftarrow B_1 \cdots B_n$ with the mgu θ and σ is a substitution, then

$$[A\theta\sigma] \subseteq T_P(\bigcup_{j=1}^{n} B_j\theta\sigma)$$

Theorem 7 (Optimistic approximations *survive* T_P).
Let $v = \mathbf{val}_{opt}^{k}\Gamma(P, G)$ be the k^{th} optimistic approximation of the value of a game with initial position $G = A_1, .., A_n$. Then

$$[G \cdot v] \subseteq T_P \downarrow k$$

Proof. The proof is by induction on the approximation level $k \geq 0$ of the game tree.

– $\mathbf{k} = \mathbf{0}$. The thesis is obvious because $T_P \downarrow 0$ is the Herbrand base.
– **Induction step** The game tree $\Gamma(P, G)$ has a player sub-tree Γ_i for each move (atom) $i = 1..n$. For each Γ_i the player has a move with a payoff θ_{ij} for the $j - th$ program clause $B^{ij} \leftarrow B_1^{ij} \cdots B_{n_{ij}}^{ij}$ used to reduce A_i. After the $i^t h$ move i of opponent and the move j of player we have a subtree that $\Gamma_{ij} = \Gamma(P, (B_1^{ij} \cdots B_{n_{ij}}^{ij})\theta_{ij})$.
Now consider the optimistic value of these subtrees: $v_{ij} = \mathbf{val}_{opt}^{k}\Gamma_{ij}$. Applying the induction hypothesis for all $h = 1..n_{ij}$, we get

$$[(B_h^{ij}\theta_{ij}) \cdot v_{ij}] \subseteq T_P \downarrow k \qquad \forall k \in 1, .., n_{ij}$$

So

$$I_{ij} = \bigcup_{h=1}^{n_{ij}} [(B_h^{ij}\theta_{ij}) \cdot v_{ij}] \subseteq T_P \downarrow k$$

By monotonicity of T_P operator, $T_P(I_{ij}) \subseteq T_P(T_P \downarrow k) = T_P \downarrow (k+1)$. Lemma 8 implies that $[A_i\theta_{ij}v_{ij}] \subseteq T_P(I_{ij})$ for any j. Then, $\bigcup_j [A_i\theta_{ij}v_{ij}] \subseteq T_P(I_{ij})$. One the one hand, $\bigcup_j [A_i\theta_{ij}v_{ij}] = [A_i \cdot v_i]$ where $v_i = \bigcup_j [\theta_{ij}v_{ij}]$ and on the other hand

$$v = \mathop{\uparrow}_{i=1}^{n} \bigcup_{j} \theta_{ij} v_{ij} = \mathop{\uparrow}_{i=1}^{n} v_i$$

Finally, if we take any element $\theta \in v$ there exists $\theta_1 \in v_1 \cdots \theta_n \in v_n$ such that $\theta = \mathop{\uparrow}_{i=1}^{n} \theta_i$, then any θ_i is more general than θ. This gives the leftmost inclusion in the following chain: $[A_i\theta] \subseteq [A_i\theta_i] \subseteq [A_i v_i] \subseteq T_P(I_{ij}) \subseteq T_P \downarrow (k+1)$.

This allows us to finally conclude

Corollary 1 (Completness of optimistic semantics with respect to $T_P \downarrow \omega$).

If A is an Herbrand atom such that $A \notin T_P \downarrow k$, then A is game-refuted at depth k.

Proof. By contradiction: if we suppose that $v = \mathbf{val}_{opt}^{k} \Gamma(P, A)$ is not the empty set, then, by previous theorem, $[A \cdot v] \subseteq T_P \downarrow k$. Since A is ground, we have $[A \cdot v] = \{A\}$, so we get $A \notin T_P \downarrow k$, which contradicts the hypothesis.

7 Conclusion and future work

We have presented in this paper a game theoretic approach to the semantics of Logic Programming, by providing an abstract framework for two-player games based on the notion of evaluation structure, which allows us to abstractly prove general properties of finite approximations, and we have stated the conditions under which the value of the game can be equivalently formulated using the value sets of strategies or the limit of successive approximations obtained by cutting the tree at deeper and deeper levels.

We have shown how, by choosing the proper evaluation structure, this framework can be instantiated to capture various forms of semantics proposed in the literature: from the minimum Herbrand model to the computed answers, to a very elegant treatment of SLDNF using sets of idempotent substitutions, and computed answers with multiplicity using multisets of idempotent substitutions. The key notion is here the notion of winning strategy, which concisely represents a very large class of equivalent SLD derivations, thus providing a powerful tool to investigate properties of derivations.

This showed a little of the potential of the game theoretic approach, but we are convinced that much more can be achieved by pursuing this direction of research :

- For lack of space, we can only hint here at another evaluation structure, where the domain is precisely the complete lattice of idempotent substitutions introduced by Palamidessi in [Pal90], equipped with $\uparrow$ and $\downarrow$ instead of $\uparrow$ and $\cup$. This domain seems very well adapted to perform abstract interpretation of logic programs to obtain an approximation of the real values
.

- It is clear that the game values of programs are AND-compositional (namely, $\mathbf{val}_{pess}\Gamma(P,(G_1,G_2)) = \mathbf{val}_{pess}\Gamma(P,G_1) \uparrow \mathbf{val}_{pess}\Gamma(P,G_2)$), but not OR-compositional ($\mathbf{val}_{pess}\Gamma((P \cup Q),G) \neq \mathbf{val}_{pess}\Gamma(P,G) \cup \mathbf{val}_{pess}\Gamma(P,G)$). This leaves open the problem to find an OR-compositional game-based denotation, hopefully more abstract and intuitive than the known ones.

- The relation we have established here between various traditional semantics and two player games could be the basis of a very promising transfer of results from traditional game theory to logic programming, especially in the field of parallel execution, where game theory provides a wealth of efficient execution algorithms that could now be carried over to the Logic Programming community (see [LP93,Bro96] for comprehensive bibliographies about parallelisation of Logic Programming and parallelisation game-search algorithms)

– To efficiently handle the problem of repeated position in a game, clever programs use a hash-based mechanism to check the linear history of the play for already visited positions. Could this technique be adapted to logic programs? Intuitively, the program $Q(X) \leftarrow Q(X)$ will loop forever for the goal $Q(X)$ because it will go through an infinite sequence of variants of the same position $Q(X)$, although $Q(X)$ is not included in the least fix point semantics of the program. If we code a clause using deBrujin indexes, variants of a same clause are mapped to the same term, so hashing could be applied, like in game programs. Could this be made into a nice semantic framework?

– Finally, the two ways, pessimistic or optimistic, of defining a cut-approximation computation of game values yields a duality clearly similar to the least fix-point *vs* greatest fix-point duality used to deal with positive and negative interrogation in logic programming, but in a unified framework, so we hope this will provide a means of giving a clean semantics to logic programs with negative goals. After all, the intuitive semantics of $not(A)$ is quite simple in term of games: when Opponent challenges us to show that A is not provable, we just turn the table, give him A, and say "your turn"!

We believe that the game semantic framework holds interesting promises for the Logic Programming community, if it accepts to play the game.

References

[Ae92] Aumann and Hart (eds). *Handbook of Game Theory with Economic Applications*. 1992.

[AJ94] Samson Abramsky and Rhada Jagadeesan. Games and full completeness for multiplicative linear logic. *The Journal of Symbolic Logic*, 59(2):543–574, 1994.

[Apt97] Krzysztof Apt. *From Logic Programming to Prolog*. Prentice Hall, 1997.

[BGLM94] A. Bossi, M. Gabbrielli, G. Levi, and M. Martelli. The s-semantics approach: Theory and applications. *Journal of Logic Programming*, 19-20:149–197, 1994.

[Bla92] A. Blass. A game semantics for linear logic. *Annals of Pure and Applied Logic*, 56:pages 183–220, 1992.

[Bro96] M.G. Brockington. A taxonomy of parallel game-tree search algorithms. *Journal of the International Computer Chess Association*, 19(3):162–174, 1996.

[CH96] P.L. Curien and H. Herbelin. Computing with abstract bohm trees. 1996.

[CM95] Levi Comici and Meo. Compositionality of sld-derivations and their abstractions. *ILPS*, 1995.

[Doe93] K. Doets. Levationis laus. *Journal of Logic Computation*, 3(5):pages 487–516, 1993.

[Ede85] E. Eder. Properties of substitutions and unifications. *Journal of Symbolic Computation*, (1):31–46, 1985.

[Joy95] A. Joyal. Free lattices, communication and money games. *Proceedings of the 10th International Congress of Logic, Methodology, and Philosophy of Science*, 1995.

[Lam95] F. Lamarche. Game semantics for full propositional linear logic. *Proceedings of the 10th Annual IEEE Symposium on Logic in Computer Science*, pages 464–473, 1995.

[LN97] Jean Loddo and Stéphane Nicolet. Theorie des jeux et langages de programmation. Technical report, ENS, 45, Rue d'Ulm, 1997. To appear.

[LP93] G. Levi and F. Patricelli. *Prolog : Linguaggio Applicazioni ed Implementazioni*. Scuola Superiore G. Reiss Romoli, 1993.

[Pal90] C. Palamidessi. Algebraic properties of idempotent substitutions, icalp 1990. *LNCS*, 443, 1990.

[PBE97] V. Danos P. Baillot and T. Ehrhard. Believe it or not, AJM's games model is a model of classical linear logic. *Proceedings of the 12th Annual IEEE Symposium on Logic in Computer Science*, pages pages 68–75, 1997.

[vN28] J. von Neumann. Zur Theorie der Gesellschaftsspiele. *Mathaematische Annalen*, (100):195–320, 1928.