A Game Semantics Foundation for Logic Programming

(Extended Abstract)

Roberto Di Cosmo[*] Jean-Vincent Loddo[†] Stephane Nicolet[‡]

December 9, 1997

Abstract

We introduce a semantics of Logic Programming based on classical Game Theory, which is proven to be sound and complete w.r.t. traditional semantics like the minimum Herbrand model and the s-semantics. This AND compositional game semantics allows a very simple characterization of the solution set of a logic program in term 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 novel approach to semantics opens the way to a better understanding of the mechanisms at work in parallel implementations of logic programs, and is of great pedagogical value.

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 nice works seem to use more the *vocabulary* of games (player, move, game, strategy) than the results and the techniques of traditional Game Theory. This shows that the vocabulary of Game Theory can be, because of 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 of its intuitive nature, a metalanguage that allows to tackle many of the aspects of modern programming languages. 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 value of a game tree are not sterile in the semantics of programming languages.

In this paper, we relate Game Theory and Logic Programming, using these classical notions, thus providing a first game semantics for Logic Programming.

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:

[*]DMI-LIENS - Ecole Normale Supérieure - 45, Rue d'Ulm - 75005 Paris, France - Email:`dicosmo@ens.fr`

[†]Université de Paris VII - 2, Place Jussieu - 75005 Paris, France - Email:`loddo@ens.fr`

[‡]DMI-LIENS - Ecole Normale Supérieure - 45, Rue d'Ulm - 75005 Paris, France - Email:`nicolet@ens.fr`

- the logical vision: if $A_1 \wedge \cdots \wedge A_n$ then B
- the effective vision: to compute B, one must first compute $A_1, A_2 \cdots A_n$. Another way of saying it is that B is a procedure whose body is $A_1 \cdots A_n$

We propose here a third alternative: to each logic program we associate a simple 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 to get a simpler understanding of the mechanisms involved in logic programming.

Nevertheless, our game is *not* the naïve one obtained by taking the tree of all possible SLD derivations; we wanted 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 a special evaluation function in the AND nodes of the game tree similar to the mean value used in games with random players.

The main result of the paper is that winning strategies in the game are in direct correspondence with successful SLD-derivations, but they are much more compact: one winning strategy represents an *exponential* number of "switch-equivalent" derivations (i.e. derivations differing by inessential details and having the same computed answer, see 3.3).

Whereas the traditional semantics of Logic Programming languages (*e.g.* Prolog) have been deeply investigated (see for instance the books by Lloyd [Llo87], Apt [Apt97] or Doets[Doe94] for good general sources on the usual semantics), we think that this novel game theoretic description is more concise, intuitive and easy to understand for the novice.

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. It would probably have been possible to arrive at the same results starting from the tradition of games for model theory (like the Ehrenfeucht-Fraissé games [Ebb][Sti97]), which have similar flavor. Nevertheless, to the author's best knowledge, no such attempt has been previously made.

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 introduce here some basic notions of game theory which we will use later in the paper, which 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 $j^t h$ 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) 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.

The paper is organized as follows: Section 2 introduces the game $\Gamma(P)$ associated to a logic program P, the notion of game tree $\Gamma(P, G)$, strategy and value of a game tree. Section 3 discusses the relationship between winning strategies and various semantics of logic programs, and establishes the soundness and completeness results, by reduction to the operational semantics of a logic program. In essence, we can exhibit a winning strategy in the game associated to (P, G) for each successful SLD-derivation starting with goal G in program P and viceversa. Finally, we give an exact estimate of the size of the equivalence class of SLD derivations represented by a winning strategy and we conclude by hinting at several potential applications of this new theory.

2 The Rules of the Game

We begin this section with an informal explanation of the idea of the game associated to a logic program P. Given a set of Horn clauses , we can define informally the mechanisms of a two player game, as follows:

- Each goal to satisfy defines an initial position for a different instance of the game.
- The first player (called *Player* in the rest of the paper) wants to prove that the current atomic goal A is satisfiable. He searches in the program all the clauses whose head unify with the current goal (so the moves of Player are found by a vertical scanning of the rules of P). For Player to win, that is, for the goal to be satisfied the goal, it would be sufficient for one of his moves to be winning, so Player's nodes will be denoted by 'OR' nodes in the game tree.
- The second player (called *Opponent*) wants to prevent Player from satisfying the current goal. For this, each time Player moves choosing a rule $H \leftarrow A_1, .., A_n$ of the program, Opponent answer in turn with all the sub-goals $A_1, .., A_n$, which thus represent his moves. So the moves of Opponent are found by an horizontal reading of the rules of P. To win, the Player must prove that he satisfy all these sub-positions proposed by Opponent, so Opponents's nodes will be denoted by 'AND' nodes in the game tree.
- Each player loses in a terminal node if he has no legal move in the current position. For Opponent, this means that the current goal is the empty goal $\square$ (Player has been able to apply a axiom $(H \leftarrow\ )$ of P at his last move). For Player, this means that Opponent has been able to reach a state where no clause of P unify with the current game position.
- The *payoff* of the game will model the price paid by Player each time he wants to make a move to advance in the game: it is just the substitution calculated by the unification process done

when he wants to apply one of the clauses of P. Of course, to get the resulting payoff for a terminal position of the game, one has just to sum (to compose) all the payoffs (the elementary substitutions) accumulated during the history of the game.

- We model the fact that queries given to a logic program generally consist of multiples atoms by saying that Opponent will be the first player to move in this game.

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

Definition 2.1 (Game semantics of a logic program) *The game associated to a logic program P has the following components:*

players : *the game has two players, one called* Player *and the other called* Opponent

positions : *a* player position *in the game is an atomic formula A, while an* opponent position *is a sequence of atomic formulae. Only opponent positions are legal starting positions of a game*

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 (we say that theta is the* **payoff** *of the move).*
If no rule unifies with the current position, Player looses and Opponent wins.

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, Opponent looses and Player wins.*

Notice that 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, in game theoretic terminology, that is related to provability. To compute such a value, we need to introduce the notion of a game tree, a set of values, and the evaluation functions.

Let's first introduce the idea of the game tree $\Gamma(P, G)$ of the goal G in the program P by a simple example, then we will formalize it.

Example 2.2 [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. $\Box$

As is immediately be seen from figure 1, the game trees we get here are *not* the SLD-trees used in the usual semantics of logic programming. The game tree is more abstract that the corresponding SLD-tree: in fact, each SLD-tree describes a particular *visit* of the game tree, given the chosen selection rule which impose the order the visit of the sub-game trees at Opponent's nodes.

We will denote game trees using (possibly infinite) *terms* built via the constant constructors `AndLeaf` and `OrLeaf`, the n-ary constructors `And` and `Or`, substitutions σ and the binary constructor `Label`, for which we will use the abbreviation $\texttt{Label}(\theta, t) = \frac{\theta}{t}$.

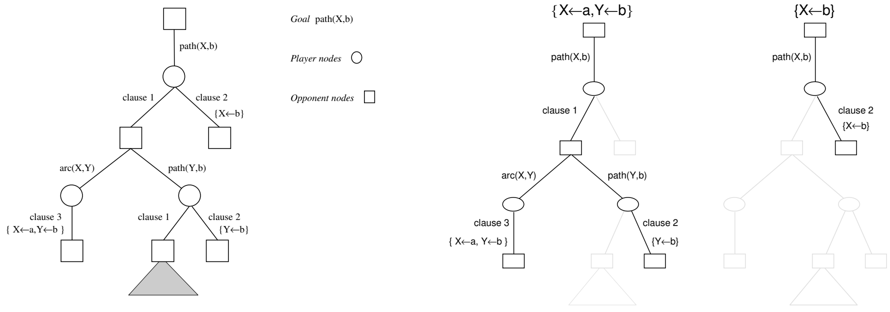

Figure 1: An example game tree and two winning strategies.

Definition 2.3 (The game tree of a goal in a logic program)
We introduce here the two mutually recursive functions f_{and} and f_{or} to construct the game tree of a goal G in a logic program P. The function f_{and} takes a goal G as argument and constructs the AND levels or the tree, whereas f_{or} takes an atom A as argument and constructs the OR levels of the tree. They are defined as follow:

- $f_{and}(\Box) = \texttt{AndLeaf}$
- $f_{and}(A_1, .., A_n) = \texttt{And}\,(f_{or}(A_1), ..., f_{or}(A_n))$
- $f_{or}(A) = \texttt{OrLeaf}$ *if Player has no move at A*
- $f_{or}(A) = \texttt{Or}\,(\frac{\theta_1}{f_{and}(G_1)}, \cdots, \frac{\theta_k}{f_{and}(G_k)})$ *if there are k moves with payoffs $\theta_1, .., \theta_k$ for Player leading to the opponent positions $G_1, .., G_k$*

A game tree is thus an AND/OR tree in which the edges going from OR nodes to AND nodes are labeled by substitutions. In the rest of the paper, we will sometimes call AND game tree and OR game tree any term beginning with the `And` and `Or` constructors, respectively.
Finally, we can define the game tree $\Gamma(G, P)$ as $f_{and}(G)$ in P.

Remark 2.4 *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.*

Following the tradition of classical theory of games, we now try to give value to the games trees associated to a logic program we've just described. Game trees will have values ranging over formal disjunctive expressions of substitutions (including a special substitution **FALSE** to denote the solution of an unsatisfiable system of equations), equipped with a partial order that extends the usual substitution partial order $\leq$ (i.e. $\sigma \leq \tau$ iff $\exists\sigma'.\tau = \sigma\sigma'$: notice we reverse the traditional notation for coherence reasons with payoffs) to disjunctive expressions. The intuition behind this is that the disjunction of two substitutions is a value representing the possibility for player to choose either one of the two, while intersection of values is the function performing the compatibility check.

We will use in what follows the usual notations and definitions for substitutions, unifiers, most general unifiers (mgu) etc. (a comprehensive survey of the properties of substitutions and unifiers can be found, for instance, in [Ede85]). We will use $\sigma, \tau, \ldots$ to range over substitutions, $\alpha, \beta, \ldots$ to range over substitutions or the special symbol **FALSE**, and v_1, v_2 to range over formal expressions (also called *values*). Also, ϵ will stand for the identity substitution. More formally

Definition 2.5 (The values of the game) *The values of the game are the formal expressions given by the grammar* $v := \textbf{FALSE} \mid \sigma \mid v \vee v$ *for which we impose that* $\vee$ *is an associative commutative symbol such that* $\textbf{FALSE} = \textbf{FALSE} \vee \textbf{FALSE}$*, and equipped with the partial order* $\sqsubseteq$ *generated by the following inequations:*

$$\begin{array}{llllll} \sigma \sqsubseteq \textbf{FALSE} & \textit{for all substitution } \sigma & (Top) & \alpha \vee v \sqsubseteq v & & (choice) \\ \sigma \sqsubseteq \tau & \textit{if } \sigma \leq \tau & (subst) & \alpha \vee v \sqsubseteq \beta \vee v & \textit{if } \alpha \sqsubseteq \beta & (\vee - monotonicity) \end{array}$$

Given a substitution θ, we note E_θ the system of equations $E_\theta = \{x = x\theta \mid x \in Dom(\theta)\}$. Since θ is clearly a solution of E_θ, E_θ is always unifiable. Moreover, in the rest of the paper, we will always consider substitutions calculated by unification algorithms, and notice that in that case, if θ is idempotent and relevant, then $\theta = mgu(E_\theta)$.

Definition 2.6 (Operation on Values) *We define the* **intersection** $\sigma \wedge \tau$ *of two substitutions* σ, τ *as* $mgu(E_\sigma \cup E_\tau)$ *if* $E_\sigma \cup E_\tau$ *is unifiable (in which case we will say that* σ *and* τ *are* compatible*), and* **FALSE** *otherwise. This associative commutative operation* $\wedge$ *is extended to arbitrary values by means of distributivity with respect to* $\vee$*. The* **composition** $\sigma \cdot v$ *of a substitution* σ *with a value* v *is defined by means of distributivity of composition of substitution with respect to* $\vee$*, with the additional equation* $\sigma \cdot \textbf{FALSE} = \textbf{FALSE}$.

It is easy to verify that $\wedge$ and $\vee$ on values are monotone with respect to the order $\sqsubseteq$.

When a game tree is finite, we can give it a value that represent the payoff for the player. Here is where it becomes apparent that we are using a special operator on AND nodes, that captures a global notion of compatibility of values, thus making the evaluation *very different* from what could be found in simple games like the ones discussed in the introduction.

Definition 2.7 (Value of a finite game tree)
The value of a game 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:

$$\begin{array}{ll} \textbf{val}\,(\texttt{AndLeaf}) = \epsilon & \textbf{val}\,(\texttt{And}\,(t_1, \cdots, t_n)) = (\textbf{val}\,t_1) \wedge \cdots \wedge (\textbf{val}\,t_n) \\ \textbf{val}\,(\texttt{OrLeaf}) = \textbf{FALSE} & \textbf{val}\,(\texttt{Or}\,(\frac{\theta_1}{t_1}, \cdots, \frac{\theta_n}{t_n})) = (\theta_1 \cdot (\textbf{val}\,t_1)) \vee \cdots \vee (\theta_n \cdot (\textbf{val}\,t_n)) \end{array}$$

2.1 Winning strategies

The central notion to relate game trees to the traditional SLD derivations is that of *strategy* for Player: intuitively, a strategy tells the player what move she/he must chose in each position where it is her/his turn to play. Formally, a strategy is a partial function from the internal Or nodes of the game tree to subtrees, or, equivalently, a subtree of a game tree which is deterministic on the OR levels:

Definition 2.8 (Strategies in a game tree)
The set $\phi(t)$ *of strategies in a term* t *is the maximum set such that:*

- $t = \texttt{AndLeaf} \Rightarrow \phi(t) = \{\texttt{AndLeaf}\}$
- $t = \texttt{And}\,(t_1, \ldots, t_n), \varphi \in \phi(t) \Rightarrow \varphi = \texttt{And}\,(\varphi_1, \ldots, \varphi_n)$ *with* $\varphi_i \in \phi(t_i)$
- $t = \texttt{OrLeaf} \Rightarrow \phi(t) = \{\texttt{OrLeaf}\}$
- $t = \texttt{Or}\,(\frac{\theta_1}{t_1}, \ldots, \frac{\theta_n}{t_n}), \varphi \in \phi(t) \Rightarrow \exists j$ *such that* $\varphi = \texttt{Or}\,(\frac{\theta_j}{\varphi'})$ *with* $\varphi' \in \phi(t_j)$

Remark 2.9 (Value of a finite strategy) *Finite strategies are just finite game trees with a special form, so the definition of the value of a game tree extends naturally to strategies. Note, however, that it can easily be seen from the definitions that the value of a strategy is either a substitution or* **FALSE** *: it cannot be a disjunction of substitutions.*

Since we are mainly interested in the calculated answers of a logical programs, we introduce the notion of *winning* strategy for the first player:

Definition 2.10 (Winning Strategy) *A winning strategy in a (possibly infinite) game tree* Γ *is a finite strategy* φ *such that* $\mathbf{val}\,\varphi < \mathbf{FALSE}$

For the game on the left of the previous figure 1, there are at least two winning strategies, shown on the right of figure 1.

3 Games and traditional semantics

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 shall prove in this section that it is also possible to give another equivalent definition of these denotations, with the notion of winning strategies.
We've seen that the initial positions of the game are just goal, so we shall in the following definitions use freely the adjective "atomic, ground, Herbrand, conjunctive, etc." to characterize them. The least Herbrand model denotation (success set) of a program P is rewritten as the set of winning Herbrand positions for Player, that is the Herbrand positions in which he has a winning strategy:

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

For denotation of computed answers (s-semantics in [BM94]), we have the most general atomic positions (*i.e.* positions with the form $A(X_1, .., X_n)$), instantiated by the computed answers:

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

We shall 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, through the use of their well-known operational characterizations:
$O_P = \{A \mid A \text{ Herbrand atom and } \exists\ A \underset{P}{\Longrightarrow}^* \boxed{}\}$ and $O_P^S = \{A\theta \mid A \text{ most general atom and } \exists\ A \overset{\theta}{\underset{P}{\Longrightarrow}}^* \boxed{}\}$

3.1 Fusion and splitting of SLD derivations

In order to relate strategies to SLD-derivations, we need to prove a key result about SLD-derivations, namely that we can fusion 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. Technically, this is done by introducing an annotated version of SLD derivation, to keep track of the system of equations solved by an SLD derivation, and using the notion of switch-equivalent derivations (informally, that two SLD-derivation steps can be *switched* provided that in the second step an instance of an "old" atom is selected, and this give raise to a symmetric relation $\leftrightarrow$, whose reflexive and transitive closure is denoted $\approx$. This technical development has an intrinsic interest, so it is fully detailed in the appendix, but for our purposes we only need to state the following final result which is an immediate consequence of lemma A.6 and lemma A.10.

Proposition 3.1 (Goal fusion and goal split for successful SLD-derivations) *Suppose that* θ_1 *and* θ_2 *are compatible, that* $G_1 \stackrel{\theta_1}{\Longrightarrow}^* \Box$ *and that* $G_2 \stackrel{\theta_2}{\Longrightarrow}^* \Box$*, then* $(G_1, G_2) \stackrel{\theta_1 \wedge \theta_2}{\Longrightarrow}^* \Box$.

Conversely, if $(G_1, G_2) \underset{P}{\stackrel{\theta}{\Longrightarrow}}^* \Box$ *then there exists* θ_1,θ_2 *s.t.* $G_1 \underset{P}{\stackrel{\theta_1}{\Longrightarrow}}^* \Box$, $G_2 \underset{P}{\stackrel{\theta_2}{\Longrightarrow}}^* \Box$ *and* $\theta = \theta_1 \wedge \theta_2$.

This is the key technical tool to convert SLD-derivations back and forth into winning strategies.

3.2 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 exists, we can find a successful SLD-derivation. Then, we show that our game denotations are also complete, that is that whenever a successful SLD-derivation exists, we can find a winning strategy. What is more, the computed answer and the value of the strategy are the same.

Theorem 3.2 (From winning strategies to successful SLD-derivations (soundness)) *If there exists a winning strategy with value* θ *in* $\Gamma(P, G)$ *then there exists a successful SLD-derivation of* G *in* P *with computed answer* θ.

Proof. The proof is by induction on the structure of the strategy. A winning strategy φ is a finite AND-OR tree that have no OR-Leaf, so we can prove the thesis by induction on the number d of and-levels of the strategy.

- $\mathbf{d = 1}$. If $\varphi = \texttt{AndLeaf}$ then $\Gamma(P, G) = \texttt{AndLeaf}$ then G is the empty goal by construction and we have a successful SLD-derivation of G with computed answer ϵ, which is the value of the strategy φ, as claimed.
- **Induction step** Suppose $G = (A_1, \ldots, A_n)$, so $\varphi = \texttt{And(Or(}\frac{\theta_1}{\varphi_1}\texttt{),}\ldots\texttt{,Or(}\frac{\theta_n}{\varphi_n}\texttt{))}$ 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 $A_i \underset{H_i \leftarrow G_i}{\stackrel{\theta_i}{\Longrightarrow}} G_i\theta_i$ where φ is finite and its value is strictly lower then **FALSE**, the sub-strategies φ_i are finite and their values v_i are strictly lower then **FALSE**. 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 $G_i\theta_i \underset{P}{\stackrel{v_i}{\Longrightarrow}}^* \Box$ 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 $v = \theta_1 v_1 \wedge \cdots \wedge \theta_n v_n$ is the value of φ, a winning strategy, it is strictly lower than **FALSE**, so we can apply corollary 3.1 to construct a successful SLD-derivation of $(A_1, .., A_n)$ with computed answer v, and we are done.

□

Theorem 3.3 (From successful SLD-derivations to winning strategies (completeness)) *If* $G \underset{P}{\stackrel{\theta}{\Longrightarrow}}^* \Box$ *then there exists a winning strategy with value* θ *in the game* $\Gamma(P, G)$.

Proof. Let ξ be a SLD-derivation $G \underset{P}{\stackrel{\theta}{\Longrightarrow}}^* \Box$. 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 $A \stackrel{\theta'}{\Longrightarrow} G' \stackrel{\theta''}{\Longrightarrow}^* \Box$. If $G' = \Box$, 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, $\varphi = \texttt{And(Or(}\frac{\theta'}{\varphi'}\texttt{))}$ 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 corollary 3.1 to get n successful SLD-derivations $\xi_i = A_i \xrightarrow{\theta_i} \Box$, with the additional property $\theta = \theta_1 \wedge \theta_2 \wedge \cdots \wedge \theta_n$. 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_2, use the strategy φ_2 when opponent plays A_2, and so on... Formally, if $\varphi_i = \texttt{And(Or(}\frac{\theta'_i}{\varphi'_i}\texttt{))}$ with value θ_i , then we can form the strategy $\varphi = \texttt{And(Or(}\frac{\theta'_1}{\varphi'_1}\texttt{)}, \ldots, \texttt{Or(}\frac{\theta'_n}{\varphi'_n}\texttt{))}$ which has the value $\theta_1 \wedge \theta_2 \wedge \cdots \wedge \theta_n = \theta$

□

3.3 Expressive power of winning strategies

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

It is quite interesting to 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 derivation 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.

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

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

Example 3.4 Consider a strategy with the structure shown in the following picture (figure 2) we can

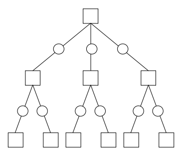

Figure 2: A strategy with three And levels

calculate 13440 different successful SLD-represented by this simple winning strategy. □

4 Cutting and evaluating game trees

Up to now, we have evaluated only strategies, for which the disjunction of substitutions was not necessary in the space of values. If we look at the value of a whole game tree, then we will find disjunctions, telling us how many ways Player has to win, allowing thus to finely analyze the multiplicity of computed answers. For this, we will need to evaluate potentially infinite trees, but here again traditional game theory comes to our rescue: for instance, the size of the chess game tree is so big that it can be considered infinite for all practical purposes, so real chess playing programs cut it at a certain level and provide an approximative evaluation of the tree using appropriate heuristics for the values of cut nodes. Here, we can do the same: cut the tree at the ith AND level, assigning a value to the cut nodes and obtaining an ith approximation.

Two main choices are available to estimate of the cut node: either we use **FALSE**, and obtain a pessimistic approximation $\mathbf{val}^i_{pess}$ of the real value, or use ϵ, and obtain an optimistic approximation $\mathbf{val}^i_{opt}$ of the real value (see appendix for details). The first choice gives us the necessary tool to establish the correspondence with SLD derivations, while the second choice allows to handle SLDNF derivations (which we will not consider here due to lack of space).

A fundamental property of these approximations is that they are, respectively, decreasing and increasing with respect to the cut level (see theorem B.3), so one can take the limit on an appropriate set of values and obtain for example the pessimistic value $\mathbf{val}\,\Gamma(P,G)$ of a game tree which has some very interesting properties:

- Its *cardinality* (the number of substitutions in the disjunction) is the number of equivalence classes of successful SLD derivations for G in P for the switch equivalence.
- Its *components* are the computed answers substitutions of the query G in G, counted with their order of multiplicity
- If we apply to it the simplification rule " $\theta \vee \theta = \theta$ ", then we come back to the *computed answers semantics*.
- If we apply to it the simplification rule " $\theta \vee \sigma = \sigma$ if σ is more general than θ ", then we get only the *most general computed answers*.

The soundness and completeness theorems established in the previous sections allows to relate this approximation to SLD derivations.

Corollary 4.1 (Pessimistic semantics and SLD derivations: soundness and completeness)
If $\exists i\,.\mathbf{val}^i_{pess}\Gamma(P,G) = v$ *such that* $v \sqsubset$ **FALSE** *then* $\exists\ G \overset{\theta}{\underset{P}{\Longrightarrow}}{}^{*} \Box$ *with* $v \sqsubseteq \theta$.
Conversely, if $G \overset{\theta}{\underset{P}{\Longrightarrow}}{}^{*} \Box$ *then* $\exists i.\mathbf{val}^i_{pess}\Gamma(P,G) = v \sqsubseteq \theta \sqsubset$ **FALSE**.

Proof. If there exists a pessimistic evaluation of the game with value v strictly lower than **FALSE**, then there exists a winning strategy in Cut(i,$\Gamma(P,G)$) with value θ such that $v \sqsubseteq \theta$. So we can use theorem 3.2 to obtain a successful SLD-derivation with computed answer θ.

On the other side, given a successful SLD-derivation $\xi = G \overset{\theta}{\underset{P}{\Longrightarrow}}{}^{*} \Box$, we can construct the winning strategy $\varphi = \mathbf{SLD2Strat}(\xi)$ in $\Gamma(P,G)$. Then take i as the depth of φ. By theorem B.3 have that $\mathbf{val}_{pess}\Gamma(P,G) \sqsubseteq \mathbf{val}^i_{pess}\Gamma(P,G) \sqsubseteq \mathbf{val}\,\varphi = \theta$. □

5 Conclusions and Perspectives

We have presented in this paper a game theoretic approach to the semantics of Logic Programming, and proved that it is flexible enough to capture various forms of semantics proposed in the literature, from the minimum Herbrand model to the computed answers, to the most general computed answers. We showed that a winning strategy concisely represents a very large class of equivalent SLD derivations, thus providing a powerful tool to investigate properties of derivations. Nevertheless, all these results that established the first bridge between Game Theory and Logic Programming seem to have tapped only a little of the potential intrinsic in this interconnection. Let us point here at some further very promising developments.

It would be interesting to link the literature in the field of parallelization of logic languages with the field of parallelization of game-search algorithms (good comprehensive bibliographies can respectively be found in [LP93] and [Bro96]). Since the works on game-algorithms began decades before the first logic programming implementations, we expect that the latter would greatly benefit from the connection.

It is clear, though we hadn't enough place to develop the idea here, that the game values of the programs are AND-compositional (namely $\mathbf{val}\,\Gamma(P,(G_1,G_2)) = \mathbf{val}\,\Gamma(P,G_1) \wedge \mathbf{val}\,\Gamma(P,G_2)$), but not OR-compositional ($\mathbf{val}\,\Gamma((P \cup Q),G) \neq \mathbf{val}\,\Gamma(P,G) \vee \mathbf{val}\,\Gamma(P,G)$). This suggests to look for an OR-compositional game-based denotation, which hopefully would be more abstract and intuitive than the known ones.

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. Since the negation as failure (SLDNF) denotation of a program P uses that maximum fix-point of the immediate consequences operator T_P, it seems that the negative goals must be treated with the optimistic approximations: if we can show that $\mathbf{val}^{i}_{opt}\Gamma(P,G)$ is **FALSE** for some i, then G has been shown to be unsatisfiable despite being very optimistic for the cut parts of the game, and G is a contradiction of the logic program P. This gives a very intuitive explanation of SLDNF.

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.

[Aum81] R.J. Aumann. *Survey of Repeated Games. In : Essays in Game Theory and Mathematical Economics in Honour of Oskar Morgenstern*. Bibliographisches Institut, Zurich, 1981.

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

[BM94] Levi Bossi, Gabbrielli and Meo. The s-semantics approach : theory and applications. *Journal of Logic Programming*, 19(20), 1994.

[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.

[Doe94] Kees Doets. *From Logic to Logic Programming*. The MIT Press, 1994.

[Ebb] Thomas Ebbinghaus, Flum. *Mathematical Logic*. Springer Verlag.

[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.

[Llo87] John Lloyd. *Foundations of Logic Programming*. Springer Verlag, 2nd edition edition, 1987.

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

[Loo46] L.H. Loomis. On a theorem of von Neumann. *Proceedings of the National Academy of Sciences of The United States of America*, (32):213–215, 1946.

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

[MFP93] M. Martelli M. Falaschi, G. Levi and C. Palamidessi. A model-theoretic reconstruction of the operational semantics of logic programs. *Information and Computation*, 102(1):86–113, 1993.

[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.

[Pea84] J. Pearl. *Heuristics. Intelligent Search Strategies for Computer Problem Solving*. Addison Wesley, 1984.

[SAM94] Rhada Jagadeesan Samson Abramsky and Pasquale Malacaria. Full abstraction for PCF, in. *Theorical Aspects of Computer Software, International Symposium TACS'94*, 1994.

[Sha53] L.S. Shappley. Stochastic games. *Proceedings of the National Academy of Sciences of The United States of America*, (39):1095–1100, 1953.

[Sti97] C. Stirling. Bisimulation, model checking and other games. Technical report, Notes for Mathfit Instructional Meeting on Games and Computation, Edinburgh, 1997.

[Sto79] G.C. Stockman. A minimax algorithm better than alpha-beta ? *Artificial Intelligence*, 12(2):179–196, 1979.

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

[vNM44] J. von Neumann and Morgenstern. *Theory of Games and Economics Behavior*. Princeton University Press. Princeton, 1944.

[Zer13] E. Zermolo. Uber eine Anwendung der Mengellehre auf die Theorie des Schachspils. *Proceedings of the Fifth International Congress of Mathematicians. Cambridge University Press. Cambridge*, 2(100):501–504, 1913.

A Technical definitions and results about SLD derivations

A.1 Goal Fusion

We need to extend the definition of SLD-derivation to include the notion of system associated to (and solved by) a traditional SLD-derivation.

Definition A.1 (System associated to an SLD-derivation) *If $G = A_1, .., A_n$ is the current gaol and $B \leftarrow H$ is a variant of a program clause, variable disjoint with G and τ, such that B and $A_i\tau$ unify with mgu θ, then we can infer the transition*

$$\langle G, \tau, E\rangle \underset{B\leftarrow H}{\overset{\theta}{\Longrightarrow}} \langle G', \tau\theta, E \cdot \{A_i = B\}\rangle$$

where $G' = A_1, .., A_{i-1}, H, A_{i+1}, .., A_n$.

The two notations are clearly equivalent: if we note λ is the empty sequence of equations, then

$$G \overset{\theta}{\Longrightarrow}{}^{*} G' \quad \textit{if and only if} \quad \langle G, \epsilon, \lambda\rangle \overset{\theta}{\Longrightarrow}{}^{*} \langle G'', \theta, E\rangle \ \textit{and} \ G' = G''\theta$$

but with the new notation, we keep track of the sequence E of formal equations solved along the steps of the SLD-derivation. In the rest of the paper we will say that E is the *system associated to an SLD-derivation.*

Remark A.2 *We shall frequently consider a sequence of equations as a system, so we will, by abuse of notation, say that a sequence is solvable or that two sequences are equivalent (in which case we will write $E_1 \sim E_2$).*

Example A.3 We illustrate the construction process of an SLD-derivation system with a simple example. Suppose we had the following successful SLD-derivation ξ :

$$A_1 \underset{B_1\leftarrow A_2,A_3}{\overset{\theta_1}{\Longrightarrow}} A_2\theta_1, A_3\theta_1 \underset{B_2\leftarrow}{\overset{\theta_2}{\Longrightarrow}} A_3\theta_1\theta_2 \underset{B_3\leftarrow}{\overset{\theta_3}{\Longrightarrow}} \Box$$

then in the new notation we have:

$$\begin{array}{lcl} \langle A_1, \epsilon, \lambda\rangle & \underset{B_1\leftarrow A_2,A_3}{\overset{\theta_1}{\Longrightarrow}} & \langle (A_2, A_3), \theta_1, \{A_1 = B_1\}\rangle \\ & \underset{B_2\leftarrow}{\overset{\theta_2}{\Longrightarrow}} & \langle A_3, \theta_1\theta_2, \{A_1 = B_1\} \cdot \{A_2 = B_2\}\rangle \\ & \underset{B_3\leftarrow}{\overset{\theta_3}{\Longrightarrow}} & \langle \Box, \theta_1\theta_2\theta_3, \{A_1 = B_1\} \cdot \{A_2 = B_2\} \cdot \{A_3 = B_3\}\rangle \end{array}$$

so the system to ξ is $\{A_1 = B_1, A_2 = B_2, A_3 = B_3\}$. $\Box$

The nice point of the notion of systems associated to SLD-derivations is that they are very convenient for complicate proofs on SLD-derivations, as there are lots of well-known and strong results on unification of systems. For instance, we will use the following lemma, which allows us to search for mgus in an iterative fashion. A proof can be found in [Apt97].

Lemma A.4 (Iteration) *Let E_1,E_2 be two sets of equations. Suppose that θ_1 is a mgu of E_1 and η_1 an mgu of $E_2\theta_1$. Then $\theta_1\eta_1$ is a mgu of $E_1 \cup E_2$. Moreover, if E_1 cupE_2 is unifiable then an mgu θ_1 of E_1 exists and for any mgu θ_1 of E_1 an mgu η_1 of $E_2\theta_1$ exists, as well.*

The system associated to any SLD-derivation has a solution, which is exactly its computed answer substitution. In fact we even show something stronger, from which we get the result by taking $\tau = \epsilon$ and $E = \lambda$.

Lemma A.5 *If $\langle G, \tau, E\rangle \stackrel{\theta}{\Longrightarrow} \langle G', \tau\theta, E \cdot E'\rangle$ then θ is the mgu of the system $E'\tau$.*

Proof. The proof is a simple induction on the length $\mathbf{l}$ of the SLD-derivation

- $\mathbf{l = 1}$. If $\langle G, \tau, E\rangle \stackrel{\theta}{\Longrightarrow} \langle G', \tau\theta, E \cdot \{A = B\}\rangle$ by definition θ is the mgu of $A\tau$ and B where A is the selected atom in G, in other words θ is the mgu of $\{A\tau = B\} = \{A\tau = B\tau\} = \{A = B\}\tau$ (notice that $B\tau = B$ because B is an arbitrary variant of a program clause disjoint to τ).
- $\mathbf{l > 1}$. We decompose the derivation in two parts strictly shorter than $\mathbf{l}$ (but longer than 1) :

$$\langle G, \tau, E\rangle \stackrel{\theta_1}{\Longrightarrow}^* \langle G_1, \tau\theta_1, E \cdot E_1\rangle \stackrel{\theta_2}{\Longrightarrow}^* \langle G_2, \tau\theta_1\theta_2, E \cdot E_1 \cdot E_2\rangle$$

  and we want to show that $(E_1 \cdot E_2)\tau$ is solved by $\theta_1\theta_2$. By induction hypothesis θ_1 is the mgu of $E_1\tau$ and θ_2 is the mgu of $E_2\tau\theta_1$. By applying the iteration lemma A.4 we have that $\theta = \theta_1\theta_2$ is the mgu of $E_1\tau \cup E_2\tau = (E_1 \cup E_2)\tau$.

□

We can now prove the following key lemma

Lemma A.6 (Goal fusion) *If $G_1 \stackrel{\theta_1}{\Longrightarrow}^* G_1'$ and $G_2 \stackrel{\theta_2}{\Longrightarrow}^* G_2'$ are two SLD-derivations such that θ_1 and θ_2 are compatible ($\theta_1 \wedge \theta_2 \leq$ **FALSE**), then starting from the fusion of the two goals we can obtain an SLD-derivation $(G_1, G_2) \stackrel{\theta_1\wedge\theta_2}{\Longrightarrow}^* (G_1'\eta_1, G_2'\eta_2)$ where η_1 and η_2 are substitutions such that $\theta_1\eta_1 = \theta_2\eta_2 = \theta_1 \wedge \theta_2$.*

Proof. The idea of the proof is to consider the extended form of SLD-derivations:

$$\langle G_1, \epsilon, \lambda\rangle \stackrel{\theta_1}{\Longrightarrow}^* \langle G_1'', \theta_1, E_1\rangle \quad \text{and} \quad \langle G_2, \epsilon, \lambda\rangle \stackrel{\theta_2}{\Longrightarrow}^* \langle G_2'', \theta_2, E_2\rangle$$

and to prove that we can concatenate them, by doing the latter after the former, thus forming the following SLD-derivation:

$$\langle (G_1, G_2), \epsilon, \lambda\rangle \stackrel{\theta_1}{\Longrightarrow}^* \langle (G_1'', G_2), \theta_1, E_1\rangle \stackrel{\eta_1}{\Longrightarrow}^* \langle (G_1'', G_2''), \theta_1 \wedge \theta_2, E_1 \cdot E_2\rangle$$

First, we prove that $E_1 \cup E_2$ is solvable. By hypothesis $\theta_1 \wedge \theta_2 <$ **FALSE**, so by definition of the intersection of substitutions $mgu(\theta_1, \theta_2)$ exists. But by lemma A.5, θ_i is the mgu of E_i, so θ_i is just the solved form of E_i and we obtain that $E_1 \cup E_2$ is solved by $\theta_1 \wedge \theta_2$.
Suppose the second SLD-derivation had n steps and E_2 has the form $E_2 = \{L_1 = R_1\} \cdots \{L_n = R_n\}$, then $E_2\theta_1 = \{L_1\theta_1 = R_1\theta_1\} \cdots \{L_n\theta_1 = R_n\theta_1\}$. Since we can choose the program clauses $R_i \leftarrow H_i$ variable disjoint with θ_1, this is equivalent to the system $\{L_1\theta_1 = R_1\} \cdots \{L_n\theta_1 = R_n\}$. The iteration lemma for solution of systems (lemma A.4) assure us that $E_2\theta_1$ can in fact be solved step by step, that is in the order

$$\begin{aligned}
\tau_1 &= mgu(L_1\theta_1, R_1) \\
\tau_2 &= mgu(L_2\theta_1\tau_1, R_2) \\
&\cdots \\
\tau_n &= mgu(L_n\theta_1\tau_1 \cdots \tau_{n-1}, R_n)
\end{aligned}$$

Having now $\langle (G_1, G_2), \epsilon, \lambda \rangle \stackrel{\theta_1}{\Longrightarrow}^* \langle (G_1'', G_2), \theta_1, E_1 \rangle$, we can complete the missing steps to construct $\langle (G_1'', G_2), \theta_1, E_1 \rangle \stackrel{\eta_1}{\Longrightarrow}^* \langle (G_1'', G_2''), \theta_1\eta_1, E_1 \cdot E_2 \rangle$, where $\eta_1 = \tau_1 \cdots \tau_n$, taking the previous order to solve the equations.

Now, by applying once more the lemma A.5, we get that $\theta_1\eta_1 = mgu(E_1 \cdot E_2) = mgu(E_1 \cup E_2)$. By doing the work in the other way round (that is, reducing G_1 after G_2), we could construct eta_2 such that $\theta_2\eta_2 = \theta_1 \wedge \theta_2 = \theta_1\eta_1$. Finally we have that $(G_1, G_2) \stackrel{\theta_1 \wedge \theta_2}{\Longrightarrow}^* (G_1''\theta_1\eta_1, G_2''\theta_1\eta_1)$ with $G_1''\theta_1\eta_1 = G_1'\eta_1$ and $G_2''\theta_1\eta_1 = G_2''\theta_2\eta_2 = G_2'\eta_2$, as claimed. $\square$

A.2 Goal Splitting

Let us recall a classical technical lemma on SLD-derivations, the *switching lemma*, which says, informally, that two SLD-derivation steps can be switched provided that in the second step an instance of an "old" atom is selected. The following formulation is taken from [Apt97].

Lemma A.7 (Switching Lemma) *Consider a query Q_n with two different atoms A_1 and A_2. Suppose that*

$$\xi = Q_0 \underset{c}{\stackrel{\theta_1}{\Longrightarrow}} Q_1 \cdots Q_n \underset{c_{n+1}}{\stackrel{\theta_{n+1}}{\Longrightarrow}} Q_{n+1} \underset{c_{n+2}}{\stackrel{\theta_{n+2}}{\Longrightarrow}} Q_{n+2} \cdots$$

is an SLD-derivation where

- *A_1 is the selected atom of Q_n,*
- *$A_2\theta_{n+1}$ is the selected atom of Q_{n+1}.*

Then for some Q'_{n+1}, θ'_{n+1} and θ'_{n+2}

- $\theta'_{n+1}\theta'_{n+2} = \theta_{n+1}\theta_{n+2}$,
- *there exists an SLD-derivation*

$$\xi' = Q_0 \underset{c}{\stackrel{\theta_1}{\Longrightarrow}} Q_1 \cdots Q_n \underset{c_{n+2}}{\stackrel{\theta'_{n+1}}{\Longrightarrow}} Q_{n+1} \underset{c_{n+1}}{\stackrel{\theta'_{n+2}}{\Longrightarrow}} Q_{n+2} \cdots$$

  where

  - *ξ and ξ' coincide up to the resolvent Q_n,*
  - *A_2 is the selected atom of Q_n,*
  - *$A_1\theta'_{n+1}$ is the selected atom of Q'_{n+1},*
  - *ξ and ξ' coincid after the resolvent Q_{n+2}.*

Definition A.8 (Equivalence of SLD-derivation) *If ξ and ξ' are two SLD-derivations which satisfy the conditions of the switching lemma A.7 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 $\leftrightarrow$.*

Lemma A.9 (Systems of equivalent SLD-derivations) *Equivalent SLD-derivations have the same associated system of equations and the same computed answers.*

Proof. We first observe that

$$\xi = \langle G, \epsilon, \lambda\rangle \overset{\theta}{\Longrightarrow}^{*} \langle G', \theta, E_1\rangle, \qquad \xi' = \langle G, \epsilon, \lambda\rangle \overset{\theta}{\Longrightarrow}^{*} \langle G', \theta, E_2\rangle, \qquad \xi \leftrightarrow \xi' \Rightarrow E \approx E'$$

since if A_1 and A_2 are the switched atoms and $c_1 = (B_1 \leftarrow H_1)$ and $c_2 = (B_2 \leftarrow H_2)$ are the switched clauses used to transform ξ into ξ', then we have $E_1 = E_1' \cdot \{A_1 = B_1\} \cdot \{A_2 = B_2\} \cdot E_1''$ and $E_2 = E_2' \cdot \{A_2 = B_2\} \cdot \{A_1 = B_1\} \cdot E_1''$. So E_1 and E_2 represent the same set of equation (but in a different order). By reflexivity and transitivity of equality of systems, $\approx$ has this property too. Having the same associated system, they have the same computed answer. □

Lemma A.10 (Goal split) *If* $(G_1, G_2) \overset{\theta}{\underset{P}{\Longrightarrow}}^{*} G'$ *then there exists* G_1',G_2',θ_1,θ_2,η_1 *and* η_2 *such that*

- $G_1 \overset{\theta_1}{\underset{P}{\Longrightarrow}}^{*} G_1'$, $G_2 \overset{\theta_2}{\underset{P}{\Longrightarrow}}^{*} G_2'$ *with* $G' = (G_1'\eta_1, G_2'\eta_2)$
- $\theta = \theta_1\eta_1 = \theta_2\eta_2 = \theta_1 \wedge \theta_2$

Proof. We consider the extended form of the SLD-derivation:

$$\langle (G_1, G_2), \epsilon, \emptyset\rangle \overset{\theta}{\underset{P}{\Longrightarrow}}^{*} \langle (G_1'', G_2''), \theta, E\rangle$$

The idea, then, is that we can isolate in ξ the steps which reduce G_1 and all its residus. A method to achieve this, for instance, is to "mark" G_1 in the SLD-derivation and all the descendants of G_i. By using the switching lemma repeatedly and sorting, we can move G_1 and all of its marked residues to the head of ξ, and get an SLD-derivation ξ' equivalent to ξ, but in which we reduce G_1 first:

$$\xi_1 \;:=\; \langle (G_1, G_2), \epsilon, \emptyset\rangle \overset{\theta_1}{\underset{P}{\Longrightarrow}}^{*} \langle (G_1'', G_2), \theta_1, E_1\rangle \overset{\eta_1}{\underset{P}{\Longrightarrow}}^{*} \langle (G_1'', G_2''), \theta_1\eta_1, E_1 \cdot E_2\rangle$$

With the same argument we can reduce first G_2 and its residues to obtain

$$\xi_2 \;:=\; \langle (G_1, G_2), \epsilon, \emptyset\rangle \overset{\theta_2}{\underset{P}{\Longrightarrow}}^{*} \langle (G_1, G_2''), \theta_2, E_2\rangle \overset{\eta_2}{\underset{P}{\Longrightarrow}}^{*} \langle (G_1'', G_2''), \theta_2\eta_2, E_2 \cdot E_1\rangle$$

Since ξ_1 and ξ_2 are both equivalent to ξ, they have the same associated system and the same calculated answer by lemma A.9, so $E = E_1 \cup E_2$ is solvable and $\theta = \theta_1\eta_1 = \theta_2\eta_2$. But notice that now, calling $G_1' = G_1''\theta 1$ and $G_2' = G_2''\theta 2$, we can extract the derivations for G_1 and G_2 separately by keeping only the first steps in ξ_1 and ξ_2, respectively :

$$\xi_1' := G_1 \overset{\theta_1}{\underset{P}{\Longrightarrow}}^{*} G_1' \quad \text{and} \quad \xi_2' := G_2 \overset{\theta_2}{\underset{P}{\Longrightarrow}}^{*} G_2'$$

E_1 is the system associated to ξ_1', E_2 the system associated to ξ_2', they are solved by θ_1 and θ_2 respectively, and $E_1 \cup E_2$ is solvable, so we can apply the fusion lemma A.6 to get that $\theta = \theta_1 \wedge \theta_2$. □

B Cutting and evaluating infinite trees in detail

Here is the formal definition of the function for cutting infinite trees, followed by the definition of the pessimistic and optimistic value of a cut game tree. Applied to a game tree, the function Cut(i, _) cuts it at the i^{th} AND level, replacing all non-leaf subtrees with the constant Ω, which represent the interruption of the tree development process.

Definition B.1 (Cut of a tree at level i)
The function Cut(o,f) a game tree is defined inductively as follow :

- *at the AND levels :*
  $Cut(0,\texttt{AndLeaf}) = \Omega$
  $Cut(k,\texttt{AndLeaf}) = \texttt{AndLeaf} \qquad \forall k \geq 1$
  $Cut(0,\texttt{And}\,(t_1, ..., t_n)\,) = \Omega$
  $Cut(i,\texttt{And}\,(t_1, ..., t_n)\,) = \texttt{And}\,(Cut(i-1, t_1), ..., Cut(i-1, t_n)\,) \qquad \forall i \geq 1$
- *at the OR levels :*
  $Cut(i,\texttt{OrLeaf}) = \texttt{OrLeaf}$
  $Cut(i,\texttt{Or}\,(\frac{\theta_1}{t_1}, ..., \frac{\theta_n}{t_n})\,) = \texttt{Or}\,(\frac{\theta_1}{Cut(i,t_1)}, ..., \frac{\theta_n}{Cut(i,t_n)}\,)$

Definition B.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 using the equation $\mathbf{val}_{pess}\Omega$ = **FALSE***. The (dual) optimistic version $\mathbf{val}_{opt}$ evaluates Ω with the best possible value, that is using the equation $\mathbf{val}_{opt}\Omega = \epsilon$ We will note $\mathbf{val}^i_{pess}t$ the approximation $\mathbf{val}_{pess}Cut(i,t)$ and $\mathbf{val}^i_{opt}t$ the dual approximation $\mathbf{val}_{opt}Cut(i,t)$.*

Theorem B.3 (Monotonicity of progressive approximations)
The function $\mathbf{val}^i_{pess}$ of a game tree is decreasing with respect to the cut level i, whereas the function $\mathbf{val}^i_{opt}$ is monotone (increasing) with respect to i.

Proof. To prove the monotonicity for the function $\mathbf{val}_{pess}$, we compare the two values $\mathbf{val}^{i+1}_{pess}t$ and $\mathbf{val}^i_{pess}t$ to establish that the latter is greater than (or equal to) the former. If t is a leaf ($t = $`AndLeaf`) then both terms are equals to ϵ and the claim is trivially proved. So we will suppose that t has the general form $\texttt{And}\,(t_1, ..., t_n)$ with $n \geq 1$ where $t_j = $`OrLeaf` or $t_j = \texttt{Or}\,(\frac{\theta^{(j)}_1}{t^{(j)}_1}, .., \frac{\theta^{(j)}_n}{t^{(j)}_n})$
The proof is by induction on the cut level i :

- when $\mathbf{i = 0}$, we have $Cut(0,t) = \Omega$ and its pessimistic evaluation is **FALSE** which is greater than to anything else.
- for $\mathbf{i > 0}$, we unfold the two values to compare them :

$$\begin{aligned}
\mathbf{val}^{i+1}_{pess}t &= \mathbf{val}_{pess}Cut(i+1, \texttt{And}\,(t_1, .., t_n)\,) \\
&= \mathbf{val}_{pess}\texttt{And}\,(Cut(i,t_1), .., Cut(i,t_n)) \\
&= \bigwedge_{j=1}^{n} \mathbf{val}_{pess}Cut(i,t_j)
\end{aligned}$$

$$\begin{aligned}
\mathbf{val}^{i}_{pess}t &= \mathbf{val}_{pess}Cut(i, \texttt{And}\,(t_1, .., t_n)\,) \\
&= \mathbf{val}_{pess}\texttt{And}\,(Cut(i-1,t_1), .., Cut(i-1,t_n)) \\
&= \bigwedge_{j=1}^{n} \mathbf{val}_{pess}Cut(i-1,t_j)
\end{aligned}$$

If one of the subtrees (say t_j) is a leaf then $Cut(i,t_j) = Cut(i-1,t_j) = $`OrLeaf` and so $\mathbf{val}^{i+1}_{pess}t = \mathbf{val}^{i+1}_{pess}t =$ **FALSE**, since $\mathbf{val}_{pess}$`OrLeaf` $=$ **FALSE** and by the definition of intersection. On the other hand, if all the t_j are of the form $\texttt{Or}\,(\frac{\theta_1^{(j)}}{t_1^{(j)}}, .., \frac{\theta_{n_j}^{(j)}}{t_{n_j}^{(j)}})$, then we further unfold the values $\mathbf{val}_{pess}^{i+1} t$ and $\mathbf{val}_{pess}^{i} t$ and get

$$\begin{aligned}
\mathbf{val}_{pess}^{i+1} t &= \bigwedge_{j=1}^{n} \mathbf{val}_{pess} \texttt{Or}\,(\frac{\theta_1^{(j)}}{Cut(i, t_1^{(j)})}, .., \frac{\theta_{n_j}^{(j)}}{Cut(i, t_{n_j}^{(j)})}) \\
&= \bigwedge_{j=1}^{n} \bigvee_{h=1}^{n_j} \theta_h^{(j)} \mathbf{val}_{pess}^{i} t_h^{(j)} \\
\mathbf{val}_{pess}^{i} t &= \bigwedge_{j=1}^{n} \mathbf{val}_{pess} \texttt{Or}\,(\frac{\theta_1^{(j)}}{Cut(i-1, t_1^{(j)})}, .., \frac{\theta_{n_j}^{(j)}}{Cut(i-1, t_{n_j}^{(j)})}) \\
&= \bigwedge_{j=1}^{n} \bigvee_{h=1}^{n_j} \theta_h^{(j)} \mathbf{val}_{pess}^{i-1} t_h^{(j)}
\end{aligned}$$

Now, by the induction hypothesis, we know that $\mathbf{val}_{pess}^{i} t_h^{(j)} \sqsubseteq \mathbf{val}_{pess}^{i-1} t_h^{(j)}$, so using monotonicity of composition of substitutions (that is, $\theta_1 \leq \theta_2$ implies $\theta\theta_1 \leq \theta\theta_2$) and of $\vee$ and $\wedge$ operators, we get the result:

$$\mathbf{val}_{pess}^{i+1} t \sqsubseteq \mathbf{val}_{pess}^{i} t$$

The proof of monotonicy for $\mathbf{val}_{opt}^{i}$ is similar (replacing $\sqsupseteq$ for $\sqsubseteq$ and ϵ for **FALSE**).

□

We now turn to the definition of the value of an infinite game-tree. Do do that, we introduce the set of winning strategies of a given height in the game tree,

$$\phi_n^{P,G} = \{\varphi \mid \varphi \text{ is a winning strategy of height } n \text{ in } \Gamma(P,G)\}$$

and the value associated to this set :

$$\omega_n = \bigvee_{\varphi \in \phi_n^{P,G}} \mathbf{val}\,\varphi \quad \text{if } \phi_n^{P,G} \text{ in not empty, and } \textbf{FALSE} \text{ otherwise}$$

The pessimistic value of a general game-tree is then the (possibly infinite) following disjunction :

$$\mathbf{val}\,\Gamma(P,G) = \lim_{i\to\infty} \mathbf{val}_{pess}^{i} \Gamma(P,G) = \bigvee_{k=1}^{\infty} \omega_k \text{ in which we quantify only on the } \omega_k \neq \textbf{FALSE}.$$