Well-Founded Semantics for Boolean Grammars*

Vassilis Kountouriotis[1], Christos Nomikos[2], and Panos Rondogiannis[1]

[1] Department of Informatics & Telecommunications
University of Athens, Athens, Greece
{grad0771, prondo}@di.uoa.gr
[2] Department of Computer Science, University of Ioannina,
P.O. Box 1186, 45 110 Ioannina, Greece
cnomikos@cs.uoi.gr

Abstract. Boolean grammars [*A. Okhotin, Information and Computation 194 (2004) 19-48*] are a promising extension of context-free grammars that supports conjunction and negation. In this paper we give a novel semantics for boolean grammars which applies to *all* such grammars, independently of their syntax. The key idea of our proposal comes from the area of *negation in logic programming*, and in particular from the so-called *well-founded semantics* which is widely accepted in this area to be the "correct" approach to negation. We show that for every boolean grammar there exists a distinguished (three-valued) language which is a *model* of the grammar and at the same time the least fixed point of an operator associated with the grammar. Every boolean grammar can be transformed into an equivalent (under the new semantics) grammar in normal form. Based on this normal form, we propose an $\mathcal{O}(n^3)$ algorithm for parsing that applies to any such normalized boolean grammar. In summary, the main contribution of this paper is to provide a semantics which applies to *all* boolean grammars while at the same time retaining the complexity of parsing associated with this type of grammars.

1 Introduction

Boolean grammars constitute a new and promising formalism, proposed by A. Okhotin in [Okh04], which extends the class of conjunctive grammars introduced by the same author in [Okh01]. The basic idea behind this new formalism is to allow intersection and negation in the right-hand side of (context-free) rules. It is immediately obvious that the class of languages that can be produced by boolean grammars is a proper superset of the class of context-free languages.

Despite their syntactical simplicity, boolean grammars appear to be non-trivial from a semantic point of view. As we are going to see in the next section, the existing approaches for assigning meaning to boolean grammars suffer from certain shortcomings (one of which is that they do not give a meaning to all such grammars).

* The first author's work is being supported by a doctoral research grant by the General Secretariat for Research and Technology under the program $\Pi ENE\Delta$ (grant number $03E\Delta$ 330).

O.H. Ibarra and Z. Dang (Eds.): DLT 2006, LNCS 4036, pp. 203–214, 2006.

In this paper we propose a new semantics (the *well-founded* semantics) which applies to all boolean grammars. More specifically, we demonstrate that for every boolean grammar there exists a distinguished (*three-valued*, see below) language that can be taken as the meaning of this grammar; this language is the unique least fixed point of an appropriate operator associated with the grammar (and therefore it is easy to see that it satisfies all the rules of the grammar).

Our ideas originate from an important area of research in the theory of logic programming, that has been very active for more than two decades (references such as [AB94, PP90] provide nice surveys). In this area, there is nowadays an almost unanimous agreement that if one seeks a unique model of a logic program with negation, then one has to search for a three-valued one. In other words, classical two-valued logic is not sufficient in order to assign a proper meaning to logic programs with negation. Actually, it can be demonstrated that every logic program with negation has a distinguished three-valued model, which is usually termed the *well-founded model* [vGRS91].

We follow the same ideas here: we consider three-valued languages, namely languages in which the membership of strings may be characterized as true, false, or *unknown*. As we will see, this simple extension solves the semantic problems associated with negation in boolean grammars. Moreover, we demonstrate that under this new semantics, every boolean grammar has an equivalent grammar in normal form (similar to that of [Okh04]). Finally, we show that for every such normalized grammar, there is an $\mathcal{O}(n^3)$ parsing algorithm under our new semantics. Our results indicate that there may be other fruitful connections between formal language theory and the theory of logic programming.

2 Why an Alternative Semantics for Boolean grammars?

In [Okh04] A. Okhotin proposed the class of boolean grammars. Formally:

Definition 1 ([Okh04]). *A Boolean grammar is a quadruple $G = (\Sigma, N, P, S)$, where Σ and N are disjoint finite nonempty sets of terminal and nonterminal symbols respectively, P is a finite set of rules, each of the form*

$$A \rightarrow \alpha_1 \& \cdots \& \alpha_m \& \neg\beta_1 \& \cdots \& \neg\beta_n \qquad (m+n \geq 1, \alpha_i, \beta_i \in (\Sigma \cup N)^*),$$

and $S \in N$ is the start symbol of the grammar. We will call the α_i's positive literals *and the $\neg\beta_i$'s* negative.

To illustrate the use of Boolean grammars, consider a slightly modified example from [Okh04]:

Example 1. Let $\Sigma = \{a, b\}$. We define:

$$\begin{aligned}
&S \rightarrow \neg(AB)\ \&\ \neg(BA)\ \&\ \neg A\ \&\ \neg B\\
&A \rightarrow a\\
&A \rightarrow CAC\\
&B \rightarrow b\\
&B \rightarrow CBC\\
&C \rightarrow a\\
&C \rightarrow b
\end{aligned}$$

It can be shown that the above grammar defines the language $\{ww \mid w \in \{a,b\}^*\}$ (see [Okh04] for details). It is well-known that this language is not context-free.

Okhotin proposed two semantics intended to capture the meaning of boolean grammars. In this section we demonstrate some deficiencies of these two approaches, which led us to the definition of the well-founded semantics. Both semantics proposed in [Okh04] are defined using a system of equations, which is obtained from the given grammar.

In the first approach, the semantics is defined only in the case that the system of equations has a *unique solution*. This is a restrictive choice: actually most interesting grammars do not correspond to systems of equations having a unique solution. For example, even the simplest context-free grammars generating infinite languages, give systems of equations which have infinitely many solutions. For such grammars, it seems that the desired property is a form of *minimality* rather than uniqueness of the solution.

Apart from its limited applicability, the unique solution semantics also demonstrates a kind of instability. For example, let $\Sigma = \{0,1\}$ and consider the boolean grammar consisting of the two rules $A \to \neg A \& \neg B$ and $B \to 0\&1$. The corresponding system of equations has no solution and therefore the unique solution semantics for this grammar is not defined. Suppose that we augment the above grammar with the rule $B \to B$. Seen from a constructive point of view, the new rule does not offer to the grammar any additional information. It is reasonable to expect that such a rule would not change the semantics of the grammar. However, the augmented grammar has unique solution semantics, namely $(A,B) = (\emptyset, \Sigma^*)$. On the other hand, suppose that we augment the initial grammar with the rule $A \to A$. Then, the unique solution semantics is also defined, but now the solution is $(A,B) = (\Sigma^*, \emptyset)$. Consequently by adding to an initially meaningless grammar two different information-free rules, we obtained two grammars defining complementary languages. To put it another way, three grammars that look equivalent, have completely different semantics.

Let's now turn to the second approach proposed in [Okh04], namely the naturally feasible solution semantics. Contrary to the unique solution semantics, the feasible solution semantics generalizes the semantics of context-free and conjunctive languages (see [Okh04][Theorem 3]). However, when negation appears, there are cases that this approach does not behave in an expected manner. Consider for example the boolean grammar with rules:

$$A \to \neg B, \quad B \to C \& \neg D, \quad C \to D, \quad D \to A$$

This grammar has the naturally feasible solution $(A,B,C,D) = (\Sigma^*, \emptyset, \Sigma^*, \Sigma^*)$. It is reasonable to expect that composing two rules would not affect the semantics of the grammar. For example in context-free grammars such a composition is a natural transformation rule that simply allows to perform two steps of the production in a single step. However, if we add $C \to A$ to the above set of rules, then the naturally feasible solution semantics of the resulting grammar is not defined. On the other hand, the technique we will define shortly, does not suffer from this shortcoming.

Furthermore, there exist grammars for which the naturally feasible solution semantics is undefined, although they may have a clear intuitive meaning. For example, let $\Sigma = \{a\}$ and consider the following set of eight rules:

$$A \to \neg B, \quad A \to D, \quad B \to \neg C, \quad B \to D,$$

$$C \to \neg A, \quad C \to D, \quad D \to aD, \quad D \to \epsilon$$

The semantics of this grammar should clearly be $(A, B, C, D) = (\Sigma^*, \Sigma^*, \Sigma^*, \Sigma^*)$, and actually this is what the well-founded semantics will produce. On the other hand the naturally feasible solution semantics is undefined.

The problem of giving semantics to recursive formalisms in the presence of negation has been extensively studied in the context of logic programming. Actually, the unique solution semantics can be paralleled with one of the early attempts to give semantics to logic programs with negation, namely what is now called *the Clark's completion semantics* (which actually presents similar shortcomings with the unique solution approach). On the other hand, the naturally feasible solution can be thought of as a first approximation to the procedure of constructing the intended minimal model of a logic program with negation (see also Theorem 3 that will follow). Since the most broadly accepted semantic approach for logic programs with negation is the well-founded semantics, we adopt this approach in this paper.

At this point we should also mention a recent work on the *stratified* semantics of Boolean grammars [Wro05], an idea that also originates from logic programming. However, the stratified semantics is less general than the well-founded one (since the former does not cover the whole class of Boolean grammars).

3 Interpretations and Models for Boolean Grammars

In this section we formally define the notion of *model* for boolean grammars. In context-free grammars, an interpretation is a function that assigns to each nonterminal symbol of the grammar a set of strings over the set of terminal symbols of the grammar. An interpretation of a context-free grammar is a model of the grammar if it satisfies all the rules of the grammar. The usual semantics of context-free grammars dictate that every such grammar has a minimum model, which is taken to be as its intended meaning.

When one considers boolean grammars, the situation becomes much more complicated. For example, a grammar with the unique rule $S \to \neg S$ appears to be meaningless. More generally, in many cases where negation is used in a circular way, the corresponding grammar looks problematic. However, these difficulties arise because we are trying to find *classical* models of boolean grammars, which are based on classical two-valued logic. If however we shift to three-valued models, every boolean grammar has a well-defined meaning. We need of course to redefine many notions, starting even from the notion of a language:

Definition 2. *Let Σ be a finite non-empty set of symbols. Then, a (three-valued) language over Σ is a function from Σ^* to the set $\left\{0, \frac{1}{2}, 1\right\}$.*

Intuitively, given a three-valued language L and a string w over the alphabet of L, there are three-cases: either $w \in L$ (ie., $L(w) = 1$), or $w \notin L$ (ie., $L(w) = 0$), or finally, the membership of w in L is unclear (ie., $L(w) = \frac{1}{2}$). Given this extended notion of language, it is now possible to interpret the grammar $S \longrightarrow \neg S$: its meaning is the language which assigns to every string the value $\frac{1}{2}$.

The following definition, which generalizes the familiar notion of concatenation of languages, will be used in the following:

Definition 3. *Let Σ be a finite set of symbols and let $L_1, \ldots, L_n$ be (three-valued) languages over Σ. We define the* three-valued concatenation *of the languages $L_1, \ldots, L_n$ to be the language L such that:*

$$L(w) = \max_{\substack{(w_1,\ldots,w_n):\\ w=w_1\cdots w_n}} \left(\min_{1\le i\le n} L_i(w_i) \right)$$

The concatenation of $L_1, \ldots, L_n$ will be denoted by $L_1 \circ \cdots \circ L_n$.

We can now define the notion of *interpretation* of a given boolean grammar:

Definition 4. *An* interpretation *I of a boolean grammar $G = (\Sigma, N, P, S)$ is a function $I : N \to \left(\Sigma^* \to \left\{0, \frac{1}{2}, 1\right\}\right)$.*

An interpretation I can be recursively extended to apply to expressions that appear as the right-hand sides of boolean grammar rules:

Definition 5. *Let $G = (\Sigma, N, P, S)$ be a boolean grammar and I be an interpretation of G. Then I can be extended to become a* truth valuation *$\hat{I}$ as follows:*

- *For the empty sequence ϵ and for all $w \in \Sigma^*$, it is $\hat{I}(\epsilon)(w) = 1$ if $w = \epsilon$ and 0 otherwise.*
- *Let $a \in \Sigma$ be a terminal symbol. Then, for every $w \in \Sigma^*$, $\hat{I}(a)(w) = 1$ if $w = a$ and 0 otherwise.*
- *Let $\alpha = \alpha_1 \cdots \alpha_n$, $n \geq 1$, be a sequence in $(\Sigma \cup N)^*$. Then, for every $w \in \Sigma^*$, it is $\hat{I}(\alpha)(w) = (\hat{I}(\alpha_1) \circ \cdots \circ \hat{I}(\alpha_n))(w)$.*
- *Let $\alpha \in (\Sigma \cup N)^*$. Then, for every $w \in \Sigma^*$, $\hat{I}(\neg\alpha)(w) = 1 - \hat{I}(\alpha)(w)$.*
- *Let $l_1, \ldots, l_n$ be literals. Then, for every string $w \in \Sigma^*$, $\hat{I}(l_1 \& \cdots \& l_n)(w) = \min\{\hat{I}(l_1)(w), \ldots, \hat{I}(l_n)(w)\}$.*

We are now in a position to define the notion of a model of a boolean grammar:

Definition 6. *Let $G = (\Sigma, N, P, S)$ be a boolean grammar and I an interpretation of G. Then, I is a* model *of G if for every rule $A \to l_1 \& \cdots \& l_n$ in P and for every $w \in \Sigma^*$, it is $\hat{I}(A)(w) \geq \hat{I}(l_1 \& \cdots \& l_n)(w)$.*

In the definition of the well-founded model, two orderings on interpretations play a crucial role (see [PP90]). Given two interpretations, the first ordering (usually called the *standard ordering*) compares their *degree of truth*:

Definition 7. *Let $G = (\Sigma, N, P, S)$ be a boolean grammar and I, J be two interpretations of G. Then, we say that $I \preceq J$ if for all $A \in N$ and for all $w \in \Sigma^*$, $I(A)(w) \leq J(A)(w)$.*

Among the interpretations of a given boolean grammar, there is one which is the least with respect to the $\preceq$ ordering, namely the interpretation $\bot$ which for all A and all w, $\bot(A)(w) = 0$.

The second ordering (usually called the *Fitting ordering*) compares the *degree of information* of two interpretations:

Definition 8. *Let $G = (\Sigma, N, P, S)$ be a boolean grammar and I, J be two interpretations of G. Then, we say that $I \preceq_F J$ if for all $A \in N$ and for all $w \in \Sigma^*$, if $I(A)(w) = 0$ then $J(A)(w) = 0$ and if $I(A)(w) = 1$ then $J(A)(w) = 1$.*

Among the interpretations of a given boolean grammar, there is one which is the least with respect to the $\preceq_F$ ordering, namely the interpretation $\bot_F$ which for all A and all w, $\bot_F\ (A)(w) = \frac{1}{2}$.

Given a set U of interpretations, we will write $lub_{\preceq}U$ (respectively $lub_{\preceq_F}U$) for the least upper bound of the members of U under the standard ordering (respectively, the Fitting ordering).

4 Well-Founded Semantics for Boolean Grammars

In this section we will define the well-founded semantics of boolean grammars. The basic idea behind the well-founded semantics is that the intended model of the grammar is constructed in stages, ie., there is a stratification process involved that is related to the levels of negation used by the grammar. For every nonterminal symbol, at each step of this process, the values of certain strings are computed and fixed (as either true or false); at each new level, the values of more and more strings become fixed (and this is a monotonic procedure in the sense that values of strings that have been fixed for a given nonterminal in a previous stage, cannot be be altered by the next stages). At the end of all the stages, certain strings for certain nonterminals may have not managed to get the status of either true or false (this will be due to circularities through negation in the grammar). Such strings are classified as unknown (ie., $\frac{1}{2}$).

Consider the boolean grammar G. Then, for any interpretation J of G we define the operator $\Theta_J : \mathcal{I} \to \mathcal{I}$ on the set $\mathcal{I}$ of all 3-valued interpretations of G. This operator is analogous to the one used in the logic programming domain (see for example [PP90]).

Definition 9. *Let $G = (\Sigma, N, P, S)$ be a boolean grammar, let $\mathcal{I}$ be the set of all three-valued interpretations of G and let $J \in \mathcal{I}$. The operator $\Theta_J : \mathcal{I} \to \mathcal{I}$ is defined as follows. For every $I \in \mathcal{I}$, for all $A \in N$ and for all $w \in \Sigma^*$:*

1. *$\Theta_J(I)(A)(w) = 1$ if there is a rule $A \to l_1 \& \cdots \& l_n$ in P such that, for all $i \leq n$, either $\hat{J}(l_i)(w) = 1$ or l_i is positive and $\hat{I}(l_i)(w) = 1$;*
2. *$\Theta_J(I)(A)(w) = 0$ if for every rule $A \to l_1 \& \cdots \& l_n$ in P, there is an $i \leq n$ such that either $\hat{J}(l_i)(w) = 0$ or l_i is positive and $\hat{I}(l_i)(w) = 0$;*
3. *$\Theta_J(I)(A)(w) = \frac{1}{2}$, otherwise.*

An important fact regarding the operator Θ_J is that it is monotonic with respect to the $\preceq$ ordering of interpretations:

Theorem 1. *Let G be a boolean grammar and let J be an interpretation of G. Then, the operator Θ_J is monotonic with respect to the $\preceq$ ordering of interpretations. Moreover, Θ_J has a unique least (with respect to $\preceq$) fixed point $\Theta_J^{\uparrow\omega}$ which is defined as follows:*

$$\begin{aligned}\Theta_J^{\uparrow 0} &= \bot \\ \Theta_J^{\uparrow n+1} &= \Theta_J(\Theta_J^{\uparrow n}) \\ \Theta_J^{\uparrow\omega} &= lub_{\preceq}\{\Theta_J^{\uparrow n} \mid n < \omega\}\end{aligned}$$

Proof. The proof essentially follows the same lines of thought as that of the logic programming case (see [Prz89]). □

We will denote by $\Omega(J)$ the least fixed point of Θ_J. Given a grammar G, we can use the Ω operator to construct a sequence of interpretations whose ω-limit M_G will prove to be a distinguished model of G:

$$\begin{aligned}M_0 &= \bot_F \\ M_{n+1} &= \Omega(M_n) \\ M_G &= lub_{\preceq_F}\{M_n \mid n < \omega\}\end{aligned}$$

Notice that here we have an essential difference with respect to the well-founded semantics of logic programming: there, the construction of the well-founded model may require a transfinite number of iterations which is greater than ω. In other words, the well-founded semantics of logic programs is not computable in the general case. However, in the case of Boolean grammars, the model is constructed in at most ω iterations:

Theorem 2. *Let G be a boolean grammar. Then, M_G is a model of G (which will be called the* well-founded model of G*). Moreover, M_G is the least (with respect to the $\preceq_F$ ordering) fixed point of the operator Ω.*

Proof. Technically, the proof is very similar to that of the logic programming case (see [Prz89]). □

Actually, it can be shown (following a similar reasoning as in [RW05]) that the model M_G is the *least* model of G according to a syntax-independent relation.

The construction of the well-founded model is illustrated by the following example:

Example 2. Let G be the grammar given in Example 1. Then, it is easy to see that $M_G = M_2$, ie., that in order to converge to the well-founded model of G we need exactly two iterations of Ω. More specifically, in $M_1 = \Omega(M_0)$ the denotations of the non-terminals A, B and C stabilize (notice that the definitions of these nonterminals are standard context-free rules). However, in order for the denotation of S to stabilize, an additional iteration of Ω is required. Notice that the language produced by this grammar is two-valued.

We can now state the relationship between the well-founded semantics and the naturally feasible semantics of boolean grammars:

Theorem 3. *Suppose that a boolean grammar G has a two-valued (ie., with values 0 and 1) well-founded semantics. Then the naturally feasible solution for this grammar either coincides with the well-founded semantics or is undefined.*

It is easy to see that if a boolean grammar has a naturally feasible solution semantics, then it is possible that this semantics differs from the well-founded one. For example, in the four-rule grammar of Section 2, the well-founded semantics assigns the $\bot_F$ interpretation to all the nonterminal symbols of the grammar. Notice that although the naturally feasible semantics for this grammar is defined, it appears to be counterintuitive.

5 Normal Form

In this section we demonstrate that every boolean grammar can be converted into an equivalent one that belongs to the following normal form:

Definition 10. *A Boolean grammar $G = (\Sigma, N, P, S)$ is said to be in binary normal form if P contains the rules $U \rightarrow \neg U$ and $T \rightarrow \neg\epsilon$, where U and T are two special symbols in $N - \{S\}$, and every other rule in P is of the form:*

$$A \rightarrow B_1C_1\& \cdots \&B_mC_m\&\neg(D_1E_1)\& \cdots \&\neg(D_nE_n)\&TT[\&U] \quad (m, n \geq 0)$$
$$A \rightarrow a[\&U]$$
$$S \rightarrow \epsilon[\&U] \quad \textit{(only if } S \textit{ does not appear in right-hand sides of rules)}$$

where $A, B_i, C_i, D_j, E_j \in N - \{U, T\}$, $a \in \Sigma$, and the brackets denote an optional part.

The basic theorem of this section states that for every boolean grammar G there exists a boolean grammar in binary normal form that defines the same language as G. More formally:

Theorem 4. *Let $G = (\Sigma, N, P, S)$ be a boolean grammar. Then there exists a grammar $G' = (\Sigma, N', P', S)$ in binary normal form such that $M_G(S) = M_{G'}(S)$.*

The proof of Theorem 4 is based on several transformations, justified by some lemmata given below. We give here an outline of how the binary normal form is constructed.

Consider a boolean grammar $G = (\Sigma, N, P, S)$. Without loss of generality we may assume that S does not appear in the right-hand side of any rule (otherwise we can replace S with S' in every rule, and add a rule $S \rightarrow S'$). Initially, we bring the grammar into a form, which we call *pre-normal form* (see Definition 11). This is performed using Lemmas 1,2 and 3. More specifically, Lemma 1 is used to eliminate terminal symbols from rules containing boolean connectives or concatenation; Lemma 2 separates boolean connectives from concatenation; and, Lemma 3 is used to eliminate "long" concatenations. Based on the pre-normal form, we then construct an ϵ-free version of the grammar (Definition 12). The ϵ-free version is then brought into binary-normal form (see Definition 10 above) using the technique described in Definition 15. Detailed proofs of lemmas comprising this procedure are lengthy, and are omitted in the current form of the paper.

Lemma 1. *Let $G = (\Sigma, N, P, S)$ be a boolean grammar, and let G' be the grammar $(\Sigma, N \cup \{A_a \mid a \in \Sigma\}, P' \cup \{A_a \to a \mid a \in \Sigma\}, S)$ where:*

- $\{A_a \mid a \in \Sigma\} \cap N = \emptyset$.
- *P' is obtained from P by replacing each occurrence of the terminal symbol a with A_a, in every rule that contains concatenation or boolean connectives.*

Then, for every $C \in N$, $M_G(C) = M_{G'}(C)$.

Lemma 2. *Let $G = (\Sigma, N, P, S)$ be a boolean grammar, and let $\beta \in N^k$, $k \geq 2$, be a sequence of non-terminal symbols. Let $G' = (\Sigma, N \cup \{B\}, P' \cup \{B \to \beta\}, S)$ where:*

- *$B \notin N$ is a new non-terminal symbol.*
- *For every rule $A \to \alpha_1 \& \cdots \& \alpha_m \& \neg\alpha_{m+1} \& \cdots \& \neg\alpha_n$ in P, P' contains the rule $A \to \alpha'_1 \& \cdots \& \alpha'_m \& \neg\alpha'_{m+1} \& \cdots \& \neg\alpha'_n$, where $\alpha'_i = B$ if $\alpha_i = \beta$, otherwise $\alpha'_i = \alpha_i$.*

Then, for every $C \in N$, $M_G(C) = M_{G'}(C)$.

Lemma 3. *Let $G = (\Sigma, N, P, S)$ be a boolean grammar, let $A \to B_1 B_2 B_3 \dots B_k$, $A, B_i \in N$, $k \geq 3$, be a rule of P and let $G' = (\Sigma, N \cup \{D\}, P', S)$ where:*

- *$D \notin N$ is a new non-terminal symbol.*
- *$P' = (P - \{A \to B_1 B_2 B_3 \dots B_k\}) \cup \{A \to D B_3 \dots B_k, D \to B_1 B_2\}$.*

Then, for every $C \in N$, $M_G(C) = M_{G'}(C)$.

Using the above lemmas it is straightforward to bring the initial grammar into the following form:

Definition 11. *A Boolean grammar $G = (\Sigma, N, P, S)$ is said to be in pre-normal form if every rule in P is of the form:*

$$\begin{array}{ll} A \to B_1 \& \cdots \& B_m \& \neg C_1 \& \cdots \& \neg C_n & (m + n \geq 1, B_i, C_j \in N \cup \{\epsilon\}) \\ A \to BC & (B, C \in N) \\ A \to a & (a \in \Sigma) \end{array}$$

Based on the pre-normal form of the grammar, we can now define its ϵ-free version:

Definition 12. *Let $G = (\Sigma, N, P, S)$ be a boolean grammar in pre-normal form. The* ϵ-free version *of G, denoted by G_ϵ, is the boolean grammar $(\Sigma, N \cup \{U\}, P', S)$ where P' is obtained as follows:*

- *P' contains a rule $U \to \neg U$, where $U \notin N$ is a special non-terminal symbol, which represents the set in which all strings have the value $\frac{1}{2}$.*
- *For every rule of the form $A \to B_1 \& \cdots \& B_m \& \neg C_1 \& \cdots \& \neg C_n$, $(m + n \geq 1, B_i, C_j \in N \cup \{\epsilon\})$ in P*
  - *If $B_i = \epsilon$ for some i, then the rule is ignored in the construction of P'.*
  - *Otherwise, if $C_i = \epsilon$ for some i, then the rule is included in P' as it is.*

- *Otherwise, P' contains the rule* $A \rightarrow B_1 \& \cdots \& B_m \& \neg C_1 \& \cdots \& \neg C_n \& \neg\epsilon$

– *For every rule of the form* $A \rightarrow BC$ *in* P
- P' *contains the rule* $A \rightarrow BC \& \neg\epsilon$
- *If* $M_G(B)(\epsilon) = 1$ *(respectively* $M_G(C)(\epsilon) = 1$*), then* P' *contains the rule* $A \rightarrow C \& \neg\epsilon$ *(respectively the rule* $A \rightarrow B \& \neg\epsilon$*).*
- *If* $M_G(B)(\epsilon) = \frac{1}{2}$ *(respectively* $M_G(C)(\epsilon) = \frac{1}{2}$*), then* P' *contains the rule* $A \rightarrow C \& U \& \neg\epsilon$ *(respectively the rule* $A \rightarrow B \& U \& \neg\epsilon$*).*

– *For every* $a \in \Sigma$ *and* $A \in N$*, if* $M_G(A)(a) = 1$ *then* P' *contains the rule* $A \rightarrow a$ *and if* $M_G(A)(a) = \frac{1}{2}$ *then* P' *contains the rule* $A \rightarrow a \& U$

Lemma 4. *Let* $G = (\Sigma, N, P, S)$ *be a boolean grammar in pre-normal form, and let* G_ϵ *be its* ϵ*-free version. Then, for every* $C \in N$ *and for every* $w \in \Sigma^*$*,* $w \neq \epsilon$ *implies* $M_G(C)(w) = M_{G_\epsilon}(C)(w)$*.*

In order to obtain a grammar in binary normal form, we need to eliminate rules of the form $A \rightarrow B_1 \& \cdots \& B_m \& \neg C_1 \& \cdots \& \neg C_n \& \neg\epsilon$. Membership in $M_G(A)$ depends only on membership in each of $\hat{M}_G(BC)$, for all BC that appear in the right-hand sides of rules. We can express this dependency directly by a set of rules. In order to do this we treat each BC that appears in the right-hand side of a rule as a boolean variable (see also [Okh04]).

Definition 13. *Let* X *be a set of variables and let* V, W *be functions from* X *to* $\{0, \frac{1}{2}, 1\}$*. We denote by* V_i *the set* $\{x \in X \mid V(x) = i\}$*. We write* $V \sqsubseteq W$ *if* $V_0 \subseteq W_0$ *and* $V_1 \subseteq W_1$

Definition 14. *Let* G *be a grammar in pre-normal form and let* $G_\epsilon = (\Sigma, N, P, S)$ *be the* ϵ*-free version of* G*. Let* $X = \{BC \mid A \rightarrow BC \in P\}$ *and let* V *be a function from* X *to* $\{0, \frac{1}{2}, 1\}$*. Then, the extension of* V *to non-terminal symbols in* N*, denoted by* $\hat{V}$*, is defined as follows:* $\hat{V}(A)$ *is the value* $M_{G_\epsilon}(A)(w)$ *when* $M_{G_\epsilon}(BC)(w) = V(BC)$*, for all* $BC \in X$ *and for arbitrary* w*.*

Notice that $\hat{V}$ is well-defined and can be computed in finitely many steps from V.

Definition 15. *Let* G *be a grammar in pre-normal form, let* $G_\epsilon = (\Sigma, N, P, S)$ *be the* ϵ*-free version of* G*. Let* $X = \{BC \mid A \rightarrow BC \in P\}$ *and let* $\mathcal{V}$ *be the set of all functions from* X *to* $\{0, \frac{1}{2}, 1\}$*. The normal form* $G_n = (\Sigma, N \cup \{T\}, P', S)$ *of* G *is the grammar obtained from* G_ϵ *as follows:*

– P' *contains all the rules in* P *of the form* $A \rightarrow a$ *and* $A \rightarrow a \& U$*, where* $a \in \Sigma$*, the rule* $U \rightarrow \neg U$ *in* P *and the rule* $T \rightarrow \neg\epsilon$*, where* $T \notin N$ *is a special symbol which represents the set in which all non-empty strings have value* 1*.*

– *For every* $A \in N$ *let* $\mathcal{T}_A = \{V \in \mathcal{V} \mid \hat{V}(A) = 1\}$*. For every minimal (with respect to* $\sqsubseteq$*) element* V *of* $\mathcal{T}_A$*,* P' *contains the rule:*

$$A \rightarrow x_{i_1} \& \ldots \& x_{i_m} \& \neg y_{j_1} \& \ldots \& \neg y_{j_n} \& TT$$

where $\{x_{i_1}, \ldots, x_{i_m}\} = V_1$ *and* $\{y_{j_1}, \ldots, y_{j_n}\} = V_0$*.*

– *For every* $A \in N$ *let* $\mathcal{U}_A = \{V \in \mathcal{V} \mid \hat{V}(A) = \frac{1}{2}\}$. *For every maximal (with respect to* $\sqsubseteq$*) element* V *of* $\mathcal{U}_A$, P' *contains the rule:*

$$A \rightarrow x_{i_1} \& \neg x_{i_1} \& \ldots \& x_{i_m} \& \neg x_{i_m} \& TT \& U$$

where $\{x_{i_1}, \ldots, x_{i_m}\} = V_{\frac{1}{2}}$.

Notice that in the former case we consider only minimal elements, because if $V' \sqsubseteq V$ and $\hat{V}'(A) = 1$ then $\hat{V}(A) = 1$. Similarly in the latter case we consider only maximal elements, because if $V' \sqsubseteq V$ and $\hat{V}(A) = \frac{1}{2}$ then $\hat{V}'(A) = \frac{1}{2}$. The above properties follow from the monotonicity of the Ω operator, with respect to the $\preceq_F$ (Fitting) ordering.

Lemma 5. *Let* $G = (\Sigma, N, P, S)$ *be a boolean grammar in pre-normal form, and let* G_n *be its binary normal form. Then, for every* $A \in N$ *and for every* $w \in \Sigma^*$, $w \neq \epsilon$ *implies* $M_G(A)(w) = M_{G_n}(A)(w)$.

Given the above lemmas, a simple step remains in order to reach the statement of Theorem 4: if in the original grammar G it is $M_G(S)(\epsilon) \neq 0$, then an appropriate rule of the form $S \rightarrow \epsilon$ or $S \rightarrow \epsilon \& U$ is added to the grammar that has resulted after the processing implied by all the above lemmas. The resulting grammar is then in binary normal form and defines the same language as the initial one.

6 Parsing Under the Well-Founded Semantics

We next present an algorithm that computes the truth value of the membership of an input string $w \neq \epsilon$ in a language defined by a grammar G, which is assumed to be in binary normal form. The algorithm computes the value of $M_G(A)(u)$ for every non-terminal symbol A and every substring u of w in a bottom up manner. By convention $min_{i=1}^{0} v_i = 1$.

Algorithm for parsing an input string $w = a_1 \cdots a_n$:

for $i := 1$ to n do begin
  for every $A \in N$ do
    if there exist a rule $A \rightarrow a_i$ then $M_G(A)(a_i) := 1$
    else if there exist a rule $A \rightarrow a_i \& U$ then $M_G(A)(a_i) := \frac{1}{2}$
    else $M_G(A)(a_i) := 0$
end

for $d := 2$ to n do
  for $i := 1$ to $n - d + 1$ do begin
    $j := i + d - 1$
    for every $B, C \in N$ such that BC appears in the right-hand side of a rule do
      $\hat{M}_G(BC)(a_i \ldots a_j) := \max_{k=i}^{j-1} \min\{M_G(B)(a_i \ldots a_k), M_G(C)(a_{k+1} \ldots a_j)\}$
    for every $A \in N$ do $M_G(A)(a_i \ldots a_j) := 0$
    for every rule $A \rightarrow B_1 C_1 \& \ldots \& B_m C_m \& \neg D_1 E_1 \& \ldots \& \neg D_r E_r \& TT \& U$ do begin
      $v := \min\{\frac{1}{2}, \min_{p=1}^{m} \hat{M}_G(B_p C_p)(a_i \ldots a_j), \min_{q=1}^{r}(1 - \hat{M}_G(D_q E_q)(a_i \ldots a_j))\}$

```
            if v > M_G(A)(a_i ... a_j) then M_G(A)(a_i ... a_j) := v
        end
        for every rule A → B_1C_1& ... &B_mC_m&¬D_1E_1& ... &¬D_rE_r&TT do begin
            v := min{min_{p=1}^{m} M̂_G(B_pC_p)(a_i ... a_j), min_{q=1}^{r}(1 − M̂_G(D_qE_q)(a_i ... a_j))}
            if v > M_G(A)(a_i ... a_j) then M_G(A)(a_i ... a_j) := v
        end
    end
return M_G(S)(a_1 ··· a_n)
```

For a fixed grammar the above algorithm runs in time $\mathcal{O}(n^3)$: the value $M_G(A)(u)$ is computed for $\mathcal{O}(n^2)$ substrings u of w; each computation requires to break u in two parts in all possible ways, and there are $\mathcal{O}(n)$ appropriate breakpoints.

7 Conclusions

We have presented a novel semantics for boolean grammars, based on techniques that have been developed in the logic programming domain. Under this new semantics every boolean grammar has a distinguished language that satisfies its rules. Moreover, we have demonstrated that every boolean grammar can be transformed into an equivalent one in a binary normal form. For grammars in binary normal form, we have derived an $\mathcal{O}(n^3)$ parsing algorithm.

We believe that a further investigation of the connections between formal language theory and the theory of logic programming will prove to be very rewarding.

References

[AB94] Apt, K., Bol, R.: Logic Programming and Negation: A Survey. Journal of Logic Programming **19,20** (1994) 9–71

[Okh01] Okhotin, A.: Conjunctive Grammars. Journal of Automata, Languages and Combinatorics **6(4)** (2001) 519–535

[Okh04] Okhotin, A.: Boolean Grammars. Information and Computation **194(1)** (2004) 19–48

[PP90] Przymusinska, H., Przymusinski, T.: Semantic Issues in Deductive Databases and Logic Programs. In R. Banerji, editor, Formal Techniques in Artificial Intelligence: a Source-Book, North Holland (1990) 321–367

[Prz89] Przymusinski, T., C,: Every Logic Program has a Natural Stratification and an Iterated Fixed Point Model. In Proceedings of the 8th Symposium on Principles of Database Systems ACM SIGACT-SIGMOD (1989) 11–21

[RW05] Rondogiannis, P., Wadge, W., W.: Minimum Model Semantics for Logic Programs with Negation-as-Failure. ACM Transactions on Computational Logic **6(2)** (2005) 441–467

[vGRS91] van Gelder, A, Ross, K., A., Schlipf, J., S.: The Well-Founded Semantics for General Logic Programs. Journal of the ACM **38(3)** (1991) 620–650

[Wro05] Wrona M.: Stratified Boolean Grammars. MFCS (2005) 801–812