This is the preprint of the article that appeared in final form in **Theoretical Computer Science** 411 : 1723-1733 (2010) , which has been published in final form at https://doi.org/10.1016/j.tcs.2010.01.013.

Translating Propositional Extended Conjunctions of Horn Clauses into Boolean Circuits ⋆

Jose Gaintzarain,

EUITI de Bilbao,
48012-Bilbao, SPAIN

Montserrat Hermo, Paqui Lucio and Marisa Navarro *

Facultad de Informática,
P.O. Box 649, 20080-San Sebastián, SPAIN

Abstract

$Horn^{\supset}$ is a logic programming language which extends usual $Horn$ clauses by adding intuitionistic implication in goals and clause bodies. This extension can be seen as a form of structuring programs in logic programming. We are interested in finding correct and efficient translations from $Horn^{\supset}$ programs into some representation type that, preserving the signature, allow us suitable implementations of these kind of programs. In this paper we restrict to the propositional setting of $Horn^{\supset}$ and we study correct translations into Boolean circuits, i.e. graphs; into Boolean formulas, i.e. trees; and into conjunctions of propositional $Horn$ clauses. Different results about the efficiency of the transformations are obtained in the three cases.

Key words: Boolean circuits, Boolean formulas, Horn clauses, Horn$^{\supset}$ clauses

1 Introduction

In logic programming, some approaches for extending $Horn$ clauses consider to incorporate into the language a new implication symbol, $\supset$, with the aim

⋆ This work has been partially supported by Spanish Project TIN2007-66523 and the Basque Project LoRea GIU07/35. It is an expanded version of a talk presented at MFCS'2005 [5].
* Corresponding author: montserrat.hermo@ehu.es

of structuring logic programs in some blocks with local clauses [1,4,6,7,10–14]. These extensions can also be seen as a sort of inner modularity in logic programming (see [3] for a survey on modularity).

A typical example, borrowed from [10], is the following program (written in Prolog terminology) for efficiently reversing a list of elements:

$$reverse(In, Out) :- D \supset revAux(In, Out, []).$$

where D is the set formed by the two following clauses

$$revAux([], K, K).$$
$$revAux([X|L], K, Aux) :- revAux(L, K, [X|Aux]).$$

By using the new symbol $\supset$, the definition of $revAux$ is local and therefore only accessible inside a call to $reverse$.

The different extensions depend on considering closed or open blocks. Moreover, for open blocks, a scope rule is required to relate the possible definitions of each predicate with each call to such predicate. There are mainly two scope rules. In the *dynamic* scope rule, the actual (when it is called) definition of a predicate depends on the history of calls till that moment whereas in the *static* scope rule, such definition depends on program block structure. We consider a particular extension, named $Horn^{\supset}$, defined for open blocks with the static scope rule. This programming language has been formally studied in [1,6,7,14]. In [1] a natural extension of classical first order logic $\mathcal{FO}$ with the intuitionistic implication ($\supset$), named $\mathcal{FO}^{\supset}$, is presented as the underlying logic of the programming language $Horn^{\supset}$. Additionally, in [8] a complete calculus for $\mathcal{FO}^{\supset}$ is introduced.

Model semantics of $\mathcal{FO}^{\supset}$ is based on Kripke structures consisting of a non-empty partially ordered set of worlds, each world associated to an interpretation. However, to deal with $Horn^{\supset}$, Kripke structures can be restricted to those with (a) Herbrand interpretations associated to their worlds, (b) a unique minimal world and (c) closure with respect to superset. Moreover, each interpretation I univocally determines a Kripke structure (formed with all the supersets of I) and, conversely, each Kripke structure satisfying conditions (a), (b) and (c) is univocally determined by (the interpretation associated to) its minimal world.

Other "good properties" that verify $Horn$ clauses (as a programming language) with respect to its underlying logic $\mathcal{FO}$ are also conserved by $Horn^{\supset}$ clauses with respect to $\mathcal{FO}^{\supset}$: each program has a canonical model, the operational semantics is an effective subcalculus of a complete calculus for $\mathcal{FO}^{\supset}$ and the goals satisfied in the canonical model are the goals that can be derived from the program in such calculus. The formalization about what are "good properties of a programming language" is borrowed from [9] and proved for $Horn^{\supset}$ in [1].

More related to implementation issues, a usual way to proceed is to translate the extended logic programs into the language of some well-known logic [2,6,13,14,16]. For instance, [14,16] present a transformation of the given structured program into a flat one. More concretely, [14] introduces a translation from $Horn^{\supset}$ programs into $Horn$ programs, in the propositional setting, by preserving the original operational semantics in $Horn^{\supset}$ by means of SLD-resolution on the resulting $Horn$ program. In [16], this transformation is lifted to the first order case, and generalized to normal constraint logic programs extended with $\supset$ as structuring mechanism. Such translation obtains the translated program in a signature which extends the original one with new predicate symbols.

Concerning models, a more appropriate comparison can be done if translations use the same signature. In these cases we can preserve the equivalence between formulas and its transformations rather than only preserving satisfiability. In this paper, our aim is to study possible correct and efficient translations from propositional $Horn^{\supset}$ programs into some ($\mathcal{FO}$ logic based) representation type preserving the signature. Since we restrict our study to the propositional case, from now on, $Horn^{\supset}$ always means propositional $Horn^{\supset}$.

In the task of representing Boolean functions, although, in principle, any valid representation is allowed, some of them may be preferred because they are more succinct, more efficient to manipulate or more indicative of the complexity of the function. The three representation types we have chosen are $Horn$ clauses, Boolean formulas and Boolean circuits. All of them are well-known data structures for representing Boolean functions. In general, the description of a Boolean function should be rather short and efficient; support the evaluation and manipulation of the function; make particular properties of the function visible; suggest ideas for a technical realization. Boolean circuit constitutes a representation type which satisfies all the above properties, but mainly the first one: the fact that the out-degree of its gates can be greater than 1, often allows very compact representations.

The study made in this paper shows how to translate programs from the extended programming language into equivalent $Horn$ programs, Boolean formulas and Boolean circuits. Such translations prove that $Horn^{\supset}$ programs can be represented efficiently by Boolean circuits, while the size is exponential when the translation is into $Horn$ programs. Regarding to Boolean formulas, we are not able to ensure that the size of the formula obtained by the translation is bounded by a polynomial and we leave this question open. In fact, we show that this question is an instance of a well-known open problem.

The paper is organized as follows: In Section 2 the programming language $Horn^{\supset}$ is introduced. In Section 3 some preliminary notions and properties about Boolean circuits are given. In Section 4 we prove that any translation from a $Horn^{\supset}$ program into an equivalent $Horn$ program obtains, in general, an exponential number of clauses. Then, looking for a more efficient representation, in Sections 5 and 6 we present two translations from $Horn^{\supset}$ goals into monotone Boolean circuits and, respectively, into monotone Boolean formulas. Both transformations are proved correct. The main result is about efficiency: the transformation from $Horn^{\supset}$ goals into monotone circuits is proved to be linear, but the question of whether the transformation into Boolean formulas is efficient remains open. We conclude, in Section 7, by summarizing our results.

2 The Extended Programming Language

In this section, after some preliminary definitions, we introduce the programming language $Horn^{\supset}$ by showing its syntax and its model semantics. We also define the persistency and equivalence of formulas and prove some useful results for later sections.

2.1 Preliminaries

We introduce here basic terminology on propositional $Horn$ clauses, $Horn$ programs and its models.
A *signature* Σ is a fixed set of propositional variables. A Σ-formula is a formula built from variables in Σ, constants (*true* and *false*) and classical connectives ($\neg$, $\wedge$, $\vee$, and $\rightarrow$).

A $Horn$ clause D is a Σ-formula of the form $G \rightarrow v$ where v is a variable in Σ and G is a $Horn$ goal, or simply of the form v. In logic programming terminology, a $Horn$ clause $G \rightarrow v$ is usually called "a rule" with head v and body G, whereas a $Horn$ clause of the form v is usually called "a fact". A $Horn$ goal G is a conjunction of one or more variables in Σ. Both definitions can be summarized in the following way:

$$G ::= v \mid G_1 \wedge G_2 \qquad\qquad D ::= v \mid G \rightarrow v$$

A $Horn$ program is a set of $Horn$ clauses, $P = \{D_1, D_2, \ldots, D_n\}$, but it can alternatively be seen as the conjunction of its clauses, $P = D_1 \wedge D_2 \wedge \ldots \wedge D_n$.

Given a signature Σ, the model semantics for $Horn$ is given by the set of all Σ-interpretations $\underline{\mathsf{Mod}}(\Sigma) = \{I \mid I \subseteq \Sigma\}$. A Σ-interpretation I assigns a truth value ($True$ or $False$) to each variable v in Σ: $I(v) = True$ if and only if $v \in I$. It is well-known that each Σ-interpretation I determines a unique truth value $I(\varphi)$ to each Σ-formula φ.

I is a *model* of φ when $I(\varphi) = True$. This is usually denoted by $I \models \varphi$. Since clauses and goals are formulas, and a program P is the conjunction of its clauses, I is a *model* of P if it is a model of all its clauses. When working with $Horn$ programs, the intersection of all models of P is also a model of P, named its *canonical* model.

Example 1 *The set* $\{c,\ d,\ (c \wedge d) \rightarrow b,\ (b \wedge a) \rightarrow a\}$ *is a Horn program with four clauses over signature* $\Sigma = \{a, b, c, d\}$*. Among all the* Σ*-interpretations, only two of them are models of the program:* $I_1 = \{c,\ d,\ b,\ a\}$ *and* $I_2 = \{c,\ d,\ b\}$*.* I_2 *is the canonical model.* ∎

2.2 The syntax of $Horn^{\supset}$

The syntax of the programming language $Horn^{\supset}$ is an extension of the propositional $Horn$ language by adding the intuitionistic implication $\supset$ in goals (and therefore in clause bodies). Let Σ be a fixed signature.

$Horn^{\supset}$ clauses, named D, and $Horn^{\supset}$ goals, named G, are recursively defined as follows (where v stands for any variable in Σ):

$$G ::= v \mid G_1 \wedge G_2 \mid D \supset G \qquad\qquad D ::= v \mid G \rightarrow v \mid D_1 \wedge D_2$$

Although the definition of clauses (respectively goals) does not include the constant *true*, sometimes, for technical reasons, we consider *true* as a clause (respectively a goal).

A $Horn^{\supset}$ program is a finite set (or conjunction) of $Horn^{\supset}$ clauses. The main difference between a $Horn^{\supset}$ program and a $Horn$ program is the use of a "local" clause set D in goals of the kind $D \supset G$.

Example 2 *The following set with three clauses is a* $Horn^{\supset}$ *program over signature* $\Sigma = \{a, b, c, d\}$

$$\{((b \rightarrow c) \supset c) \rightarrow a,\ b,\ ((a \wedge (b \rightarrow c)) \supset (((b \rightarrow c) \wedge (a \rightarrow d)) \supset a)) \rightarrow d\}$$

The second clause is simply b*. The first and the third program clauses are of the form* $G \rightarrow v$*. In the first one, the goal* G *is* $(b \rightarrow c) \supset c$*. That is, it contains a local set with one clause. In the third clause, the goal* G *is of the form* $D_1 \supset (D_2 \supset G_3)$*, where* $D_1 = a \wedge (b \rightarrow c)$ *and* $D_2 = (b \rightarrow c) \wedge (a \rightarrow d)$ *are both local sets with two clauses, and* $G_3 = a$*.* ∎

In the underlying logic of the programming language $Horn^{\supset}$, well-formed formulas are built from propositional variables in Σ, using constants (*true* and *false*), classical connectives ($\neg$, $\wedge$, $\vee$, and $\rightarrow$) and the intuitionistic implication ($\supset$). Given a signature Σ, the model semantics for $Horn^{\supset}$ is given by the set of all Σ-interpretations $\underline{\mathsf{Mod}}(\Sigma) = \{I \mid I \subseteq \Sigma\}$.

The *satisfaction* relation[1], denoted by $\Vdash_{\Sigma}$ (or simply $\Vdash$ if there is no confusion about the signature), between an interpretation I and a formula φ is given below. $Horn^{\supset}$ clauses and goals are particular formulas in this logic.

Definition 1 *Let $I \in \underline{\mathsf{Mod}}(\Sigma)$ and φ be a Σ-formula. The binary* satisfaction relation $\Vdash$ *is inductively defined as follows:*

$I \nVdash false$
$I \Vdash v$ *iff* $v \in I$ *for* $v \in \Sigma$
$I \Vdash \neg\varphi$ *iff* $I \nVdash \varphi$
$I \Vdash \varphi \wedge \psi$ *iff* $I \Vdash \varphi$ *and* $I \Vdash \psi$
$I \Vdash \varphi \vee \psi$ *iff* $I \Vdash \varphi$ *or* $I \Vdash \psi$
$I \Vdash \varphi \rightarrow \psi$ *iff if* $I \Vdash \varphi$ *then* $I \Vdash \psi$
$I \Vdash \varphi \supset \psi$ *iff for all* $J \subseteq \Sigma$ *such that* $I \subseteq J$*: if* $J \Vdash \varphi$ *then* $J \Vdash \psi$

Definition 2 *Let $I \in \underline{\mathsf{Mod}}(\Sigma)$ and φ be a Σ-formula. I is a model of φ if and only if $I \Vdash \varphi$.*

Note that the satisfaction of a formula $\varphi \supset \psi$ in an interpretation I depends on the satisfaction of ψ in all the interpretations J containing I that satisfy φ. If the formula does not contain the connective $\supset$, then $\Vdash$ coincides with the satisfaction relation in classical logic $\models$.

Example 3 *Let φ be the formula $((a \wedge c) \rightarrow b) \supset (c \wedge b)$ on signature $\{a, b, c\}$. Among its eight interpretations, we have that $I \Vdash \varphi$ for $I = \{a, b, c\}$, $I = \{a, c\}$ and $I = \{b, c\}$. $I \nVdash \varphi$ for $I = \{a, b\}$, $I = \{a\}$, $I = \{b\}$, $I = \{c\}$ and $I = \emptyset$. Note, for instance, that $\{a, b\} \Vdash (a \wedge c) \rightarrow b$ and $\{a, b\} \nVdash (c \wedge b)$.* ∎

Finally we point out that once the semantic has been defined, we can justify that *true* is both a goal ($v \supset v$) and a clause ($v \rightarrow v$).

[1] Also called *forcing* relation in Kripke models for intuitionistic logic.

$\underline{\mathsf{Mod}}(\Sigma)$ is partially ordered by the inclusion relation. The satisfaction relation does not behave monotonically with respect to this relation.
For instance, $a \rightarrow b$ is satisfied in the interpretation $I = \emptyset$ but it is not satisfied in $J = \{a\}$. We say that a formula is *persistent* whenever the satisfaction relation behaves monotonically for it.

Definition 3 *A formula φ is* persistent *when for each interpretation I, if $I \Vdash \varphi$ then $J \Vdash \varphi$ for any interpretation J such that $I \subseteq J$.*

Proposition 1 *Any $v \in \Sigma$ is persistent. Any formula $\varphi \supset \psi$ is persistent. If φ and ψ are persistent then $\varphi \vee \psi$ and $\varphi \wedge \psi$ are persistent.*

Proof. For variables and formulas of the form $\varphi \supset \psi$ the property is a trivial consequence of the satisfaction relation (Definition 1). The other two cases are easily proved, by induction, using the satisfaction relation definition for $\wedge$ and $\vee$. ∎

From this proposition we obtain the two following results. The second result is a consequence of the former one and it can be proved by induction on the definition of D.

Corollary 1 *Any goal G is a persistent formula.*

Corollary 2 *For any clause D and interpretations I_1, I_2, if $I_1 \Vdash D$ and $I_2 \Vdash D$ then $I_1 \cap I_2 \Vdash D$.*

Definition 4 *Two formulas φ and ψ are* (semantically) equivalent *if both have the same meaning in each I of $\underline{\mathsf{Mod}}(\Sigma)$. In other words, if both are satisfied in the same interpretations.*

Next, we provide some examples of equivalence between goals. These results will be useful later.

Proposition 2 *G and $true \supset G$ are equivalent goals.*

Proof. $I \Vdash true \supset G \Leftrightarrow$ for all $J \supseteq I$, $J \Vdash G \Leftrightarrow I \Vdash G$. The last step uses the persistency of G. ∎

Proposition 3 *$((G_1 \rightarrow v) \wedge D) \supset G_2$ and $((D \supset G_1) \rightarrow v) \supset (D \supset G_2)$ are equivalent goals.*

Proof. Let us prove that, for every I in $\underline{\mathsf{Mod}}(\Sigma)$, $I \Vdash ((G_1 \rightarrow v) \wedge D) \supset G_2$ if and only if $I \Vdash ((D \supset G_1) \rightarrow v) \supset (D \supset G_2)$.

From left to right: Let us assume that $I \not\Vdash ((D \supset G_1) \to v) \supset (D \supset G_2)$. Then there exists J such that $J \supseteq I$, $J \Vdash (D \supset G_1) \to v$ and $J \not\Vdash D \supset G_2$. Moreover, there exists J_1 such that $J_1 \supseteq J$, $J_1 \Vdash D$ and $J_1 \not\Vdash G_2$. We distinguish two cases:

- If $v \in J$ also $v \in J_1$. Then $J_1 \Vdash (G_1 \to v) \wedge D$ and $J_1 \not\Vdash G_2$. Therefore $I \not\Vdash ((G_1 \to v) \wedge D) \supset G_2$.
- If $v \notin J$ then $J \not\Vdash D \supset G_1$. That is, there exists J_2 such that $J_2 \supseteq J$, $J_2 \Vdash D$ and $J_2 \not\Vdash G_1$. By using Corollaries 1 and 2, it is easy to prove that the interpretation $J_3 = J_1 \cap J_2$ verifies: $J_3 \Vdash D$, $J_3 \not\Vdash G_1$ and $J_3 \not\Vdash G_2$. Then $J_3 \Vdash (G_1 \to v) \wedge D$, $J_3 \not\Vdash G_2$ and $J_3 \supseteq I$. Therefore $I \not\Vdash ((G_1 \to v) \wedge D) \supset G_2$.

From right to left: Now assume that $I \not\Vdash ((G_1 \to v) \wedge D) \supset G_2$. There must exist J such that $J \supseteq I$, $J \Vdash (G_1 \to v)$, $J \Vdash D$ and $J \not\Vdash G_2$. Again two cases are distinguished:

- If $v \in J$ then trivially $J \Vdash (D \supset G_1) \to v$ and $J \not\Vdash D \supset G_2$. Therefore $I \not\Vdash ((D \supset G_1) \to v) \supset (D \supset G_2)$.
- If $v \notin J$ then $J \not\Vdash G_1$. Since $J \Vdash D$ then $J \not\Vdash D \supset G_1$ and hence $J \Vdash (D \supset G_1) \to v$. As we also have $J \not\Vdash D \supset G_2$, we conclude $I \not\Vdash ((D \supset G_1) \to v) \supset (D \supset G_2)$. ∎

3 Boolean Circuits, Boolean Formulas and Horn clauses

An n-ary Boolean function is a function $f : \{True, False\}^n \mapsto \{True, False\}$. In this section we revise from [15] some representations of Boolean functions. Namely, we give a formal definition of the syntax and semantics of Boolean circuits, and present Boolean formulas and Horn clauses as special cases of Boolean circuits. Finally, we remark some properties to be used in next sections.

3.1 The syntax and semantics of Boolean Circuits

A *Boolean circuit* over signature Σ is a graph $C = (V, E)$, where the nodes $V = \{1, 2, \ldots, n\}$ are called the *gates* of C. Graph C has a rather special structure. First, there are no cycles in the graph, so we can assume that all edges are of the form (i, j) where $i < j$. All nodes in the graph have indegree equal to $0, 1$ or 2. Also, each gate $i \in V$ has a *sort* $s(i)$ associated with it, where $s(i) \in \{true, false, \wedge, \vee, \neg\} \cup \Sigma$.
If $s(i) \in \{true, false\} \cup \Sigma$, then the indegree of i is 0, that is, i must have no incoming edges. Gates with no incoming edges are called the *inputs* of C. If $s(i) = \neg$ then i has indegree one. If $s(i) \in \{\wedge, \vee\}$, then the indegree of i

must be two. Finally, node n (the largest numbered gate in the circuit, which necessarily has no outgoing edges) is called the *output gate* of the circuit. Figure 1 shows an example of a circuit.

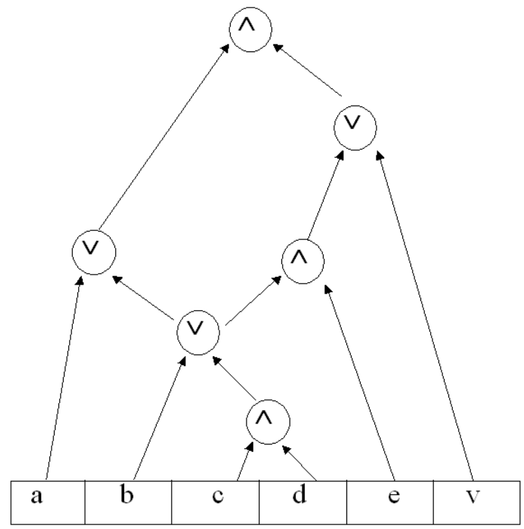

Fig. 1. Example of a circuit

Given a signature Σ, each $I \subseteq \Sigma$ can be seen as a Σ-interpretation where, for every $v \in \Sigma$, $I(v) = True$ if and only if $v \in I$. The semantics of a circuit $C = (V, E)$ specifies a truth value $I(C)$ for each interpretation $I \subseteq \Sigma$. The *truth value of gate* $i \in V$, $I(i)$, is defined by induction as follows: If $s(i) = true$ then $I(i) = True$ and similarly if $s(i) = false$ then $I(i) = False$. If $s(i) \in \Sigma$ then $I(i) = I(s(i))$. If $s(i) = \neg$ then there is a unique gate $j < i$ such that $(j, i) \in E$. By induction we know $I(j)$, and then $I(i) = True$ if and only if $I(j) = False$. If $s(i) = \vee$ then there are two edges (j, i) and (j', i) entering i. $I(i)$ is then $True$ if and only if at least one of $I(j)$, $I(j')$ is $True$. If $s(i) = \wedge$, then $I(i) = True$ if and only if both $I(j)$, $I(j')$ are $True$, where (j, i) and (j', i) are the incoming edges. Finally, the *value of the circuit*, $I(C)$, is $I(n)$, where n is the output gate.

Given a Boolean circuit C (over Σ), a Σ-interpretation I is a Σ-model of C, denoted $I \models_\Sigma C$, or $I \models C$ for short, if the value $I(C)$ is $True$. For instance, the interpretation $I = \{c, d, v\}$ is a model for the circuit in Figure 1.

A *Boolean formula* over signature Σ is built on constants ($true$, $false$) and variables in Σ, by using the connectives in $\{\wedge, \vee, \neg\}$. Each formula can be seen as a tree. That is, it is a particular case of circuit where sub-circuits (in particular variables) are not shared. In general, the possibility of sharing sub-circuits (gates with out-degree greater than one) makes circuits more economical than formulas in representing Boolean functions.

Finally any *Horn* clause $((v_1 \wedge v_2 \wedge \ldots \wedge v_n) \rightarrow v)$ is the Boolean formula $(\neg v_1 \vee \neg v_2 \vee \ldots \vee \neg v_n \vee v)$.

3.2 Notation and properties

Given two circuits $C_1 = (V_1, E_1)$ and $C_2 = (V_2, E_2)$ over the same signature Σ and given $v \in \Sigma$, the new circuit $C_1|_v^{C_2}$ is obtained by changing C_2 for v in C_1. That is, $C_1|_v^{C_2}$ is a pair (V, E) which is the result of combining C_1 and C_2 as follows: V is an adequate enumeration for the union of V_1 and V_2. The edges of the new circuit are the union of E_1 and E_2, according to such enumeration, except those outgoing edges from v in E_1 that now come out from the output gate of C_2. Figure 2 shows $C_1|_v^{C_2}$ from two given circuits C_1 and C_2. Note that

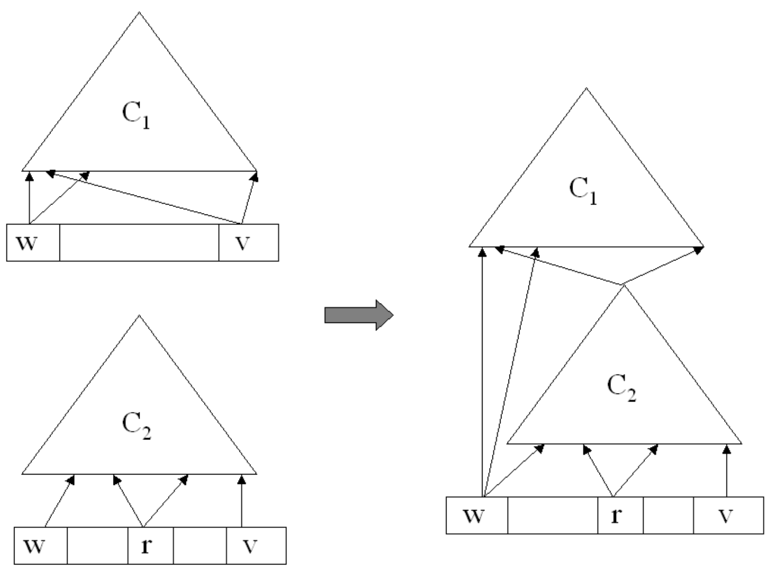

Fig. 2. $C_1|_v^{C_2}$ from C_1 and C_2

many circuits can compute the same Boolean function, but we are interested in those that have minimum size. Therefore we can assume that input gates only appear once in Boolean circuits.

A *monotone* Boolean function f is one that has the following property: If one of the inputs changes from $False$ to $True$, the value of the function cannot change from $True$ to $False$. f is monotone if and only if it can be expressed as a circuit without gates of the sort $\neg$. These are called *monotone Boolean circuits*.

Next lemmas present some properties on monotone and non-monotone Boolean circuits. From now on, we consider Boolean circuits over signature Σ.

Lemma 1 *Let C_1 be a monotone Boolean circuit, $I \subseteq \Sigma$, and $v \in \Sigma$. The following holds:*

(a) *(Monotonicity) If $I \models C_1$ then $J \models C_1$, for every $J \supseteq I$.*
(b) *If $I \models C_1$ then $I \models C_1|_v^{C_2 \vee v}$ for any Boolean circuit C_2.*

Lemma 2 *Let C be a Boolean circuit, $I \subseteq \Sigma$, and $v \in \Sigma$. The following holds:*

(a) *$I \cup \{v\} \models C$ if and only if $I \models C|_v^{true}$*
(b) *$I - \{v\} \models C$ if and only if $I \models C|_v^{false}$*

4 Translation into conjunctions of Horn clauses

Our first proposal is to simulate $Horn^{\supset}$ programs with $Horn$ clauses, therefore suitable for SLD resolution. This problem is efficiently solved in [14,16] but the original signature needs to be extended in the translation process. Similarly, in [2], a two step translation method is presented. In the first step of this method, the introduction of new modal operators is required for eliminating all intuitionistic implications ($\supset$). In the second step, modalities are eliminated by adding to all predicates an extra argument representing the modal context. This implies again to change the original signature.

If we want to translate the original program to Horn clauses maintaining the same signature, the cost of any simulation becomes exponential. This result is proved in this section. Namely, we present a particular $Horn^{\supset}$ program D for which any translation into an equivalent $Horn$ program $\widehat{D}$ yields a number of clauses that is exponential in the size of D.

Definition 5 *For each $Horn^{\supset}$ clause D over signature Σ, let $Models(D)$ be the set $\{I \subseteq \Sigma \mid I \Vdash D\}$. Let $Min(D)$ be the set $\{I \subseteq \Sigma \mid I \nVdash D$ but $J \Vdash D$, for all $J \subset I\}$. That is, $Min(D)$ contains the "minimal" interpretations not satisfying D.*

In the sequel, we intentionally consider I as a set or as a conjunction, as convenient. The set of all subsets of Σ is denoted by $\mathcal{P}(\Sigma)$.

Definition 6 *For each Horn$^{\supset}$ clause D, the Horn program $\widehat{D}$ is defined as follows:*

$$\text{For } D = v, \qquad \widehat{D} = \{v\}$$

$$\text{For } D = D_1 \wedge D_2, \quad \widehat{D} = \widehat{D_1} \cup \widehat{D_2}$$

$$\text{For } D = G \rightarrow v, \qquad \widehat{D} = \begin{cases} \emptyset & \text{if } Models(D) = \mathcal{P}(\Sigma) \\ \bigcup_{I \in Min(D)} \{I \rightarrow v\} & \text{in other case} \end{cases}$$

Example 4 *Let D be the Horn$^{\supset}$ clause $([(a \rightarrow b) \supset b] \wedge [(c \rightarrow b) \supset b]) \rightarrow a$. Obviously, all interpretations containing the variable a belong to $Models(D)$. The interpretation $I_0 = \emptyset$ also belongs to $Models(D)$. The other three interpretations $I_1 = \{b\}$ and $I_2 = \{c\}$ and $I_3 = \{b, c\}$ do not belong to $Models(D)$. Among them only I_1 and I_2 belong to $Min(D)$, since they do not satisfy D and I_0 satisfies D. Then $\widehat{D} = \{I_1 \rightarrow a\} \cup \{I_2 \rightarrow a\} = \{b \rightarrow a,\ c \rightarrow a\}$.* ■

In the following theorem we prove that D and $\widehat{D}$ are semantically equivalent, in other words, they have the same models.

Theorem 1 *For each interpretation I and each Horn$^{\supset}$ clause D it holds that $I \Vdash \widehat{D}$ if and only if $I \Vdash D$*

Proof. For $D = v$, the theorem is trivial. For $D = D_1 \wedge D_2$, it holds by induction. Let D be $G \rightarrow v$.

From left to right Let us suppose that $I \nVdash D$. Then $v \notin I$. Since $I \nVdash D$, $Min(D)$ is not empty. Therefore there exists some $J \in Min(D)$ such that $J \subseteq I$ and $J \rightarrow v$ is a clause of the program $\widehat{D}$. Then $I \Vdash J$ and $v \notin I$ imply $I \nVdash \widehat{D}$.

From right to left Let us suppose that $I \Vdash D$. If $v \in I$ then trivially $I \Vdash \widehat{D}$. If $v \notin I$ then $I \nVdash G$. Let $J \rightarrow v$ be a clause in the program $\widehat{D}$ for some $J \in Min(D)$ (if $\widehat{D}$ were empty then trivially $I \Vdash \widehat{D}$). If J were a (proper) subset of I, by persistence of goals we obtain $J \nVdash G$. But this implies $J \Vdash D$ which contradicts $J \in Min(D)$. That is, each $J \in Min(D)$ is not a subset of I and then trivially $I \Vdash J \rightarrow v$. Therefore $I \Vdash \widehat{D}$. ■

Corollary 3 *Each Horn$^{\supset}$ program P is equivalent to the Horn program $\widehat{P}$.*

Now we are going to consider a concrete *Horn*$^{\supset}$ clause D whose $\widehat{D}$ needs to have an exponential number of clauses with respect to the symbols in D.

Lemma 3 *Let D be the Horn$^{\supset}$ clause*

$$([(a_{11} \rightarrow b) \wedge (a_{12} \rightarrow b) \wedge \ldots \wedge (a_{1n} \rightarrow b)] \supset b \wedge$$

$$[(a_{21} \to b) \wedge (a_{22} \to b) \wedge \ldots \wedge (a_{2n} \to b)] \supset b \wedge$$

$$\ldots$$

$$[(a_{n1} \to b) \wedge (a_{n2} \to b) \wedge \ldots \wedge (a_{nn} \to b)] \supset b\ ) \to a$$

over signature $\Sigma = \{a_{ij} \mid i, j \in \{1, \ldots, n\}\} \cup \{b, a\}$. *Each interpretation of the form* $\{a_{1k_1}, a_{2k_2}, \ldots, a_{nk_n}\}$, *with* $k_j \in \{1, \ldots, n\}$, *belongs to* $Min(D)$.

Proof. Let I be one of such interpretations. Without loss of generality, let us suppose I to be $\{a_{11}, \ldots, a_{n1}\}$. The given clause D is $(G_1 \wedge \ldots \wedge G_n) \to a$ where each G_i is the goal $((a_{i1} \to b) \wedge \ldots \wedge (a_{in} \to b)) \supset b$. First let us prove that for each proper subset $J \subset I$, it holds that $J \Vdash D$. Since there exists some $a_{i1} \notin J$, then $J \Vdash (a_{i1} \to b) \wedge \ldots \wedge (a_{in} \to b)$ and $J \nVdash b$. Then $J \nVdash G_i$ and therefore $J \Vdash (G_1 \wedge \ldots \wedge G_n) \to a$. Now let us see that $I \nVdash D$. Since $a_{11} \in I$, for every interpretation K such that $I \subseteq K$ and $K \Vdash (a_{11} \to b) \wedge \ldots \wedge (a_{1n} \to b)$ it holds that $K \Vdash b$ and therefore $I \Vdash G_1$. Similarly, we can obtain $I \Vdash G_i$ for each $i \in \{1, \ldots, n\}$ and then, since $a \notin I$, $I \nVdash D$. ∎

By Definition 6, the *Horn* program $\widehat{D}$ obtained from the clause D given in Lemma 3 contains at least these n^n clauses:

$$\{I \to a \mid I \text{ is } \{a_{1k_1}, a_{2k_2}, \ldots, a_{nk_n}\},\ \text{with } k_j \in \{1, \ldots, n\}\} \tag{1}$$

The next result shows that this set of *Horn* clauses is non-redundant.

Lemma 4 *Any set of Horn clauses equivalent to (1) has at least* n^n *clauses.*

Proof. Denote by $I_r \to a$ the r-th clause in (1), for $1 \leq r \leq n^n$. I_r is not a model of the r-th clause in (1), but satisfies any other clause in (1). In addition, for each pair I_i, I_j with $1 \leq i \neq j \leq n^n$, the intersection $I_i \cap I_j$ is a model of (1).
Suppose that there exists a set H of *Horn* clauses equivalent to (1) whose number of clauses is smaller than n^n. There must be at least two different interpretations I_i and I_j that falsify the same clause c in H. Since we are dealing with *Horn* clauses, the interpretation $I_i \cap I_j$ falsifies c and therefore $I_i \cap I_j$ is not a model of H which is a contradiction. ∎

5 Translation into Boolean Circuits

Unlike previous section, we present here a linear transformation μ from goals into monotone Boolean circuits. Due to the fact that programs are of the form

$(G_1 \to v_1) \wedge \ldots \wedge (G_k \to v_k)$, where each G_i is a goal, the corresponding translation of programs by μ should be the (now non-monotone) Boolean circuit: $(\neg\mu(G_1) \vee v_1) \wedge \ldots \wedge (\neg\mu(G_k) \vee v_k)$.

Definition 7 *Let μ be the following function. It is defined by induction on the definition of G (on the three cases v, $G_1 \wedge G_2$, and $D \supset G$), but splitting as well the third case $D \supset G$ depending on D.*

$$\mu(G) = \begin{cases} v & \text{if } G = v & (1) \\ \mu(G_1) \wedge \mu(G_2) & \text{if } G = G_1 \wedge G_2 & (2) \\ \mu(G_2)|_v^{true} & \text{if } G = v \supset G_2 & (3) \\ \mu(G_2)|_v^{\mu(G_1)\vee v} & \text{if } G = (G_1 \to v) \supset G_2 & (4) \\ \mu(D \supset G_2)|_v^{true} & \text{if } G = (v \wedge D) \supset G_2 & (5) \\ \mu(D \supset G_2)|_v^{\mu(D\supset G_1)\vee v} & \text{if } G = ((G_1 \to v) \wedge D) \supset G_2 & (6) \end{cases}$$

Figure 3 shows the transformation of the goal $((a \wedge c) \to b) \supset (c \wedge b)$ by μ. This transformation is correct, both the goal and the obtained circuit represent the same Boolean function, and it is efficient, since it obtains a circuit whose size is linear with respect to the goal. In the next points we prove, respectively, the correctness and the efficiency of μ.

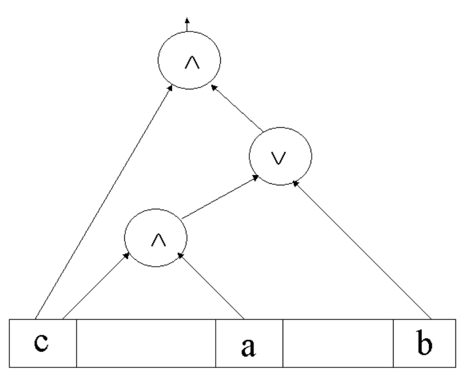

Fig. 3. Circuit for $((a \wedge c) \to b) \supset (c \wedge b)$

It is worth to note that for each goal G, the corresponding Boolean circuit $\mu(G)$ is monotone. This fact is ensured by the own definition of μ, and it is used, in the following theorem, to prove the correctness of the transformation.

Theorem 2 *Let G be any goal, for all $I \subseteq \Sigma$, $I \Vdash G$ if and only if $I \models \mu(G)$.*

Proof. By structural induction on G. Case (1) is trivial and so is case (2) by using induction on G_1 and G_2. Also note that (3) and (5) are respectively particular cases of (4) and (6) because v and $true \rightarrow v$ are equivalent clauses. Let us see cases (4) and (6) in detail.

case (4) For $G = (G_1 \rightarrow v) \supset G_2$, $\mu(G)$ is defined as $\mu(G_2)|_v^{\mu(G_1)\vee v}$.

From left to right: Let $I \Vdash (G_1 \rightarrow v) \supset G_2$.

- If $I \Vdash G_2$ then, by induction hypothesis on G_2, $I \models \mu(G_2)$ and hence, by Lemma 1(b), $I \models \mu(G_2)|_v^{\mu(G_1)\vee v}$.
- If $I \nVdash G_2$ then $I \Vdash G_1$, $I \nVdash v$ and $I \cup \{v\} \Vdash G_2$. By induction hypothesis on G_1 and G_2: $I \models \mu(G_1)$ and $I \cup \{v\} \models \mu(G_2)$. Now by Lemma 2(a), $I \models \mu(G_2)|_v^{true}$ and due to the fact that $I \models \mu(G_1) \vee v$, $I \models \mu(G_2)|_v^{\mu(G_1)\vee v}$ also holds.

From right to left: Let $I \nVdash (G_1 \rightarrow v) \supset G_2$. There must exist J such that $J \supseteq I$, $J \Vdash G_1 \rightarrow v$ and $J \nVdash G_2$. By induction hypothesis on G_2: $J \not\models \mu(G_2)$.

- If $v \in J$ then, by Lemma 2(a), $J \not\models \mu(G_2)|_v^{true}$. Then $J \not\models \mu(G_2)|_v^{\mu(G_1)\vee v}$ since $J \models \mu(G_1) \vee v$. And, by monotonicity (Lemma 1(a)), $I \not\models \mu(G_2)|_v^{\mu(G_1)\vee v}$.
- If $v \notin J$ then $J \nVdash G_1$. On the one hand, by induction hypothesis on G_1, $J \not\models \mu(G_1)$ and then $J \not\models \mu(G_1) \vee v$. On the other hand, by Lemma 2(b), $J \not\models \mu(G_2)|_v^{false}$. Then $J \not\models \mu(G_2)|_v^{\mu(G_1)\vee v}$ and as before, by monotonicity, $I \not\models \mu(G_2)|_v^{\mu(G_1)\vee v}$.

case (6) $G = ((G_1 \rightarrow v) \wedge D) \supset G_2$. Then $\mu(G) = \mu(D \supset G_2)|_v^{\mu(D\supset G_1)\vee v}$.

By Proposition 3, G is equivalent to the formula

$$G' = ((D \supset G_1) \rightarrow v) \supset (D \supset G_2)$$

which is a formula of the form $(G'_1 \rightarrow v) \supset G'_2$, for $G'_1 = D \supset G_1$ and $G'_2 = D \supset G_2$. Then, as the case (4) has been proved, $I \Vdash G$ if and only if $I \models \mu(G')$. But, by the definition of μ, $\mu(G') = \mu((G'_1 \rightarrow v) \supset G'_2) = \mu(G'_2)|_v^{\mu(G'_1)\vee v} = \mu(D \supset G_2)|_v^{\mu(D\supset G_1)\vee v} = \mu(G)$.

Then for all $I \subseteq \Sigma$, $I \Vdash G$ if and only if $I \models \mu(G)$. ∎

Now we show that the size of any monotone Boolean circuit $\mu(G)$ with respect to the size of its original goal G is linear. The size of a Boolean circuit is defined as the number of its gates. Respectively, the size of a goal is the number of its connectives $(\wedge, \rightarrow, \supset)$ and variables.

Theorem 3 *Let G be a goal. The size of $\mu(G)$ is linear in the size of G.*

Proof. The proof is made by induction on the construction of $\mu(G)$. Cases $(1), (2)$, $(3), (4)$, and (5) are trivial. Case (4) can be seen in Figure 4 which shows the transformation of $\mu(G_2)$ when v is changed by $\mu(G_1) \vee v$.

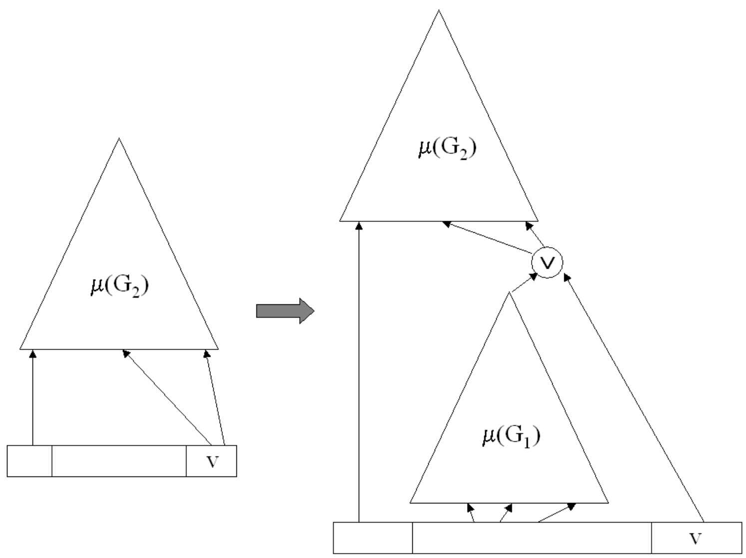

Fig. 4. Circuit for $(G_1 \rightarrow v) \supset G_2$

We study the transformation in case (6). In the easiest situation the goal to transform is the following:

$$G = ((G_{11} \rightarrow v_1) \wedge \underbrace{(G_{12} \rightarrow v_2)}_{D}) \supset G_2$$

Applying Proposition 3, this goal is equivalent to

$$(\underbrace{[(G_{12} \rightarrow v_2) \supset G_{11}]}_{G'} \rightarrow v_1) \supset \underbrace{[(G_{12} \rightarrow v_2) \supset G_2]}_{G''}$$

and by using case (4) of μ,

$$\mu(G) = \mu(G'')|_{v_1}^{\mu(G') \vee v_1}$$

$$= \mu((G_{12} \to v_2) \supset G_2)|_{v_1}^{\mu(G')\vee v_1}$$
$$= \mu(G_2)|_{v_2}^{\mu(G_{12})\vee v_2}|_{v_1}^{\mu(G')\vee v_1}$$
$$= \mu(G_2)|_{v_2}^{\mu(G_{12})\vee v_2}|_{v_1}^{\mu((G_{12}\to v_2)\supset G_{11})\vee v_1}$$
$$= \mu(G_2)|_{v_2}^{\mu(G_{12})\vee v_2}|_{v_1}^{(\mu(G_{11})|_{v_2}^{\mu(G_{12})\vee v_2})\vee v_1}$$

Let us see graphically the circuit for the goal $((G_{11} \to v_1) \wedge (G_{12} \to v_2)) \supset G_2$. Figure 5 represents three Boolean circuits $\mu(G_{11})$, $\mu(G_{12})$, and $\mu(G_2)$ and the corresponding $\mu(G)$. Since the substitution $|_{v_2}^{\mu(G_{12})\vee v_2}$ is shared by $\mu(G_2)$ and

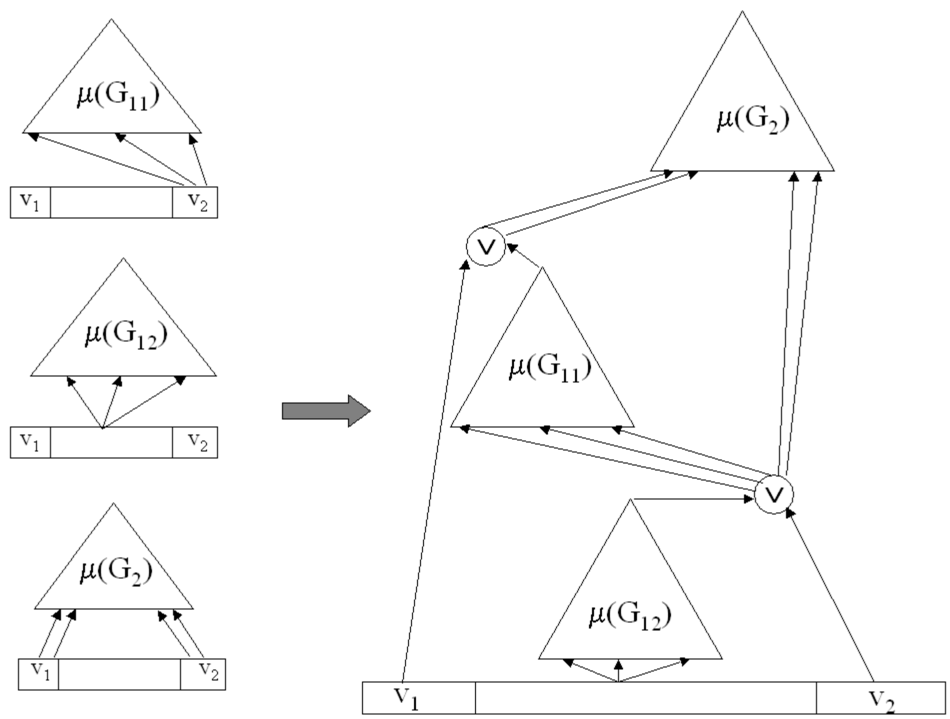

Fig. 5. Circuit for $((G_{11} \to v_1) \wedge (G_{12} \to v_2)) \supset G_2$

by $\mu(G_{11})$, the size of the circuit $\mu(G)$ is linear with respect to the size of G. This reasoning can be extended to any D in the goal $((G_1 \to v) \wedge D) \supset G_2$, since D always induces a substitution σ_D such that $\mu(G) = \mu(G_2)\sigma_D|_v^{\mu(G_1)\sigma_D \vee v}$ and σ_D is shared by $\mu(G_2)$ and by $\mu(G_1)$. ■

6 Translation into Boolean Formulas

In previous sections we have found an exponential lower bound for the problem of simulating $Horn^{\supset}$ programs with $Horn$ clauses, and a linear upper bound when the simulation is made using Boolean circuits. Our next proposal is the study of the relationship between $Horn^{\supset}$ programs and general Boolean formulas. These are represented by trees, therefore a coarse translation from circuits to formulas there exists, by repeating the shared sub-circuits so many times as necessary. However we try to find a more concise translation.

In this section we present a transformation, γ, from $Horn^{\supset}$ goals into monotone Boolean formulas. Obviously, likewise previous μ, this function γ defines the corresponding transformation from programs into Boolean formulas.

The function γ is essentially based on a well-known result due to Ingo Wegener [17] whose details we explain next: for each monotone Boolean function f, let T be a monotone Boolean formula computing f. One can choose a subtree T' (computing f') of the largest tree T. Let f'_0 respectively f'_1 be the functions computed by T if we replace T' by $False$ respectively by $True$. Thus

$$f = f'_0 \vee (f' \wedge f'_1)$$

Before formalizing our transformation, we explain how to use this idea to convert previous circuits into formulas.

Example 5 *Suppose we have a goal $G = (G_1 \rightarrow v) \supset G_2$. Using Theorem 2, G and $C = \mu(G_2)|_v^{\mu(G_1)\vee v}$ are equivalent. Moreover, as the circuit C is monotone, we can choose the subcircuit $C' = \mu(G_1)$, and the corresponding C'_0 and C'_1:*

$$C'_0 = \mu(G_2)|_v^{false\vee v} = \mu(G_2) \quad \textit{and} \quad C'_1 = \mu(G_2)|_v^{true\vee v} = \mu(G_2)|_v^{true}$$

Therefore G is equivalent to $C'_0 \vee (C' \wedge C'_1) = \mu(G_2) \vee (\mu(G_1) \wedge \mu(G_2)|_v^{true})$. ∎

The transformation γ uses this idea recursively and it is given by induction on the definition of G.

Definition 8 *Let γ be the following function:*

$$\gamma(G) = \begin{cases} v & \textit{if } G = v & (1) \\ \gamma(G_1) \wedge \gamma(G_2) & \textit{if } G = G_1 \wedge G_2 & (2) \\ \gamma(G_2)|_v^{true} & \textit{if } G = v \supset G_2 & (3) \\ \gamma(G_2) \vee (\gamma(G_1) \wedge \gamma(G_2)|_v^{true}) & \textit{if } G = (G_1 \rightarrow v) \supset G_2 & (4) \\ \gamma(D \supset G_2)|_v^{true} & \textit{if } G = (v \wedge D) \supset G_2 & (5) \\ \gamma(D \supset G_2) \vee (\gamma(D \supset G_1) \wedge & & \\ \gamma(D \supset G_2)|_v^{true}) & \textit{if } G = ((G_1 \rightarrow v) \wedge D) \supset G_2 & (6) \end{cases}$$

Theorem 4 *The transformation γ is correct, that is, for all $I \subseteq \Sigma$ and goal G, $I \Vdash G$ if and only if $I \models \gamma(G)$.*

Proof. This result is a direct consequence of the equivalence between $\gamma(G)$ and $\mu(G)$. This equivalence, $\gamma(G) \equiv \mu(G)$, can be proven by structural induction on G. Cases (1) and (2) are trivial. Cases (3) and (5) are respectively particular cases of (4) and (6) because γ produces monotone formulas, and v and $true \rightarrow v$ are equivalent clauses.
Case (4) is formally proven as follows: $\mu(G) = \mu(G_2)|_v^{\mu(G_1)\vee v}$ which is equivalent, by induction hypothesis, to the monotone formula $\gamma(G_2)|_v^{\gamma(G_1)\vee v}$. This formula has so many occurrences of $\gamma(G_1)$ as v are in $\gamma(G_2)$. By applying the Wegener's result to all these occurrences, we obtain the equivalent formula $\gamma(G_2) \vee (\gamma(G_1) \wedge \gamma(G_2)|_v^{true})$ which is the definition of $\gamma(G)$.
Finally, case (6) can be reduced to case (4) by using Proposition 3. ∎

Example 6 *The application of γ to the goal $((a \wedge c) \rightarrow b) \supset (c \wedge b)$ produces the Boolean formula $(c \wedge b) \vee ((a \wedge c) \wedge (c \wedge true))$.* ∎

It should be pointed out that γ allows us to obtain Boolean formulas with small sizes. In general, it is more efficient applying directly γ than first computing μ, and then translating it into a Boolean formula. The following example shows this fact.

Example 7 *Let G be the goal $(G_1 \rightarrow c) \supset G_2$, where G_1 and G_2 are the following goals*

$$\begin{array}{l} G_1 = ((a_{11} \rightarrow b) \supset b) \wedge ((a_{12} \rightarrow b) \supset b) \wedge \ldots \wedge ((a_{1n} \rightarrow b) \supset b) \\ G_2 = ((a_{21} \rightarrow c) \supset c) \wedge ((a_{22} \rightarrow c) \supset c) \wedge \ldots \wedge ((a_{2n} \rightarrow c) \supset c) \end{array}$$

The application of μ to G produces the Boolean circuit

$$\mu(G) = \mu(G_2)|_c^{\mu(G_1)\vee c}$$

Since for each i, with $1 \leq i \leq n$, $\mu((a_{2i} \rightarrow c) \supset c) = c|_c^{a_{2i}\vee c} = a_{2i} \vee c$, then $\mu(G_2) = (a_{21} \vee c) \wedge (a_{22} \vee c) \wedge \ldots \wedge (a_{2n} \vee c)$ and in a symmetrical manner $\mu(G_1) = (a_{11} \vee b) \wedge (a_{12} \vee b) \wedge \ldots \wedge (a_{1n} \vee b)$.

Now, the formula directly obtained from the circuit $\mu(G)$ is

$$\begin{array}{ll} (a_{21} \vee [(a_{11} \vee b) \wedge (a_{12} \vee b) \wedge \ldots \wedge (a_{1n} \vee b)] \vee c) & \wedge \\ (a_{22} \vee [(a_{11} \vee b) \wedge (a_{12} \vee b) \wedge \ldots \wedge (a_{1n} \vee b)] \vee c) & \wedge \\ \qquad \ldots & \wedge \\ (a_{2n} \vee [(a_{11} \vee b) \wedge (a_{12} \vee b) \wedge \ldots \wedge (a_{1n} \vee b)] \vee c) & \end{array}$$

However, the application of γ to G produces the Boolean formula

$$\gamma(G) = \gamma(G_2) \vee (\gamma(G_1) \wedge \gamma(G_2)|_c^{true})$$

$$\begin{aligned} \textit{where } \gamma(G_2) &= ((c \vee a_{21}) \wedge (c \vee a_{22}) \wedge \ldots \wedge (c \vee a_{2n})) \\ \gamma(G_1) &= ((b \vee a_{11}) \wedge (b \vee a_{12}) \wedge \ldots \wedge (b \vee a_{1n})) \\ \gamma(G_2)|_c^{true} &= ((true \vee a_{21}) \wedge (true \vee a_{22}) \wedge \ldots \wedge (true \vee a_{2n})) = true \end{aligned}$$

then $\gamma(G) = ((c \vee a_{21}) \wedge \ldots \wedge (c \vee a_{2n})) \vee ((b \vee a_{11}) \wedge \ldots \wedge (b \vee a_{1n}))$

We have used simplification rules as $(true \wedge \varphi) = \varphi$ or $(true \vee \varphi) = true$. However, even without using them the size of $\gamma(G)$ is smaller than the size of the previous formula obtained from $\mu(G)$. ∎

Although γ works well, it does not ensure that the size of the obtained formula always is bounded by a polynomial in the size of the input. In fact, even though applying natural simplification rules, the size of the obtained formulas appreciably decrease, we have not found a systematic method that works efficiently. Moreover, we have not found either a super-polynomial lower bound for this problem. This is a difficult task as we will see in the next section.

7 Conclusions and Open Problems

We have studied three possible representations of $Horn^{\supset}$ programs maintaining the signature.

The main result presented is a linear transformation from $Horn^{\supset}$ programs into Boolean circuits, which preserves the semantic equivalence between the original program and its translation. Since the representation of Boolean functions by circuits is well established, this translation allows us to work with $Horn^{\supset}$ clauses in an easy and compact way.

In addition, we have shown that any possible transformation of $Horn^{\supset}$ programs into $Horn$ clauses requires an exponential number of clauses. Therefore, the first language is exponentially more succinct than the second representation.

Finally, we have given a procedure that constructs a Boolean formula from a $Horn^{\supset}$ program. Unfortunately, this method is not efficient but we have not been able to find a super-polynomial lower bound. Therefore, the problem of whether there exists a polynomial-size translation from $Horn^{\supset}$ programs into general Boolean formulas remains open. In fact, it turns out that this is a deep question, related to whether *all efficient computation can be parallelized*. On the one hand, if we were able to find a super-polynomial lower bound for the problem of transforming Horn$^{\supset}$ programs into formulas, then we would obtain that circuits cannot be simulated by formulas with only a polynomial cost, and therefore that $\mathbf{P} \neq \mathbf{NC}_1$ [15]. On the other hand, if we were able to find a polynomial upper bound for the problem, then we would obtain a subclass of *nontrivial* circuits that can be converted into equivalent Boolean formulas with only polynomial increase in its size. Although the latter question seems easier to deal with, we think it is a non-trivial task.

References

[1] Arruabarrena R., Lucio P. and Navarro M. *A Strong Logic Programming View for Static Embedded Implications*. In: Proc. of FOSSACS'99, Springer-Verlag Lect. Notes in Comput. Sciences 1578: 56–72 (1999).

[2] Baldoni M., Giordano L. and Martelli A. *Translating a Modal Language with Embedded Implication into Horn Clause Logic*. In: Proc. of 5 Int. Workshop of Extensions of Logic Programming, ELP'96, Springer-Verlag Lect. Notes in Comput. Sciences 1050 (1996).

[3] Bugliesi M., Lamma E. and Mello P. *Modularity in Logic Programming*. Journal of Logic Programming, (19-20): 443–502, (1994).

[4] Gabbay D. M. *N-Prolog: An Extension of Prolog with Hypothetical Implications. II. Logical Foundations and Negation as Failure*. Journal of Logic Programming 2(4): 251–283 (1985).

[5] Gaintzarain J., Hermo M. and Navarro M. *Goals in the Propositional Horn$^{\supset}$ Language are Monotone Boolean Circuits*. In: Proc. of MFCS'2005, Springer-Verlag Lect. Notes in Comput. Sciences 3618: 376–386 (2005).

[6] Giordano L. and Martelli A. *Structuring Logic Programs: A Modal Approach*. Journal of Logic Programming 21: 59–94 (1994).

[7] Giordano L., Martelli A. and Rossi G. *Extending Horn Clause Logic with Implication Goals*. Theoretical Computer Science 95: 43–74, (1992).

[8] Lucio P. *Structured Sequent Calculi for Combining Intuitionistic and Classical First-Order Logic*. In: Proc. of FroCoSS'2000, Springer-Verlag Lect. Notes in Artificial Intelligence 1794: 88–104 (2000).

[9] Meseguer J. *Multiparadigm Logic Programming.* In: Proc. of ALP'92, Springer-Verlag Lect. Notes in Comput. Sciences 632: 158–200, (1992).

[10] Miller D. *A Logical Analysis of Modules in Logic Programming.* In: Journal of Logic Programming 6: 79–108, (1989).

[11] Miller D., Nadathur G., Pfenning F. and Scedrov A. *Uniform Proofs as a Foundation for Logic Programming.* Annals of Pure and App. Logic 51: 125–157, (1991).

[12] Monteiro L. and Porto A. *Contextual Logic Programming.* In: Proc. 6th International Conf. on Logic Programming 284–299, (1989).

[13] Moscowitz Y. and Shapiro E. *Lexical Logic Programs.* In: Proc. 8th International Conf. on Logic Programming 349–363, (1991).

[14] Navarro M. *From Modular Horn Programs to Flat Ones: a Formal Proof for the Propositional Case.* In: Proc. of ISIICT 2004 (Int. Symp. on Innovation in Information and Communication Technology), Amman, Jordan. April 2004. Technical Report UPV-EHU/ LSI/ TR 01-2004.

[15] Papadimitriou Christos H. *Computational Complexity.* Addison-Wesley Publishing Company, Inc. (1995).

[16] Pasarella E., Orejas F., Pino E. and Navarro M. *A Transformational Semantics of Static Embedded Implications.* In: Proc. of LOPSTR'05, Springer-Verlag Lect. Notes in Comput. Sciences 3901, pp. 113-146, (2006).

[17] Wegener I. *Relating Monotone Formula Size and Monotone Depth of Boolean Functions.* In: Information Processing Letters 16: 41–42, (1983).