Hybrid Probabilistic Programs: Algorithms and Complexity *

Michael I. Dekhtyar
Tver State University
Michael.Dekhtyar@tversu.ru

Alex Dekhtyar
University of Maryland
dekhtyar@cs.umd.edu

V.S. Subrahmanian
University of Maryland
vs@cs.umd.edu

Abstract

Hybrid Probabilistic Programs (HPPs) are logic programs that allow the programmer to explicitly encode his knowledge of the dependencies between events being described in the program. In this paper, we classify HPPs into three classes called HPP_1, HPP_2 and $HPP_r, r \geq 3$. For these classes, we provide three types of results for HPPs. First, we develop algorithms to compute the set of all ground consequences of an HPP. Then we provide algorithms and complexity results for the problems of entailment ("Given an HPP P and a query Q as input, is Q a logical consequence of P?") and consistency ("Given an HPP P as input, is P consistent?"). Our results provide a fine characterization of when polynomial algorithms exist for the above problems, and when these problems become intractable.

1 Introduction

Computing the probability of a complex event from the probability of the primitive events constituting it depends upon the dependences (if any) known to exist between the events being composed. For example, consider two events e_1, e_2. The probability, $\mathbf{P}(e_1 \wedge e_2)$ of the occurrence of both is events is 0 if the events are mutually exclusive. However, if the events are independent, then $\mathbf{P}(e_1 \wedge e_2) = \mathbf{P}(e_1) \times \mathbf{P}(e_2)$. If we are ignorant of the relationship between these two events, then, as stated by Boole[1], the best we can say about $\mathbf{P}(e_1 \wedge e_2)$ is that it lies in the interval $[\max(0, \mathbf{P}(e_1) + \mathbf{P}(e_2) - 1), \min(\mathbf{P}(e_1), \mathbf{P}(e_2)]$.

In short, computing the probability of a complex event depends fundamentally upon our knowledge about the dependences between the events involved. In [2] we proposed a language called *Hybrid Probabilistic (Logic) Programs* (or HPPs, for short), that extended logic programs to deal with diverse types of probabilistic dependencies, and we defined the semantics of such a language. HPPs build upon the idea of an annotated logic program introduced in [29], and studied extensively by many researchers over the years [8, 18, 16, 11, 10] In this technical note, we make two classes of contributions.

1. First, we study the complexity of a variety of problems related to the semantics of HPPs. In particular, we show that the complexity of the entailment problem (answers to queries to HPPs) is polynomial for HPPs with atomic heads of rules, and in many cases for HPPs with at most two atoms in the heads. However, when formulas of size three or more are allowed in the heads of the rules, the complexity of query processing becomes NP-complete. We establish some other complexity results for related problems, such as checking the consistency of an HPP.

2. Second, we propose a proof system HGR_P for HPPs that may be used for query processing. This is a Hilbert-style proof system and it is shown to be sound and complete. We show that proofs in HGR_P are polynomially bounded in size (this is consistent with the preceding NP-completeness result because the search space may involve exponentially many derivations each of polynomially bounded length). This is an interesting and counterintuitive result — it says that (the answers to) all queries to HPPs have at least one polynomial explanation.

*The work of the first author had been partially supported by Russian Fundamental Studies Foundation (Grants 97-01-00973). The other authors were supported by the Army Research Office under Grants DAAH-04-95-10174, DAAH-04-96-10297, and DAAH04-96-1-0398, by the Army Research Laboratory under contract number DAAL01-97-K0135, by an NSF Young Investigator award IRI-93-57756, and by a TASC/DARPA grant J09301S98061.

Section 2 recapitulates the syntax and semantics of HPPs as described in [2]. In Section 3, we describe results on the computation complexity of HPPs. Section 4 introduces the proof system HGR_P and shows it is sound and complete, while Section 5 presents results showing the proofs in HGR_P are polynomially bounded.

2 Background

The aim of this section is to describe the syntax and semantics of HPPs — the content of this section is not new and overviews results in [2]. HPPs are based on an abstract class of functions called *probabilistic strategies*. Associated with each such strategy s, we can introduce a new "conjunction like" connective $\wedge_s$ and a new "disjunction like" connective, $\vee_s$, which may then be used to define a syntax for HPPs.

2.1 Probabilistic Strategies (p-strategies)

It is well-known that the probability of a compound event may be an *interval*, rather than a point even if point probabilities are known for the primitive events involved. This was first shown by Boole[1] in 1854. Thus, p-strategies will be defined on intervals – points, in any case, are special cases of intervals.

Let $\mathcal{C}[0,1]$ denote the set of all closed intervals of $[0,1]$. If $[a,b] \in \mathcal{C}[0,1]$, $[c,d] \in \mathcal{C}[0,1]$ then we write $[a,b] \leq_t [c,d]$ if $a \leq c$ and $b \leq d$.

Definition 1 *A probabilistic strategy (p-strategy) is a pair of functions:* $\rho = \langle c, md \rangle$, *such that:*

1. $c : \mathcal{C}[0,1] \times \mathcal{C}[0,1] \longrightarrow \mathcal{C}[0,1]$ *is called a probabilistic composition function if it satisfies the following axioms:*
   - *(a)* Commutativity : $c([a_1,b_1],[a_2,b_2]) = c([a_2,b_2],[a_1,b_1])$
   - *(b)* Associativity : $c(c([a_1,b_1],[a_2,b_2]),[a_3,b_3]) = c([a_1,b_1],c([a_2,b_2],[a_3,b_3]))$
   - *(c)* Monotonicity : $c([a_1,b_1],[a_2,b_2]) \subseteq c([a_3,b_3],[a_2,b_2])$ if $[a_1,b_1] \subseteq [a_3,b_3]$
   - *(d)* Separation:*there exist two functions* c^1 *and* c^2 *such that* $c([a_1,b_1],[a_2,b_2]) = (c^1(a_1,a_2), c^2(b_1,b_2))$
2. $md : \mathcal{C}[0,1] \longrightarrow \mathcal{C}[0,1]$ *is called a maximal interval function.*

Intuitively, a composition function determines, given the probability ranges of two events, the probability range of their (either and- or or-composition). A max-interval function md returns the best estimate for the probability of simple event given the probability of a compound event. For the discussion on why we specify max-interval functions as above see [2].

The two combinations of events we plan on dealing with are conjunctions of events and disjunction of events. Among all possible p-strategies, we identify *conjunctive* and *disjunctive* p-strategies, which will handle the computation of probabilities of these two combinations respectively.

Since composition functions are both commutative and associative, all terms constructed by applications of composition function c to $n \geq 2$ intervals $\mu_1 = [a_1,b_1], \ldots, \mu_n = [a_n,b_n]$ will have the same value which we will denote as $c(\mu_1,\ldots,\mu_n)$ with it's lower bound $c^1(a_1,...,a_n)$ and upper bound $c^2(b_1,...,b_n)$. For technical reasons it's convenient in the case $n=1$ for any $\mu = [a,b]$ to set $c(\mu) = \mu$, $c^1(a) = a$ and $c^2(b) = b$.

Definition 2 Conjunctive and Disjunctive p-strategies

A *p-strategy* $< c, d >$ *is called* conjunctive (disjunctive) if it satisfies the following axioms:

Axiom	*Conjunctive*	*Disjunctive*
Bottomline	$c([a_1,b_1],[a_2,b_2]) \leq [min(a_1,a_2), min(b_1,b_2)]$	$c([a_1,b_1],[a_2,b_2]) \geq [max(a_1,a_2), max(b_1,b_2)]$
Identity	$c([a,b],[1,1]) = [a,b]$	$c([a,b],[0,0]) = [a,b]$
Annihilator	$c([a,b],[0,0]) = [0,0]$	$c([a,b],[1,1]) = [1,1]$
Max.Interval	$md([a,b]) = [a,1]$	$md([a,b]) = [0,b]$

For a more complete discussion of the axioms we refer the reader to [2].

Example 1 *Below are some examples of p-strategies. We provide definitions of composition functions only, since max-interval functions are defined uniquely by the type of p-strategy [2].*

- inc:
  p-strategies for independence assumption
  Conjunctive: $c_{inc}([a,b],[c,d]) = [ac, bd]$
  Disjunctive: $c_{ind}([a,b],[c,d]) = [a+c-ac, b+d-bd]$.
- igc: p-strategies for ignorance assumption
  Conjunctive: $c_{igc}([a,b],[c,d]) = [\max(0, a+c-1), \min(b,d)]$
  Disjunctive: $c_{igd}([a,b],[c,d]) = [\max(a,c), \min(1, b+d)]$.
- pcc: p-strategies for positive correlation assumption
  Conjunctive: $c_{pcc}([a,b],[c,d]) = [\min(a,c), \min(b,d)]$
  Disjunctive: $c_{pcd}([a,b],[c,d]) = [\max(a,c), \max(b,d)]$.
- p-strategies for negative correlation assumption
  Disjunctive: $c_{ncd}([a,b],[c,d]) = [\min(a+c,1), \min(b+d,1)]$

Since in this paper we investigate complexity of some algorithmic problems related to HPPs all the interval bounds are supposed to be rational numbers in interval $[0,1]$ represented for example by finite binary numbers. It is easy to see that composition functions c_{inc}, c_{igc} and c_{igd} specified by Example 1 as well as a number of other examples of strategies presented in [2] are computable in polynomial time with respect to the length of their arguments. To make our results independent of complexity of particular strategies we will suppose bellow that the computation of a composition function for each p-strategy is provided by some oracle, or in other words can be done in constant time. This way, all bounds obtained in this paper should be multiplied by the complexity of computing the composition. However, for composition functions computable in polynomial time such multiplication will not result in the change in the polynomiality (deterministic or nondeterministic) of the bounds.

2.2 Syntax of hp-programs

Let L be a language which has predicate, variable and constant symbols, but has no function symbols. Let B_L be the set of all ground atoms of L.

In hybrid probabilistic programs, we assume the existence of an arbitrary, but fixed set of conjunctive and disjunctive p-strategies. The programmer may augment this set with new strategies when s/he needs new ones for their application. The following definition says that each strategy has an associated conjunction operator, and a disjunction operator.

Definition 3 *Let $\mathcal{CONJ}$ be a finite set of conjunctive p-strategies and $\mathcal{DISJ}$ be a finite set of disjunctive p-strategies. Let $\mathcal{S}$ denote $\mathcal{CONJ} \cup \mathcal{DISJ}$ If $\rho \in \mathcal{CONJ}$ then a connective $\wedge_\rho$ is called an* ρ-annotated conjunction *If $\rho \in \mathcal{DISJ}$, connective $\vee_\rho$ is called an* ρ-annotated disjunction

Hybrid basic formulas, defined below, are either conjunctions of atoms, or disjunctions of atoms (but not mixes of both) w.r.t. a single connective.

Definition 4 *Let ρ be a* conjunctive p-strategy, *ρ' be a* disjunctive p-strategy *and $A_1, \ldots, A_k$ be distinct atoms. Then $A_1 \wedge_\rho \ldots \wedge_\rho A_k$ and $A_1 \vee_{\rho'} A_2 \ldots \vee_{\rho'} A_k$ are called* hybrid basic formulas. *Suppose $bf_\rho(B_L)$ denotes the set of all ground hybrid basic formulas for the $\vee_\rho$ and $\wedge_\rho$ connectives. Let $bf_{\mathcal{S}}(B_L) = \cup_{\rho \in \mathcal{S}} bf_\rho(B_L)$. Similarly, $bf_{\mathcal{CONJ}} = \cup_{\rho \in \mathcal{CONJ}} bf_\rho(B_L)$ and $bf_{\mathcal{DISJ}} = \cup_{\rho \in \mathcal{DISJ}} bf_\rho(B_L)$.*

We define a notion of *annotations* inductively as follows: (1) Any real number or variable over real numbers is an annotation term. (2) If f is an interpreted function over the reals of arity k and $t_1, \ldots, t_k$ are annotation terms, then $f(t_1, \ldots, t_k)$ is an annotation term. An *annotation* is pair $[at_1, at_2]$ where at_1, at_2 are annotation terms. Thus, for instance, $[0.5, 0.6], [0..5, \frac{V+1}{2}]$ are both annotations.

Definition 5 *A hybrid probabilistic annotated basic formula (hp-annotated basic formula) is an expression of the form $B : \mu$ where B is a hybrid basic formula and μ is an annotation.*

Intuitively, if $B : \mu$ is ground, this says that the probability of B being true is in the interval μ.

Definition 6 *Let $B_0, B_1, \ldots, B_k$ be hybrid basic formulas. Let $\mu_0, \mu_1, \ldots, \mu_k$ be annotations. A hybrid probabilistic clause (hp-clause) is a construction of the form:*

$$B_0 : \mu_0 \leftarrow B_1 : \mu_1 \wedge \ldots \wedge B_k : \mu_k$$

Informally speaking, the above rule is read: "If the probability of B_1 falls in the interval μ_1 and $\cdots$ the probability of B_k falls within the interval μ_k, then the probability of B_0 falls within the interval μ_0. Note that it is entirely possible that B_i uses a connective $\wedge_\rho$ corresponding to a particular (conjunctive) p-strategy, while B_j may use a connective $\vee_{\rho'}$ corresponding to some other disjunctive p-strategy. HPPs allow mixing and matching of different kinds of p-strategies, both in the B_i's in the body of a rule, as well as in B_0 - the head of a rule.

Definition 7 *A hybrid probabilistic program (hp-program) over set $\mathcal{S}$ of p-strategies is a finite set of hp-clauses involving only connectives from $\mathcal{S}$.*

An hp-program is *ground* iff its every clause is ground, i.e. all its clauses do not contain neither variables nor variable annotations.

2.3 Fixpoint and Model Theory for hp-programs

In this chapter, we briefly describe the model theory underlying HPPs. [2] contains a more comprehensive description. Before proceeding further we first introduce some notation for "splitting" a complex formula into two parts.

Definition 8 *Let $F = F_1 *_\rho \ldots *_\rho F_n$, $G = G_1 *_\rho \ldots *_\rho G_k$, $H = H_1 *_\rho \ldots *_\rho H_m$ where $* \in \{\wedge, \vee\}$. We will write $G \oplus_\rho H = F$ (or $G \oplus H$ if the p-strategy ρ is irrelevant)* **iff:**

1. $\{G_1, \ldots, G_k\} \cup \{H_1, \ldots, H_m\} = \{F_1, \ldots, F_n\}$ *and*
2. $\{G_1, \ldots, G_k\} \cap \{H_1, \ldots, H_m\} = \emptyset$.
3. $k > 0$ *and* $m > 0$.

The analog of an Herbrand interpretation in classical logic programs is what we call a *hybrid formula function*.

Definition 9 *A function $h : bf_{\mathcal{S}}(B_L) \longrightarrow \mathcal{C}[0,1]$, is called a* hybrid formula function *iff it satisfies the following three conditions:*

1. Commutativity. *If $F = G_1 \oplus_\rho G_2$ then $h(F) = h(G_1 *_\rho G_2)$.*
2. Composition. *If $F = G_1 \oplus_\rho G_2$ then $h(F) \subseteq c_\rho(h(G_1), h(G_2))$.*
3. Decomposition. *For any basic formula F, $h(F) \subseteq md_\rho(h(F *_\rho G))$ for all $\rho \in \mathcal{S}$ and $G \in bf_{\mathcal{S}}(B_L)$.*

From the first condition it follows that $h(F) = h(F')$ for any F and F' which are permutations of one another. This property of models allows us, in fact not to distinguish between the formulas and the sets of atoms they are composed of together with a strategy attached. Second condition states that the probability of a complex formula is bounded by the probabilities of its subformulas. Conversely, the third condition bounds the probability of a subformula by the probability of a formula it is a part of.

We are now in a position to specify what it means for a hybrid basic formula function to satisfy a formula.

Definition 10 Satisfaction. Let h be a hybrid basic formula function, $F \in bf_{\mathcal{S}}(B_L)$, $\mu \in \mathcal{C}[0,1]$. We say that

- $h \models F : \mu$ **iff** $h(F) \subseteq \mu$.
- $h \models F_1 : \mu_1 \wedge \ldots \wedge F_n : \mu_n$ **iff** $(\forall\, 1 \leq j \leq n) h \models F_j : \mu_j$.
- $h \models F : \mu \longleftarrow F_1 : \mu_1 \wedge \ldots \wedge F_n : \mu_n$ **iff** either $h \models F : \mu$ or $h \not\models F_1 : \mu_1 \wedge \ldots \wedge F_n : \mu_n$.
- $h \models (\exists x)(F : \mu)$ **iff** $h \models F(t/x) : \mu$ *for some ground term* t.
- $h \models (\forall x)(F : \mu)$ **iff** $h \models F(t/x) : \mu$ *for every ground term* t.

A formula function h is called a model *of an hp-program P ($h \models P$)* **iff** $(h \models C)$ *for all clauses* $C \in P$.

As usual, we say that $F : \mu$ is a *consequence* of P iff for every model h of P, it is the case that $h(F) \subseteq \mu$.

It is possible for a hybrid formula function h to assign $\emptyset$ to some formula. When $h(F) = \emptyset$, h is "saying" that F's probability lies in the empty set. This corresponds to an inconsistency because, by definition, nothing is in the empty set.

Definition 11 *Formula function h is called* fully defined *iff* $\forall(F \in bf_{\mathcal{S}}(B_L))(h(F) \neq \emptyset)$.

Now we introduce the fixpoint semantics for the hp-programs. Operator S_P is a preliminary operator, restricted only to the clauses which have the same head as the argument. It is then extended to full fixpoint operator T_P

Definition 12 *Let P be* a hybrid probabilistic program. *Operator $S_P : \mathcal{HFF} \longrightarrow \mathcal{HFF}$ is defined as follows (where F is a basic formula): $S_P(h)(F) = \cap M$ where $M = \{\mu\sigma | F : \mu \longleftarrow F_1 : \mu_1 \wedge \ldots \wedge F_n : \mu_n$* is a ground instance of some clause in P ; *σ is a ground substitution of annotation variables and $(\forall j \leq n) h(F_j) \subseteq \mu_j\sigma\}$ if $M = \emptyset$* $S_P(h)(F) = [0,1]$.

We use the definition of S_P to define the immediate consequence operator T_P.

Definition 13 *Let P be* a hybrid probabilistic program. We inductively define operator $T_P : \mathcal{HFF} \longrightarrow \mathcal{HFF}$ as follows:

1. *Let F be an* <u>*atomic*</u> *formula.*
   - *if $S_P(h)(F) = \emptyset$ then $T_P(h)(F) = \emptyset$.*
   - *if $S_P(h)(F) \neq \emptyset$, then let $M = \{\langle\mu\sigma, \rho\rangle | (F \oplus_\rho G) : \mu \longleftarrow F_1 : \mu_1 \wedge \ldots \wedge F_n : \mu_n$* where σ *is a ground substitution for the annotation varables and $i \in \mathcal{S}$ and $(\forall j \leq n) h(F_j) \subseteq \mu_j\sigma\}$.* We define $T_P(h)(F) = (\cap\{md_\rho(\mu\sigma) | \langle\mu\sigma, i\rangle \in M\}) \cap S_P(h)(F)$.
2. *(F <u>not atomic</u>) Let $F = F_1 *_\rho \ldots *_\rho F_n$.*
   Let $M' = \{\langle\mu\sigma, \rho\rangle | D_1 *_\rho \ldots *_\rho D_k : \mu \longleftarrow E_1 : \mu_1 \wedge \ldots E_m : \mu_m \in ground(P)$;
   $(\forall 1 \leq j \leq m) h(E_j) \subseteq \mu_j$; $\{F_1, \ldots, F_n\} \subset \{D_1, \ldots D_k\}, n < k\}$
   Then:
   $$T_P(h)(F) = S_P(h)(F) \cap (\cap\{c_\rho(T_P(h)(G), T_P(h)(H)) | G \oplus_\rho H = F\}) \cap (\cap\{md_\rho(\mu\sigma) | \langle\mu\sigma, i\rangle \in M'\})$$

In [2] it was shown that both S_P and T_P are monotonic if the annotations of the atoms in P are constant.

Definition 14

1. *$T_P^0 = h_\perp$ where $\perp$ is the atomic function that assigns $[0,1]$ to all ground atoms A.*
2. *$T_P^\alpha = T_P(T_P^{\alpha-1})$ where α is a successor ordinal whose predecessor is denoted by $\alpha - 1$.*
3. *$T_P^\gamma = \sqcup\{T_P^\alpha | \alpha < \gamma\}$, where γ is limit ordinal.*

In [2], we showed that given any HPP P, we could associate with P, an operator T_P, that maps hybrid formula functions to hybrid formula functions. We recapitulate its definition here.

Theorem 1 *Let P be any hp-program. Then:*

1. *h is a model of P iff $T_P(h) \leq h$.*
2. *P has a model iff $lfp(T_P)$ is fully defined.*
3. *If $lfp(T_P)$ is fully defined, then it is the least model of P, and $F : \mu$ is a logical consequence of P iff $lfp(T_P) \subseteq \mu$.*

The above results ties together, the fixpoint theory and the model theoretical characterizations of hp-programs, regardless of which p-strategies occur in the hp-program being considered.

In what follows we will consider only ground hp-programs. It is clear that for any such program P, the least fixpoint of its T_P operator, $lfp(T_P)$ is achieved in a finite number of iterations, i.e., at least, $lfp(T_P) = T_P^{\omega}$. For brevity we will denote $lfp(T_P)$ as h_P.

3 Algorithms and Complexity of ground HPPs

In this section, we will develop algorithms, and associated complexity results, for three kinds of HPP problems.

1. **Computation:** We will first study (Section 3.1) the problem of computing the set of all ground basic formulas that are logical consequences of an HPP P.
2. **Entailment:** Second (in Section 3.2), given a query Q and an HPP P as input, we will study the complexity of checking if Q is a logical consequence of P.
3. **Consistency:** Finally, in Section 3.3, given as input, a HPP P, we will examine the complexity of checking if P is consistent.

Obviously, these three problems are closely related to one another.

3.1 Complexity of model computation

Let P be a **ground** hp-program over the set of atoms $\{A_1, ..., A_N\}$.

Definition 15 *For a clause $C \equiv F : \mu \leftarrow F_1 : \mu_1 *_\rho \ldots *_\rho F_n : \mu_n \in P$ let*

$$act_P(C) = min\{i | T_P^i(F_j) \subseteq \mu_j \ (1 \leq j \leq n)\}.$$

In other words, $act_P(C)$ denotes the minimal step i at which C becomes active (i.e., just after all formulas in the body of C are satisfied). Let $Fired_P(i)$ denote the set $\{C | act_P(C) = i\}$.

The following result provides us with an exact bound on the number of steps required to compute the fixpoint of the T_P operator.

Lemma 1
1. *If for some i we have $Fired_P(i) = Fired_P(i+1) = \emptyset$, then $lfp(T_P) = T_P^{i+1}(h_\perp)$*
2. *If P consists of m ground clauses then $lfp(T_P) = h_P = T_P^{2m}$.*

The following example demonstrates that the bound of $2m$ in Lemma 1.2 cannot be decreased.

Example 2 *Let igc and inc be the ignorance and independence strategies defined in Example 1. Let us consider the following program P:*

$$
\begin{array}{rrcl}
r_0: & a \wedge_{igc} b_0 : [0.5,1] & \leftarrow & \\
r_1: & b_1 : [0.5,1] & \leftarrow & a \wedge_{inc} b_0 : [0.5,1] \\
r_2: & b_2 : [0.5,1] & \leftarrow & b_0 \wedge_{inc} b_1 : [0.5^2,1] \\
 & & \ldots & \\
r_{i+1}: & b_{i+1} : [0.5,1] & \leftarrow & b_0 \wedge_{inc} b_1 \wedge_{inc} \ldots \wedge_{inc} b_i : [0.5^{i+1},1] \\
 & & \ldots & \\
r_n: & b_n : [0.5,1] & \leftarrow & b_0 \wedge_{inc} b_1 \wedge_{inc} \ldots \wedge_{inc} b_{n-1} : [0.5^n,1]
\end{array}
$$

P consists of $n+1$ rules $r_0, r_1, ...r_n$. It easy to check that $act_P(r_i) = 2i+1$ for every $0 \leq i \leq n$. Then

$T_P^1(a) = T_P^1(b_0) = [0.5;1]$, and for $i > 1$ $T_P^{2i+1}(b_i) = [0.5,1]$ and at the step $2(i+1)$ we have $T_P^{2(i+1)}(b_0 \wedge_{ind} b_1 \wedge_{ind} \ldots \wedge_{ind} b_i) = [0.5^{i+1}, 1]$. Hence, it takes $2(n+1)$ steps to get h_P. ■

The following lemma shows how one can compute T_P^{i+1} knowing T_P^i.

Lemma 2 *Let P be an hp-program.*

1. *For any <u>atomic</u> formula F*

   $$T_P^{i+1}(F) = T_P^i(F) \cap (\cap\{md_\rho(\mu) | \langle \mu, \rho \rangle \in M^{i+1}\}) \cap (\cap\{\mu | F : \mu \longleftarrow Body \in Fired_P(i)\})$$

   where $M^{i+1} = \{\langle \mu, \rho \rangle | (F *_\rho G) : \mu \longleftarrow Body \in Fired_P(i)\}$

2. *For any <u>non-atomic</u> formula F*
   $T_P^{i+1}(F) = T_P^i(F) \cap (\cap\{\mu | F : \mu \longleftarrow Body \in Fired_P(i)\}) \cap (\cap\{md_\rho(\mu) | \langle \mu, \rho \rangle \in M^{i+1}\}) \cap$
   $(\cap\{c_\rho(T_P^{i+1}(G), T_P^{i+1}(H)) | G \oplus_\rho H = F\})$

Given a basic formula G, we define the *width*(G) to be the number of atoms in G.
Given an hp-clause $C = B_0 : \mu_0 \leftarrow B_1 : \mu_1 \wedge \ldots \wedge B_k : \mu_k$, we say that the *head-width* of C is the width of B_0, and the *body-width* of C is $\max\{width(B_1), \ldots, width(B_k)\}$.

We may now define a hierarchy of subclasses of HPPs in terms of the head/body widths of the clauses involved.

Definition 16 *Let $HPP_{k,r}$ denote the class of HPP-programs P such that for all clauses $C \in P$, the head-width of C is less than or equal to k and the body-width of C is less than or equal to r. Let $HPP_k = \bigcup_{r \geq 0} HPP_{k,r}$.*

Algorithm **LFP** below shows how we may compute the least fixpoint of T_P for class $HPP_{k,r}$.

Algorithm LFP.
Input: $P \in HPP_{k,r}$, $N \geq \max(k, r)$, m - number of clauses in P, $F_1, \dots F_M$ - all formulas of *width* $\leq N$, lexicographically ordered.
Output: table $t_{2m}(F_k)$, $1 \leq k \leq M$.

BEGIN (algorithm)
1. FOR $j = 1$ TO M DO $t_0(F_j) := [0, 1]$;
2. FOR $i = 0$ TO $2m - 1$ DO
BEGIN
3. FOR EACH $C = G : \mu \leftarrow G_1 : \mu_1 \wedge \ldots \wedge G_n : \mu_l \in P$ such that for all $1 \leq j \leq n$ $t_i(G_j) \subseteq \mu_j$ DO
BEGIN
4. $t_{i+1}(G) := t_i(G) \cap \mu$;
5. delete C from P;
6. FOR EACH H included in G, i.e. $G = H \oplus_\rho H'$ DO
7. $t_{i+1}(H) := t_i(H) \cap md_\rho(\mu)$;
END
8. FOR $j = 1$ TO M DO
9. FOR $k = j + 1$ TO M DO
10. IF $F_k = F_j \oplus_\rho F_l$ (for some $l < k$)
11. THEN $t_{i+1}(F_k) := t_{i+1}(F_k) \cap c_i(t_{i+1}(F_j), t_{i+1}(F_l))$;
END
END.

The following theorem proves that algorithm **LFP** is a correct way of computing the least fixpoint of T_P for class $HPP_{k,r}$ and establishes its complexity.

Theorem 2 *Let P be any program in $HPP_{k,r}$ with m clauses. Let a be the number of different atoms in P, s be the number of different strategies in P, and $N \geq max\{k, r\}$. Then* **Algorithm LFP** *computes h_P on all the formulas of $bf_S(B_L)$ of width $\leq N$ in time $O(2m(2sa^N)^2) = O(m(sa^N)^2)$.*

The proof of this result is long and complex, and uses the property that for every $i = 0, 1, \dots, 2m$ and for every $k = 1, \dots M$, the quantity $t_i(F_k)$ in algorithm **LFP** coincides with $T_P^i(F_k)$ and therefore $t_{2m}(F_k) = h_P(F_k)$.

It is important to note that this theorem tells us that computing the least fixpoint of an HPP is *exponential* in the width of the largest formula of interest. In other words, if we were to develop an implementation of HPPs, and we required that no basic formulas of length greater than δ for some fixed δ are allowed, then the above theorem yields a polynomial result. This is a reasonable assumption, as we do not expect that formulas of width greater than some small constant (e.g., 4) would be of interest in any practical application. This is stated in the following corollary.

Corollary 1 *Let P be any hp-program, and suppose δ is a fixed bound on the width of basic formulas occurring in P. Then h_P can be computed in polynomial time of size of P for all formulas of width $\leq \delta$.*

<u>Note:</u> It is important to note that all the results of this subsection are valid for HPPs over p-strategies satisfying axioms (a)-(c) of Definition 1. Satisfaction of axiom (d) is not required.

3.2 Complexity of Entailment

While it is important to know the complexity of computing the entire model of an HPP, it is really the entailment problem which gets solved over and over when queries are asked to the program. In this section we will consider the complexity of entailment problem on HPPs: given a *consistent* program P and a query $F : \mu$, check whether $P \models F : \mu$. The problem of determining the consistency of an HPP is studied in Section 3.3.

In order to carry out our analysis of the entailment problem, we will split this subsection into three parts based on the syntax of HPPs.

- In Section 3.2.1, we will analyze what happens when we consider HPPs in the class HPP_1 — such HPPs only have atomic annotated atoms in rule heads, though the bodies may contain arbitrarily complex basic formulas.
- In Section 3.2.2, we will analyze what happens when we consider HPPs in the class HPP_r for $r \geq 3$.
- In Section 3.2.3, we will analyze what happens when we consider HPPs in the class HPP_2.

As usual we fix some standard encoding which is used to represent programs and queries. If P is an HPP, $|P|$ will denote the size of the representation of P in this encoding, and similarly, if $F : \mu$ is an annotated basic formula, $|F : \mu|$ will denote the size of its representation. The complexity results in the sections that follow will be relative to $|P|$ and $|F : \mu|$.

3.2.1 Algorithms and Complexity for Entailment w.r.t. HPP_1

Below, we show that if we consider the class HPP_1 containing only atoms in rule heads, then we can specialize algorithm **LFP** to a better algorithm, $\mathbf{LFP}_1$ for computing the least fixpoint of T_P.

Algorithm $\mathbf{LFP}_1$.
Input: $P \in HPP_1$, m – number of clauses in P, $F_1, \ldots, F_M$ - lexicographical enumeration of all formulas in P, $F = A_1 *_\rho \ldots *_\rho A_n$, $A_i \in B_L, i \in \{1, \ldots, n\}$.
Output: μ' - a subinterval of $[0, 1]$
BEGIN
(1) FOR $j = 1$ TO M DO $t(F_j) := [0; 1]$;
(2) FOR $i = 0$ TO $2m - 1$ DO
BEGIN
(3) FOR EVERY $C = G : \mu \leftarrow G_1 : \mu_1, ..., G_l : \mu_l \in P$ such that $t(G_j) \subseteq \mu_j$ for all $1 \leq j \leq l$ DO
BEGIN
(4) $t(G) := t(G) \cap \mu$;
(5) delete C from P
END
(6) FOR $k = 1$ TO M DO
(7) IF $F_k = B_1 *_\rho ... *_\rho B_r,\ r > 0, B_j$ is an atom for all $1 \leq j \leq r$
(8) THEN $t(F_k) := c_\rho(t(B_1), ..., t(B_r))$;
END;
(9) $\mu' := c_\rho(t(A_1), \ldots, t(A_n))$
END. (algorithm)

The following result specifies that the above algorithm may be directly used to check if an annotated basic formula $F : \mu$ is entailed by an HPP $P \in HPP_1$. Find the value μ' returned by Algorithm $\mathbf{LFP}_1$ — if $\mu' \subseteq \mu$, then $P \models F : \mu$, else it does not.

Theorem 3 *For any program $P \in HPP_1$ and annotated basic formula $F : \mu$ the entailment problem "$P \models F : \mu$" is solvable in time $O(|P|^2 + |F : \mu|)$ via* **Algorithm $\mathbf{LFP}_1$**.

3.2.2 Algorithms and Complexity for Entailment w.r.t. $HPP_r, r \geq 3$

In this section, we will develop algorithms and provide complexity results for checking entailment when we consider hybrid probabilistic programs for HPP_r when $r \geq 3$. The main results in this section are : **(i)** a *nondeterministic* polynomial algorithm to compute entailment for arbitrary HPPs in $HPP_r, r \geq 3$, and **(ii)** proof that this problem is NP-complete.

We start our analysis by first considering the class HPP^0 of HPPs that only consist of facts, i.e. all rules in such HPPs have an empty body. Proposition 1 below is an important technical proposition that may be used to develop an algorithm for entailment in class HPP^0. This in turn may be used to show that entailment for arbitrary consistent HPPs (including HPPs not in class HPP^0) is in class NP.

Proposition 1 *Let $P \in HPP^0$ be a program consisting of m clauses $C_1, ..., C_m$ with empty bodies: $C_k = H_k : [a_k, b_k],\ k = 1, ..., m$. Then for every formula $F = B_1 *_\rho ... *_\rho B_n,\ (n \geq 1,\ B_j$ is an atom, $1 \leq j \leq n)$, and a number $0 \leq x \leq 1$:*

1. *$P \models F : [0, x]$ iff there are $k \geq 1$ numbers $x_1, ..., x_k$ and such partition of F into k subformulas $F = F_1 *_\rho ... *_\rho F_k$ that:*
   - **(i)** *each F_i is either some head H_k and $x_i \geq b_k$, or it is a subformula of some head $H_k = F_i \oplus_\rho H'_k$ and $md_\rho([a_k, b_k]) \subseteq [0, x_i]$, and*
   - **(ii)** $c^2_\rho(x_1, ..., x_k) \leq x$.
2. *$P \models F : [x, 1]$ iff there are k numbers $x_1, ..., x_k$ and such partition of F into k subformulas $F = F_1 *_\rho ... *_\rho F_k$ that:*
   - **(i)** *each F_i is either some head H_k and $x_i \leq a_k$, or a subformula of some head $H_k = F_i \oplus_\rho H'_k$ and $md_\rho([a_k, b_k]) \subseteq [x_i, 1]$, and*
   - **(ii)** $c^1_\rho(x_1, ..., x_k) \geq x$.

Proposition 1 allows us to use the following *nondeterministic* algorithm to check entailment for hybrid probabilistic programs in class HPP^0.

Algorithm Ent-HPP^0
Input: program $P \in HPP^0$, formula $F = B_1 *_\rho ... *_\rho B_n$, and an interval $[a, b]$.

1. Guess such $k \leq width(F)$, sequence of bounds $x_1, ..., x_k$ and partition $F = F_1 *_\rho ... *_\rho F_k$ which satisfy conditions (i) and (ii) of Proposition 1.(1) for $x = b$.
2. Guess such $k \leq width(F)$, sequence of bounds $x_1, ..., x_k$ and partition $F = F_1 *_\rho ... *_\rho F_k$ which satisfy conditions (i) and (ii) of Proposition 1.(2) for $x = a$.
3. If both attempts are successful then output "Yes".

The following lemma which uses Proposition 1 establishes correctness of Algorithm **Ent-HPP^0** and its complexity.

Lemma 3 *(1) For any consistent hp-program $P \in HPP^0$ and any query $F : [a, b]$ algorithm Ent-HPP^0 output "Yes" iff $P \models F : [a, b]$.*
(2) Algorithm Ent-HPP^0 works in nondeterministic polynomial time.

Let $P \in HPP$ consists of m clauses $C_1, ..., C_m$ and $C_j = H_j : \mu_j \longleftarrow body_j$. For $0 \leq i \leq 2m$ we define an hp-program $P_i \in HPP^0$ as $P_i = \{H_j : \mu_j | act(C_j) \leq i\}$. The following important lemma shows us that it is possible to reduce the entailment problem for all HPPs to the entailment problem for the subclass HPP^0.

Lemma 4 *(1) For every $0 \leq i \leq 2m - 1$ and any formula F $P_i \models F : T_P^{i+1}(F)$.*
(2) For every $0 \leq i \leq 2m$ and any clause $C_j = (H_j : \mu_j \longleftarrow F_1 : \nu_1 \wedge ... \wedge F_r : \nu_r) \in P$
$C_j \in Fired(i + 1)$ iff $P_i \models F_l : \nu_l$ for all $l = 1, ..., r$.
(3) For any query $F : \mu$
$P \models F : \mu$ iff $P_{2m} \models F : \mu$.

We are now ready to present a generic algorithm, Algorithm **Ent-HPP**, that computes entailment by HPPs.
The following result establishes that algorithm **Ent-HPP** correctly computes entailment in nondeterministic polynomial time.

Lemma 5 *Algorithm* **Ent-HPP** *determines nondeterministically if $P \models F : [a, b]$ in polynomial time.*

Algorithm Ent-HPP
Input: program $P \in HPP$, consisting of m clauses $C_1, ..., C_m$ of the form $C_j = H_j : \mu_j \longleftarrow F_1^j : \nu_1 \wedge ... \wedge F_{r_j}^j$, formula $F = B_1 *_\rho ... *_\rho B_n$, and an interval $[a, b]$.
(1) $P_0 := \{H_j : \mu_j | body\ of\ C_j\ is\ empty\}$;
(2) FOR $i = 1$ TO $2m$ DO
 (3) guess $Fired(i)$;
(4) FOR EACH $C_j \in Fired_P(i)$ DO
 (5) FOR $l = 1$ TO r_j DO
  (6) Call Ent-$HPP^0(P_i, F_l^j : \nu_l)$;
 (7) END_DO
 (8) END_DO
 (9) $P_i := P_{i-1} \cup \{H_j : \mu_j | C_j \in Fired(i)\}$
(10) END_DO
(11) Call Ent-$HPP^0(P_{2m}, F : [a, b])$
(12) Output "Yes" if all (nondeterministic) calls of Ent-HPP^0 were successful.

An immediate consequence of the above result is the claim that the entailment problem for HPPs is in NP.

Theorem 4 *The entailment problem for the class of consistent HPPs belongs to NP.*

The following result shows that the problem of entailment in HPPs is NP-complete even when we only consider head-widths of size 3, and rule bodies are forced to be empty (which occurs when we set body-widths to 0).

Theorem 5 *The entailment problem is NP-complete for the class $HPP_{3,0}$.*

The proof of NP-hardness of the entailment problem can be obtained, by reducing a well-known NP-complete problem 3-Dimensional Matching to it.

The proof of Theorem 5 allows us immediately infer the following result.

Corollary 2 *The entailment problem is NP-complete for the classes HPP and $HPP_{k,r}(k \geq 3)$.*

3.2.3 Algorithms and Complexity for Entailment w.r.t. HPP_2

So far we have have shown that entailment problem is polynomial for hp-programs in HPP_1 and is NP-complete for hp-programs in HPP_k, $k \geq 3$. We now turn our attention to HPP_2. Here, our results are most interesting — it will turn out that for many different types of p-strategies, the entailment problem is polynomially solvable, though this does not appear to be the case for all p-strategies.

Recall that given a graph $G = (V, E)$, a *matching*[17] is a set $E' \subseteq E$ such that no two edges in E' share a common vertex. A matching E' is *maximal* iff every edge in $(E - E')$ shares a vertex with some edge in E'. If $V = 2m$, we say a matching E is *complete* iff every vertex $v \in V$ is the endpoint of some edge $e_v \in E'$. It will turn out that entailment is *polynomial time equivalent* to the following *generalized matching problem* on general graphs.

Generalized weighted matching problem

Given an edge-weighted, undirected graph $G = \langle V, E, w\rangle$ and a goal weight combination function c, find a complete matching for which the goal function on weights of selected edges is maximized (minimized).

More formally, we define two classes of "yes-no" matching problems:
$GWM_{max}(c) = \{G = \langle V, E, w : E \to [0, 1]\rangle, B \in [0, 1] || V| = 2m\}$ and there exists a complete matching $\{e_1, \ldots, e_m\} \subseteq E$ such that $c(w(e_1), \ldots, w(e_m)) \geq B\}$ and
$GWM_{min}(c) = \{G = \langle V, E, w : E \to [0, 1]\rangle, B \in [0, 1] || V| = 2m\}$ and there exists a complete matching $\{e_1, \ldots, e_m\} \subseteq E$ such that $c(w(e_1), \ldots, w(e_m)) \leq B\}$.

To show that entailment in HPPs is polynomial-time equivalent to the above generalized matching problem, we need to show that each problem can be polynomially reduced to the other.

Reducing Entailment to Generalized weighted matching. In order to reduce the entailment problem, we can use the composition function $c = \langle c^1, c^2 \rangle$ to create a generalized weighted matching which uses the c^1, c^2 as goal weight combination functions. Recall that as p-strategies are commutative and associative, we can consider them to be functions of any number of arguments.

Theorem 6 *Let $P \in HPP_2$ use probabilistic strategies with combination functions $c = \langle c^1, c^2 \rangle$ where $c^1 \in C^1, c^2 \in C^2$ for some sets of functions C^1 and C^2. Then the entailment problem for P and annotated basic formula $F : \mu$ is polynomially reducible to the problems $GWM_{max}(c^1)$ and $GWM_{min}(c^2)$, where $c^1 \in C^1, c^2 \in C^2$.*

Conversely, any generalized weighted matching problem whose associated goal function satisfies axioms (a)-(d) of Definition 1 may be polynomially reduced to an equivalent entailment problems for hp-programs of $HPP_{2,0}$.

Theorem 7 *Any generalized weighted matching problem for goal functions, satisfying axioms (a)-(d) of Definition 1 , is reducible in polynomial time to entailment problems for hp-programs of $HPP_{2,0}$.*

It is well-known [17] that weighted matching problem is solvable in polynomial time for the sum of edges weights. This allows to get effective algorithms for almost all of strategies considered in [2].

Corollary 3 *The entailment problem for the class of HPP_2 programs over strategies $\mathcal{S} = \{inc, igc, pcc, igd, pcd, ncd\}$ is solvable in polynomial time.*

The above result is interesting because it provides polynomial results for programs in HPP_2 for all composition strategies studied in [2] — in fact, we have been unable to find a single composition strategy for which entailment for the class of HPP_2 programs is not polynomial. This leads to an interesting open question.

Open question. Is there polynomial time computable composition function $c = (c^1, c^2)$ satisfying axioms (a)-(d) for which generalized matching problem $GWM_{min}(c^2)$ ($GWM_{max}(c^1)$) is NP-complete? If so, we would know that there are some polynomially computable composition functions for which the entailment problem for programs in HPP_2 is NP-complete.

3.3 Algorithms and Complexity Results for the Consistency Problem

In this section, we establish the complexity of determining if an HPP is consistent, i.e. is there a hybrid formula function h that satisfies all rules in P? As in the case of the Entailment Problem, we will split our work into three cases — where programs are from HPP_1, from HPP_2 and from HPP_3 or larger.

3.3.1 Algorithms and Complexity for Consistency w.r.t. HPP_1

It is easy to see that even a simple HPP_1 program containing two simple facts, viz. $a : [0, 0]$, $a : [1, 1]$, is inconsistent. The complicated interactions between logic and probabilities can engender more devious inconsistencies in HPPs.

The following result tells us that to check if P is consistent, if our language allows n ground atoms, we need to create only all ground basic formulas F containing all the n atoms and check if $h_P(F) \neq \emptyset$ for them. If so, P is guaranteed to be consistent.

Lemma 6 *Let $P \in HPP$ over set $\mathcal{S} = \{\rho_1, \ldots, \rho_m\}$ of p-strategies and let $\mathcal{A} = \{A_1, \ldots, A_n\}$ be all the atoms found in P. Then P is consistent* **iff** *for all formulas F_i of form $F_i = A_1 *_{\rho_i} \ldots *_{\rho_i} A_n$, $i = 1, \ldots, m$, $h_P(F_i) \neq \emptyset$.*

However, for programs from the class HPP_1, the situation with checking consistency is much simpler. As it turns out, it is sufficient to check for consistency only formulas of length 1, i.e. atoms.

Theorem 8 *Given a program $P \in HPP_1$ its consistency can be established in polynomial time.*

The proof is based on the following lemma which says that when a program $P \in HPP_1$ is inconsistent, then there exists a ground atom A such that $h_P(A) = \emptyset$.

Lemma 7 *Let $P \in HPP_1$. If P is inconsistent, then there exists an atom $A \in B_L$ such that*

1. *A is the head of at least one rule in P and*
2. *$h_P(A) = \emptyset$.*

Using the result of the lemma, we can now present the following polynomial algorithm, based on **Algorithm LFP**$_1$ from Theorem 3 for consistency check:

Algorithm Consistency_Check_for_HPP$_1$.

- Run **Algorithm LFP**$_1$ until $t(A) = \emptyset$ for some $A \in B_L$. If such an A exists, then P is inconsistent.
- If **Algorithm LFP**$_1$ finished successfully, then P is consistent.■

3.3.2 Algorithms and Complexity for Consistency w.r.t. HPP_2

It would be nice if result similar to Lemma 7 held when we consider programs with complex formulas in the heads. However, the following example shows that even for $P \in HPP_{2,0}$ the shortest inconsistent formula can be longer than the longest head of the rule.

Example 3 *Let P be*

$$\begin{array}{cc} a \wedge_{ind} b : [0.5, 0.7] \longleftarrow & c \wedge_{ind} d : [0.3, 0.48] \longleftarrow \\ a \wedge_{ind} c : [0.6, 1] \longleftarrow & b \wedge_{ind} d : [0.8, 1] \longleftarrow \end{array}$$

We can easily compute $h_P = lfp(T_P)$:

$$\begin{array}{cc} h_P(a) = [0.6, 1] & h_P(b) = [0.8, 1] \\ h_P(c) = [0.6, 1] & h_P(d) = [0.8, 1] \\ h_P(a \wedge_{ind} b) = [0.5, 0.7] & h_P(a \wedge_{ind} c) = [0.6, 1] \\ h_P(a \wedge_{ind} d) = [0.4, 1] & h_P(b \wedge_{ind} c) = [0.48, 1] \\ h_P(b \wedge_{ind} d) = [0.8, 1] & h_P(c \wedge_{ind} d) = [0.48, 0.48] \end{array}$$

However, $h_P(a \wedge_{ind} b \wedge_{ind} c \wedge_{ind} d) \subseteq c_{ind}(h_P(a \wedge_{ind} b), h_P(c \wedge_{ind} d)) \cap c_{ind}(h_P(a \wedge_{ind} c), h_P(b \wedge_{ind} d)) = [0.5 * 0.48, 0.7 * 0.48] \cap [0.6 * 0.8, 1 * 1] = [0.24, 0.336] \cap [0.48, 1] = \emptyset$.

The following result tells us that the inconsistency problem for HPP_2 is polynomially reducible to the generalized weighted matching problem.

Theorem 9 *Inconsistency problem for HPP_2 is polynomially reducible to $GWM_{\min}$ and $GWM_{\max}$.*

This is an important result, because we know that for many generalized weighted matching problems, polynomial algorithms exist. We may therefore immediately state the following corollary.

Corollary 4 *Let P be any HPP_2 program over the set of p-strategies $\{inc, igc, pcc, igd, pcd, ncd\}$. Then the consistency problem for P is solvable in polynomial time.*

3.3.3 Algorithms and Complexity for Consistency w.r.t. $HPP_r,\ r \geq 3$

We are now ready to examine the general consistency problem. Checking if an *arbitrary* HPP is inconsistent may be performed by the *nondeterministic* algorithm **InCon** below.

Algorithm InCon.
Input. An arbitrary HPP P.

1. Guess the shortest "inconsistent" formula F.
2. Guess two partitions of F into $G_1 *_\rho \ldots *_\rho G_m$ and $H_1 *_\rho \ldots *_\rho H_k$ two sets of numbers: $x_1, \ldots, x_m$ and $y_1, \ldots, y_k$, $0 \leq x_i, y_j \leq 1$ such that $c^1_\rho(x_1, \ldots, x_m) > c^2_\rho(y_1, \ldots y_k)$.
3. Using **Algorithm Ent-HPP** check that for all $i \in \{1, \ldots m\}$ $P \models G_i : [0, x_i]$ and all $j \in \{1, \ldots k\}$ $P \models H_j : [y_j, 1]$.
4. If all calls to **Algorithm Ent-HPP** of previous step are successful then output "Yes".

The following result shows that for arbitrary HPPs, the problem of checking if an HPP is inconsistent is NP-complete.

Theorem 10 *The inconsistency problem for HPP is NP-complete.*

An immediate corollary is that the consistency problem for arbitrary HPPs is co-NP-complete.

Corollary 5 *Consistency problem for HPP is* co-NP-complete.

4 Proof Procedure

In this section, we present a sound and complete proof procedure for HPPs. The first proof procedure for probabilistic programs, introduced in [2] and [25] is based upon expanding the program P to a larger set of clauses (a closure of the program) and then resolving queries against that set. Since this procedure is computationally inefficient, other tabulation based proof procedures have also been developed. Here, we present a Hilbert-style proof system for ground hp-programs which guarantees that all proofs are polynomially bounded in length ! This is consistent with the preceding NP-completeness result because the search space may involve exponentially many derivations each of polynomially bounded length. This is an interesting and counterintuitive result — it says that (the answers to) all queries to HPPs have at least one polynomial explanation. This is not true of classical logic proof systems [31] as well as proof systems for most nonmonotonic logics.

4.1 System HGR_P

Let us now define the axioms and inference rules of the proof system, HGR_P.

Definition 17 *Let P be a ground hp-program over set $\mathcal{S}$ of p-strategies. We define the* *formal system HGR_P as follows:*

1. *Axioms of HGR_P are all expressions of the form:*

$$\overline{A : [0, 1]}$$

   where $A \in B_L$.

2. *Inference Rules. There are 7 types of inference rule schemas in HGR_P. One type (Program) depends on the clauses of program P while other 6 types of inference rule schemas are independent of clauses in P but do depend on which p-strategies are in $\mathcal{S}$.*

- Program: *Let* $F : \mu \leftarrow G_1 : \mu_1, ..., G_k : \mu_k \in P$,

$$\frac{G_1 : \mu_1 \ \ ... \ G_k : \mu_k}{F : \mu}$$

Note: rules corresponding to clauses with empty body ($k = 0$) *are actually axioms.*

- A-Composition: *Let* $A_1, A_2 \in B_L$,$\rho \in \mathcal{S}$

$$\frac{A_1 : \mu_1 \quad A_2 : \mu_2}{(A_1 *_\rho A_2) : c_\rho(\mu_1, \mu_2)}$$

- F-Composition: *Let* $A_1, \dots A_k, B_1 \dots B_k \in B_L$,$\rho \in \mathcal{S}$

$$\frac{(A_1 *_\rho \dots *_\rho A_k) : \mu_1 \quad (B_1 *_\rho \dots *_\rho B_l) : \mu_2}{(A_1 *_\rho \dots *_\rho A_k *_\rho B_1 *_\rho \dots *_\rho B_l) : c_\rho(\mu_1, \mu_2)}$$

- Decomposition (cut): *Let* $\rho \in \mathcal{S}$

$$\frac{(F \oplus_\rho G) : \mu}{F : md_\rho(\mu)}$$

- Clarification:

$$\frac{F : \mu_1 \quad F : \mu_2}{F : \mu_1 \cap \mu_2}$$

- Exchange: *Let* $A_1, \dots, A_k \in B_L$, $\rho \in \mathcal{S}$, *and let* $B_1, \dots, B_k$ *be a* **permutation** *of* $A_1, \dots, A_k$

$$\frac{(A_1 *_\rho \dots *_\rho A_k) : \mu}{(B_1 *_\rho \dots *_\rho B_k) : \mu}$$

- Interval Weakening:

$$\frac{F : \mu \quad \mu \subseteq \mu_1}{F : \mu_1}$$

3. *A finite sequence* $C_1 \dots C_r$ *of annotated formulas is called an* HGR_P**-derivation** iff *each formula* $C_j = F_j : \mu_j$ *can be deduced from zero (in the case of axiom), one, or more previous of* $C_1 \dots C_{j-1}$ *by applying one of the* inference rules *to them. We call formula* C_r the result *of the* HGR_P*-derivation.*

4. *An annotated formula* $C = F : \mu$ *is derivable in* HGR_P **iff** *there exists such an* HGR_P*-derivation* $C_1, \dots C_r$ *that* $C_r = C$*. We denote it by* $P \vdash_{HGR_P} C$*,or just by* $P \vdash C$ *in the absence of other inference systems.*

The following theorem states that HGR_P is a sound inference system.

Theorem 11 (soundness of HGR_P**)** *Let* P *be an* $hp-program$ *and let* Q *be a an hp-formula. If* $P \vdash_{HGR_P} Q$*, then* $P \models Q$.

5 Proof Complexity

We are now ready to study the complexity of proofs in this proof system. It is known that all "natural" proof systems for standard classical propositional logic have proofs of exponential size (see e.g. [31]). In this section we show that this is not the fact in our proof system HGR_P.

The following result states that for programs $P \in HPP^0$, HGR_P is both a *complete inference system* and that the length of the proofs in HGR_P is linearly bounded.

Proposition 2 *For any* $P \in HPP_0$ *and annotated formula* $F : \mu$ *if* $P \models F : \mu$ *there exists such an* HGR_P*-derivation* $C_1, \dots C_r$ *that* $C_r = F : \mu$ *that* $r \leq 10|F| + 6$.

It is possible to generalize the above result to show that HGR_P is a complete inference system for all HPPs P and further that it guarantees polynomially-sized proofs.

Theorem 12 *For any $P \in HPP$ and annotated formula $F : \mu$ if $P \vdash F : \mu$ then there exists such an HGR_P-derivation $C_1, \ldots C_r$ that $C_r = F : \mu$ and $r \leq O(|P|^2 + |F : \mu|)$.*

This result may seem somewhat counterintuitive as it is well-known (see e.g. [28]) that for *classical propositional logic* existence of proof systems with polynomially bounded length of proofs implies that $NP = coNP$. This is due to the fact that the entailment problem for propositional classical logic is $coNP$-complete. In contrast, for hybrid probabilistic logic programs, entailment is NP-complete and our polynomiality bound on proof lengths therefore does not imply $NP = coNP$ as the search space may involve exponentially many derivations. Still, this result is interesting, since for many proof systems for classical propositional logic (e.g. resolution based) and for variety of nonmonotonic logics superpolynomial lower bounds were established ([31, 28]).

6 Conclusions

As described in the introduction, there are numerous kinds of dependencies that might exist between uncertain events. Probability theory mandates that the probability of a complex event be computed not only in terms of the probabilities of the primitive events involved, but also it should take into account, dependencies between the events involved. Pioneering efforts in this area were made by Lakshmanan and Shiri [15]. Hybrid Probabilistic Programs (HPPs) [2] represent one of the first frameworks that allow a logic program to explicitly encode a variety of different probability assumptions explicitly into the program, for use in inferencing. Most existing frameworks for uncertainty in logic programming [3, 4, 5, 10, 11, 14, 20, 21, 22, 25, 27, 29, 9] do not permit this. A few important initial attempts to incorporate different probabilistic strategies were made by Thone *et al.*[30], and Lakshmanan [15], which culminated in an extension of the relational algebra that accommodated different probabilistic strategies [12].

In this paper, we have made three contributions. First, we have developed algorithms to efficiently perform a variety of computations for hybrid probabilistic programs. Each of these algorithms is "tuned" to fit the class within which an HPP falls (i.e. class HPP_1, HPP_2 or HPP_r, $r \geq 3$). We have given algorithmic complexity analyses of these problems. To date, with the exception of the work by Kiessling's group [9, 30] and by Lukasiewicz [22], almost no work on bottom up algorithms for computing probabilistic logic programs exists. Our algorithms are the first to apply not only to HPPs, but to have finer complexity bounds for different classes of HPPs.

Second, we have studied the computational complexity of the Entailment and Consistency problems for the abovementioned classes of HPPs. The results may be neatly summarized via the following table.

Problem	HPP_1	HPP_2	HPP_r, $r \geq 3$
Entailment	polynomial	polynomial for composition functions in $\{inc, igc, pcc, igd, pcd, ncd\}$	NP-complete
Consistency	polynomial	polynomial for composition functions in $\{inc, igc, pcc, igd, pcd, ncd\}$	co-NP-complete

In effect, this result says that from the point of view of complexity, it is possible to safely write HPPs over class HPP_1 (with any set of composition strategies), or over class HPP_2 (but with certain composition strategies only), and be guaranteed a polynomial computation. To our knowledge, this paper is the first paper to contain a detailed analysis of complexity results in probabilistic logic programs, though [12] contains some results for probabilistic relational algebra, and [14] contains some results for a different probabilistic framework, and Lukasiewicz[20, 21, 22, 23] proves some elegant complexity results for a mix of multivalued and probabilistic logic programming.

Finally, we have described a proof system for HPPs that guarantees that for every $F : \mu$ that is a ground logical consequence of an HPP P, we have a polynomially bounded proof of $F : \mu$, which in turn, means that an *explanation* for $F : \mu$ is polynomially bounded. Though many proof systems have been developed for annotated logic programs (notably by Hahnle[7], Henschen and Lu[8], Lu and Leach[16], and Lu, Murray and Rosenthal[18, 19], these do not apply to probabilistic logic programs Our proof system HGR_P is new (and is also different from the proof system in [2]), and to our knowledge none of the existing proof systems for annotated logic have been shown to have polynomially bounded proofs (and hence succinct explanations).

References

[1] G. Boole. (1854) *The Laws of Thought*, Macmillan, London.

[2] A. Dekhtyar and V.S. Subrahmanian. (1997) *Hybrid Probabilistic Logic Programs*, Proc. 1997 Intl. Conf. on Logic Programming (ed. L. Naish), MIT Press. Extended version submitted to Journal of Logic Programming, Jan. 1998, revised Sep. 1998. Available as University of Maryland Tech Report CS-TR-3883, from http://www.cs.umd.edu/TRs/authors/Alex_Dekhtyar.html.

[3] D. Dubois and H. Prade. Certainty and Uncertainty of Vague Knowledge and Generalized Dependencies in Fuzzy Databases. In *Proceedings International Fuzzy Engineering Symposium*, pp. 239–249, Yokohama, Japan, 1988.

[4] R. Fagin and J. Halpern. (1988) *Uncertainty, Belief and Probability*, in Proc. IJCAI-89, Morgan Kaufman.

[5] M. C. Fitting. (1988) *Logic Programming on a Topological Bilattice*, Fundamenta Informatica, 11, pps 209–218.

[6] U. Guntzer, W. Kiessling and H. Thone. (1991) *New Directions for Uncertainty Reasoning in Deductive Databases*, Proc. 1991 ACM SIGMOD, pp 178–187.

[7] R. Hahnle. (1990) *Towards an Efficient Tableau Proof Procedure for Multiple-Valued Logics*, Proc. Computer Science Logic Conference, 1990.

[8] L.J. Henschen and J. Lu. (1992) *The Completeness of GP-resolution for Annotated Logics*, Information Processing Letters, 44, 1992, 135-140.

[9] W. Kiessling, H. Thone and U. Guntzer. (1992) *Database Support for Problematic Knowledge*, Proc. EDBT-92, pps 421–436, Springer LNCS Vol. 580.

[10] M. Kifer and A. Li. (1988) *On the Semantics of Rule-Based Expert Systems with Uncertainty*, 2-nd Intl. Conf. on Database Theory, Springer Verlag LNCS 326, (eds. M. Gyssens, J. Paredaens, D. Van Gucht), Bruges, Belgium, pp. 102–117.

[11] M. Kifer and V. S. Subrahmanian. (1992) Theory of Generalized Annotated Logic Programming and its Applications, JOURNAL OF LOGIC PROGRAMMING, 12, 4, pps 335–368, 1992.

[12] V.S. Lakshmanan, N. Leone, R. Ross and V.S. Subrahmanian. ProbView: A Flexible Probabilistic Database System. ACM TRANSACTIONS ON DATABASE SYSTEMS, Vol. 22, Nr. 3, pps 419–469, Sep. 1997.

[13] V.S. Lakshmanan and F. Sadri. (1994) *Modeling Uncertainty in Deductive Databases*, Proc. Int. Conf. on Database Expert Systems and Applications, (DEXA'94), September 7-9, 1994, Athens, Greece, Lecture Notes in Computer Science, Vol. 856, Springer (1994), pp. 724-733.

[14] V.S. Lakshmanan and F. Sadri. (1994) *Probabilistic Deductive Databases*, Proc. Int. Logic Programming Symp., (ILPS'94), November 1994, Ithaca, NY, MIT Press.

[15] V.S. Lakshmanan and N. Shiri. (1997) *A Parametric Approach with Deductive Databases with Uncertainty*, accepted for publication in IEEE Transactions on Knowledge and Data Engineering.

[16] S.M. Leach and J.J. Lu. (1994) *Computing Annotated Logic Programs*, Proceedings of the International Conference on Logic Programming (ICLP '94), MIT Press, 1994.

[17] E. Lowler. *Combinatorial Optimization: Networks and Matroids*, Holt, Reinhart & Winston, New York, 1976.

[18] J. Lu, N.V. Murray, and E. Rosenthal. (1993) *Signed Formulas and Annotated Logics*, Proceedings of the 23rd IEEE International Symposium on Multiple-Valued Logic, (ISMVL '93), IEEE Computer Society Press, 1993.

[19] J. Lu, N.V. Murray and E. Rosenthal. (1998) *A Framework for Automated Reasoning in Multiple-Valued Logics*, Journal of Automated Reasoning 21(1), pages 39–67.

[20] T. Lukasiewicz. (1997) *Efficient Global Probabilistic Deduction from Taxonomic and Probabilistic Knowledge-Bases over Conjunctive Events*, Proc. Conf. on Information and Knowledge Management, pps 75–82.

[21] T. Lukasiewicz. (1998) *Probabilistic Logic Programming*, in Procs. 13th biennial European Conference on Artificial Intelligence, pps 388-392, Brighton, UK, August 1998.

[22] T. Lukasiewicz. (1998) *Magic Inference Rules for Probabilistic Deduction under Taxonomic Knowledge*, Proceedings of the 14th Conference on Uncertainty in Artificial Intelligence, pps 354-361, Madison, Wisconsin, USA, July 1998.

[23] T. Lukasiewicz. (1998) *Many-Valued First-Order Logics with Probabilistic Semantics*, in Proc. Computer Science Logic Conference, Brno, Czech Republic, Aug. 1998.

[24] J.W. Lloyd. (1987) *Foundations of Logic Programming*, Springer.

[25] R. Ng and V.S. Subrahmanian. (1993) Probabilistic Logic Programming, INFORMATION AND COMPUTATION, 101, 2, pps 150–201, 1993.

[26] R. Ng and V.S. Subrahmanian. (1995) Stable Semantics for Probabilistic Deductive Databases, INFORMATION AND COMPUTATION, 110, 1, pps 42-83.

[27] N. Nilsson. (1986) *Probabilistic Logic*, Artificial Intelligence, 28, pp 71–87.

[28] P. Pudlak. *The lengths of proofs*, Handbook of proof theory, Stud. Logic Found. Math.,127, North-Holland, Amsterdam, pps. 547-637, 1998.

[29] V.S. Subrahmanian. (1987) *On the Semantics of Quantitative Logic Programs*, Proc. 4th IEEE Symp. on Logic Programming, pps 173-182, Computer Society Press. Sep. 1987.

[30] H. Thone, W. Kiessling and U. Guntzer. (1995) *On Cautious Probabilistic Inference and Default Detachment*, Annals of Operations Research, 55, pps 195–224.

[31] A. Urquhart. *The Complexity of Propositional Proofs*, The Bulletin of Symbolic Logic, 1(4):425-467, December 1995.

Appendix. Proofs of the results

This Appendix is included for the convenience of the referees. It may be removed if this paper is accepted for publication. The paper, together with this appendix, will be available as a University of Maryland Technical Report.

Proof of Lemma 1.

1. We will use induction on the number of atoms in F to prove that $T_P^{i+1}(F) = T_P^{i+2}(F)$. The statement of the claim will then follow immediately.

   Base case. Let F be an *atomic formula*.

   From def. 12 it follows that if $Fired_P(i) = Fired_P(i+1) = \emptyset$ then for any basic formula F, $S_P(T_P^i)(F) = S_P(T_P^{i+1})(F)$.

   Since no new rules are fired on steps i and $i+1$ the set

   $$M^i = \{\langle\mu, \rho\rangle | (F *_\rho G) : \mu \longleftarrow F_1 : \mu_1 \wedge \ldots \wedge F_n : \mu_n \in P; (\forall 1 \leq j \leq n)(T_P^i(F_j) \subseteq \mu_j)\}$$

   is equal to the set

   $$M^{i+1} = \{\langle\mu, \rho\rangle | (F *_\rho G) : \mu \longleftarrow F_1 : \mu_1 \wedge \ldots \wedge F_n : \mu_n \in P; (\forall 1 \leq j \leq n)(T_P^{i+1}(F_j) \subseteq \mu_j)\}.$$

   Then, $T_P(T_P^i)(F) = T_P(T_P^{i+1}(F))$, therefore, $T_P^{i+1}(F) = T_P^{i+2}(F)$.

   Induction step. Let F consist of n atoms. Similarly to the atomic case $S_P(T_P^i)(F) = S_P(T_P^{i+1})(F)$ and the set M' in def.13.2 remains the same at steps i and $i+1$. Also, for any two basic formulas G and H such that $G \oplus_\rho H = F$ by induction hypothesis we get that $c_\rho(T_P(T_P^i(G)), T_P(T_P^i(H))) = c_\rho(T_P(T_P^{i+1}(G)), T_P(T_P^{i+1}(G)))$. Then $T_P(T_P^i)(F) = T_P(T_P^{i+1}(F))$, and therefore, $T_P^{i+1}(F) = T_P^{i+2}(F)$.

2. If P consists of m clauses then $lfp(T_P) = T_P^{2m}$ follows immediately from the first claim of the lemma since for every two steps i, $i+1$ *at least one new* clause from P should be fired. ■

Proof of Lemma 2.
The proof follows directly from definitions 12 and 13.

Proof of Theorem 2. Let $F_1, ..., F_M$ be a lexicographically ordered sequence of all formulas of width $\leq N$ constructed by atoms and strategies of P. Then $M \leq 2sa^N$. **Algorithm LFP** constructs a sequence of tables $t_0, t_1, ..., t_{2m}$. The correctness of the algorithm follows from the following proposition.

Proposition 3 *For every* $i = 0, 1, \ldots, 2m$ *and for every* $k = 1, \ldots M$ $t_i(F_k) = T_P^i(F_k)$.

Proof of Proposition 3. We prove this proposition using induction on i.

Base Case. $i = 0$.

Table t_0 is computed in line 1 of the algorithm. For each formula F_j, $t_0(F_j)$ is assigned $[0, 1]$, which is equal to $T_P^0(F_j)$.

Induction Step. Assume that after $i-1$ iterations of loop 2-11 table t_i constructed is such that for each F_k $(1 \leq k \leq M)$ $t_i(F_k) = T_P^i(F_k)$ and that the set of clauses C deleted via line 5. is equal to $\{C | act_P(C) \leq i\} = \cup_{j=0}^i Fired_P(j)$.

We will show that after i iterations of this loop $t_{i+1}(F_k) = T_P^{i+1}(F_k)$ for all $(1 \leq k \leq M)$ and that the set of clauses deleted on ith iteration via line 5. is exactly $Fired_P(i+1)$.

Since $T_P^i(F_k) = t_i(F_k)$, the set of clauses selected from P in line 3. on ith iteration of the loop 2.-11. should belong to $\cup_{j=0}^{i+1} Fired_P(j)$. But by induction hypothesis, after $i-1$ iterations of loop $2-11$ all clauses in $\cup_{j=0}^i Fired_P(j)$ are deleted from P. So, if for some clause C, the condition in line 3 is satisfied on ith iteration of the loop, then $C \in Fired_P(i+1)$. All such clauses are then deleted in executions of line 5. of this iteration of the loop.

Then the assignment statements in lines 4. and 7. imply that after loop 3.–7. is completed on ith iteration of the algorithm $t_{i+1}(F_k) = t_i(F_k) \cap (\cap\{\mu|F_k : \mu \longleftarrow Body \in Fired_P(i+1)\}) \cap (\cap\{md_\rho(\mu)|(F \oplus_\rho G) : \mu \longleftarrow Body \in Fired_P(i+1)\})$.

If F_k is atomic the condition in line 10. will never be true for it, therefore the assignment statement in line 11. will never be executed for F_k. Therefore, by Lemma2.1 $T_P^{i+1}(F_k) = t_{i+1}(F_k)$ already after the completion of loop 3.–7.

We proceed by induction on k to prove that after k executions of loop 8.–11. on ith iteration, $T_P^{i+1}(F_k) = t_{i+1}(F_k)$.

Base Case For all k such that F_k is atomic, the statement is true since because $T_P^{i+1}(F_k)$ was equal to $t_{i+1}(F_k)$ even before the first iteration of the loop, and $t_{i+1}(F_k)$ is never changed inside the loop.

Induction Step. Suppose that for each F_j, $(j < k)$, $t_{i+1}(F_j) = T_P^{i+1}(F_j)$ after j iterations of loop 8.–11. Notice that $t_{i+1}(F_j)$ will never appear again on the left side of the assignment statement in any further execution of line 11. Therefore, the value of $t_{i+1}(F_j)$ will not change.
Let M_{11} be the set of all intervals $c_\rho(t_{i+1}(F_j), t_{i+1}(F_l))$ which intersect with $t_{i+1}(F_k)$ in line 11. Since by induction assumption $t_{i+1}(F_j) = T_P^{i+1}(F_j)$ and $t_{i+1}(F_l) = T_P^{i+1}(F_L)$ we get that $M_{11} \subseteq \{c_\rho(T_P^{i+1}(G), T_P^{i+1}(H))|F_k = G \oplus_\rho H\}$. Suppose now that for some basic formulas $G, H, F_k = G \oplus H$. Then, there exist such numbers $j, l < k$, that $G = F_j$, $H = F_l$ and on iteration j of loop 8.–11. on iteration $k-j$ of loop 9.–11., the assignment $t_{i+1}(F_k) := t_{i+1}(F_k) \cap c_\rho(t_{i+1}(F_j), t_{i+1}(F_l))$ in line 11. will be executed. Thus $c_\rho(T_P^{i+1}(G), T_P^{i+1}(H)) = c_\rho(t_{i+1}(F_j), t_{i+1}(F_l)) \in M_{11}$. Therefore $M_{11} = \{c_\rho(T_P^{i+1}(G), T_P^{i+1}(H))|F_k = G \oplus_\rho H\}$. Then, by Lemma2.2 we get that $t_{i+1}(F_K) = t_i(F_k) \cap (\cap\{\mu|F_k : \mu \longleftarrow Body \in Fired_P(i+1)\}) \cap (\cap\{md_\rho(\mu)|(F \oplus_\rho G) : \mu \longleftarrow Body \in Fired_P(i+1)\}) \cap (\cap\{\mu|\mu \in M_{11}\}) = T_P^{i+1}(F_k)$.
This completes the proof of Proposition 3. ■

Now, from Proposition 3 and Lemma 1 it follows that t_{2m} represents $h_P = lfp(T_P)$. We have only to compute the time complexity of the algorithm. Loop 3.–11. will be executed exactly $2m$ times. The running time of loop 3.–7. is determined by the running time of the loop 8.–11. which requires $O(M^2) = O((2sa^N)^2)$ steps. Combining together this yields the $O(2m(2sa^N)^2)$ bound.■

Proof of Theorem 3.
Our algorithm takes as input $P \in HPP_1$ and F and produces as output $\mu' = h_P(F)$. Then by the definition of entailment, $P \models F : \mu$ iff $\mu' \subseteq \mu$. Suppose $F = A_1 *_\rho \ldots *_\rho A_n,\ n > 0$.

The algorithm proceeds in two steps. In the first step we use procedure $\mathbf{LFP}_1$ to compute h_P for all atoms in P and F. In the second step, *if* $n > 1$ we compute $c_\rho(h_P(A_1), \ldots, h_P(A_n))$ and return the obtained interval μ' as the output (if $n = 1$ then F is equal to some atom A for which we have obtained the value of h_P on the first step. This value will be returned).

Now to compute h_P on the set of atoms we use lines (1)–(8) of **Algorithm** $\mathbf{LFP}_1$. Algorithm $\mathbf{LFP}_1$ iteratively constructs table t of size M such that after i iterations of the main loop (1)–(8) $t(F_j) = T_P^i(F_j)$ for all $1 \le j \le M$. The proof of correctness of algorithm $\mathbf{LFP}_1$ is analogous to the proof of correctness of the algorithm **LFP** from Theorem 2. This proof uses the following proposition:

Proposition 4 *For any $P \in HPP_1$ and basic formula $F = A_1 *_\rho \ldots *_\rho A_n$*

1. *for all $i \ge 1$ $T_P^i(F) = c_\rho(T_P^i(A_1), \ldots, T_P^i(A_n))$.*
2. $h_P(F) = c_\rho(h_P(A_1), \ldots, h_P(A_n))$.

Proof of Proposition 4. To prove part 1 of the proposition we notice that from Lemma 2.2 it follows that $T_P^{i+1}(F) = T_P^i \cap (\cap\{c_\rho(T_P^{i+1}(G), T_P^{i+1}(H))|F = G \oplus_\rho H\})$. Then using double induction on i and on $width(F)$ and the properties of associativity and commutativity of c_ρ we get the required result.

Part 2 follows immediately from part 1. ■

Proposition 4.2 also assures us that $\mu' = c_\rho(h_P(A_1), \ldots, h_P(A_n))$ computed in line (9) the algorithm and returned as output is indeed equal to $h_P(F)$ which our algorithms claims to compute.

To evaluate the complexity of this algorithm we note that it is determined by the complexity of loop (1)–(8). Its running time is $O(2mM) \le O(|P|^2)$ because $M \le |P|$. The running time of line (9) and comparison of μ' and μ is $O(|F : \mu|)$. ■

Proof of Proposition 1. Proof (1)If-part. Since $P \in HPP^0$ then all it's clauses belong to $Fired_P(0)$ and $h_P(F) = T_P^1(F)$. Let given F $A = \{\mu | F : \mu \in P\}$, $B = \{md_\rho(\mu) | (F \oplus_\rho G) : \mu \in P\}$, and $C = \{c_\rho(h_P(G), h_P(H)) | F = G \oplus_\rho H\}$. We will use induction on width of F.
Base case. Let F be an atomic formula. By Lemma 2.1 we get $h_P(F) = [0,1] \cap \bigcap\{\mu | \mu \in A\} \cap \bigcap\{\mu | \mu \in B\}$. Then $h_P(F) = \mu_1 \cap \mu_2$ for some intervals $\mu_1, \mu_2 \in A \cup B \cup \{[0,1]\}$. Let $h_P(F) = [y, z]$. Assume without loss of generality that $\mu_1 = [y_1, z_1], \mu_2 = [y_2, z_2]$, and $z = min\{z_1, z_2\} = z_1$. Since $P \models F : [0, x]$ then $z_1 \le x$. Now set $F_1 = F$ and $x_1 = z_1$. If $\mu_1 \in A$ then for some k $F_1 = F = H_k$ and $\mu_1 = [y_1, z_1] = [a_k, b_k]$. So, $x_1 \ge b_k$. If $\mu_1 \in B$ then for some k $H_k = F \oplus_\rho H'_k$ and $\mu_1 = md_\rho([a_k, b_k]) = [y_1, z_1] \subseteq [0, x_1]$. In the both cases condition (i) is satisfied. To establish (ii) it's enough to notice that $c^2(x_1) = x_1 = z_1 \le x$.
Induction step. Suppose that for all formulas of width less than $n \ge 2$ Proposition 1 (1) is valid. Let $F = B_1 *_\rho ... * B_n$, $(n \ge 2)$.

By Lemma 2.2 we get $h_P(F) = [0,1] \cap \bigcap\{\mu | \mu \in A\} \cap \bigcap\{\mu | \mu \in B\} \cap \bigcap\{\mu | \mu \in C\}$. Then $h_P(F) = \mu_1 \cap \mu_2$ for some intervals $\mu_1, \mu_2 \in A \cup B \cup C \cup \{[0,1]\}$. Let $h_P(F) = [y, z]$. Assume without loss of generality that $\mu_1 = [y_1, z_1], \mu_2 = [y_2, z_2]$, and $z = min\{z_1, z_2\} = z_1$. Since $P \models F : [0, x]$ then $z_1 \le x$. If $\mu_1 \in A \cup B$ we proceed as in the basic case. If $\mu_1 \in C$ then $F = G \oplus_\rho H$ for some formulas G, H of width less than n, and $\mu_1 = c_\rho(h_P(G), h_P(H))$. Let $h_P(G) = [y_G, z_G]$ and $h_P(H) = [y_H, z_H]$. Since $P \models F : [0, x]$ we deduce that $c_\rho(h_P(G), h_P(H)) = c_\rho([y_G, z_G], [y_H, z_H]) \subseteq [0, x]$ and therefore $c_\rho^2(z_G, z_H) \le x$. ¿From the definition of bounds z_G and z_H it follows that $P \models G : [0, z_G]$ and $P \models H : [0, z_H]$. Then by the induction assumption there exist such two sequence of bounds $u_1, ..., u_s$ and $v_1, ..., v_p$ and two partions $G = G_1 *_\rho ... *_\rho G_s$ and $H = H_1 *_\rho ... *_\rho H_p$ which satisfy conditions (i) and (ii) of Proposition 1 (1). Uniting these sequences of bounds and partitions we get the sequence of bounds $x_1 = u_1, ..., x_s = u_s, x_{s+1} = v_1, ..., x_{s+p} = v_p$ of the length $k = s + p$ and partition of $F = G_1 *_\rho ... *_\rho G_s *_\rho H_1 *_\rho ... *_\rho H_p$ which satisfies condition (i). To prove (ii) we notice that $c_\rho^2(u_1, ..., u_s) \le z_G$ and $c_\rho^2(v_1, ..., v_p) \le z_H$ and due to monotonicity of c_ρ^2 we get $c_\rho^2(x_1, ..., x_k) = c_\rho^2(c_\rho^2(u_1, ..., u_s), c_\rho^2(v_1, ..., v_p)) \le c^2(z_G, z_H) \le x$.
The proof of **Only-If** part of proposition follows immediately from Lemma 2. It's enough to notice that from condition (i) it follows that for every F_j in the partition of F $h_P(F_j) \subseteq [0, x_j]$.

The proof of (2) is on the same lines as the proof of (1) with minor changes.
■

Proof of Lemma 4. The proof of point (1) follows directly from Lemma 2. Points (2) and (3) follows from (1).
■

Proof of Lemma 5. The proof of correctness of the algorithm follows from Lemmas 3 and 4. If for all i sets $Fired(i)$ are guessed correctly in line (3) then the loop in lines (4)-(8) can (nondeterministically) confirm this guess. If so, then in line (9), the set P_i is defined correctly. Then after finishing the loop in lines (2)-(10), set P_{2m} consists of the heads of all fired clauses and call of Ent-HPP^0 in line (11) can correctly establish if $P_{2m} \models F : [a, b]$ which due to Lemma 4.3 assures the required result.

To evaluate complexity of the algorithm we notice that the total number of calls of algorithm Ent-HPP^0 in line (6) does not exceed number of formulas in bodies of all clauses of P. Let $t_{Ent-HPP^0}(P)$ denotes the maximal time among all calls of Ent-HPP^0. Then the length $t_{Ent-HPP}(P, F : [a, b])$ of the shortest successful computation of algorithm Ent-HPP on input $(P, F : [a, b])$ can be bounded as $t_{Ent-HPP}(P, F : [a, b]) \le O(|P| t_{Ent-HPP^0} + |F : [a, b]|) \le pol(|P| + |F : [a, b])$ for some polynomial $pol(N)$. ■

Proof of Theorem 3.2.2.
The proof will proceed in two stages. In the first stage we use the nondeterministic polynomial time algorithm **Ent-HPP**^{0} to solve the entailment problem for the class HPP^0 programs consisting only in facts, i.e. $HPP^0 = \bigcup_{i=1}^{\infty} HPP_{i,0}$. At the second stage this algorithm will be extended to all programs of HPP.

Proof of Theorem 5.
Upper bound follows by Theorem 3.2.2.

To prove the lower bound we show that well-known NP-complete problem 3-Dimensional Matching (3-DM) is reducible in polynomial time to the entailment problem for class $HPP_{3,0}$. Let I be any instance of 3-dimensional matching problem, consisting of three sets W, X and Y of equal size q and a set of triples $M \subseteq W \times X \times Y$. $I \in$3-DM iff there is a complete matching, i.e. such subset $M' \subseteq M$ of size q that $W = \{w| < w, x, y > \in M', X = \{x| < w, x, y > \in M'\}$ and $Y = \{y| < w, x, y > \in M'\}$. We use I to construct HPP-program P and query $F : \mu$ such that $I \in$ 3-DM iff $P \models F : \mu$.

As a strategy we choose the conjunctive independence strategy, with c_{inc} defined in Example 1 as follows: $c_{inc}([a, b], [c, d]) = [ac, bd]$. Program P uses atoms of the set $A = W \cup X \cup Y$. For every $< w, x, y > \in M$ we include in P the clause:
$w \wedge_{inc} x \wedge_{inc} y : [0; 0.5] \longleftarrow$.
Let $F : \mu = \bigwedge_{inc}\{a|a \in A\} : [0, 0.5^q]$.

It is easy to check that for every atom $a \in A$ $h_P(a) = [0, 1]$, for every pair of atoms (a, b) $h_P(a \wedge_{inc} b) = [0, 1]$ and for every triple (a, b, c) which is not constitute the head of some clause of P $h_P(a \wedge_{inc} b \wedge_{inc} c) = [0, 1]$. Of course, if $w \wedge_{inc} x \wedge_{inc} y$ is the head of some clause of P then $h_P(w \wedge_{inc} x \wedge_{inc} y) = [0, 0.5]$. Suppose now that $I \in$3-DM and M' is it's complete matching. Then F can be partitioned by q triples corresponding to the heads of clauses of P:
$F = \bigwedge_{inc}(w \wedge_{inc} x \wedge_{inc} y| < w, x, y > \in M'\}$.
This implies that $h_P(F) \subseteq c_i([0, 0.5], ..., [0, 0.5]) = [0, 0.5^q]$, and $P \models F : [0, 0.5^q]$.

On the other hand, if$P \models F : [0, 0.5^q]$ then $h_P(F) \subseteq [0, 0.5^q]$. It is possible only if F can be partitioned into q subformulas: $F = F_1 \wedge_{inc} \ldots \wedge_{inc} F_q$ such that each $F_i, 1 \leq i \leq q$, is a head of some clause of P. Indeed, any other partition of F which includes $k < q$ heads of clauses of P provides interval $[0, 0.5^k]$. But $[0, 0.5^k] \not\subseteq [0, 0.5^q]$.

Then the set $M' = \{< w_i, x_i, y_i > |F_i = w_i \wedge_{inc} x_i \wedge_{inc} y_i, i = 1, ..., q\}$ evidently is a complete matching and $I \in$3-DM. ■

Proof of Theorem 6.
Due to Lemma 4 any problem $''P \models F : \mu ?''$ for $P \in HPP_2$ can be reduced in polynomial time to polynomial number of entailment problems of the form $''P' \models F' : \mu' ?''$ for programs $P' \in HPP_{2,0}$. So we will suppose that $P \in HPP_{2,0}$.

We will prove the theorem for $F = a_1 \wedge_\rho ... \wedge_\rho a_n$, where $a_i \in B_L$ for $i = 1, ..., n$. The proof for formulas $F = a_1 \vee_\rho ... \vee_\rho a_n$, will be analogous. Without loss of generality one can assume that the width of F is even, i.e. $n = 2m$ (otherwise one can add a new true fact t in the query and in P: $P \models F : \mu \iff P \cup \{t : [1, 1]\} \models F \wedge_\rho t : \mu$).

Using algorithm LFP of theorem 2 we compute (in polynomial time) the least fixpoint
$h_P(a_i \wedge_\rho a_j) = \mu_{i,j} = [a_{i,j}, b_{i,j}]$ for any pair $1 \leq i < j \leq 2m$. Define now a new program $P' \in HPP_{2,0}$ consisting of $2m(2m - 1)$ clauses of the form $a_i \wedge_\rho a_j : \mu_{i,j}$. From proposition 1 it follows that $P \models F : \mu \iff P' \models F : \mu$. Now let $c_\rho = \langle c^1_\rho, c^2_\rho \rangle$. Define undirected graph $G = G(P', F) = (V, E)$ as follows: $V = \{a_1, ..., a_{2m}\}, E = \{(a_i, a_j)|1 \leq i < j \leq 2m\}$, and attach to it two weight functions: $w_1(a_i, a_j) = a_{i,j}$ and $w_2(a_i, a_j) = b_{i,j}$. Then the theorem follows from

Proposition 5 $P' \models F : [a, b] \iff \{G, w_1, a\} \in GWM_{max}(c^1_\rho)$ *and* $\{G, w_2, b\} \in GWM_{min}(c^2_\rho)$.

Proof $\Longrightarrow$ If $P' \models F : [a, b]$ then by proposition 1 there are two partitions of F into heads (or subformulas of heads) of clauses of P': $F = G_1 \wedge_\rho ... \wedge_\rho G_k = H_1 \wedge_\rho ... \wedge_\rho H_l$ such that $c^1_\rho(h_{P'}(G_1), ..., h_{P'}(G_k)) \geq a$ and $c^2_\rho(h_{P'}(H_1), ..., h_{P'}(H_l)) \leq b$ (where for any formula E we set $h_{P'}(E) = [h^1_{P'}(E), h^2_{P'}(E)]$). Now we can assume that $k = l = m$ and all G_i's and H_j's are the heads of clauses of P'. Indeed, if e.g. $G_1 = a_i$ and $G_2 = a_j$ then we can combine them into the head $a_i \wedge_\rho a_j$ because $c_\rho(h_{P'}(a_i), h_{P'}(a_j)) \subseteq h_{P'}(a_i \wedge_\rho a_j) = \mu_{i,j}$ and therefore $c^1_\rho(h^1_{P'}(a_i), h^1_{P'}(a_j)) \leq a_{i,j}$. Then it is easy to check that the first of these partitions $\{G_1, ..., G_m\}$ corresponds to the complete matching $\{e_p = (a_i, a_j)|G_p = a_i \wedge_\rho a_j,\ 1 \leq p \leq m\}$ which solves the matching problem $\{G, w_1, a\} \in GWM_{max}(c^1_\rho)$, and the second partition $\{H_1, ..., H_m\}$ corresponds to the complete matching $\{e'_p = (a_i, a_j)|G_p = a_i \wedge_\rho a_j,\ 1 \leq p \leq m\}$ which solves the matching problem $\{G, w_2, b\} \in GWM_{min}(c^2_\rho)$.
$\Longleftarrow$ It is easy to see that two complete matchings for problems $\{G, w_1, a\} \in GWM_{max}(c^1_\rho)$ and $\{G, w_2, b\} \in GWM_{min}(c^2_\rho)$ provides two partitions of F which satisfy conditions of (1) and (2) of proposition 1. Then by this proposition $P' \models F : [a, b]$. ■

Proof of Theorem 7.

Suppose that we have a problem of class $GWM_{max}(c^1)$. Then given a graph $G = \langle V, E\rangle$, weight function $w : E \rightarrow [0, 1]$, and a goal function c^1, one can define the composition function $c_\rho = (c^1, min)$ and construct the hp-program $P = P(G, w)$ which for every edge $e = (v_i, v_j) \in E$ includes the clause $v_i \wedge_\rho v_j : [w(e), 1]$. Then there exists a complete matching $\{e_1, ..., e_m\} \subseteq E$ such that $c^1(w(e_1), ..., w(e_m)) \geq B$ iff $P \models v_1 \wedge_\rho ... \wedge_\rho v_2m : [B, 1]$.

Analogously, one can reduce a problem of $GWM_{min}(c^2)$.

Proof of Theorem 9. As in the proof of theorem 6 lemma 4 allows us to reduce consistency problem for HPP_2 to consistency problem for $HPP_{2,0}$.

Let now $P \in HPP_{2,0}$ be a program over set $\mathcal{S}$ of p-strategies that uses atoms from the set $\mathcal{A} = \{A_1, \ldots, A_{2m}\}$ and only those. By lemma 6 it is sufficient to check the consistency of all formulas of form $F = A_1 *_\rho \ldots *_\rho A_{2m}$, $\rho \in \mathcal{S}$. As in theorem 6 we construct (in polynomial time) program $P' \in HPP_{2,0}$ which consists of all possible clauses of the form $A_i *_\rho A_j : h_P(A_i *_\rho A_j) \longleftarrow$. From proposition 1 it follows that P is inconsistent iff for some strategy $\rho \in \mathcal{S}$ for formula $F = A_1 *_\rho \ldots *_\rho A_{2m}$ there exist two partitions of F into subformulas $F = G_1 *_\rho \ldots *_\rho G_m$ and $F = H_1 *_\rho \ldots *_\rho H_m$ where all $G_1, \ldots, G_m$ and $H_1, \ldots, H_m$ are heads of clauses in P', such that there exist such numbers $0 \leq b < a \leq 1$ that
$c_\rho(h_P(G_1), \ldots h_P(G_m)) \subseteq [0, b]$ and
$c_\rho(h_P(H_1), \ldots h_P(H_m)) \subseteq [a, 1]$.

To verify the existence of such partitions it is sufficient to optimize two General Weighted Matching Problems. Indeed, let graph $G(P', F) = \langle V, E\rangle$ and two weight functions w_1 and w_2 be defined as in theorem 6. Then using standard dichotomy technique one can find such minimal b' and maximal a' that $\{G, w_2, b'\} \in GWM_{min}(c^2_\rho)$ and $\{G, w_1, a'\} \in GWM_{max}(c^1_\rho)$ and check if $b' < a'$. ■

Proof of Theorem 10. Algorithm **InCon** is easily seen to be in NP.

To prove that inconsistency problem is NP-hard we notice that entailment problem for consistent HPPs is reducible in polynomial time to the inconsistency problem. To see why, suppose P is consistent. Then it is easy to check that for any annotated formula $F : \mu$ $P \models F : \mu$ iff the following program P' is inconsistent. $P' = P \cup \{z : [0; 0.1] \leftarrow F : \mu\} \cup \{z : [0.2; 0.3] \leftarrow F : \mu\}$.

Now it follows from Theorem 5 (on NP-completeness of the entailment problem) that inconsistency problem is NP-hard even for class $HPP_{3,0}$. ■

Proof of Theorem 11.

We prove soundness of the system HGR_P by showing that every axiom and rule of the system preserves truth.

Let h be any model of program P, i.e. $h \models P$.

- Axioms. Let $A \in B_L$. By definition of a formula function $h(A) \subseteq [0, 1]$, therefore, $h \models A : [0, 1]$.
- Program. Let $F : \mu \longleftarrow G_1 : \mu_1 \wedge \ldots \wedge G_k : \mu_k$ be a rule of P. Let $h \models G_1 : \mu_1, \ldots, h \models G_k : \mu_k$. Then, $h \models G_1 : \mu_1 \wedge \ldots \wedge G_k : \mu_k$. Since $h \models F : \mu \longleftarrow G_1 : \mu_1 \wedge \ldots \wedge G_k : \mu_k$, by definition of satisfaction we conclude, $h \models F : \mu$.
- A-Composition. Let $A_1, A_2 \in B_L$ and $h \models A_1 : \mu_1$ and $h \models A_2 : \mu_2$. Let ρ be some p-strategy from $\mathcal{S}$. Then, by the definition of a formula function, $h(A_1 *_\rho A_2) \subseteq c_\rho(h(A_1), h(A_2))$. By the *monotonicity* property of composition function, and since $h(A_1) \subseteq \mu_1$ and $h(A_2) \subseteq \mu_2$, $h(A_1 *_\rho A_2) \subseteq c_\rho(h(A_1), h(A_2)) \subseteq c_\rho(\mu_1, \mu_2)$ and therefore, $h \models (A_1 *_r hoA_2) : c_\rho(\mu_1, \mu_2)$.
- F-Composition. Let $A_1, \ldots A_, B_1 \ldots B_l \in B_L$ and $h \models (A_1 *_\rho \ldots *_\rho A_k) : \mu_1$ and $h \models B_1 *_r ho \ldots *_\rho B_l : \mu_2$ for some p-strategy $\rho \in \mathcal{S}$.

  Then, by the definition of a formula function, $h(A_1 *_\rho \ldots *_\rho A_k *_\rho B_1 *_\rho \ldots *_\rho B_l) \subseteq c_\rho(h(A_1 *_\rho \ldots *_\rho A_k), h(B_1 *_\rho \ldots *_\rho B_l))$. By the *monotonicity* property of composition function, and since $h(A_1 *_\rho \ldots *_\rho A_k) \subseteq \mu_1$ and $h(B_1 *_\rho \ldots *_\rho B_l) \subseteq \mu_2$ we get $h(A_1 *_\rho \ldots *_\rho A_k *_\rho B_1 *_\rho \ldots *_\rho B_l) \subseteq c_\rho(\mu_1, \mu_2)$, i.e., $h \models (A_1 *_\rho \ldots *_\rho A_k *_\rho B_1 *_\rho \ldots *_\rho B_l) : c_\rho(\mu_1, \mu_2)$.
- Decomposition. Let $h \models H : \mu$ and $H = F \oplus_\rho G$. By the *Commutativity* property of formula functions, $h(H) = h(F *_\rho G) \subseteq \mu$. Then, by the *Decomposition* property of formula functions, $h(F) \subseteq md_\rho(h(F *_\rho$

$G)$). Since $h(F *_\rho G) \subseteq \mu$, $md_\rho(h(F *_\rho G)) \subseteq md_\rho(\mu)$[1] Therefore $h \models F : md_\rho(h(F *_\rho G))$ and $h \models F : md_\rho(\mu)$.

- Clarification. Let $h \models F : \mu_1$ and $h \models F : \mu_2$. Then $h(F) \subseteq \mu_1$ and $h(F) \subseteq \mu_2$. Therefore $h(F) \subseteq \mu_1 \cap \mu_2$, hence, $h \models F : \mu_1 \cap \mu_2$.
- Exchange. Let $h \models (A_1 *_\rho \ldots *_\rho A_k) : \mu$ and let $B_1, \ldots, B_k$ be a permutation of $A_1, \ldots, A_k$. By the *commutativity* property of formula functions $h(B_1 *_\rho \ldots *_\rho B_k) = h(A_1 *_\rho \ldots *_\rho A_k) \subseteq \mu$. Then $h \models (B_1 *_\rho \ldots *_\rho B_k) : \mu$.
- Interval Weakening. Let $h \models F : \mu$ and $\mu \subseteq \mu_1$. Since $h(F) \subseteq \mu$, also $h(F) \subseteq \mu_1$ and therefore $h \models F : \mu_1$.$\blacksquare$

Proof of Proposition 2.
Let $P \in HPP^0$ and $P \models F : [a, b]$. Let us fix two partitions of F, $F = G_1 *_\rho \ldots *_\rho G_k$ and $F = H_1 *_\rho \ldots *_r hoH_l$ that and numbers $x_1, \ldots, x_k$, $y_1, \ldots, y_l$ which satisfy proposition 1. Then a canonical proof $C_1, \ldots, C_r = F : [a, b]$ of $F : [a, b]$ can be constructed in the following way:

1. The last formula, $C_r = F : [a, b]$ is obtained by the *interval weakening* rule applied to formula $C_{r-1} = F : [a', b']$ for some $[a', b'] \supseteq [a, b]$;
2. The inference rule of *clarification* is used only once in the entire proof to obtain a formula $C_{r-1} = F : [a', b']$ from two formulas $C_{r-2} = F : [a', 1]$, $C_{r-3} = F : [0, b']$.
3. Formula $C_{r-2} = F : [a', 1]$ is obtained via the rule of *exchange* from formula $C_{r-4} = F' : [a', 1]$ where F' is a formula which is a permutation of atoms in F.
4. Formula $C_{r-3} = F : [0, b']$ is obtained via the rule of *exchange* from formula $C_{r-5} = F'' : [0, b']$ where F'' is a formula which is a permutation of atoms in F.
5. For some integer $s < r - 5$ such that $C_1, \ldots C_s, C_{r-5} = F'' : [0, b']$ is a proof of $F'' : [0, b']$ and $C_{s+1}, \ldots C_{r-6}, C_{r-4} = F' : [a', 1]$ is proof of $F' : [a', 1]$.
6. In partition $F' = G_1 *_\rho \ldots *_\rho G_k$, for every i if G_i is a head of a clause $G_i : [a'_i, b'_i] \longleftarrow$ the proof includes two formulas:

   (a) $G_i : [a'_i, b'_i]$ (as an axiom);

   (b) $G_i : \mu_i$, $\mu_i = [x_i, 1]$ (obtained from previous formula by the rule of *interval weakening*).

   If G_i is a subformula of some head of a clause $H = G_i \oplus_\rho H' : [a'_i, b'_i] \longleftarrow$ of P the proof includes the following four formulas:

   (a) $H : [a'_i, b'_i]$ (as an axiom);

   (b) $G_i *_\rho H' : [a'_i, b'_i]$ (obtained from previous formula by the rule of *exchange*);

   (c) $G_i : [c_i, d_i]$ where $[c_i, d_i] = md_\rho([a'_i, b'_i])$ (obtained by *decomposition* rule from previous formula);

   (d) $G_i : \mu_i$, $\mu_i = [x_i, 1]$ ((obtained from previous formula by the rule of *interval weakening*).

   After all the formulas $G_i : \mu_i$ ($i \in \{1, \ldots, k\}$) are derived, the proof of $F' : [a', 1]$ will contain the formula $G_1 *_\rho G_2 : c_\rho(\mu_1, \mu_2)$ obtained from $G_1 : \mu_1$ and $G_2 : \mu_2$ by either *A-composition* or *F-composition* rule (depending on whether both G_1 and G_2 are atomic) and for all $2 < i \leq k$ the formula $G_1 *_\rho \ldots *_\rho G_i : c_\rho(\mu_1, \ldots, \mu_i)$ which is obtained by the rule of *F-composition* from formulas $G_1 *_\rho \ldots *_\rho G_{i-1} : c_\rho(\mu_1, \ldots, \mu_{i-1})$ and $G_i : \mu_i$.

   The last of such formulas, $G_1 *_\rho \ldots *_\rho G_k : c_\rho(\mu_1, \ldots, \mu_k) = F' : [0, a'] = C_{r-4}$.
7. The proof of $C_{r-5} = F'' : [0, b']$ is constructed from the partition $F'' = H_1 *_\rho \ldots *_r hoH_l$ in a manner similar to the construction of the proof of $F' : [0, a'] = C_{r-4}$ on previous step.

[1]Let $[a, b] \subseteq [a', b']$. If ρ is a conjunctive p-strategy, then $md_\rho([a, b]) = [a, 1] \subseteq [a', 1] = md_\rho([a', b'])$. If ρ is a disjunctive p-strategy then $md_\rho([a, b]) = [0, b] \subseteq [0, b'] = md_rho([a', b'])$.

We can now compute the length of the proof constructed as above. It is bounded by $5(k+l)+6 \leq 10|F|+6$. ■

Proof of Theorem 12.
If $P \in HPP$ then we can construct the proof of $F : \mu$ in two stages. At the first stage we will derive all formulas of $P_0, P_1, \ldots P_{2m}$ (as defined in theorem 3.2.2) where m is the number of clauses in P. If a clause $C_j \in Fired_P(i+1) \setminus Fired_P(i)$ and $C_j = D : \nu \longleftarrow D_1 : \mu_1 \wedge \ldots \wedge D_s : \mu_s$ then $P_i \models D_l : \mu_l$ for all $1 \leq l \leq s$ and we include in our proof the proofs of all $D_l : \mu_l$ and using them will derive $D : \nu$ via the rule *program* which corresponds to the clause C_j.

At the second stage we use the construction from proposition 2 to derive $F : \mu$ using the formulas of P_{2m} derived during the first stage.

The number of formulas of form $D_i : \mu_i$ we derived at the first stage of the proof less than the length of P. The length of each D_i is less then the length of P as well. So, the total length of the proof is less then $O(|P|^2 + |F|)$.
■