On Normal Forms and Equivalence for Logic Programs

James Harland
Department of Computer Science
University of Melbourne
Parkville, 3052, Australia
jah@cs.mu.oz.au

Abstract

It is known that larger classes of formulae than Horn clauses may be used as logic programming languages. One such class of formulae is *hereditary Harrop formulae*, for which an operational notion of provability has been studied, and it is known that operational provability corresponds to provability in intuitionistic logic. In this paper we discuss the notion of a normal form for this class of formulae, and show how this may be given by removing disjunctions and existential quantifications from programs. Whilst the normal form of the program preserves operational provability, there are operationally equivalent programs which are not intuitionistically equivalent. As it is known that classical logic is too strong to precisely capture operational provability for larger classes of programs than Horn clauses, the appropriate logic in which to study questions of equivalence is an intermediate logic. We explore the nature of the required logic, and show that this may be obtained by the addition of the *Independence of Premise* axioms to intuitionistic logic. We show how equivalence in this logic captures the notion of operational provability, in that logically equivalent programs are operationally equivalent. This result suggests that the natural logic in which to study logic programming is a slightly stronger constructive logic than intuitionistic logic.

1 Introduction

It is well known that variables which appear in Horn clauses may be considered to be universally quantified at the front of the clause, so that the Prolog clause `p(X) :- q(X,Y)` may be considered a shorthand for the formula $\forall x \forall y \; p(x) \subset q(x,y)$. However, we may also consider variables which do not appear in the head of the clause as being existentially quantified at the front of the *body*, so that the above clause is equivalent to $\forall x \, p(x) \subset \exists y \, q(x,y)$. Thus we may consider Horn clauses to have the expressive power of the latter style (i.e. in which existential quantifiers are allowed in the bodies of clauses), but we need only implement the former style. In this way we may think of the more restrictive definition as a normal form for programs corresponding to the second definition.

A more general class of programs and goals was given in [11], the definition of which is given below.

$$D := A \mid \forall x D \mid D_1 \wedge D_2 \mid G \supset A$$
$$G := A \mid \exists x G \mid G_1 \wedge G_2 \mid G_1 \vee G_2 \mid D \supset G$$

where A denotes an atom. A program is a set of closed D formulae and a goal is a closed G formula.

In [11], it was shown how disjunctions may be removed from programs in the above class, in that for any program which contains a disjunction, there is a program with the same operational behaviour, which, although usually much larger, contains no disjunctions. We may think of this as removing redundancies from the program, and so programs may be given a simpler definition, in that there is a smaller class of formulae with the same expressive power. In this paper we extend this result by showing that existential quantifiers may be similarly removed from programs. Thus, the normal form will be a program which contains no disjunctions and no existential quantifications.

Another interesting property of Horn clauses [7, 8] when used as a programming language is that the class of goals used, i.e. (existentially quantified) conjunctions of atoms, sometimes seems to be somewhat restrictive, as it does not allow goals to inspect the structure of the program. For example, the success or failure of any Horn clause goal is the same no matter which of the following two programs is used.

$$\begin{array}{lll} p & \qquad\qquad & p \\ r \supset q & & q \supset r \end{array}$$

The problem is that conjunctions of atoms only "look" at the head of the clause, rather than the body. Were we able to ask a query such as "add q to the program and then ask r", we would indeed observe different behaviour from each of these two logically inequivalent programs.

Hence it seems that a larger class of goals (and/or a larger class of programs), such as those mentioned above, may lead to notions of operational equivalence which are more in tune with logical equivalence. It is also not at all clear what the appropriate notion of logical equivalence will be, as it is known that classical logic, intuitionistic logic and minimal logic all coincide on the Horn clause fragment [12].

The larger class of programs and goals mentioned above indeed allows additions to the program to be made during computation, and hence will be useful for determining an appropriate notion of operational equivalence for logic programs. An operational notion of provability, denoted $\vdash_o$, may be given for this class of formulae [12, 11], and $\vdash_o$ may be thought of as a generalisation of the notion of SLD-resolution [8]. The nature of proofs in this system, known as uniform proofs, was also discussed in [12].

It is known that the relation $\vdash_o$ corresponds to intuitionistic provability, in that if P is a program and G is a goal, then $P \vdash_o G$ iff $P \vdash_I G$. Hence intuitionistically equivalent formulae have the same operational behaviour. However, the converse is not true. For example the nature of $\vdash_o$ is such that $P \vdash_o D \supset (G_1 \vee G_2)$ iff $P \vdash_o (D \supset G_1) \vee (D \supset G_2)$, and these two goals are not intuitionistically equivalent. Similarly there are programs P and P' for which $P \vdash_o G$ iff $P' \vdash_o G$ but P and P' are not intuitionistically equivalent [4]. Moreover, the transformation to the normal form does not preserve intuitionistic equivalence, in that the original program and its normal form are not necessarily intuitionistically equivalent. Thus the natural logic for this class of programs needs to be stronger than intuitionistic logic. As it is known that classical logic is too strong for this purpose [11, 12], this leads us to intermediate logics. In this paper we also show how an appropriate logic may be obtained by adding the *Independence of Premise* (IP) rules to intuitionistic logic. We may think of this logic as being "fully abstract" with respect to $\vdash_o$, in that logical equivalence implies operational equivalence and vice-versa. In particular, a program and its normal form are equivalent in this logic.

This paper is organized as follows. In Section 2 we provide the technical preliminaries, and in Section 3 we show how a normal form, in which disjunctions and existential quantifiers may be eliminated from programs, may be given, and in Section 4 we discuss the relevance of the normal form for computation. Section 5 presents a useful "minimality" result for operational equivalence, and in Section 6 we discuss the intermediate logic and show the full abstraction theorem. Finally in section 7 we discuss some further work and remaining issues.

Due to space limitations, it is not possible to include proofs of all the results in this paper. Complete proofs of all results stated here and further discussion may be found in [2, 4, 5].

2 Preliminaries

Definition 2.1 *D_{iH} and G_{iH} formulae are given by the grammar*

$$D := A \mid \forall x D \mid D_1 \wedge D_2 \mid G \supset A$$
$$G := A \mid \exists x G \mid G_1 \wedge G_2 \mid G_1 \vee G_2 \mid D \supset G$$

We refer to this class of formulae as implicative Horn formulae.[1]

D_{hhf} and G_{hhf} formulae are given by the grammar

$$D := A \mid \forall x D \mid D_1 \wedge D_2 \mid G \supset A$$
$$G := A \mid \exists x G \mid \forall x G \mid G_1 \wedge G_2 \mid G_1 \vee G_2 \mid D \supset G$$

[1] In earlier work we have used the term D_{mod} and G_{mod} formulae; however, implicative Horn formulae seems to be more descriptive name.

We refer to this class of formulae as hereditary Harrop formulae*.*

In either case, we refer to the D formulae as definite formulae *and the G formulae as* goal formulae*.*

A program *is a set of closed definite formulae, and a* goal *is a closed goal formula.*

In this paper we will be mainly concerned with implicative Horn formulae, although we will refer to hereditary Harrop formulae at times.

In [12] it is shown how an operational notion of proof may be given for both of the above classes of programs and goals. Such proofs are known as *uniform* proofs, and a corresponding operational notion of provability $\vdash_o$ may be defined as follows.

We assume the existence of a finite set of constant and function symbols, and a countable set of variables.

Definition 2.2 *We refer to the set of all ground terms as the* Herbrand universe*, denoted by $\mathcal{U}$.*

We refer to the set of all terms (not necessarily ground) as the general Herbrand universe*, denoted by $\mathcal{U}'$.*

As noted in [12], we may think of a definite formula in much the same way that we think of a Horn clause. We denote the substitution of the term t for the variable x via $[t/x]$.

Definition 2.3 *We define a mapping $\mid - \mid$ from sets of definite formulae to sets of definite formulae by cases as follows:*

$$\begin{array}{lcl} \mid \{D_1, \ldots, D_n\} \mid & = & \bigcup_{i=1}^{n} \mid D_i \mid \\ \mid A \mid & = & \{A\} \\ \mid G \supset A \mid & = & \{G \supset A\} \\ \mid \forall x D \mid & = & \bigcup_{t \in \mathcal{U}'} \mid D[t/x] \mid \\ \mid D_1 \wedge D_2 \mid & = & \mid D_1 \mid \cup \mid D_2 \mid \end{array}$$

Definition 2.4 *Let P be a D_{hhf} program. Then we define $\vdash_o$ as the least relation satisfying*

- $P \vdash_o A$ *iff* $A \in \mid P \mid$ *or* $\exists G \supset A \in \mid P \mid$ *such that* $P \vdash_o G$
- $P \vdash_o G_1 \vee G_2$ *iff* $P \vdash_o G_1$ *or* $P \vdash_o G_2$
- $P \vdash_o G_1 \wedge G_2$ *iff* $P \vdash_o G_1$ *and* $P \vdash_o G_2$
- $P \vdash_o \exists x G$ *iff* $P \vdash_o G[t/x]$ *for some* $t \in \mathcal{U}'$
- $P \vdash_o \forall x G$ *iff* $P \vdash_o G[y/x]$ *where* y *is not free in* P *or* G

- $P \vdash_o D \supset G$ *iff* $P \cup \{D\} \vdash_o G$

In [12] it is shown how the above relation corresponds to the notion of *uniform proof*, i.e. a proof in which the outermost connective of the goal is introduced in the previous step. Further discussion on this point is beyond our scope; for now we note that it is known that the above relation corresponds to a well-understood notion of proof. It was also shown in [12] that $P \vdash_o G$ iff $P \vdash_I G$ where $\vdash_I$ denotes intuitionistic provability.

3 A Normal Form for Programs

In this section we show how any D_{iH} program P is operationally equivalent to a program P' containing no disjunctions or existential quantifiers, i.e. that $P \vdash_o G$ iff $P' \vdash_o G$. This process is inspired by the intuitionistic equivalences

$$(G_1 \vee G_2) \supset A \equiv_I (G_1 \supset A) \wedge (G_2 \supset A)$$
$$(\exists x G) \supset A \equiv_I \forall x (G \supset A)$$

where x does not appear in A.

As in [11], $\mathrm{dnf}(G)$ and $\mathrm{dfnf}(P)$ denote respectively the *disjunctive normal form* of a goal G and the *disjunction-free normal form* of a program P. The definitions in [11] are given in a slightly modified form here, so that we may consider $\mathrm{dnf}(G)$ as a formula, rather than a set. We introduce along similar lines the *existential normal form* of a goal and the *existential-free normal form* of a program, here denoted by enf and efnf respectively. We will assume that each existentially quantified variable is unique, so that a goal such as $\exists x\, p(x) \wedge \exists x\, q(x)$ is written as $\exists x\, p(x) \wedge \exists y\, q(y)$. Note that goals must be closed formulae, and so cannot contain free variables.

Definition 3.1 *Let D be a D_{iH} definite formula and let G be a G_{iH} goal formula. Then we define* $\mathrm{dfnf}(D)$, $\mathrm{efnf}(D)$, $\mathrm{dnf}(G)$ *and* $\mathrm{enf}(G)$ *as follows:*

$\mathrm{enf}(G) = \exists \tilde{x}\ \mathrm{enf}'(G)$	$\mathrm{dnf}(G) = \bigvee \mathrm{dnf}'(G)$
$\mathrm{enf}'(A) = A$	$\mathrm{dnf}'(A) = \{A\}$
$\mathrm{enf}'(G_1 \vee G_2) = \mathrm{enf}'(G_1) \vee \mathrm{enf}'(G_2)$	$\mathrm{dnf}'(G_1 \vee G_2) = \mathrm{dnf}'(G_1) \cup \mathrm{dnf}'(G_2)$
$\mathrm{enf}'(G_1 \wedge G_2) = \mathrm{enf}'(G_1) \wedge \mathrm{enf}'(G_2)$	$\mathrm{dnf}'(G_1 \wedge G_2) = \{G' \wedge G'' \mid G' \in \mathrm{dnf}'(G_1), G'' \in \mathrm{dnf}'(G_2)\}$
$\mathrm{enf}'(\exists x G) = \mathrm{enf}'(G)$	$\mathrm{dnf}'(\exists x G) = \{\exists x G' \mid G' \in \mathrm{dnf}'(G)\}$
$\mathrm{enf}'(D \supset G) = \mathrm{efnf}(D) \supset \mathrm{enf}'(G)$	$\mathrm{dnf}'(D \supset G) = \{\mathrm{dfnf}(D) \supset G' \mid G' \in \mathrm{dnf}'(G)\}$
$\mathrm{efnf}(A) = A$	$\mathrm{dfnf}(A) = A$
$\mathrm{efnf}(G \supset A) = \forall \tilde{x}\ (\mathrm{enf}'(G) \supset A)$	$\mathrm{dfnf}(G \supset A) = \bigwedge \{G' \supset A \mid G' \in \mathrm{dnf}'(G)\}$
$\mathrm{efnf}(D_1 \wedge D_2) = \mathrm{efnf}(D_1) \wedge \mathrm{efnf}(D_2)$	$\mathrm{dfnf}(D_1 \wedge D_2) = \mathrm{dfnf}(D_1) \wedge \mathrm{dfnf}(D_2)$
$\mathrm{efnf}(\forall x D) = \forall x\ \mathrm{efnf}(D)$	$\mathrm{dfnf}(\forall x D) = \forall x\ \mathrm{dfnf}(D)$

where $\tilde{x}$ are all the existentially quantified variables of G.

Let P be a D_{iH} program. Then we define $\text{dfnf}(P) = \{\text{dfnf}(D) \mid D \in P\}$, *and* $\text{efnf}(P) = \{\text{efnf}(D) \mid D \in P\}$.

An example of each of these transformations is given below.

P	$\text{dfnf}(P)$	$\text{efnf}(P)$
$s \subset (\exists x p(x) \vee q(x)) \wedge r$	$s \subset \exists x p(x) \wedge r$	$\forall x\ s \subset (p(x) \vee q(x)) \wedge r$
	$s \subset \exists x q(x) \wedge r$	

We may think of the process described by $\text{dnf}(G)$ as pushing all disjunctions to the top level of the goal, so that $\text{dnf}(G)$ is a disjunction of disjunction-free formulae. We may then apply the identity above to obtain the disjunction-free program. A similar remark applies to $\text{enf}(G)$; all existential quantifiers are pushed to the top of the formula, and so we may apply the corresponding identity to obtain a program free of existential quantifications. Thus the above equivalences imply that

$$\text{dfnf}(G \supset A) = \bigwedge\{G' \supset A \mid G' \in \text{dnf}'(G)\} \equiv_I \text{dnf}(G) \supset A$$
$$\text{efnf}(G \supset A) = \forall(\text{enf}'(G \supset A)) \equiv_I \text{enf}(G) \supset A$$

In this way we may specify the program equivalent to D which contains no occurrences of $\exists$ or $\vee$ as $\text{efnf}(\text{dfnf}(D))$. We will refer to this program as $\text{normal}(D)$. Similarly we will refer to $\text{enf}(\text{dnf}(G))$ as $\text{normal}(G)$. The formal results which establish the desired operational equivalence of D and $\text{normal}(D)$ are given below.

Proposition 3.1 *Let P be a D_{iH} program, and let G be a G_{iH} goal. Then*

1. $P \vdash_o G \Leftrightarrow \text{dfnf}(P) \vdash_o G \Leftrightarrow P \vdash_o \text{dnf}(G) \Leftrightarrow \text{dfnf}(P) \vdash_o \text{dnf}(G)$
2. $P \vdash_o G \Leftrightarrow \text{efnf}(P) \vdash_o G \Leftrightarrow P \vdash_o \text{enf}(G) \Leftrightarrow \text{efnf}(P) \vdash_o \text{enf}(G)$

The equivalence of the first and the fourth cases of 1 was shown in [11].

Corollary 3.2 *Let P be a D_{iH} program, and G be a G_{iH} goal. Then*

$$P \vdash_o G \Leftrightarrow \text{normal}(P) \vdash_o G \Leftrightarrow P \vdash_o \text{normal}(G) \Leftrightarrow \text{normal}(P) \vdash_o \text{normal}(G)$$

Thus the corollary above not only ensures that for any program there exists an operationally equivalent program which contains no disjunctions or existential quantifiers, but also shows how such a program may be derived from the original one.

4 Normal Forms and Computation

Whilst the normalisation results above are not particularly deep, they provide an insight into the way that computation is performed in this setting. In particular, we may think of the process of converting a program into its normal form as a transformation on programs which takes place at compile time, and so we may think of the above processes as statically converting the program into a more specific form.

We may also apply the above results to the goal, in that given program P and a goal G we have that $P \vdash_o G$ iff $P \vdash_o \text{normal}(G)$. Moreover, we may write normal(G) as $\exists(G_1 \vee \ldots \vee G_n)$ where each of the G_i are goal formulae which do not contain disjunctions or existential quantifiers, and $\exists(G)$ is the existential closure of all free variables in G. Now as the relation $\vdash_o$ has the disjunctive and existential properties, we have that $P \vdash_o \exists(G_1 \vee \ldots \vee G_n)$ iff $P \vdash_o G_i\theta$, for some $1 \leq i \leq n$ and some substitution θ such that $G_i\theta$ is ground. In fact, we may be more precise than this. It is well known in the logic programming literature that a Prolog interpreter returns not only an answer substitution for the (existentially quantified) variables of the goal, but also a *most general* answer substitution, which essentially means that, without changing the clauses used in the derivation, we cannot find a more general way of representing the successful instances of the goal [2].

A similar property holds in our more general framework, in that we can find a most general substitution θ such that the success of $\exists(G_1 \vee \ldots \vee G_n)$ implies the success of $G_i\theta$, and moreover the success of every instance G' of $G_i\theta$, including those instances in which the variables of $G_i\theta$ are renamed. Hence we have that $P \vdash_o \exists(G_1 \vee \ldots \vee G_n)$ iff $P \vdash_o \forall(G_i\theta)$. [3] Now as G_i may any one of the formulae $G_1, \ldots, G_n$, in our framework we need not only the notion of an answer substitution, but a more general notion of an *answer formula*, in that if $P \vdash_o \forall(G_i\theta)$, then $\forall(G_i\theta)$ is an answer formula for $\exists(G_1 \vee \ldots \vee G_n)$. For example, consider the program $P = \{p(a), q(b)\}$. It is easy to see that the goal $G = \exists x\, p(x) \vee \exists y\, q(y)$ succeeds, with the answer formulae being $p(a)$ and $q(b)$. In this way an answer formula may be thought of as a *realizer* of the original goal in the sense of Kleene [6], in that the answer formula supplies information, missing from the statement that $P \vdash_o G$, which is sufficient to establish the truth of the statement. Thus the computation process determines a formula $\forall(G_i\theta)$ of sufficient precision that it satisfies the relatively weak constraint $\exists(G_1 \vee \ldots \vee G_n)$ represented by the original goal. Naturally there may be many realizers for a given goal, and the less "definite" the goal (i.e. the looser the constraint), the

[2] Of course, other such maximally general answers may be found by unifying the goal with different clauses in the program.

[3] Note that the universal quantification of the goal gives a particular neat characterisation of the property that if a goal G succeeds with most general answer substitution θ, then every instance of $G\theta$ succeeds. In this way we may capture an important property of a most general answer substitution simply by enlarging the class of formulae that may be used as goals.

larger is the number of answer formulae which satisfy the constraint. In this way indefinite formulae may be used to extract related pieces of definite information from the program.

An interesting footnote to this "conversion" of an existential quantifier into a more specific universal quantifier is to note that Horn clauses may be used as hereditary Harrop formulae goals, and hence a Horn clause (or in fact any closed definite formula) is its own answer formula. For example, consider the clause $\{\forall x\ p(f(x))\}$. The goal $\exists y\ p(y)$ succeeds with the answer substitution being $y \leftarrow f(x)$, and so the answer form of the goal is $\forall x\ p(f(x))$. Thus the process of converting the normal form of a goal into an answer formula makes it clear that computation in this setting is a matter of converting an indefinite constraint into a definite answer.

The answer formula is a particularly definite piece of information; in fact, it is not hard to rewrite $\forall(G_i\theta)$ as a definite formula (see [2, 3, 4]). Hence we may add answer formulae to the program, and hence *memoise* [10] successful goals. Memoisation is a well known technique, and may considerably improve performance, as we do not need to re-compute previously computed results. Whilst we cannot in general store successful goals themselves, this analysis shows that we can in fact store something more useful — the answer formulae — and the normal form of a goal partially shows how the answer formulae are related to the original goal. In this sense the normalisation process may be thought of as a pre-computation, in that by converting a program and a goal into normal form before computation we can save ourselves work which would be performed later anyway.

Hence the normalisation results, whilst simple, provide a useful insight into the computation process. It is possible to derive some stronger results than those above, in that it is possible to show that the above transformations not only preserve success, but also preserve (finite) failure. In other words, if we use $\vdash_f$ to denote (finite) failure, $P \vdash_f G$ iff $\mathrm{dfnf}(P) \vdash_f G$ iff $P \vdash_f \mathrm{dnf}(G)$ iff $\mathrm{dfnf}(P) \vdash_f \mathrm{dnf}(G)$, and similarly for the existential case. This means that the above results will still hold in the presence of Negation as Failure. More detail (including proofs) may be found in [2].

5 "Minimal" Operational Equivalence

As mentioned above, we may think of $\mathrm{normal}(P)$ as a transformation of P which produces an "equivalent" program P', in that every goal derivable from one is derivable from the other. Maher [9] has shown that there are several meaningful conceptions of equivalence between logic programs; one obvious such notion is to consider two programs P_1 and P_2 to be *operationally equivalent* if for every goal G, $P_1 \vdash_o G \Leftrightarrow P_2 \vdash_o G$. This may be thought of as treating the programs as two black boxes, so that the only way that we may distinguish between them is by how they react to queries from the outside world. We will denote the operational equivalence of two programs

P_1 and P_2 as $P_1 \equiv_o P_2$.

At this point, it is instructive to consider the question of "minimal testing" for observational equivalence. More formally, the question is what is the smallest class of goals for which operational equivalence needs to be established in order to show operational equivalence for all goals. It is easy to see that this smallest class must properly include all atomic goals. However as noted above, atomic goals alone are not enough, and suggests that implications are needed in goals in such a minimal class. In fact, implications are all that is necessary, as is shown in the proposition below.

Proposition 5.1 *Let Q formulae be defined by*

$$Q := A \mid D \supset Q$$

where D is any D_{iH} definite formula.

Let D_1 and D_2 be D_{iH} programs, and let G range over all G_{hhf} goals. Then

$$(\forall Q \ D_1 \vdash_o Q \Rightarrow D_2 \vdash_o Q) \Rightarrow (\forall G \ D_1 \vdash_o G \Rightarrow D_2 \vdash_o G)$$

This proposition turns out to be remarkably useful in proving subsequent results. Note that as $P \vdash_o Q$ iff $P \vdash_o \text{normal}(Q)$, we in fact need only consider formulae which contain no existential quantifiers or disjunctions.

We may think of the above result as establishing that operational equivalence for atoms and implications is sufficient to establish operational equivalence for all goals.

Alternatively, the above result may be interpreted as stating that if we can establish that all *extensions* of two programs enable the same atoms to be derived, then we have established operational equivalence. Thus if we cannot distinguish between two programs by arbitrary mutual extensions of them, even when the only goals which may be asked are atomic, then the two programs are operationally equivalent. There may be weaker conditions under which operational equivalence holds, but it would seem that any weakening of this condition would come from restricting the extensions that may be made to the programs.

6 Logical Equivalence

We saw above that intuitionistic logic is clearly important in the study of operational equivalence for hereditary Harrop formulae. In fact, it is easy to see that intuitionistic equivalence implies operational equivalence, as if $P_1 \vdash_I P_2$ and $P_2 \vdash_o G$, then as $P_2 \vdash_o G$ implies that $P_2 \vdash_I G$, we have that $P_1 \vdash_I G$, i.e. $P_1 \vdash_o G$. The corresponding case for $P_2 \vdash_I P_1$ is similar.

A natural question to ask at this point is whether programs which are operationally equivalent are logically equivalent, i.e. if $\forall G \ D_1 \vdash_o G$ iff $D_2 \vdash_o G$, does it follow that $D_1 \equiv_I D_2$? The answer is negative, as whilst the transformations above preserve operational equivalence, they do not preserve intuitionistic equivalence. A counterexample, due to Dale Miller, is given as follows:

D_1	$\mathrm{dfnf}(D_1)$
$(r \supset (p \vee q)) \supset s$	$((r \supset p) \supset s) \wedge ((r \supset q) \supset s)$

A Kripke model in which $\mathrm{dfnf}(D_1)$ is true but D_1 is not is given by a model with three worlds w_1, w_2 and w_3 where $w_1 \leq w_2$ and $w_1 \leq w_3$, and the true atomic formulae at w_1, w_2 and w_3 are $\emptyset$, $\{r, p, s\}$ and $\{r, q, s\}$ respectively. D_1 is not true as it is not true at w_1.

A similar counterexample may be given for the program $(p \supset \exists x q(x)) \supset r$.

Thus operational equivalence is not strong enough to establish (intuitionistic) logical equivalence. One way to interpret this result is that the natural choice of logic for programming in this context needs to have a stronger consequence relation than $\vdash_I$, so that operationally equivalent programs are logically equivalent. Hence, any natural logic of equivalence for logic programs will need to be stronger than intuitionistic logic. [4] Note that classical logic is too strong for this purpose; for example, the two programs below are equivalent in classical logic, but they are not operationally equivalent.

$$(q \vee (p \supset p)) \supset r \qquad\qquad ((p \supset q) \vee p) \supset r$$

The problem is that whilst $q \vee (p \supset p)$ and $(p \supset q) \vee p$ are classically equivalent, $q \vee (p \supset p)$ succeeds (i.e. $\vdash_o q \vee (p \supset p)$) whereas $(p \supset q) \vee p$ fails.

Thus any intermediate logic, say I', appropriate for this application will need to satisfy the following equivalences:

$$D \supset (G_1 \vee G_2) \equiv_{I'} (D \supset G_1) \vee (D \supset G_2)$$
$$D \supset \exists x\, G \equiv_{I'} \exists x\, D \supset G$$

where x is not free in D. Note that the $\Leftarrow$ direction of both equivalences hold in intuitionistic logic. These equivalences are natural ones to choose given that we expect that goals which are operationally equivalent are logically equivalent. The $\Rightarrow$ directions of these rules are known as the *Independence of Premise* axioms [13], and although there are some results regarding the addition of these and similar axioms to Heyting arithmetic [13], not much seems to be known about the logic obtained by adding these rules to intuitionistic logic.

We may think of such a logic as a logic of "present choice", in that the choice of witness for the existentially quantified variable cannot be postponed; if we can ever choose such a witness, then we can do so immediately,

[4] This may be thought of as a full abstraction problem for the relation $\vdash_o$, in that we are trying to determine a notion of consequence for which equivalence is precisely operational equivalence.

without investigating future worlds. Similar remarks apply to the disjunctive case, in that if we can ever choose between the two alternatives we can do so immediately, without waiting to see what will happen in the future (and hence the choice is "independent" of the assumption used to prove its truth). This immediacy is reflected in the computation process by the fact that in order to establish the truth of r from either P_3 or P_4 we do so by trying to prove the body of the clause, rather than considering possible future choices. In this way the knowledge which we may use to make the relevant choice is already present in the program, and so it is merely a matter of seeing if such a choice can be made. If it cannot, then we have no other way to proceed. Hence, the logic of present choice will reflect the fact that we can only use information encoded in the program.

We have seen how it is necessary for the logic I' to satisfy the IP axioms so that operational equivalence implies logical equivalence. Now for the converse to hold as well, we need to ensure that I' is not too strong. One way to analyse this requirement on the nature of I' is to explore the relation $\vdash_o \subseteq \mathcal{D} \times \mathcal{G}$ in terms of a relation on $\mathcal{D} \times \mathcal{D}$ and a relation on $\mathcal{G} \times \mathcal{G}$. The main obstacle to viewing $\vdash_o$ directly as a consequence relation in the traditional sense is that such relations are usually defined on $\mathcal{F} \times \mathcal{F}$ where $\mathcal{F}$ is the set of all well-formed (first-order) formulae. Hence, as is done in [11], we may view $\vdash_o$ as the restriction of $\vdash_I$ to $\mathcal{D} \times \mathcal{G}$. However, we cannot use $\vdash_o$ directly for questions of equivalence between programs or between goals, and so for such equivalences we need to use $\vdash_{I'}$.

It should be noted that there are some reasonably well-known intermediate logics. Perhaps the best known one is known as the *logic of constant domains*, and for various reasons, this seems an appropriate notion for the desired logic. However, this is not particularly suitable for our purposes. The standard proof-theoretic characterisation of the logic of constant domains (CD) is to add the following rule to intuitionistic logic (in a Hilbert-type proof system):

$$\forall x\ (\phi \vee \psi(x)) \supset (\phi \vee \forall x\ \psi(x))$$

We may incorporate this rule into the sequent system by adding the following inference rule:

$$\frac{\Gamma \longrightarrow \forall x\ (\phi \vee \psi(x))}{\Gamma \longrightarrow \phi \vee \forall x\ \psi(x)}\ \mathrm{CD}$$

We refer to the extended proof system as I_{CD}. However, if we restrict our attention to D_{iH} and G_{iH} formulae, then this rule can never be used, as neither of the above two consequents are D_{iH} formulae, and as we do not allow universal quantification in G_{iH} goals, they are not G_{iH} formulae either. Hence, this rule will not affect the provability relations between D_{iH} formulae or between G_{iH} formulae. We give a formal statement of this result below.

Definition 6.1 *A sequent $\Gamma \longrightarrow F$ is an* $\mathbf{I}'$*-sequent iff every element of Γ is either a D_{iH} or G_{iH} formula, and F is either a D_{iH} formula or a G_{iH} formula.*

Proposition 6.1 *Let $\Gamma \longrightarrow F$ be an* **I'***-sequent. Then $\Gamma \vdash_{I_{CD}} F$ iff $\Gamma \vdash_I F$.*

Hence the logic of constant domains is not much help in this context. Another intermediate logic of interest is the extension of intuitionistic propositional calculus (IPC) called *lc* by Gabbay [1]. This may be characterised by adding to IPC the rule (in a Hilbert-style proof system)

$$(p \supset (q \vee r)) \supset ((p \supset q) \vee (p \supset r))$$

This is clearly an instance of the IP rule for disjunction. However, *lc* appears to be too strong, as it loses the disjunctive property, i.e. it is not true that if $A \vee B$ is provable then either A is provable or B is provable. Also, some of the alternative characterisations of *lc* do not fit in well with computational intuition. For example, another way to characterise this logic is to add the following rule to IPC:

$$(p \supset q) \vee (q \supset p)$$

This does not seem to be a rule which can be justified in terms of computation, especially due to the restrictions which first-order hereditary Harrop formulae place on implications, and so *lc* does not seem particularly helpful in this context either.

As mentioned above, we want $\vdash_{I'}$ to include all intuitionistic consequences (i.e. $\vdash_I \subseteq \vdash_{I'}$), but also for the following rules to hold:

$$D \supset \exists x G \vdash_{I'} \exists x D \supset G$$
$$D \supset (G_1 \vee G_2) \vdash_{I'} (D \supset G_1) \vee (D \supset G_2)$$

where x is not free in D.

The simplest way to define $\vdash_{I'}$ is to add the two rules above to the deduction rules of first-order intuitionistic logic. This will also ensure that I' is not too strong. In a sequent-style proof system, this means adding the following two rules to the standard system for intuitionistic logic:

$$\frac{\Gamma \longrightarrow F_1 \supset \exists x F_2}{\Gamma \longrightarrow \exists x F_1 \supset F_2} \qquad \frac{\Gamma \longrightarrow F \supset (F_1 \vee F_2)}{\Gamma \longrightarrow (F \supset F_1) \vee (F \supset F_2)}$$

We refer to proofs in the above system as $\mathbf{I}'$-proofs. Note that the two extra rules are only applicable when the consequents are a particular kind of formulae, but that the extra rules ensure that there are programs D_1 and D_2 for which $D_1 \not\vdash_I D_2$ but $D_1 \vdash_{I'} D_2$. For example, $p \supset \exists x\, q(x) \equiv_{I'} \exists x\, p \supset q(x)$, and so $(p \supset \exists x\, q(x)) \supset r \equiv_{I'} (\exists x\, p \supset q(x)) \supset r$.

One important feature of the above proof rules is that the restriction of $\vdash_{I'}$ to $\mathcal{D} \times \mathcal{G}$ is precisely $\vdash_I$. In this way we have not altered the way that goals are derived from programs, but strengthened the provability relation between programs and between goals. In fact, this result also holds for hereditary Harrop formulae.

Proposition 6.2 *Let P be a D_{hhf} program and G be a G_{hhf} goal. Then* $P \vdash_{I'} G \Leftrightarrow P \vdash_I G$.

In this way I' proves the desired equivalences between programs and between goals, but does not affect computations, i.e. the consequence relation between programs and goals. It follows immediately from this proposition that the same goals may be derived from programs which are provably equivalent in I', and goals which are provably equivalent in I' behave identically.

Corollary 6.3 *Let P_1 and P_2 be D_{hhf} programs and let G be a G_{hhf} goal. If $P_1 \vdash_o G$ and $P_2 \vdash_{I'} P_1$ then $P_2 \vdash_o G$.*

Corollary 6.4 *Let P be a D_{hhf} program and let G_1 and G_2 be G_{hhf} goals. If $P \vdash_o G_1$ and $G_1 \vdash_{I'} G_2$ then $P \vdash_o G_2$.*

It is also not hard to show that a program is equivalent to its normal form in I', and similarly for a goal.

Proposition 6.5 *Let P be a D_{iH} program and G be a G_{iH} goal. Then* $P \equiv_{I'} \mathrm{normal}(P)$ *and* $G \equiv_{I'} \mathrm{normal}(G)$.

Naturally we expect operational equivalence to imply logical equivalence for an intermediate logic I' in which the equivalences mentioned above hold. We may think of this requirement on I' as demanding that all connectives be treated similarly, since the corresponding equivalences for the other connectives, given below, are intuitionistically valid.

$$\begin{gathered} D \supset (G_1 \wedge G_2) \equiv_I (D \supset G_1) \wedge (D \supset G_2) \\ D \supset (D' \supset G) \equiv_I (D \wedge D') \supset G \\ D \supset \forall x\, G \equiv_I \forall x\, (D \supset G) \end{gathered}$$

where x is not free in D.

It is possible to show that operational equivalence implies logical equivalence in I'. A formal statement is given below.

Theorem 6.6 *Let P_1 and P_2 be D_{iH} programs, and let G range over G_{iH} goals. Then*

$$(\forall G\ D_1 \vdash_o G \Leftrightarrow D_2 \vdash_o G) \Rightarrow D_2 \equiv_{I'} D_1.$$

Proof: Assume that $\forall G\ D_1 \vdash_o G \Rightarrow D_2 \vdash_o G$. From lemma 5.1, we know that this implies that $\forall G'\ D_1 \vdash_o G' \Rightarrow D_2 \vdash_o G'$, where G' ranges over G_{hhf} goals. Now by Proposition 6.5, $D_1 \equiv_{I'} \mathrm{normal}(D_1)$, and as $\mathrm{normal}(D_1)$ may be re-written as a G_{hhf} formula G'', we have that $D_1 \equiv_{I'} G''$ and so $D_1 \vdash_o G''$, and hence $D_2 \vdash_o G''$, i.e. $D_2 \vdash_{I'} D_1$.

A similar argument shows that $D_1 \vdash_{I'} D_2$. □

Thus in I', operational equivalence implies logical equivalence. In fact it is not hard to see that this result will hold in any logic for which $D \equiv \mathrm{normal}(D)$, and so this property also holds in classical logic.

An interesting interpretation of this result is that if two programs D_1 and D_2 differ, i.e. $D_1 \not\equiv_{I'} D_2$, then there is a goal which distinguishes the two. More precisely, from the contrapositive of theorem 6.6 we have that if $D_2 \not\vdash_{I'} D_1$, then there is a goal G such that $D_1 \vdash_o G$ but it is not the case that $D_2 \vdash_o G$. As noted in the preceding remark, a similar property will also hold for classical logic.

7 Conclusions and Discussion

We have seen how a normal form for programs and goals may be given, and have shown that a slightly stronger logic than intuitionistic logic is needed in order to obtain a full abstraction property, i.e. that operationally equivalent programs are logically equivalent.

A point of further work on this matter is to obtain a normal form for full hereditary Harrop formulae, i.e. the class of formulae below.

$$D := A \mid \forall x D \mid D_1 \wedge D_2 \mid G \supset A$$
$$G := A \mid \exists x G \mid \forall x G \mid G_1 \wedge G_2 \mid G_1 \vee G_2 \mid D \supset G$$

The main difficulty is how to push the existential quantifiers and disjunctions outside universal quantifiers. One technique which may be of use is Skolemisation; another possibility is to use some form of infinitary logic. For example, Skolem functions will be useful for a goal such as $\forall x \exists y\ p(x,y)$, and some infinitary construction would presumably be necessary for a goal such as $\forall x\ p(x) \vee q(x)$.

Another point of further interest is the precise nature of the logic I'. Whilst the proof rules given above are sufficient, they are not very natural rules, and hence there may be a simpler and more informative proof system for I'. More natural inference rules would presumably shed more light on the nature of I'.

Another possibility is to show the analogous result to Theorem 6.6 for goal formulae, i.e. that

$$(\forall P\ P \vdash_o G_1 \Rightarrow P \vdash_o G_2) \Rightarrow G_1 \vdash_{I'} G_2$$

The main technical difficulty is to find a program "equivalent" to the goal G_1, which will then allow the proof to proceed as before. The techniques of infinitary logic may be useful here too.

8 Acknowledgements

The author would like to thank Dale Miller and David Pym for stimulating discussions of this material.

The support of the Australian Research Council and the Centre for Intelligent Decision Systems is gratefully acknowledged. Some of this work was performed whilst the author was a student in the Department of Computer Science, University of Edinburgh.

References

[1] D. Gabbay, *Semantical Investigations in Heyting's Intuitionistic Logic*, Reidel, 1981.

[2] J. Harland, *On Hereditary Harrop Formulae as a Basis for Logic Programming*, Ph.D. thesis, University of Edinburgh, July, 1991.

[3] J. Harland, Structural Properties of Logic Programs, *Proceedings of the Fourteenth Australian Computer Science Conference*, Sydney, February, 1991. Published as *Australian Computer Science Communications*:13:1, February, 1991.

[4] J. Harland, *Structural Properties of Logic Programs*, Technical Report 90/20, Department of Computer Science, University of Melbourne.

[5] J. Harland, *An Intermediate Logic for Logic Programs*, Technical Report 90/29, Department of Computer Science, University of Melbourne.

[6] S.C. Kleene, *Introduction to Metamathematics*, North-Holland, 1952.

[7] R.A. Kowalski, Predicate Logic as a Programming Language, *Information Processing 74*, North-Holland, Amsterdam, 1974.

[8] J.W. Lloyd, *Foundations of Logic Programming*, Springer-Verlag, Berlin, 1984.

[9] M.J. Maher, Equivalences of Logic Programs, in *Foundations of Deductive Databases and Logic Programming* 627-658, J. Minker (ed.), Morgan Kaufmann, 1988.

[10] D. Michie, Memo functions and machine learning, *Nature* 218:19-22, April, 1968.

[11] D.A. Miller, A Logical Analysis of Modules in Logic Programming, *Journal of Logic Programming* 6:79-108, 1989.

[12] D.A. Miller, G. Nadathur, F. Pfenning and A. Scedrov, Uniform Proofs as a Foundation for Logic Programming, *Annals of Pure and Applied Logic* 51:125-157, 1991.

[13] A.S. Troelstra, *Metamathematical Investigations of Intuitionistic Arithmetic and Analysis*, Lecture Notes in Mathematics 344, Springer-Verlag, Berlin, 1973.