Bunched Logic Programming
(Extended Abstract)

Pablo A. Armelín and David J. Pym

Queen Mary
University of London
England, UK
{pablo,pym}@dcs.qmw.ac.uk

Abstract. We give an operational semantics for the logic programming language `BLP`, based on the hereditary Harrop fragment of the logic of bunched implications, **BI**. We introduce **BI**, explaining the account of the sharing of resources built into its semantics, and indicate how it may be used to give a logic programming language. We explain that the basic input/output model of operational semantics, used in linear logic programming, will not work for bunched logic. We show how to obtain a complete, goal-directed proof theory for hereditary Harrop **BI** and how to reformulate the operational model to account for the interaction between multiplicative and additive structure. We give a prototypical example of how the resulting programming language handles, in contrast with Prolog, sharing (or local) and non-sharing (or global) use of resources purely logically.

1 Bunched Logic and Logic Programming

The logic of bunched implications, **BI**, freely combines an additive (intuitionistic) and a multiplicative (linear) implication as connectives of equal status [12,14,15]. Thus it stands in stark contrast with linear logic [4], in which intuitionistic implication is available via an exponential [12,14,15].

The semantics of **BI** may be motivated directly by modelling the notion of *resource*. Consider the following, very simple, axiomatization of the notion of resource (clearly, refinements are possible):

- An underlying *set* of resources, M;
- A way of *combining* resources, $\cdot$;
- A representative for *zero* resources, e;
- A way of *comparing* resources, $\sqsubseteq$.

Mathematically, we recognize that we have naturally identified a (for now, commutative) preordered monoid $\mathcal{M} = (M, e, \cdot, \sqsubseteq)$ of resources.

First, we may also exploit the presence of the monoidal combining operation to define the following multiplicative conjunction [17,12,14,15], in the possible-worlds style [9]:

$$m \models \phi * \psi \quad \text{iff} \quad \text{there are } n \text{ and } n' \text{ such that } m \sqsubseteq n \cdot n', \; n \models \phi \text{ and } n' \models \psi$$

This conjunction is interpreted as follows: the resource required to establish $\phi * \psi$ is obtained by combining the resources required to establish each of ϕ and ψ. Similarly, we can define the corresponding implication

$$m \models \phi \mathrel{-\!\!*} \psi \quad \text{iff} \quad \text{for all } n \text{ such that } n \models \phi,\ m \cdot n \models \psi.$$

This implication is interpreted as follows: if the resource required to establish the "function", $\phi \mathrel{-\!\!*} \psi$, is m and the resource required to establish the "argument" ϕ, is n, then the resource required to establish the result is $m \cdot n$. Thus the function and the argument do not share resources. Rather, their respective resources are taken from distinct worlds.

Second, the presence of the preorder suggests the possibility of a satisfaction relation for the intuitionistic connectives, using worlds $m, n \in M$ as usual:

$$\begin{array}{rcl} m \models \phi \wedge \psi & \text{iff} & m \models \phi \text{ and } m \models \psi \\ m \models \phi \vee \psi & \text{iff} & m \models \phi \text{ or } m \models \psi \\ m \models \phi \rightarrow \psi & \text{iff} & \text{for all } n \sqsubseteq m,\ n \models \phi \text{ implies } n \models \psi. \end{array}$$

The conjunction (and the disjunction) are interpreted as follows: each of the conjuncts (disjuncts) may share resources with the other. The implication is interpreted as a "function" which may share resources with its argument. We refer to the meaning of the semantics described here as the *sharing interpretation* [12,15]. Similarly, the additives may be seen as describing *local* properties whereas the mutliplicatives are *global*. We return to this point in § 4, in which we give a concrete, and implemented, programming example.

Proof-theoretically, the presence of the two implications is, at first sight, problematic. To see this consider that the whilst we may easily distinguish between multiplicative and additive elimination (or left) rules:

$$\frac{\Gamma \vdash \phi \mathrel{-\!\!*} \psi \quad \Delta \vdash \phi}{\Gamma, \Delta \vdash \psi} \mathrel{-\!\!*} E \qquad \frac{\Gamma \vdash \phi \rightarrow \psi \quad \Gamma \vdash \phi}{\Gamma \vdash \psi} \rightarrow E.$$

How are we to distinguish the corresponding introduction rules ? We may write

$$\frac{\Gamma, \phi \vdash \psi}{\Gamma \vdash \phi \mathrel{-\!\!*} \psi} \mathrel{-\!\!*} I \quad \text{but how then to write a rule for} \rightarrow I \quad \frac{\Gamma \;?\; \vdash \psi}{\Gamma \vdash \phi \rightarrow \psi} \;?$$

A semantically clean solution is provided by moving to sequents built not out of finite sequences of hypotheses but rather out of *bunches of hypotheses*, *i.e.*, finite trees, with the leaf nodes being formulæ, and the internal nodes denoted by either "," or ";", and referred to as *bunches*. The grammar of bunches is given as follows:

$$\begin{array}{rll} \Gamma ::= & \phi & \text{propositional assumption} \\ & \mid \emptyset_m & \text{multiplicative unit} \\ & \mid \Gamma, \Gamma & \text{multiplicative combination} \\ & \mid \emptyset_a & \text{additive unit} \\ & \mid \Gamma; \Gamma & \text{additive combination} \end{array}$$

We write $\Gamma(\Delta)$ to denote that Δ is a sub-bunch of Γ in the evident sense. Equality of bunches is given by the commutative monoid laws for "," and ";", together with substitution congruence: if $\Delta \equiv \Delta'$, then $\Gamma(\Delta) \equiv \Gamma(\Delta')$. A bunch is said to be *multiplicative* if its top-level combinator is "," and *additive* if its top-level combinator is ";". Contraction and Weakening are permitted for ";" but not for ",":

$$\frac{\Gamma(\Delta;\Delta) \vdash \phi}{\Gamma(\Delta) \vdash \phi}\ \mathit{Contraction} \qquad \frac{\Gamma(\Delta) \vdash \phi}{\Gamma(\Delta;\Delta') \vdash \phi}\ \mathit{Weakening}.$$

In much of what follows, we regard "," and ";" as multi-ary operators, and refer to their operands as *components*.

The introduction and elimination rules for the multiplicative and additive implications now go as follows:

$$\frac{\Gamma, \phi \vdash \psi}{\Gamma \vdash \phi \mathrel{-\!\!*} \psi}\ {-\!\!*}\,I \qquad \frac{\Gamma; \phi \vdash \psi}{\Gamma \vdash \phi \to \psi}\ {\to}\,I$$

$$\frac{\Gamma \vdash \phi \mathrel{-\!\!*} \psi \quad \Delta \vdash \phi}{\Gamma, \Delta \vdash \psi}\ {-\!\!*}\,E \qquad \frac{\Gamma \vdash \phi \to \psi \quad \Delta \vdash \phi}{\Gamma; \Delta \vdash \psi}\ {\to}\,E.$$

Turning to predication, consider that we can express a first-order sequent over a collection X of first-order variables as $(X)\Gamma \vdash \phi$. Given this point view, we can see that it is possible to allow not only Γ to be bunched but also X. So for each propositional rule, we have two possible forms of variable maintenance, *i.e.*, additive and multiplicative. For example, the two choices for the predicate $*R$ rule are

$$\frac{(X)\Gamma \vdash \phi \quad (Y)\Delta \vdash \psi}{(X;Y)\Gamma, \Delta \vdash \phi * \psi}\ *R_a \quad \text{and} \quad \frac{(X)\Gamma \vdash \phi \quad (Y)\Delta \vdash \psi}{(X,Y)\Gamma, \Delta \vdash \phi * \psi}\ *R_m.$$

The former choice is the one taken in linear logic and in this paper. It may be simplified, via Weakening and Contraction, to

$$\frac{(X)\Gamma \vdash \phi \quad (X)\Delta \vdash \psi}{(X)\Gamma, \Delta \vdash \phi * \psi}\ *R_a$$

The latter is the one taken in the basic version of **BI** [12,14,15].

The presence of bunched variables has one very significant consequence: It permits the definition of both additive, or extensional, *and* multiplicative, or intensional, quantifiers. For example,

$$\frac{(X;x)\Gamma \vdash \phi}{(X)\Gamma \vdash \forall x.\phi}\ \forall I \qquad \frac{(X,x)\Gamma \vdash \phi}{(X)\Gamma \vdash \forall_{\mathbf{new}} x.\phi}\ \forall_{\mathbf{new}} I,$$

where x is not free in Γ, and

$$\frac{(X)\Gamma \vdash \forall x.\phi \quad Y \vdash t : \mathrm{Term}}{(X;Y)\Gamma \vdash \phi[t/x]}\ \forall E \qquad \frac{(X)\Gamma \vdash \forall_{\mathbf{new}} x.\phi \quad Y \vdash t : \mathrm{Term}}{(X,Y)\Gamma \vdash \phi[t/x]}\ \forall_{\mathbf{new}} E$$

Here, we assume a simple bunched calculus of term formation [12,14,15] and, as before, the additive case may be simplified to use just one bunch of variables. The corresponding existentials are similar.

Semantically, the additive quantifier is handled intuitionistically,

$$m \models \forall x.\phi \text{ iff for all } n \sqsubseteq m \text{ and all } t \text{ defined at } n,\ n \models \phi[t/x],$$

i.e., the resource required to establish each instance of the quantified proposition must be available at the starting world. The semantics of the multiplicative $\forall_{\mathbf{new}}$ explains our use of the term "new":

$$m \models \forall x.\phi \text{ iff for all } n \text{ and all } t \text{ defined at } n,\ m \cdot n \models \phi[t/x],$$

i.e., the resource required to instantiate the proposition is taken from a *new* world or location. Again, the existentials are similar.

Our notion of *logic programming* is that introduced in [11,10], based on the sequent calculus. We start with the fragment of the logic for which *uniform proofs* are complete for logical consequence. Reading proofs from the root upwards, uniform proofs require that right rules be applied whenever possible, so that left rules are applied only when the right-hand side is atomic. Uniform proofs are said to be *simple* just in case the implicational left rules are restricted to be essentially unary. For example, in first-order intuitionistic logic, we get

$$\frac{\Gamma \vdash \phi[t/x] \quad \alpha[t/x] \vdash \beta[t/x]}{\Gamma, \phi \supset \alpha \vdash \beta} \supset L,$$

with α, β atomic and $\alpha[t/x] = \beta[t/x]$ (often, $\phi \supset \alpha$ is retained in the left-hand premiss).

In intuitionistic logic, simple uniform proofs, which are *goal-directed* and in which the non-determinism is confined to the choice of implicational formula, are complete for hereditary Harrop sequents [11,10]. Simple uniform proofs amount to the analytic notion of *resolution*. Taking all this together, we interpret hereditary Harrop sequents $\Gamma \vdash_o \exists x.\phi$ as a logic program, Γ, together with a query, or goal, ϕ, in which there is a logical variable x [8]. We use $\vdash_o$ to denote the simple, uniform, *i.e.*, resolution, proof, read from root to leaves.

In **BI**, the corresponding class of sequents may be defined. *Bunched hereditary Harrop formulæ* are given by the following grammar, in which A denotes atoms (we simplify a bit, for brevity):

$$\begin{array}{lrl} \text{Definite formulæ} & D ::= & A \mid D \wedge D \mid G \to A \mid D * D \mid G \mathrel{-\!\!*} A \\ & & \mid \forall x.D \mid \forall_{\mathbf{new}} x.D \\[1ex] \text{Goal formulæ} & G ::= & A \mid G \wedge G \mid D \to G \mid G * G \mid D \mathrel{-\!\!*} G \mid G \vee G \\ & & \mid \exists x.G \mid \exists_{\mathbf{new}} x.G \end{array}$$

A *bunched hereditary Harrop sequent* is a sequent $\mathcal{P} \vdash G$, where $\mathcal{P}$ is a bunch of definite formulæ. Such sequents are the basis of the bunched logic programming language, `BLP`, to which we give an operational semantics in § 3.

In intuitionistic logic, each of the reduction operators used in the execution of goal-directed search is additive. In **BI**, however, as in linear logic [16,10,5], we have multiplicatives which introduce a computationally significant difficulty. The typical case is $*R$:

$$\frac{\Gamma_1 \vdash_o \phi_1 \quad \Gamma_2 \vdash_o \phi_2}{\Gamma \vdash_o \phi_1 * \phi_2} \quad \Gamma = \Gamma_1\,,\,\Gamma_2.$$

Faced with $\Gamma \vdash_o \phi_1 * \phi_2$, the division of Γ into Γ_1 and Γ_2 must be calculated.

The basic solution, described for linear logic in [7,6,16,5], is the so-called *input/output* model. First pass all of Γ to the left-hand branch of the proof, leaving the right-hand branch undetermined. Proceed with the left-hand branch until it is completed. Then calculate which of the formulæ in Γ have been used to complete the left-hand branch and collect them into a finite set, Γ_{left}; The remaining, unused formulæ may now be passed to the right-hand branch:

$$\frac{\dfrac{\vdots}{\Gamma_{\text{left}} \vdash_o \phi_1} \quad \Gamma \backslash \Gamma_{\text{left}} \vdash_o \phi_2}{\Gamma \vdash_o \phi_1 \otimes \phi_2} \quad \Gamma = \Gamma_{\text{left}}, \Gamma \backslash \Gamma_{\text{left}}.$$

We refer to $\backslash$ as a "remainder operator" because it removes from Γ the consumed formulæ and passes the remainder to the next branch.

In **BI**, the problem is made *much* more complex by the mixing of additive and multiplicative structure enforced by the presence of bunches and the basic input/output idea will not work. To see this, consider the following search for a proof of the (provable) sequent $\phi, \psi \vdash (\chi \to \psi \wedge \chi) * \phi$ (note that it is convenient to put the remainder operator, read as "without", in the "current" computation):

$$\frac{\dfrac{\dfrac{}{(\chi;(\phi,\psi))\backslash\phi \vdash_o \psi\wedge\chi}\,(\text{not provable})}{(\phi,\psi)\backslash\phi \vdash_o \chi\to(\psi\wedge\chi)}\to R \qquad \dfrac{}{\phi\backslash\emptyset_m \vdash_o \phi}}{(\phi,\psi)\backslash\emptyset_m \vdash_o (\chi\to(\psi\wedge\chi))*\phi}*R$$

Consider the left-hand branch of the $*R$. In order to get an axiom of the form $\psi \vdash \psi$, we must first remove the χ from the program by performing a Weakening and then perform a subtraction of ϕ, which is required on the right-hand branch of the $*R$, so that the result of $\chi;(\phi,\psi)\backslash\phi$ is ψ, *i.e.*, the remainder operator first throws away, via Weakening, the additive bunch surrounding the multiplicative bunch within which ψ, the formula which must be removed, is contained. Now, the remaining bunch is sufficient to form, after an $\wedge R$ reduction, the axiom $\psi \vdash \psi$ but insufficient to form the necessary axiom for χ.

At first sight, thinking of axioms of the form $\Gamma;\alpha \vdash \alpha$, it might seem like we need a subtraction operation which does not perform Weakening. This would solve the problem in the particular case above but it would worsen it in other cases. From being incomplete the system would become unsound. To see this, consider a search for a proof of the (unprovable) sequent $\tau;(\phi,\psi) \vdash (\chi\to(\psi\wedge\tau))*\phi$. Once the $*R$ rule is applied, after doing $\to R$, the propositions τ and χ are at the same level in $(\chi;\tau;(\phi,\psi))\backslash\phi$ and, if this is taken to be equal to $\chi;\tau;\psi$, both χ and τ are provable.

Again it looks like this could be fixed, by requiring, for example, that the $*R$ rule be applied only on multiplicative bunches. Aside from the unpleasant non-determinism introduced by this solution, it doesn't quite solve the problem either. To see this, it is enough to consider a slight modification of the last search. Consider the unprovable $\upsilon,(\tau;(\phi,\psi)) \vdash (\tau \rightarrow (\psi \wedge \tau)) * (\phi * \upsilon)$. Here, after the application of $*R$ and $\rightarrow R$ it would be possible to prove both ψ and τ, which is unsound. It seems clear that the root of the difficulty lies with the interaction between the additives, in particular $\rightarrow R$, and the multiplicatives.

The essential idea for the solution, though far from simple in detail, is to introduce a system of stacks which keep a record of which resources have been added to the program as a result of $\rightarrow R$ and which manage their interaction with the formation of axioms and with subtraction and passing by continuations. The detailed formulation of the operational semantics is rather complex and before giving it, in § 3, we must look, in § 2 at uniform, and simple, proofs in **BI**.

2 Uniform Proofs in BI

So far we have discussed **BI** semantically, and informally considered its use as a logic programming language. Formally, as we have indicated, the basis of logic programming in **BI** relies on the availability of *goal-directed* proofs.

Proofs in **BI** may be presented as a sequent calculus, here given in Definition 1. The Cut-elimination theorem holds [15] and semantic completeness theorems are available [12,14,15,13]. In order to explain uniform proofs, and so also the subsequent operational semantics, we restrict our attention to propositional **BI**: although our logic programming language uses predicate **BI**, it exploits, in its core language, only additive predication and quantification, just as in Prolog [2,1], so that its use of logical variables and unification is completely standard. Consequently, our presentation involves no significant loss of generality.

The use of multiplicative predication and quantification is possible but more complex. We conjecture that its main use will be in BLP's module system, adapted to **BI** from the basic ideas presented in [10], in which we conjecture the importing of functions from one module to another may exploit the additive–multiplicative distinction to valuable effect for the programmer. This topic is beyond our present scope.

Definition 1. The sequent calculus **LBI** is defined as follows:

IDENTITIES

$$Axiom \quad \frac{}{\phi \vdash \phi} \qquad Cut \quad \frac{\Gamma \vdash \phi \quad \Delta(\phi) \vdash \psi}{\Delta(\Gamma) \vdash \psi}$$

STRUCTURALS

$$W \quad \frac{\Gamma(\Delta) \vdash \phi}{\Gamma(\Delta;\Delta') \vdash \phi} \qquad C \quad \frac{\Gamma(\Delta;\Delta) \vdash \phi}{\Gamma(\Delta) \vdash \phi} \qquad E \quad \frac{\Gamma \vdash \phi}{\Delta \vdash \phi} \quad (\Delta \equiv \Gamma)$$

UNITS

$$\frac{\Gamma(\emptyset_m) \vdash \phi}{\Gamma(I) \vdash \phi}\, I\,L \qquad \frac{}{\emptyset_m \vdash I}\, I\,R \qquad \frac{}{\bot \vdash \phi}\, \bot\,L \qquad \frac{\Gamma(\emptyset_a) \vdash \phi}{\Gamma(\top) \vdash \phi}\, \top\,L \qquad \frac{}{\emptyset_a \vdash \top}\, \top\,R$$

MULTIPLICATIVES

$$-\!\!*\,L \quad \frac{\Gamma \vdash \phi \quad \Delta(\Delta',\psi) \vdash \chi}{\Delta(\Delta',\Gamma,\phi -\!\!* \psi) \vdash \chi} \qquad -\!\!*\,R \quad \frac{\Gamma,\phi \vdash \psi}{\Gamma \vdash \phi -\!\!* \psi}$$

$$*L \quad \frac{\Gamma(\phi, \psi) \vdash \chi}{\Gamma(\phi * \psi) \vdash \chi} \qquad * R \quad \frac{\Gamma \vdash \phi \quad \Delta \vdash \psi}{\Gamma, \Delta \vdash \phi * \psi}$$

Additives

$$\to L \quad \frac{\Gamma \vdash \phi \quad \Delta(\Delta'; \psi) \vdash \chi}{\Delta(\Delta'; \Gamma; \phi \to \psi) \vdash \chi} \qquad \to R \quad \frac{\Gamma; \phi \vdash \psi}{\Gamma \vdash \phi \to \psi}$$

$$\wedge L \quad \frac{\Gamma(\phi_1; \phi_2) \vdash \psi}{\Gamma(\phi_1 \wedge \phi_2) \vdash \psi} \qquad \wedge R \quad \frac{\Gamma \vdash \phi \quad \Delta \vdash \psi}{\Gamma; \Delta \vdash \phi \wedge \psi}$$

$$\vee L \quad \frac{\Gamma(\phi) \vdash \chi \quad \Delta(\psi) \vdash \chi}{\Gamma(\phi \vee \psi); \Delta(\phi \vee \psi) \vdash \chi} \qquad \vee R \quad \frac{\Gamma \vdash \phi}{\Gamma \vdash \phi \vee \psi} \qquad \vee R' \quad \frac{\Gamma \vdash \psi}{\Gamma \vdash \phi \vee \psi}$$

The predicate substitution and quantifier rules may also be formulated sequentially. □

Most of the permutations, $\rightsquigarrow$, necessary to show that uniform proofs are complete for bunched hereditary Harrop formulæ are very straightforward. For example,

$$\dfrac{\Gamma \vdash \phi \qquad \dfrac{\Delta(\psi) \vdash \chi \quad \Delta(\psi) \vdash \chi'}{\Delta(\psi) \vdash \chi \wedge \chi'}\wedge R}{\Delta(\Gamma, \phi \mathrel{-\!\!*} \psi) \vdash \chi \wedge \chi'} \mathrel{-\!\!*} L \quad \rightsquigarrow \quad \dfrac{\dfrac{\Gamma \vdash \phi \quad \Delta(\psi) \vdash \chi}{\Delta(\Gamma, \phi \mathrel{-\!\!*} \psi) \vdash \chi} \mathrel{-\!\!*} L \qquad \dfrac{\Gamma \vdash \phi \quad \Delta(\psi) \vdash \chi'}{\Delta(\Gamma, \phi \mathrel{-\!\!*} \psi) \vdash \chi'} \mathrel{-\!\!*} L}{\Delta(\Gamma, \phi \mathrel{-\!\!*} \psi) \vdash \chi \wedge \chi'} \wedge R$$

or

$$\dfrac{\Gamma \vdash \phi \qquad \dfrac{\Delta(\psi) \vdash \chi \quad \Delta' \vdash \chi'}{\Delta(\psi), \Delta' \vdash \chi * \chi'} * R}{\Delta(\Gamma, \phi \mathrel{-\!\!*} \psi), \Delta' \vdash \chi * \chi'} \mathrel{-\!\!*} L \quad \rightsquigarrow \quad \dfrac{\dfrac{\Gamma \vdash \phi \quad \Delta(\psi) \vdash \chi}{\Delta(\Gamma, \phi \mathrel{-\!\!*} \psi) \vdash \chi} \mathrel{-\!\!*} L \qquad \Delta' \vdash \chi'}{\Delta(\Gamma, \phi \mathrel{-\!\!*} \psi), \Delta' \vdash \chi * \chi'} * R.$$

Note, however, that uniform proofs in **BI** must always perform any possible (and trivial) $*L$s (and $\wedge L$s) before performing any right rule: to see this, consider that we should like to have a uniform proof of $\phi * \psi \vdash \phi * \psi$.

Weakening is, at first sight, a source of difficulty. Weakenings may be permuted above all rules *except* $*R$. To see this, consider that $*R$ must divide a multiplicative bunch between its two premisses. So if a $*R$ has a Weakening immediately below it, then there is no way in which $*R$ may be applied directly to the resulting sequent. However, it turns out that this difficulty may be handled within the operational semantics by examining in turn each of the multiplicative bunches below the ";" introduced by the Weakening. To make this work, we must consider a canonical form for bunches: a bunch is in *canonical form* iff

1. its left-hand branch is either a proposition, a unit or a canonical bunch of the opposite (additive or multiplicative) type, and
2. its right-hand branch is canonical.

For example, if Γ and Γ' are additive and canonical and if Δ is canonical, then $(\Gamma, (\Gamma', \Delta))$ is canonical. Our operational semantics, in § 3, assumes that bunches are in canonical form.

Lemma 2. *Every bunch may be written in canonical form.* □

Proposition 3. *Uniform proofs in* **BI** *are complete for bunched hereditary Harrop sequents.* □

As we have seen, uniform proofs do not, however, characterize resolution. For that, we must ensure that the choice of clause in each implicational left rule is also goal-directed. For that we require proofs which are not merely uniform but simple, in which all instances of implicational left rules are of the following, essentially unary, forms (I is the unit of $*$):

$$\frac{\Gamma \vdash \phi \quad \Delta(\alpha) \vdash I}{\Delta(\Gamma; \phi \to \alpha) \vdash \alpha} \to L \qquad \frac{\Gamma \vdash \phi}{\Gamma, \phi \mathrel{-\!\!*} \alpha \vdash \alpha} \mathrel{-\!\!*} L$$

These rules are clearly admissible in **LBI** but to understand why the $\mathrel{-\!\!*} L$ rule is complete we must consider not only the canonical form for bunches but also that we may replace the basic axiom sequent with the following CutAxiom rule:

$$\frac{\Gamma \vdash I}{\Gamma, \alpha \vdash \alpha} \text{ CutAxiom (CA).}$$

The effect of this rule, which may be seen as a form of garbage collection, is to absorb, trivially, unused multiplicative bunches. Then we can make the following transformation of proof figures ($\emptyset_m$ is the unit of ",", *i.e.*, I on the left):

$$\dfrac{\Gamma_1 \vdash \phi \qquad \dfrac{\Gamma_2 \vdash I}{\Gamma_2, \alpha \vdash \alpha}\text{CA}}{\Gamma_1, \Gamma_2, \phi \mathrel{-\!\!*} \alpha \vdash \alpha} \mathrel{-\!\!*} L \quad \rightsquigarrow \quad \dfrac{\dfrac{\Gamma_2 \vdash I \qquad \dfrac{\Gamma_1 \vdash \phi}{\Gamma_1, \emptyset_m \vdash \phi}\text{Unit of ``,''}}{\Gamma_1, \Gamma_2 \vdash \phi}\text{Cut} \qquad \dfrac{}{\alpha \vdash \alpha}\text{CA}}{\Gamma_1, \Gamma_2, \phi \mathrel{-\!\!*} \alpha \vdash \alpha} \mathrel{-\!\!*} L$$

in which the right-hand premiss of the $\mathrel{-\!\!*} L$ is rendered trivial.

This step, together with an analysis of the permutations of $\mathrel{-\!\!*} L$ and $\to L$ rules with respect to one another, is sufficient to give us the following:

Proposition 4. *Simple uniform proofs in* **BI** *are complete for bunched hereditary Harrop formulæ.*

3 An Operational Semantics for BLP

As we have suggested, the operational semantics of BLP may be seen as a development of the *input/output model*, described in [7], to account for the bunched structure of sequents. This development represents a substantial generalization of the method and involves a degree of technical complexity.

Starting from uniform proofs, presented in § 2, we see that the remaining source of non-deterministic is the splitting of bunches in the $*R$ operator. Thus a remainder can be valid only if it is calculated on the left-hand branch of a search above a $*R$ operator. However, additional operational complexity arises from the bunched structure itself, *i.e.*, from the interaction between between the multiplicative and additive implications. (Notation: for clarity, we use just Γs, ϕs, *etc.*, rather than the more cumbersome Ds and Gs notation. We distinguish atomic formulæ as αs. Nevertheless, we are working with hereditary Harrop sequents.)

Consequently, operational sequents have the form $\langle\Gamma, s\rangle\backslash\langle\Gamma', s'\rangle$, which should be interpreted as follows:

- Γ is the bunch which is passed *up* the current branch of the search tree;
- Γ' is the bunch which is passed *down* the current branch of the search tree;
- s and s' denote stacks of bunches and are used to manage the interaction between the reduction operators used and the formulæ available:
  - s manages upward propagation via *open boxes* and *closed boxes*;
  - s' manages downward propagation via *full boxes*, *open boxes* and *locked boxes*.

  Informally, full, open and locked boxes, which are used in the definitions of the $*R$, $\wedge R$ and $\rightarrow R$ operators, arise as follows:

Full box:	$\{\Delta, \xi\}$	used downwards, arising from assignment to an unknown remainder — creating a full box corresponds to an assignment in the CPS execution
Open box:	$\boxed{\xi}$	used upwards, arising from a search-figure in which a $*R$ occurs below an $\rightarrow R$
Locked box:	$\boxtimes$	used upwards, arising from a $*R$

  Their meanings will be made clear in our description (below) of the operational semantics.
- Finally, $\backslash$ is a remainder operator: informally, it is read as "without" or "leaves unused". Formally, its meaning is defined by the operators for $\vdash_o$.

Overall, an operational sequent, $\langle\Gamma, s\rangle\backslash\langle\Gamma', s'\rangle \vdash_o \phi$, is read as "given a program Γ with a stack s as input, after proving ϕ we are left with a program Γ' and stack s' for the subsequent computation".

At each occurrence of a $*R$ rule, a box is created and a flag, represented by ϵ (for ϵmpty flag), is placed at the left-hand end of the bunch which is the remainder, is set to indicate whether remainders from the left-hand branch of the search are permitted (if ϵ is present, then remainders are not permitted, if it is absent they are; it is rather like I). Initially, the box is locked, and is put at the top of the upward stack.

When a $\rightarrow R$ operator is used, the box is unlocked and the antecedent of the implication $\phi \rightarrow \psi$ is put into it: so we get a stack of the form $\boxed{\phi} :: s$. If the box was unlocked already, with previous implicational antecedents in it, then the new antecedent is additively combined with the existing ones: so we get a stack of the form $\boxed{\Delta; \phi} :: s$. Also, the flag is reset, thereby allowing no remainder to pass except through the box.

A box that contains a remainder is denoted by $\{\Delta, \phi\} :: s$. Such boxes occur in downward stacks and indicate that the computation occurred under the wrapping of ϕ, arising from additives, and leaves a remainder Δ.

Finally, in order to give the operators of the operational semantics, we need to formally define a *subtraction operator*. We are now using $\backslash$ to calculate remainders but we still need a basic way of removing a bunch from a super-bunch within which it lives. The substraction of Γ from Δ is definable just in case Γ is a sub-bunch of Δ, written $\Gamma \subseteq \Delta$ and is defined, exploiting the canonical form for bunches, as follows (here $\overset{;}{\Gamma}$ means that Γ is additive and $\overset{,}{\Gamma}$ means that Γ is multiplicative; we label the components of a bunch Δ as Δ_is):

- If $\Gamma = \emptyset_m$, then $\Gamma \subseteq \Delta$, any Δ;
- $\overset{;}{\Gamma} \subseteq \overset{;}{\Delta}$ iff $\overset{;}{\Gamma} \equiv \overset{;}{\Delta}$ or $\overset{;}{\Gamma} \subseteq \overset{,}{\Delta}_i$, some i;
- $\overset{;}{\Gamma} \subseteq \overset{,}{\Delta}$ iff $\overset{;}{\Gamma} \subseteq \overset{;}{\Delta}_i$, some i;
- $\overset{,}{\Gamma} \subseteq \overset{;}{\Delta}$ iff $\overset{,}{\Gamma} \subseteq \overset{,}{\Delta}_i$, some i;
- $\overset{,}{\Gamma} \subseteq \overset{,}{\Delta}$ iff $\overset{,}{\Gamma} \equiv \overset{,}{\Delta}$ or $\overset{,}{\Gamma} \subseteq \overset{;}{\Delta}_i$, some i.

We then write $\Gamma - \Delta$ to denote the following subtraction operation: find the additive/multiplicative sub-bunch of Δ within which Γ lives and delete all of that sub-bunch. For example, $(\phi, ((\psi, \psi'); \chi)) - (\psi, \psi') = \phi$. (Formally, $\Gamma - \Delta$ is defined by recursion over additive/multiplicative cases.)

The operators, read from conclusion to premisses, which describe the operational semantics of `BLP`, are *summarized* in Table 1. The presentation relies on the execution of the left-hand branches before the right-hand. We describe the CPS execution of the operational semantics by considering the key cases of the operational reductions, given in Table 1, in turn:

CutAxiom: We are given a bunch Γ which contains the atom α, *i.e.*, the principal formula of the axiom, in any position, and a stack s. There is exactly one way of performing the minimum number of Weakening reductions,

$$\frac{\Gamma', \alpha \vdash \alpha}{\Gamma(\alpha) \vdash \alpha},$$

on Γ so as to bring α to top-level. This is done and we give to the success continuation the resulting bunch *without* α, *i.e.*, the remainder Γ'. In fact, since the only possible reduction above a CutAxiom is a Unit or Unit$'$ reduction, we can be more specfic about what the form of Δ and s' must be. For example, if Δ is ε, Γ' is not equal to $\emptyset_m$ and s has an open box on top, then s' must have a full box on top. Otherwise, Δ must be equal to Γ' and s' must be equal to s.

$-\!\!*L$: We start with a bunch in which the clause $\phi -\!\!* \alpha$ occurs in an arbitrary position. The unary, or "resolution" version of $-\!\!*L$ is invoked but the bunch taken in the premiss is that which is obtained, as in the CutAxiom case, by performing the minimum number of Weakening reductions on Γ so as to bring α to top-level.

$*R$: We start with Γ and s given and try to prove ϕ from Γ and $\boxtimes :: s$. Upon success, we get a remainder, Γ'. The top of the stack might have been modified but we don't care about it any more. At this point, we can try to prove

$$\frac{}{\langle \Gamma, s\rangle \backslash \langle \Gamma, s\rangle \vdash_o I}\ \mathrm{Unit} \qquad \frac{}{\langle \Gamma, \boxed{\phi} :: s\rangle \backslash \langle \varepsilon, \{\Gamma, \phi\} :: s\rangle \vdash_o I}\ \mathrm{Unit}'$$

$$\frac{\langle \Gamma', s\rangle \backslash \langle \Delta, s'\rangle \vdash_o I}{\langle (\Gamma(\alpha)), s\rangle \backslash \langle \Delta, s'\rangle \vdash_o \alpha}\ \mathrm{CutAxiom}^{\dagger}$$

$$\frac{\langle \Gamma', s\rangle \backslash \langle \Delta, s'\rangle \vdash_o \phi}{\langle \Gamma(\phi -\!\!* \alpha), s\rangle \backslash \langle \Delta, s'\rangle \vdash_o \alpha}\ -\!\!*L^{\dagger} \qquad \frac{\langle (\Gamma; \phi \to \alpha), []\rangle \backslash \langle \varepsilon, []\rangle \vdash_o \phi \qquad \langle \Delta, s\rangle \backslash \langle \Delta', s'\rangle \vdash_o I}{\langle \Delta(\Gamma; \phi \to \alpha), s\rangle \backslash \langle \Delta', s'\rangle \vdash_o \alpha} \to L$$

$$\frac{\langle \Gamma, \boxtimes :: s\rangle \backslash \langle \Gamma', \_ :: s\rangle \vdash_o \phi \quad \langle \Gamma', s\rangle \backslash \langle \Delta, s'\rangle \vdash_o \psi}{\langle \Gamma, s\rangle \backslash \langle \Delta, s'\rangle \vdash_o \phi * \psi} *R \qquad \frac{\langle (\phi, \Gamma), s\rangle \backslash \langle \Delta, s\rangle \vdash_o \psi}{\langle \Gamma, s\rangle \backslash \langle \Delta, s\rangle \vdash_o \phi -\!\!* \psi}\ -\!\!*R$$

$$\frac{\langle \Gamma, s\rangle \backslash \langle \Delta, s\rangle \vdash_o \phi \quad \langle \Gamma - \Delta, []\rangle \backslash \langle \varepsilon, s\rangle \vdash_o \psi}{\langle \Gamma, s\rangle \backslash \langle \Delta, s\rangle \vdash_o \phi \wedge \psi} \wedge R$$

$$\frac{\langle \Gamma, \boxed{\xi} :: s\rangle \backslash \langle \varepsilon, \{\Delta, \xi\} :: s\rangle \vdash_o \phi \quad \langle \xi; (\Gamma - \Delta), []\rangle \backslash \langle \varepsilon, []\rangle \vdash_o \psi}{\langle \Gamma, \boxed{\xi} :: s\rangle \backslash \langle \varepsilon, \{\Delta, \xi\} :: s\rangle \vdash_o \phi \wedge \psi} \wedge R'$$

$$\frac{\langle \Gamma, \boxed{\xi} :: s\rangle \backslash \langle \varepsilon, \boxed{\xi} :: s\rangle \vdash_o \phi \quad \langle \Gamma, \boxed{\xi} :: s\rangle \backslash \langle \varepsilon, \{\Delta, \xi\} :: s\rangle \vdash_o \psi}{\langle \Gamma, \boxed{\xi} :: s\rangle \backslash \langle \varepsilon, \{\Delta, \xi\} :: s\rangle \vdash_o \phi \wedge \psi} \wedge R''$$

$$\frac{\langle (\phi; \Gamma), []\rangle \backslash \langle \varepsilon, []\rangle \vdash_o \psi}{\langle \Gamma, []\rangle \backslash \langle \varepsilon, []\rangle \vdash_o \phi \to \psi} \to R \qquad \frac{\langle (\phi; \Gamma), \boxed{\phi} :: s\rangle \backslash \langle \varepsilon, \{\Delta, \phi\} :: s\rangle \vdash_o \psi}{\langle \Gamma, \boxtimes :: s\rangle \backslash \langle \Delta, \boxtimes :: s\rangle \vdash_o \phi \to \psi} \to R'$$

$$\frac{\langle (\phi; \xi; \Gamma), \boxed{\xi; \phi} :: s\rangle \backslash \langle \varepsilon, \{\Delta, (\xi; \phi)\} :: s\rangle \vdash_o \psi}{\langle \xi; \Gamma, \boxed{\xi} :: s\rangle \backslash \langle \varepsilon, \{\Delta, \xi\} :: s\rangle \vdash_o \phi \to \psi} \to R''$$

$$\frac{\langle \Gamma, s\rangle \backslash \langle \Delta, s'\rangle \vdash_o \phi}{\langle \Gamma, s\rangle \backslash \langle \Delta, s'\rangle \vdash_o \phi \vee \psi} \vee R \qquad \frac{\langle \Gamma, s\rangle \backslash \langle \Delta, s'\rangle \vdash_o \phi}{\langle \Gamma, s\rangle \backslash \langle \Delta, s'\rangle \vdash_o \phi \vee \psi} \vee R$$

† Γ' is obtained uniquely by performing the minimal number if Weakenings required to bring $\phi -\!\!* \alpha$ to top-level.

Table 1. Summary of the Operational Semantics of BLP

ψ from Γ' and s. Upon success, we get as a result a remainder Δ together with an arbitrary modification of the stack. Because we managed to prove $\phi * \psi$ from Γ, leaving Δ, this result is given to the final success continuation. The special case in which Δ is ε works in the same way but for the fact that Γ' and s, which must be [], will prove ψ only when Γ' proves ψ without leaving any remainder. So, Γ and s are given, Γ' and Δ are calculated, and s is modified as necessary, depending on the reductions encountered above. An example which includes this case follows after the description of the operators.

$\wedge R$s: There are two subcases. In the first, a remainder is allowed, which may occur only if we are on some multiplicative branch of the search. In the first, handled by $\wedge R$, the bunch Γ and the stack s are given. Notice that the stack s is the same on both sides of the left-hand branch: modifications to the stack before passing to the next branch occur only in operators in which remainders are not permitted. Moreover, notice that the right-hand branch gets an empty stack: it has already all the information needed to search for a proof. For example, this case of the rule is used for the proof of $(\phi;\psi),\chi \vdash_o (\phi \wedge \psi) * \chi$ and, indeed, the same program, with this rules, proves $(\psi \wedge \phi) * \chi$, $(\phi \wedge \phi) * \chi$, $(\psi \wedge \psi) * \chi$, *etc.*. In each case, the context given to the left-hand branch will be equivalent to $(\phi;\psi),\chi$ and the subtraction, "$\Gamma - \Delta$", will leave χ for the right-hand branch. In the second, handled by $\wedge R'$ and $\wedge R''$, no remainder is permitted. Here we have two subcases, depending on whether the stack is full on return of the left or right branch. An example which includes this case follows after the description of the operators.

$\to R$s: There are three subcases. All three share the property that the antecedent *of the implication* leaves no remainder. Note that we assume that any occurrences of $\wedge$ or $*$ in the antecendents ϕ in $\phi \to \psi$ which are (inductively) principal connectives are immediately removed using operators corresponding to the operationally trivial $*L$ or $\wedge L$ rules. An example which illustrates $\to R$ follows after the description of the operators.

The remaining cases are similar. In all cases, failure invokes backtracking.

A worked example, as mentioned above, will clarify these complex constructions (here, "CA" denotes CutAxiom):

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{\langle \phi, \boxed{\chi} \rangle \backslash \langle \varepsilon, \{\phi,\chi\}\rangle \vdash_o I}\,\mathrm{Unit}'}{\langle (\chi;(\phi,\psi)), \boxed{\chi}\rangle \backslash \langle \varepsilon, \{\phi,\chi\}\rangle \vdash_o \psi}\,\mathrm{CA} \quad \dfrac{\dfrac{}{\langle \emptyset_m, []\rangle\backslash\langle\varepsilon,[]\rangle \vdash_o I}\,\mathrm{Unit}}{\langle(\chi;\psi),[]\rangle\backslash\langle\varepsilon,[]\rangle\vdash_o \chi}\,\mathrm{CA}}{\langle (\chi;(\phi,\psi)), \boxed{\chi}\rangle\backslash\langle\varepsilon,\{\phi,\chi\}\rangle \vdash_o \psi\wedge\chi}\,\wedge R'}{\langle(\phi,\psi),\boxtimes\rangle\backslash\langle\phi,\boxtimes\rangle\vdash_o \chi\to(\psi\wedge\chi)}\,\to R' \quad \dfrac{\dfrac{}{\langle\emptyset_m,[]\rangle\backslash\langle\varepsilon,[]\rangle\vdash_o I}\,\mathrm{Unit}}{\langle\phi,[]\rangle\backslash\langle\varepsilon,[]\rangle\vdash_o\phi}\,\mathrm{CA}}{\langle(\phi,\psi),[]\rangle\backslash\langle\varepsilon,[]\rangle\vdash_o(\chi\to(\psi\wedge\chi))*\phi}\,*R$$

Thus we revisit our earlier, problematic example, showing how the use of boxes manages the interaction between the multiplicatives and additives, specially $\to R$.

In order to establish the soundness and completeness, with respect to logical consequence, of the operational semantics, we need to relate intermediate states of a computation, *i.e.*, operational sequents, to logical consequences, *i.e.*, **BI** sequents. To this end we introduce a mapping $\backslash\backslash$ from pairs of bunch-stack pairs, together with the goal, *i.e.*, $\langle \Gamma, s\rangle\backslash\langle \Gamma', s'\rangle \vdash_o \phi$, to bunches. The idea is to extend the basic subtraction operation, defined for bunches, to bunch-stack pairs. There are three cases.

1. Remainders are allowed and a full box is on the top of the downward stack: $\langle \Gamma, \_\rangle\backslash\backslash\langle \epsilon, (\{\Delta, \xi\} :: s)\rangle = f(\xi)\,;\,((\Gamma \sim \xi) - \Delta)$, in which we require the subtraction operation $\sim$, given by $\Gamma(\Delta; \Theta; \Delta') \sim \Gamma(\Delta; \Delta') = \Theta$, which leaves the difference between a bunch and an extension of it using ";". $f(\xi)$ is ξ if ϕ is an additive conjunction or implication, and $\top$ otherwise.
2. Remainders are allowed and there is something other than a full box on top of the downward stack: $\langle \Gamma, \_\rangle\backslash\backslash\langle \epsilon, s' :: s\rangle = \Gamma \quad (= \Gamma - \emptyset_m)$.
3. Finally, in case neither (1) nor (2): $\langle \Gamma, \_\rangle\backslash\backslash\langle \Delta, \_\rangle = \Gamma - \Delta$.

Then we get, by induction on the structure of proofs, the following:

Lemma 5. *The stacks used in the operational semantics do not corrupt the logical consequence relation:*

$$\langle \Gamma, s\rangle\backslash\langle \Gamma', s'\rangle \vdash_o \phi \quad \textit{if and only if} \quad \langle \Gamma, s\rangle\backslash\backslash\langle \Gamma', s'\rangle \vdash \phi. \qquad \square$$

The soundness and completeness of the operational semantics follows as a corollary of Lemma 5.

Theorem 6. *The operational semantics is sound and complete with respect to the bunched sequent calculus,* **LBI***:*

$$\Gamma \vdash \phi \textit{ if and only if } \langle \Gamma, []\rangle\backslash\langle \epsilon, []\rangle \vdash_o \phi. \qquad \square$$

We conjecture that our operational techniques will be applicable to a wide range of substructural logics.

4 Programming in BLP: An Example

Recall that our semantics for **BI**'s connectives, as set out in § 1, is couched in terms of *sharing*. Here, we give a quite generic, yet small, example of the kinds of problems for which bunched logic programming is well-suited. Consider the bunch $(p(a1); p(a2)), (p(b1); p(b2))$. Here, $p(x)$ means "x is a person". The bunch structure shows that $a1$ and $a2$ belong to the same group and that $a1$ and $b1$ belong to beligerent groups. To say that two individuals are possibly in a fighting relation we say simply $\forall x, y.p(x)*p(y) \mathrel{-\!\!*} fight(x, y)$, which is to say that x and y may fight if they belong to different groups. A complete BLP program (we write on one line to save space) would be (here, T is $\top$, the unit of $\wedge$)

```
(p(a1);p(a2)), (p(b1);p(b2)), [x,y]fight(x,y)*- p(x)*p(y)*T
```

Notice that the definition of fight has been slightly modified to take into account that there might be more than two groups; but they may be disregarded.

An alternative solution would be to decorate each group with a multiplicative unit to signal that it can be ignored. So we might have for example

```
(p(a1);p(a2);I), (p(b1);p(b2);I), (p(a5);p(a6);I)
```

However, the first approach is to be recommended since it doesn't produce redundant solutions. Adding a unit to each group allows the unit operation to be performed in different places, but without changing the solution.

The following is an equivalent Prolog program for this problem. It uses tags to distinguish the groups:

```
p(a1,t1). p(a2,t1). p(b1,t2). p(b2,t2).
fight(X,Y):- p(X,T),p(Y,U),T\=U.
```

Thinking of political parties, sometimes they split into rival factions but each faction in turn might want to keep its former allies. This situation might be represented by the bunch $(p(a1);\, p(a2)), (p(b1);\, (p(b21); p(b22)),\, (p(b23); p(b24)))$. Notice that $b21$ fights with $a1$ and $a2$ but also with $b23$ and $b24$. If we call x and y allies if they don't fight, then despite $b1$'s being an ally of $b21$, and also of $b23$, $b21$ and $b23$ are not allies. The modification of the program to reflect this state of affairs is straightforward:

```
(p(a1);p(a2)), (p(b1); (p(b21);p(b22)), (p(b23);p(b24))),
[x,y]fight(x,y)*- p(x)*p(y)*T
```

Notice that *the defining clause didn't need any modification.*

To modify the Prolog program we could start by adding an extra tag to reflect the structure of the problem like this

```
p(a1,t1,_). p(a2,t1,_).      fight(X,Y):- p(X,T,_),p(Y,U,_),T\=U.
p(b1,t2,_).                  fight(X,Y):- p(X,T,V),p(Y,U,W),T=U,V\=W.
p(b21,t2,t1). p(b22,t2,t1).
p(b23,t2,t2).p(b24,t2,t2).
```

and we should be aware that the whole program has had to be modified to account for the extra tag. Or a new, more flexible implementation may be dreamed up, like using lists of tags as a second argument:

```
p(a1,[t1]).       fight(X,Y):- p(X,U),p(Y,S),mismatch(U,S).
p(a2,[t1]).       mismatch([H1|_],[H2|_]):- H1\=H2.
p(b1,[t2]).       mismatch([H1|T1],[H2|T2]):- H1=H2,mismatch(T1,T2).
p(b21,[t2,t1]).
p(b22,[t2,t1]).
p(b23,[t2,t2]).
p(b24,[t2,t2]).
```

Please compare the heavy machinery used in this example with the simplicity of the BLP version.

The bunch structure also helps to give fine control over the scope of predicates, specially implications. In the example above, we can think of a variety of ways in which constants can be predicated. For example $a2$ might be a special kind of person. It would be possible to modify the program in the following way:

```
(p(a1);q(a2);[x]p(x) <- q(x)),
(p(b1);(p(b21);q(b22)),(p(b23);p(b24))),
[x,y]fight(x,y)*- p(x)*p(y)*T
```

Now this program says that $a2$ is a q but also that all qs are ps. However, this relation between ps and qs holds only for the group formed by $a1$ and $a2$, *i.e.*, is *local* to that world. Other qs appearing in other places in the program, for example $b22$, will not be picked up by the *local* implication, or $\rightarrow$, (which matches the ";" combining $p(a1)$ and $q(a2)$).

The language BLP has been implemented, in the continuation-passing style, by one of us (Armelín), using the OCaml system [3].

Acknowledgements. Armelín is supported by an EPSRC Research Studentship. Pym acknowledges the support of the EPSRC via an Advanced Fellowship.

References

1. W. Clocksin. *Clause and effect.* Springer-Verlag, 1997.
2. W. Clocksin and C. Mellish. *Programming in Prolog.* Springer-Verlag, 1994.
3. G. Cousineau and M. Mauny. *The Functional Approach to Programming.* Cambridge University Press, 1998.
4. J.-Y. Girard. Linear logic. *Theoretical Computer Science*, pages 1–102, 1987.
5. J.A. Harland, D.J. Pym, and M. Winikoff. Programming in Lygon: an overview. In M. Wirsing and M. Nivat, editors, *Proc. AMAST '96*, volume 1101 of *LNCS*, pages 391–405. Springer, 1996.
6. I. Cervesato J. Hodas and F. Pfenning. Efficient resource management for linear logic proof search. *Theoretical Computer Science*, 232:133–163, 2000.
7. J.S. Hodas and D. Miller. Logic programming in a fragment of intuitionistic linear logic. *Information and Computation*, 110(2):327–365, 1 May 1994.
8. R. Kowalski. *Logic for Problem-solving.* North-Holland, Elsevier, 1979.
9. S. A. Kripke. Semantical analysis of intuitionistic logic I. In J. N. Crossley and M. A. E. Dummett, editors, *Formal Systems and Recursive Functions*, pages 92–130. North-Holland, Amsterdam, 1965.
10. D. Miller. A logical analysis of modules in logic programming. *J. Logic. Programming*, 6(1& 2):431–483, 1981.
11. D. Miller, G. Nadathur, F. Pfenning, and A. Ščedrov. Uniform proofs as a foundation for logic programming. *Annals of Pure and Applied Logic*, 51:125–157, 1991.
12. P.W. O'Hearn and D.J. Pym. The logic of bunched implications. *Bulletin of Symbolic Logic*, 5(2):215–244, June 1999.
13. P.W. O'Hearn, D.J. Pym, and H. Yang. Possible worlds and resources: The semantics of **BI**. Submitted. Manuscript available at http://www.dcs.qmw.ac.uk/~pym, 2000.
14. D.J. Pym. On bunched predicate logic. In *Proc. LICS'99*, pages 183–192. IEEE Computer Society Press, 1999.
15. D.J. Pym. The Semantics and Proof Theory of the Logic of the Logic of Bunched Implications. Draft of research monograph, available at http://www.dcs.qmw.ac.uk/ pym, 2000.
16. D.J. Pym and J.A. Harland. A uniform proof-theoretic investigation of linear logic programming. *J. Logic. Computat.*, 4:175–207, 1994.
17. A. Urquhart. Semantics for relevant logics. *Journal of Symbolic Logic*, pages 1059–1073, 1972.