QUEEN MARY
AND WESTFIELD COLLEGE
UNIVERSITY OF LONDON

Department of Computer Science

Research Report No. RR-02-02 ISSN 1470-5559 September 2002

Programming with Bunched Implications

Pablo Armelin

Programming with Bunched Implications

Pablo Armelín

Submitted for the degree of Doctor of Philosophy

Queen Mary, University of London

2002

Acknowledgements

I wish to thank David Pym for his seemly inexhaustible amount of patience during the supervision of this thesis.

Also to my fellow PhD students Mike Samuels and Jules Bean for endless discussions on and off (and on again) topic.

For the writing of this thesis I used Jules Bean's "mathlig" package. Jules also implemented TeX macros used in Chapter 8.

Finally, thanks to all the academic and non-academic members of the Department of Computer Science, Queen Mary, Universisy of London for their unconditional support through all these years.

Programming with Bunched Implications

Pablo Armelín

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 examples of how the resulting programming language handles sharing and non-sharing use of resources purely logically and contrast them with Prolog. We describe the use of modules and their applications and discuss the possibilities offered in this context by multiplicative quantifiers. We provide a denotational semantics based on the construction of a least fixed point of Herbrand interpretations. Finally we provide an annotated implementation of the operational semantics using the continuation-passing style (CPS).

Submitted for the degree of Doctor of Philosophy

Queen Mary, University of London

2002

Contents

Chapter 1

Introduction

1.1 The logic landscape before BI

From the early days of philosophy people has been fascinated by the possibilities opened by our capacity for thought. Aristotle seems to be the first that realized that thought processes can be abstracted and judged by their shape rather than their content, and gave a series of formal rules known as predication rules.

For thousands of years these rules were considered the ultimate truth about valid reasoning. It wasn't until the appearance of Boolean logic that the development of logic took new impetus. The different, albeit equivalent, systems arising are nowadays known as classical logic.

Intuitionistic logic arises from some problems observed with classical logic. The main philosophical objection to classical logic was raised by Brower and it consists in its non-constructive character. For a good description of classical and intuitionistic logic see [van Dalen 97]. Classical and intuitionistic logic are well suited to work with universal, unchangeable truths. But problems arise when we are dealing with finite *resources*.

As an example of the problems that might arise, consider this theorem of intuitionistic logic: $(A \to B) \wedge (A \to C) \vdash A \to (B \wedge C)$

A proof in the sequent calculus is:

$$\dfrac{\dfrac{\dfrac{A, A \to B \vdash B}{A, A \to B, A \to C \vdash B}\,\text{Weakening} \qquad \dfrac{A, A \to C \vdash C}{A, A \to B, A \to C \vdash C}\,\text{Weakening}}{A, A \to B, A \to C \vdash B \wedge C}\,\wedge R}{A \to B, A \to C \vdash A \to (B \wedge C)}\,\to R$$

This theorem should be valid for any interpretations of A, B or C. But it looks a bit strange if we interpret them as

- A: “I have a pound”
- B: “I can buy a drink”
- C: “I can buy a sandwich”

because even though it is true that with a pound we can buy a drink and also that with a pound we can buy a sandwich, we certainly cannot buy *both* a drink and a sandwich with only one pound.

Linear logic serves to reason about *consumption* of resources.

A typical rule in linear logic is $\otimes R$ where the resources needed to prove each branch are explicitly different

$$\dfrac{\Gamma \vdash \phi \qquad \Delta \vdash \psi}{\Gamma, \Delta \vdash \phi \otimes \psi}\,\otimes R$$

and, given that no weakenings or contractions are allowed, it is easy to see that Γ and Δ need to be disjoint. Effectively this amounts to having strict control over the resources used.

However, the elimination of the structural rules turns out to be somewhat restrictive. To recover the power of intuitionistic and classical logic it is necessary to provide a way for this structural rules to be available, at least in some situations. The way that this problem is solved in linear logic is through the exponentials “!” (pronounced “of course”) and “?” (pronounced “why not?”). A proposition ϕ that can be used an arbitrary number of times is written $!\phi$. With the exponentials it is possible to find an embedding of linear logic into classical logic. In particular the intuitionistic implication ($\to$) uses an exponential together with the linear implication ($\phi \to \psi =_{def} !\phi \multimap \psi$).

1.2 A semantic view of a logic for resources

The use of linear logic as a logic for control of resources is a *post-hoc* development. Suffice to say that there is no mention of resources, their consumption or interaction in [Girard 87].

Moreover, the notion of resource that linear logic affords is very crude: either something is available or it has been consumed.

However, if we take the notion of resource as a primitive, there are some behaviours and operations that arise naturally from the concept. Consider the following, very simple, axiomatisation of the notion of resource (clearly, refinements are possible):

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

Mathematically, we recognise that we have naturally identified a commutative preordered monoid $\mathcal{M} = (M, e, \cdot, \sqsubseteq)$ of resources. Commutativity is not essential in this context.

First, we may exploit the presence of the monoidal combining operation to define the following multiplicative conjunction [Urquhart 72, O'Hearn Pym 99, Pym 99a, Pym 02], in the possible-worlds style [Kripke 65]:

$$m \models \phi * \psi \quad \text{iff} \quad \begin{array}{l}\text{there are } n \text{ and } n' \text{ such that if } m \sqsubseteq n \cdot n', \text{ then it is} \\ \text{the case that } n \models \phi \text{ and } n' \models \psi.\end{array}$$

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

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

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:

$$\begin{array}{rll} m \models \phi \wedge \psi & \text{iff} & m \models \phi \text{ and } m \models \psi. \\ m \models \phi \vee \psi & \text{iff} & m \models \phi \text{ or } m \models \psi. \\ m \models \phi \to \psi & \text{iff} & \text{for all } n \sqsubseteq m, \text{ it is the case that } n \models \phi \text{ implies } n \models \psi. \end{array}$$

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.

The unit of the additive disjunction, $\perp$, holds nowhere. However, letting $\perp$ be part of the language causes problems that will be discussed in more detail in §2.4.

We refer to the meaning of the semantics described here as the *sharing interpretation* [O'Hearn Pym 99, Pym 02]. Similarly, the additives may be seen as describing *local* properties whereas the multiplicatives are *global*. We return to this point in Chapter 5 where we give a concrete, and implemented, programming example.

It is from the resource semantics just described that the logic of Bunched Implications (**BI**) arise. It freely combines an additive (intuitionistic) and a multiplicative (linear) implication as connectives of equal status [O'Hearn Pym 99, Pym 99a, Pym 02]. Thus it stands in stark contrast with linear logic [Girard 87], in which intuitionistic implication is available via the exponential [O'Hearn Pym 99, Pym 99a, Pym 02].

BI is also a resource-aware logic, but its emphasis is on *interference* between resources. The difference can be seen from the fact that in linear logic it is possible to say for a given resource whether it is linear or not: it is enough to see whether it is qualified with the exponential "!". In contrast, resources in **BI** can be said to be additive or multiplicative *in relation* to some other resources, but they don't have an additive or multiplicative character of their own. For example in the bunch $((a;b),c)$ a is additively related to b, and multiplicatively related to c.

BI is a logic well suited for situations where it is important to keep track of (possibly harmful) interferences between resources.

One of the most obvious examples is the problem of *pointer aliasing* in programming languages which support pointers. To illustrate this point consider two variables x and y. An assignments x:= 3 might change the content of y if the variables were aliases for the same location in memory. In the following example we will use a forward version of Hoare's triples to assert properties of a program, with intuitionistic logic as the underlying logic.

Suppose that x and y are pointers and we have the assertion $\{\mathtt{x} \mapsto 9 \wedge \mathtt{y} \mapsto 9\}$. If we want to know what can be said about x and y after executing the instruction x := 3 we need to know more about y: if it was defined as

```
let y = ref 9
```

then the assertion that holds after the assignment x:= 3 will be $\{\mathtt{x} \mapsto 3 \wedge \mathtt{y} \mapsto 9\}$, but if y was defined as

```
let y = x
```

then y and x are aliased and after the assignment there is a different assertion that holds: $\{\mathtt{x} \mapsto 3 \wedge \mathtt{y} \mapsto 3\}$. The important point to notice is that considering the assertion $\{\mathtt{x} \mapsto 9 \wedge \mathtt{y} \mapsto 9\}$ on its own we have not got enough information to distinguish between these two cases.

BI let us deal with this case because it provides a way of saying explicitly whether two pointers are aliases or not. The first case, where x and y are not aliased, can be expressed by the assertion $\{\mathtt{x} \mapsto 9 * \mathtt{y} \mapsto 9\}$, to mean that x and y point to different regions of the store. In the second case we would need to use the normal intuitionistic conjunction.

Linear logic cannot help us much in this situation, since the problem of aliasing is not related with consumption of resources, but with interference between them. If we try to use linear logic anyway and x and y were made linear then we would not be able to refer to them more than once, which is certainly an undesirable effect of their linearity.

1.3 A proof-theoretic view of bunched logic

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

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

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

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

but how then to write a rule for $\to I$?

$$\frac{\Gamma \,?\, \vdash \psi}{\Gamma \vdash \phi \to \psi}$$

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.*

We give the semantics and proof theory of **BI** in Chapter 2.

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 of 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 thesis. 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** [O'Hearn Pym 99, Pym 99a, Pym 02].

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,

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

where x is not free in Γ, and

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

Here, we assume a simple bunched calculus of term formation [O'Hearn Pym 99, Pym 99a, Pym 02] 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,

$m \models \forall x.\phi$ iff for all $n \sqsubseteq m$ and all t defined at n, it is the case that $n \models \phi[t/x]$

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 $\forall_{\mathsf{new}}$ explains our use of the term "new":

$m \models \forall_{\mathsf{new}} x.\phi$ iff for all n and all t defined at n, it is the case that $m \cdot n \models \phi[t/x]$

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

1.4 Introduction to logic programming

Logic programming, together with functional and imperative programming, is a paradigm based on the clever observation that proof search can be seen as a form of computation. From this point of view, a program is the context where the computation will take place, and a specific request can be asked by presenting a goal to be proved from this program. Given a program Γ, a query, lets call it ϕ, to this program is interpreted as a request to see whether ϕ can be proved from Γ. Another way of saying the same is to see whether there is a proof $\Gamma \vdash \phi$. If ϕ has got existentially quantified variables, the meaning of the query is to find terms t_i such that $\Gamma \vdash \phi[t_i/x_i]$.

1.4.1 Proof search as Computation

However, for this scheme to *work* as a programming language we need to specify how the flow of control will be handled. It is natural, since we are talking about *logic* programming, to think of the logic connectives as means of achieving this flow of control.

To limit the non-determinism of the proof search, proofs should satisfy some properties. First, the proofs should satisfy the *subformula property*: every formula in the premises should appear in the conclusion. This is necessary because in the context of proof search the computation goes backwards, that is from conclusions to premises. If one of these premises didn't appear in the conclusion the program has to somehow guess what is an appropriate premise, or backtrack on this choice, making the performance intolerably inefficient. A typical rule that does not fulfil the subformula property is Cut:

$$\frac{\Gamma \vdash \phi \quad \Delta, \phi \vdash \psi}{\Delta, \Gamma \vdash \psi}\,\mathrm{Cut}$$

Here we can see clearly that given the conclusion, the proposition ϕ had to be created out of thin air. This is the reason why proof search always use *Cut free* calculi.

In the context of a natural deduction system for intuitionistic logic, another rule that does not satisfy the subformula property is the $\to E$ rule:

$$\frac{\Gamma \vdash \phi \to \psi \quad \Gamma \vdash \phi}{\Gamma \vdash \psi} \to E$$

ϕ has to be guessed if we are doing proof search, that is going backwards from $\Gamma \vdash \psi$. For the purpose of logic programming we need to use a sequent calculus presentation of the logic. For intuitionistic logic, a left rule for the implication is

$$\frac{\Gamma \vdash \phi \quad \Gamma, \psi \vdash \chi}{\Gamma, \phi \to \psi \vdash \chi} \to L$$

which does have the subformula property.

1.4.2 Uniform Proofs

Another requirement for a useful programming language is that non-determinism should be kept to a minimum. Towards this end first we define *uniform proofs* as proofs which apply left rules only when no right rules are possible, that is when the right hand side is atomic. We say then that the proof search is *goal directed,* since the behaviour of the program is defined by the connectives given in the goal.

Our notion of *logic programming* is that introduced in [Miller et al. 91, Miller 81], based on the sequent calculus.

Definition 1.4.1 *We say that a proof is* uniform *if all left rules (including weakening and, obviously, axioms) are applied above right rules.*

Sometimes it is possible to transform a proof into a uniform proof, by *permuting* the left rules above the right rules. For example, a proof with shape

$$\dfrac{\Gamma\vdash\phi \qquad \dfrac{\Gamma,\psi\vdash\xi \quad \Gamma,\psi\vdash\chi}{\Gamma,\psi\vdash\xi\wedge\chi}\wedge R}{\Gamma,\phi\to\psi\vdash\xi\wedge\chi}\to L$$

can be transformed to a proof with the shape

$$\dfrac{\dfrac{\Gamma\vdash\phi \quad \Gamma,\psi\vdash\xi}{\Gamma,\phi\to\psi\vdash\xi}\to L \qquad \dfrac{\Gamma\vdash\phi \quad \Gamma,\psi\vdash\chi}{\Gamma,\phi\to\psi\vdash\chi}\to L}{\Gamma,\phi\to\psi\vdash\xi\wedge\chi}\wedge R$$

and if we succeed in permuting any occurrence of a left rule above any right rule, we would have as a result a uniform proof.

But even in intuitionistic logic permutations are not always possible. Consider, for example, the proof of the sequent $p\vee q\vdash p\vee q$. All proofs of this involve applying $\vee L$ before $\vee R$ in the following fashion:

$$\dfrac{\dfrac{p\vdash p}{p\vdash p\vee q}\vee R \qquad \dfrac{q\vdash q}{q\vdash p\vee q}\vee R}{p\vee q\vdash p\vee q}\vee L$$

and the left rule cannot be permuted upwards.

Other left rule that does not permute above all right rules is $\exists L$. This can be seen even in the simple proof of $\exists x.P(x)\vdash\exists x.P(x)$:

$$\dfrac{\dfrac{P(v)\vdash P(v)}{P(v)\vdash\exists x.P(x)}\exists R}{\exists x.P(x)\vdash\exists x.P(x)}\exists L$$

where the left rule cannot be applied first due to the fact that v is not free in the right side of the sequent.

This proof also serves to illustrate another characteristic that the provability relation should satisfy, the *existential property*: every time that $\Gamma\vdash\exists x.\phi$ there should be a term t such that $\Gamma \vdash \phi[t/x]$. Only in this way an answer substitution can be calculated via an application of an $\exists R$ rule. This requirement can be seen as a corollary of the previous case in the predicate setting. It is easy to see that in the proof above this requirement is not fulfilled, since to find the term to be substituted we need to perform first the $\exists L$ rule.

In intuitionistic logic, 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 [Miller et al. 91, Miller 81].

$$\begin{array}{lcll} \text{Definite formulæ} & D & ::= & A \mid D \wedge D \mid G \to A \mid \forall x.D \\ \text{Goal formulæ} & G & ::= & \top \mid A \mid G \wedge G \mid D \to G \mid G \vee G \\ & & & \mid \exists x.G \end{array}$$

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).

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, consisting only of definite formulæ, together with a query, or goal formula, ϕ, in which there is a logical variable x [Kowalski 79]. We use $\vdash_o$ to denote the simple, uniform, *i.e.*, resolution, proof, read from root to leaves.

1.4.3 Context management in linear logic

In intuitionistic logic, each of the reduction operators used in the execution of goal-directed search is additive. This means that it is possible to store the program once and refer to it as needed. In **BI**, however, as in linear logic [Pym Harland 94, Miller 81, Harland et al. 96], we have multiplicative connectives which introduce a computationally significant difficulty. The typical case is $*R$:

$$\frac{\Gamma_1 \vdash_o \phi_1 \quad \Gamma_2 \vdash_o \phi_2}{\Gamma \vdash_o \phi_1 * \phi_2} *R(\Gamma = \Gamma_1, \Gamma_2)$$

Faced with $\Gamma \vdash_o \phi_1 * \phi_2$, the division of Γ into Γ_1 and Γ_2 must be calculated. A naïve approach to this problem is simply to try the two subproofs and check whether the remainders coincide, but the complexity of this algorithm is exponential on the size of Γ.

The basic solution is the so-called *input/output* model. It is described for linear logic in [Hodas Miller 94, Cervesato et al. 00, Pym Harland 94, Harland et al. 96]. 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:

A simplified version of this method is shown below, to clarify the ideas involved.

$$\dfrac{\dfrac{\vdots}{\Gamma_{\mathrm{L}} \vdash_o \phi_1} \qquad \Gamma\backslash\Gamma_{\mathrm{L}} \vdash_o \phi_2}{\Gamma \vdash_o \phi_1 \otimes \phi_2}\ \Gamma = \Gamma_{\mathrm{L}}, \Gamma\backslash\Gamma_{\mathrm{L}}$$

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

1.4.4 Context management in BI

The input/output model is not enough for **BI**. The main reason is that the interaction between additive and multiplicative connectives in linear logic is far simpler. Indeed, for the operational semantics of Lolli, a programming language based on this model, there are two regions, one of them for additive resources and another for multiplicative resources. But the mutually recursive nature of bunches breaks this model. We will give a solution to these problems in Chapter 4, when we develop the operational semantics for `BLP`.

In **BI**, the problem is made 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):

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

Consider the left-hand branch of the candidate $*R$ rule 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 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 $\tau;(\phi,\psi)\vdash(\chi\to(\psi\wedge\tau))*\phi$.

$$\dfrac{\dfrac{\dfrac{\dfrac{}{\chi;\tau;(\phi,\psi)\backslash\phi\vdash\psi}\,\text{Axiom} \qquad \dfrac{}{\chi;\tau;(\phi,\psi)\backslash\phi\vdash\tau}\,\text{Axiom}}{\chi;\tau;(\phi,\psi)\backslash\phi\vdash\psi\wedge\tau}\wedge R}{\tau;(\phi,\psi)\backslash\phi\vdash\chi\to(\psi\wedge\tau)}\to R \qquad \dfrac{}{\phi\backslash\emptyset_m\vdash\phi}}{\tau;(\phi,\psi)\backslash\emptyset_m\vdash(\chi\to(\psi\wedge\tau))*\phi}\,*R$$

Once the $*R$ rule is applied, and after doing $\to R$, the propositions τ and χ are at the same level in $(\chi;\tau;(\phi,\psi))\backslash\phi$. If this is taken to be equal to $\chi;\tau;\psi$, both χ and τ are provable. But this would be unsound; a proof of $\tau;(\phi,\psi)\vdash(\chi\to(\psi\wedge\tau))*\phi$ should fail, since ψ and ϕ are separated by a $*$ and, therefore, any search for a proof should start by weakening τ so that ϕ and ψ can be sent to the two subproofs of $*R$:

$$\dfrac{\dfrac{\dfrac{}{\psi\vdash\chi\to(\psi\wedge\tau)}\,\text{not provable} \qquad \dfrac{}{\phi\vdash\phi}}{\phi,\psi\vdash(\chi\to(\psi\wedge\tau))*\phi}}{\tau;(\phi,\psi)\vdash(\chi\to(\psi\wedge\tau))*\phi}\,\text{Weakening}$$

but after weakening τ it is unavailable for the proof of the left branch.

Again it looks like this could be fixed, by requiring, for example, that not only we avoid doing weakenings in the subtraction operator, but also that the $*R$ rule be applied only on multiplicative bunches. The first objection with this approach is the unpleasant non-determinism introduced, since any additive bunch will have to use each of its components consecutively trying to find a proof.

But far worst is that 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 sequent $\xi,(\tau;(\phi,\psi)) \vdash (\chi \to (\psi \wedge \tau)) * (\phi * \xi)$. Using a non-weakening axiom plus the restriction mentioned above, we find that $\chi;(\xi,(\tau;(\phi,\psi)))\backslash(\xi,\phi) \equiv \chi;\tau;\psi$ and a proof can be built as follows:

$$\dfrac{\dfrac{\dfrac{\chi;\tau;\psi \vdash \psi \quad \chi;\tau;\psi \vdash \tau}{\chi;(\xi,(\tau;(\phi,\psi)))\backslash\xi,\phi \vdash \psi \wedge \tau}\wedge R}{\xi,(\tau;(\phi,\psi))\backslash\xi,\phi \vdash \chi \to (\psi \wedge \tau)}\to R \qquad \dfrac{\xi,\phi\backslash\emptyset_m \vdash \phi * \xi}{}}{\xi,(\tau;(\phi,\psi))\backslash\emptyset_m \vdash (\chi \to (\psi \wedge \tau)) * (\phi * \xi)} *R$$

Here, after the application of $*R$ and $\to R$ it would be possible to prove both ψ and τ, which is again unsound. The reason is similar to the previous example: for a proof of the sequent the bunch $\xi,(\tau;(\phi,\psi))$ must be split in a way that ψ goes to the left branch and ξ and ϕ go to the right branch, and this cannot be done without weakening first τ, making it unavailable thereafter.

The main source of difficulty is that weakening is not permutable with the $*R$ rule. The shape of the rule in **BI** is

$$\dfrac{\Gamma \vdash \psi \qquad \Delta \vdash \phi}{\Gamma,\Delta \vdash \psi * \phi} *R$$

where it can be seen that the $*R$ rule *requires* a multiplicative bunch for its application. If this problem was unsolvable we could as well give up trying to use **BI** as the basis for a programming language, because there are exponentially many ways of performing weakenings, given an arbitrary bunch with nested additive subbunches.

The basic idea for the solution, though far from simple in detail, is to introduce stacks which keep a record of which resources have been added to the program as a result of $\to R$ and which manage their interaction with the formation of axioms, and with subtraction, and with passing by continuations. The detailed formulation of the continuation-passing style (CPS) operational semantics is rather complex and before giving it, in Chapter 4, we must look, in Chapter 3, at uniform, and simple, proofs in **BI**.

1.5 Thesis Overview

In Chapter 2 we give a natural deduction system for **BI** and show soundness and completeness of this system with respect to the resource semantics explained in §1.2. We also give a sequent calculus presentation of the logic.

Chapter 3 and Chapter 4 form the core work of the thesis.

In Chapter 3 we extend the concepts introduced in this chapter to the case of logic programming in **BI**. In particular we show how to obtain uniform proofs, resolution proofs, and simple proofs for the hereditary Harrop fragment of **BI**.

In Chapter 4 we present an operational semantics that solves the problems arising from the interactions between the multiplicative and additive structures of the logic. We prove soundness and completeness of this semantics with respect to the sequent calculus presented in Chapter 2.

In Chapter 5 we offer some basic examples of how this programming language works and for which kind of problems it could be useful. We present the case of belligerent groups and how their behaviour is easily expressed in this logic, and how to program a type inference system for the $\alpha\lambda$-calculus. We also discuss issues related to linear recursion.

Chapter 6 provides a denotational semantics based on the construction of a least fixed point of Herbrand interpretations. This is joint work with David Pym.

We discuss some conclusions and further work in Chapter 7 and in Chapter 8 we provide a fully annotated implementation of the operational semantics using the continuation-passing style(CPS). This implementation was done in OCaml **??**.

Chapter 2

The proof theory and semantics of BI

2.1 Introduction

In this chapter we present **NBI**, a propositional and predicate natural deduction system for **BI**, describe the term language for **BI** and discuss soundness and completeness of **NBI** without $\perp$ with respect to the resource semantics described in the previous chapter.

- Give a grammar for bunches and explain the additive and multiplicative aspects of the logic.
- Present **NBI**, a natural deduction system for **BI**.
- Discuss soundness and completeness of **NBI** without $\perp$ with respect to the resource semantics described in the previous chapter.
- Present **LBI**, a sequent calculus system for **BI**.

2.2 Bunches

The grammar of bunches is given as follows:

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

We write $\Gamma(\Delta)$ to denote that Δ is a sub-bunch of Γ in the evident sense. Equality of bunches, $\equiv$, 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 ",":

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

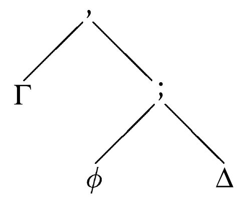
The introduction and elimination rules for the multiplicative and additive implications now go as follows:

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

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

Definition 2.2.1 *A bunch is* additive *if it's main connective is a semicolon (;) and* multiplicative *otherwise.*

As an example the following bunch is multiplicative, with an additive right branch which in turn has an additive proposition.



This bunch will be denoted as $\Gamma,(\phi;\Delta)$.

Due to the two connectives used to form bunches, almost every aspect of the logic can be seen as *additive* or *multiplicative* depending on whether is related to

";" or ",". For example we can talk about additive or multiplicative bunches, additive or multiplicative connectives, and additive or multiplicative units ($\emptyset_a$ and $\emptyset_m$ respectively).

So the meaning of additive or multiplicative theorems should be clear. We give an example of each:

$$\dfrac{\dfrac{\dfrac{}{a \vdash a}\;\text{Axiom}}{\emptyset_m, a \vdash a}\;\text{Monoid equation}}{\emptyset_m \vdash a \mathrel{-\!\!*} a}\;{-\!\!*}R \qquad \dfrac{\dfrac{\dfrac{}{a \vdash a}\;\text{Axiom}}{\emptyset_a; a \vdash a}\;\text{Monoid equation}}{\emptyset_a \vdash a \to a}\;\to R$$

One thing to notice in the additive theorem is that the monoid operation could have been replaced by a Weakening and the proof would still have been valid.

An example of a non-theorem is $\emptyset_a \nvdash a \mathrel{-\!\!*} a$.

We are ready now to prove a small and useful lemma:

Lemma 2.2.2 *Additive theorems are also multiplicative.*

Proof: Assume $\emptyset_a \vdash \phi$. Then the following transformation is possible:

$$\dfrac{\dfrac{\emptyset_a \vdash \phi}{\emptyset_m; \emptyset_a \vdash \phi}\;\text{Weakening}}{\emptyset_m \vdash \phi}\;\text{Monoid equation}$$

□

Because a crucial step in the proof of this lemma is the use of Weakening, the converse is not true.

2.3 A Natural Deduction System

We present here a natural deduction system for **BI**. We give introduction and elimination rules for the multiplicative and additive connectives of propositional **BI** and call the resulting system **NBI**. The rules are displayed in Table 2.1.

2.3.1 Term Language

The corresponding term language for **BI** is the $\alpha\lambda$-calculus. The types of the language represent propositions, and terms of a particular type are proofs of the corresponding proposition.

$$\frac{}{\phi \vdash \phi}\,\text{Axiom} \qquad \frac{\Gamma \vdash \phi}{\Delta \vdash \phi}\,(\text{where } \Delta \equiv \Gamma)E$$

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

$$\frac{}{\emptyset_m \vdash I}\,II \qquad \frac{\Gamma(\emptyset_m) \vdash \phi \quad \Delta \vdash I}{\Gamma(\Delta) \vdash \phi}\,IE$$

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

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

$$\frac{}{\emptyset_a \vdash \top}\,\top I \qquad \frac{\Gamma(\emptyset_a) \vdash \phi \quad \Delta \vdash \top}{\Gamma(\Delta) \vdash \phi}\,\top E$$

$$\frac{\Gamma \vdash \bot}{\Gamma \vdash \psi}\,\bot E$$

$$\frac{\Gamma; \phi \vdash \psi}{\Gamma \vdash \phi \to \psi}\to I \qquad \frac{\Gamma \vdash \phi \to \psi \quad \Delta \vdash \phi}{\Gamma; \Delta \vdash \psi}\to E$$

$$\frac{\Gamma \vdash \phi \quad \Delta \vdash \psi}{\Gamma; \Delta \vdash \phi \wedge \psi}\,\wedge I \qquad \frac{\Gamma(\phi; \psi) \vdash \chi \quad \Delta \vdash \phi \wedge \phi}{\Gamma(\Delta) \vdash \chi}\,\wedge E$$

$$\frac{\Gamma \vdash \phi_i}{\Gamma \vdash \phi_1 \vee \phi_2}\,(i = 1, 2) \vee I \qquad \frac{\Delta \vdash \phi \vee \psi \quad \Gamma(\phi) \vdash \chi \quad \Gamma(\psi) \vdash \chi}{\Gamma(\Delta) \vdash \chi}\,\vee E$$

Table 2.1: Natural Deduction System **NBI**

We give here the rules for conjunction and implication, both in their additive and multiplicative version, for illustrative purposes. A full discussion is given in Chapter 5.

$$\frac{\Gamma \vdash M : \phi \quad \Delta \vdash N : \psi}{\Gamma, \Delta \vdash M * N : \phi * \psi} *I \qquad \frac{\Gamma(x : \phi, y : \psi) \vdash M : \chi \quad \Delta \vdash N : \phi * \psi}{\Gamma(\Delta) \vdash \mathbf{let}(x, y)\mathbf{be}\ N\ \mathbf{in}\ M : \chi} *E$$

$$\frac{\Gamma, x : \phi \vdash M : \psi}{\Gamma \vdash \lambda x : \phi.M : \phi \mathrel{-\!\!*} \psi} \mathrel{-\!\!*} I \qquad \frac{\Gamma \vdash M : \phi \mathrel{-\!\!*} \psi \quad \Delta \vdash N : \phi}{\Gamma, \Delta \vdash MN : \psi} \mathrel{-\!\!*} E$$

$$\frac{\Gamma \vdash M : \phi \quad \Delta \vdash N : \psi}{\Gamma; \Delta \vdash M \wedge N : \phi \wedge \psi} \wedge I \qquad \frac{\Gamma \vdash M : \phi_1 \wedge \phi_2}{\Gamma \vdash \pi_i M : \phi_i} (i = 1, 2) \wedge E$$

$$\frac{\Gamma; x : \phi \vdash M : \psi}{\Gamma \vdash \alpha x : \phi.M : \phi \to \psi} \to I \qquad \frac{\Gamma \vdash M : \phi \to \psi \quad \Delta \vdash N : \phi}{\Gamma; \Delta \vdash M@N : \psi} \to E$$

Lemma 2.3.1 *The $\alpha\lambda$-calculus is strongly normalising.*

Proof: Given strong normalisation for the simply-typed λ-calculus, we can translate the multiplicative connectives into their additive counterparts ($\mathrel{-\!\!*}$ into $\to$ and $*$ into $\wedge$) and we get strong normalisation for the $\alpha\lambda$-calculus. It is easy to see that if there is a reduction in the λ-calculus of a term obtained via this translation from the $\alpha\lambda$-calculus, then the original term can also perform this reduction step.

□

2.4 Models for BI

We have already mentioned the resources interpretation. Here we give a formal definition of what a model is, briefly state soundness and completeness of Kripke models, and mention other possible semantics.

Definition 2.4.1 (Kripke Models) *Let $\mathcal{P}(L)$ be a collection of* **BI** *propositions over a language L of propositional letters.*

Then a Kripke model *is a triple* $\langle[\mathcal{M}^{op},\mathbf{Set}],\models,[\![-]\!]\rangle$ *where* $\mathcal{M}=(M,e,\cdot,\sqsubseteq)$ *is the commutative preordered monoid mentioned in 1.2,* $[\mathcal{M}^{op},\mathbf{Set}]$ *is the category of presheaves over the preorder category* $\mathcal{M}$ *to* **Set**, $\models\,\subseteq M\times\mathcal{P}(L)$ *is a satisfaction relation as described in 1.2:*

$$\begin{array}{rll}
m\models\bot & & \textit{never.} \\
m\models\top & & \textit{always.} \\
m\models I & \text{iff} & m\sqsubseteq e. \\
m\models\phi*\psi & \text{iff} & \textit{there are } n \textit{ and } n' \textit{ such that if } m\sqsubseteq n\cdot n'\textit{, then it is} \\
 & & \textit{the case that } n\models\phi \textit{ and } n'\models\psi. \\
m\models\phi\mathrel{-\!\!*}\psi & \text{iff} & \textit{for all } n \textit{ such that } n\models\phi\textit{, it is the case that } m\cdot n\models\psi. \\
m\models\phi\wedge\psi & \text{iff} & m\models\phi \textit{ and } m\models\psi. \\
m\models\phi\vee\psi & \text{iff} & m\models\phi \textit{ or } m\models\psi. \\
m\models\phi\to\psi & \text{iff} & \textit{for all } n\sqsubseteq m\textit{, it is the case that } n\models\phi \textit{ implies } n\models\psi.
\end{array}$$

and $[\![-]\!]$ *is a partial function from the* **BI** *propositions over* L *to the objects of* $[\mathcal{M}^{op},\mathbf{Set}]$ *such that Kripke monotonicity is preserved.*

Theorem 2.4.2 (Soundness of BI for Kripke models) *If* $\Gamma\vdash\phi$ *in* **NBI** *without* $\bot$*, then* $\Gamma\models\phi$.

Proof: For more details of the proof see [Pym 02]. Here we provide a sketch analysing two typical cases:

$*I$ The induction hypotheses are that if a world forces Γ it also forces ϕ, and if a world forces Δ it also forces ψ. We need to show that if a world forces Γ,Δ it also forces $\phi*\psi$.

That a world forces Γ,Δ means that it is the result of the monoid operation applied to two worlds, call them m_1 and m_2, which respectively force Γ and Δ. But then $m_1\models\phi$ and $m_2\models\psi$. According to the forcing semantics, this means that $m_1\cdot m_2\models\phi*\psi$.

$\mathrel{-\!\!*}I$ By the induction hypothesis we know that for all worlds w, if $w\models\Gamma,\phi$ then $w\models\psi$. We have to prove that if a world m forces Γ, then it also forces $\phi\mathrel{-\!\!*}\psi$. According to the forcing semantics, $\phi \mathrel{-\!\!*} \psi$ is forced in a world m' iff for all worlds n where ϕ is forced, $m' \cdot n \models \psi$. So given a world where Γ is forced, and a world where ϕ is forced, we can construct a world where Γ, ϕ is forced. Applying then the induction hypothesis we find that this world also forces ψ, which proves that m forces also $\phi \mathrel{-\!\!*} \psi$.

□

To show completeness of **NBI** without $\bot$ we need the following lemma:

Lemma 2.4.3 *If $\Gamma \nvdash \phi$ in* **NBI** *without $\bot$, then there is a Kripke model K and world $w \in K$ such that $w \models \Gamma$ but $w \not\models \phi$.*

Proof: The proof of this lemma is difficult and highly technical. For an exhaustive explanation we refer the reader to [Pym 02]. It extends the concept of *prime theory* described in [van Dalen 97] to the setting of bunches (defining *prime bunches*) and adds the idea of *prime evaluation*, a transformation of bunches closed under consequence.

□

With the help of Lemma 2.4.3 it is easy to establish completeness.

Theorem 2.4.4 (Completeness of BI for Kripke models) *If $\Gamma \models \phi$ in* **NBI** *without $\bot$, then $\Gamma \vdash \phi$.*

Proof: If $\Gamma \nvdash \phi$ then, by Lemma 2.4.3 we would have a world $w \in K$ in which $w \models \Gamma$ but $w \not\models \phi$, which contradicts the antecedent of the theorem.

□

The fact that $\bot$ holds in no world, does not present problems for soundness, but addition of $\bot$ to the logic would make it incomplete. An example of a sequent that holds in the Kripke resource semantics, but for which there is no proof in **NBI** is

$$(\phi \mathrel{-\!\!*} \bot) \to \bot ; (\psi \mathrel{-\!\!*} \bot) \to \bot \models ((\phi * \psi) \mathrel{-\!\!*} \bot) \to \bot$$

To see that this sequent holds in all Kripke resource semantics, we will first prove a lemma:

Lemma 2.4.5 $m \models (\phi \mathrel{-\!\!*} \bot) \to \bot$ *iff there is an* n *such that* $n \models \phi$.

Proof: From the forcing semantics we have that $m \models (\phi \mathrel{-\!\!*} \bot) \to \bot$ iff for all $m' \sqsubseteq m$, if $m' \models \phi \mathrel{-\!\!*} \bot$ then $m' \models \bot$, but since $\bot$ is forced nowhere, then there should be no such m'.

Using again the forcing semantics we have that $m' \not\models \phi \mathrel{-\!\!*} \bot$ iff $\exists n, n \models \phi$ and $n \not\models \bot$. The second conjunct holds always, so this is equivalent to $\exists n, n \models \phi$

□

Going back to the forcing sequent, note that from Lemma 2.4.5 the antecedent will hold iff there are n_1 and n_2 such that $n_1 \models \phi$ and $n_2 \models \psi$. Therefore, by the resource semantics, there is a world, namely $n_1 \cdot n_2$ which forces $\phi * \psi$, and using again Lemma 2.4.5 this implies that $((\phi * \psi) \mathrel{-\!\!*} \bot) \to \bot$ holds everywhere.

When we are looking for a proof of this sequent in **NBI**, after the first $\to I$ rule there are essentially two attempts that can be tried, but both of them fail. If we try the $\mathrel{-\!\!*} E$ rule the failure is obvious:

$$\dfrac{\dfrac{\dfrac{}{(\phi * \psi) \mathrel{-\!\!*} \bot \vdash (\phi * \psi) \mathrel{-\!\!*} \bot}\text{Axiom} \qquad \dfrac{}{(\phi \mathrel{-\!\!*} \bot) \to \bot ; (\psi \mathrel{-\!\!*} \bot) \to \bot \vdash \phi * \psi}\text{Not provable}}{(\phi \mathrel{-\!\!*} \bot) \to \bot ; (\psi \mathrel{-\!\!*} \bot) \to \bot ; (\phi * \psi) \mathrel{-\!\!*} \bot \vdash \bot}{\mathrel{-\!\!*} E}}{(\phi \mathrel{-\!\!*} \bot) \to \bot ; (\psi \mathrel{-\!\!*} \bot) \to \bot \vdash ((\phi * \psi) \mathrel{-\!\!*} \bot) \to \bot}{\to I}$$

where it is obvious that $\phi * \psi$ cannot be derived from the premises, since they should consist on a multiplicative bunch with ϕ and ψ in different subbunches, which is not the case.

If we tried instead applying a $\to E$ rule, we would reach the following figure:

$$\dfrac{\dfrac{\dfrac{}{(\phi \mathrel{-\!\!*} \bot) \to \bot \vdash (\phi \mathrel{-\!\!*} \bot) \to \bot}\text{Axiom} \qquad \dfrac{\dfrac{}{((\psi \mathrel{-\!\!*} \bot) \to \bot ; (\phi * \psi) \mathrel{-\!\!*} \bot), \phi \vdash \bot}\text{Not provable}}{(\psi \mathrel{-\!\!*} \bot) \to \bot ; (\phi * \psi) \mathrel{-\!\!*} \bot \vdash \phi \mathrel{-\!\!*} \bot}{\mathrel{-\!\!*} I}}{(\phi \mathrel{-\!\!*} \bot) \to \bot ; (\psi \mathrel{-\!\!*} \bot) \to \bot ; (\phi * \psi) \mathrel{-\!\!*} \bot \vdash \bot}{\mathrel{-\!\!*} E}}{(\phi \mathrel{-\!\!*} \bot) \to \bot ; (\psi \mathrel{-\!\!*} \bot) \to \bot \vdash ((\phi * \psi) \mathrel{-\!\!*} \bot) \to \bot}{\to I}$$

To see why $((\psi -\!\!* \bot) \to \bot; (\phi * \psi) -\!\!* \bot), \phi \vdash \bot$ is not provable, note that to apply the $-\!\!*E$ rule, one first has to weaken the premise $(\psi -\!\!* \bot) \to \bot$, losing any chance of finding a proof of $\phi * \psi$.

The way that this incompleteness is solved in [Pym 02] is using topological Kripke models, which provides an inconsistent world where $\bot$ holds, more specifically a Grothendieck topology. Alternatively it is possible to give a partial monoid semantics as described in [Galmiche et al. 02].

2.5 Predicate BI

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 of 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 $*I$ rule are

$$\frac{\{X\}\,\Gamma \vdash \phi \quad \{Y\}\,\Delta \vdash \psi}{\{X;Y\}\,\Gamma,\Delta \vdash \phi * \psi}\,*R_a \qquad \frac{\{X\}\,\Gamma \vdash \phi \quad \{Y\}\,\Delta \vdash \psi}{\{X,Y\}\,\Gamma,\Delta \vdash \phi * \psi}\,*R_m$$

The former choice is the one taken in linear logic and in this thesis. 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** [O'Hearn Pym 99, Pym 99a, Pym 02].

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,

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

where x is not free in Γ, and

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

Here, we assume a simple bunched calculus of term formation [O'Hearn Pym 99, Pym 99a, Pym 02] and, as before, the additive case may be simplified to use just one bunch of variables. The corresponding existentials are similar.

2.6 A sequent calculus system

We have seen in Chapter 1 that a set of rules based in the natural deduction system is not appropriate for proof search or logic programming because the non-determinism associated to some of the rules, in particular the elimination of the implications.

The natural deduction system presented earlier is Cut free, or rather Cut can be eliminated from it without losing completeness. However there are other rules where the subformula property does not hold. Specifically the elimination rules for the two implications:

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

Notice that ϕ in both rules has to be guessed if we were doing proof search, that is going upwards from $\Gamma, \Delta \vdash \psi$.

We conclude that **NBI** is not an adequate system for our purposes. We therefore describe a sequent calculus for **BI** which doesn't suffer from this drawback.

Proofs in **BI** may be presented as a sequent calculus, here given in Definition 2.6.1. The Cut-elimination theorem holds [Pym 02] and semantic completeness theorems are available [O'Hearn Pym 99, Pym 99a, Pym 02, Pym et al. 02].

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 [Miller 81], 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.

Definition 2.6.1 (LBI) *The sequent calculus* **LBI** *is defined as follows:*

Identity and Structure

$$\frac{X_1 \vdash \phi(X_1) : \mathbf{Prop} \quad X_2 \vdash \phi(X_2) : \mathbf{Prop}}{\{X_1 \odot X_2\}\ \phi(X_1) \vdash \phi(X_2)}\ \mathrm{Axiom}(\mathrm{X_1}, \mathrm{X_2})(\text{where } \odot = ; \text{or},)$$

$$\frac{\{X\}\ \Gamma \vdash \phi}{\{Y\}\ \Delta \vdash \phi}\ (\text{where } \{Y\}\ \Delta \equiv \{X\}\ \Gamma)\ E$$

$$\frac{\{Y(X)\}\ \Gamma(\Delta) \vdash \phi}{\{Y(X;X')\}\ \Gamma(\Delta;\Delta') \vdash \phi}\ (X' \vdash \Delta' : \mathbf{Prop})\ W$$

$$\frac{\{Y(X;X')\}\ \Gamma(\Delta;\Delta') \vdash \phi}{\{Y(X)\}\ \Gamma(\Delta) \vdash \phi}\ (\Delta' \cong \Delta, X' \cong X, X' \vdash \Delta' : \mathbf{Prop})\ C$$

$$\frac{\{X(x)\}\ \Gamma \vdash \phi \quad Y \vdash t : \mathit{Term}}{\{X(Y)\}\ \Gamma[t/x] \vdash \phi[t/x]}\ \mathrm{Axiom}(Y, x')\text{Substitution}$$

for all x' s.t. Axiom(x, x').

Multiplicatives

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

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

$$\frac{\{X, x\}\ \Gamma \vdash \phi(x)}{\{X\}\ \Gamma \vdash \forall_{\mathsf{new}} x.\phi(x)}\ \forall_{\mathsf{new}} R$$

$$\frac{\{X(Y)\}\ \Gamma(\phi(t)) \vdash \psi}{\{X(I)\}\ \Gamma(\forall_{\mathsf{new}} x.\phi(x)) \vdash \psi}\ (Y \vdash t : \mathit{Term})\ \forall_{\mathsf{new}} L$$

$$\frac{\{X, Y\}\ \Gamma \vdash \phi[t/x]}{\{X\}\ \Gamma \vdash \exists_{\mathsf{new}} x.\phi(x)}\ (Y \vdash t : \mathit{Term})\ \exists_{\mathsf{new}} R$$

$$\frac{\{X(x)\}\ \Gamma(\phi(x)) \vdash \psi}{\{X(I)\}\ \Gamma(\exists_{\mathsf{new}} x.\phi(x)) \vdash \psi}\ \exists_{\mathsf{new}} L$$

Additives

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

$$\frac{\{X\}\,\Gamma\vdash\psi \quad \{Y\}\,\Delta\vdash\phi}{\{X;Y\}\,\Gamma;\Delta\vdash\psi\wedge\phi}\wedge R \qquad \frac{\{X\}\,\Gamma(\phi;\psi)\vdash\chi}{\{X\}\,\Gamma(\phi\wedge\psi)\vdash\chi}\wedge L$$

$$\frac{\{X\}\,\Gamma\vdash\phi_i}{\{X\}\,\Gamma\vdash\phi_1\vee\phi_2}\,(i=1,2)\vee R \qquad \frac{\{X\}\,\Gamma(\phi)\vdash\chi \quad \{X\}\,\Gamma(\psi)\vdash\chi}{\{X\}\,\Gamma(\phi\vee\psi)\vdash\chi}\vee L$$

$$\frac{\{X;x!\}\,\Gamma\vdash\phi(x)}{\{X\}\,\Gamma\vdash\forall x.\phi(x)}\forall R$$

$$\frac{\{X(Y!)\}\,\Gamma(\phi(t))\vdash\psi}{\{X(1)\}\,\Gamma(\forall x.\phi(x))\vdash\psi}\,(Y!\vdash t:\mathit{Term})\ \forall L$$

$$\frac{\{X;Y!\}\,\Gamma\vdash\phi[t/x]}{\{X\}\,\Gamma\vdash\exists x.\phi(x)}\,(Y!\vdash t:\mathit{Term})\ \exists R$$

$$\frac{\{X(x!)\}\,\Gamma(\phi(x))\vdash\psi}{\{X(1)\}\,\Gamma(\exists x.\phi(x))\vdash\psi}\exists L$$

There are some restrictions that have to be taken into account in the quantifier rules. First, all variables are subjected to the linearity condition. This restriction interacts with the well-formedness of the sequents in the following way: looking at the rules upwards, it means that *new* variables have to be used in the instantiation of the bound variables in all cases. Looking at the rules downwards, it means that the variables cut off cannot be free in the rest of the sequent.

Extra conditions for $\forall_{\textsf{new}}R$ and $\forall R$:

$$x\notin\{y|\text{Axiom}(y,z)\text{ and }z\in\text{FV}(\Gamma)\}$$

Extra conditions for $\exists_{\textsf{new}}L$ and $\exists L$:

$$x\notin\{y|\text{Axiom}(y,z)\text{ and }z\in\text{FV}(\psi)\text{ or }\text{FV}(\Gamma)\}$$

Lemma 2.6.2 (Implicit contraction) *The following additive rules which use implicit contraction are admissible in* **LBI***:*

$$\frac{\{X\}\,\Gamma\vdash\phi \quad \{X\}\,\Gamma\vdash\psi}{\{X\}\,\Gamma\vdash\phi\wedge\psi}\wedge R \qquad \frac{\{X\}\,\Gamma\vdash\phi \quad \{Y(X)\}\,\Delta(\Gamma;\psi)\vdash\chi}{\{Y(X)\}\,\Delta(\Gamma;\phi\to\psi)\vdash\chi}\to L$$

Proof: Given the rules in Definition 2.6.1 the rules above can be derived as follows:

$$\cfrac{\cfrac{\{X\}\,\Gamma \vdash \phi \quad \{X\}\,\Gamma \vdash \psi}{\{X;X\}\,\Gamma;\Gamma \vdash \phi \wedge \psi}\wedge R}{\{X\}\,\Gamma \vdash \phi \wedge \psi}\text{Contraction}$$

$$\cfrac{\cfrac{\{X\}\,\Gamma \vdash \phi \quad \{Y(X)\}\,\Delta(\Gamma;\psi) \vdash \chi}{\{Y(X;X)\}\,\Delta(\Gamma;\Gamma;\phi \to \psi) \vdash \chi}\to L}{\{Y(X)\}\,\Delta(\Gamma;\phi \to \psi) \vdash \chi}\text{Contraction}$$

□

A Cut rule is admissible in **LBI**:

$$\cfrac{\{X(X_1)\}\,\Gamma \vdash \phi(X_1) \qquad \{Y(X_2)\}\,\Delta(\phi(X_2)) \vdash \psi}{\{Y(X(U))\}\,\Delta(\Gamma) \vdash \psi}\text{Cut}$$

where U stands for units replacing the bunches of variables cut away.

Lemma 2.6.3 (Cut elimination) *All instances of the Cut rule can be eliminated from* **LBI**.

Proof: As an illustration of the process, we provide the elimination for $-\!\!*$:

$$\cfrac{\cfrac{\{X,X'\}\,\Gamma \vdash \phi \quad \{W(Y')\}\,\Delta(\psi) \vdash \chi}{\{W(X,X',Y')\}\,\Delta(\Gamma,\phi -\!\!* \psi) \vdash \chi}-\!\!*L \qquad \cfrac{\{Y,X',Y'\}\,\Theta,\phi \vdash \psi}{\{Y,X',Y'\}\,\Theta \vdash \phi -\!\!* \psi}-\!\!*R}{\{W(X,Y)\}\,\Delta(\Gamma,\Theta) \vdash \chi}\text{Cut}$$

where $X' \vdash \phi : \mathbf{Prop}$ and $Y' \vdash \psi : \mathbf{Prop}$, translates into

$$\cfrac{\cfrac{\{Y,X',Y'\}\,\Theta,\phi \vdash \psi \quad \{W(Y')\}\,\Delta(\psi) \vdash \chi}{\{W(Y,X')\}\,\Delta(\Theta,\phi) \vdash \chi}\text{Cut} \qquad \{X,X'\}\,\Gamma \vdash \phi}{\{W(X,Y)\}\,\Delta(\Gamma,\Theta) \vdash \chi}\text{Cut}$$

where each Cut rule is of lower degree than the first.

The other cases are similar.

□

Chapter 3

Logic Programming with BI

3.1 Introduction

In this chapter we see how the concepts described in Chapter 1 for logic programming in general apply specifically to logic programming with **BI**. Specifically we

- Define *canonical bunches*
- Define *hereditary Harrop formulae* for **BI**.
- Describe how the concept of *uniform proof* applies to **BI** proofs.
- Describe *resolution* and *simple* proofs for **BI**.
- Give an exhaustive analysis of permutations for the hereditary Harrop fragment of **BI**.

We will use the arrow $\rightsquigarrow$ to denote a proof transformation. This transformation will consist on equivalent proofs, with the same premises and the same conclusion, and will be used extensively in this chapter.

3.2 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.

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

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

or

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

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 $\phi * \psi \vdash \phi * \psi$.

An exhaustive analysis of the permutations is given at the end of this chapter, in section 3.10.

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 premises. 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.

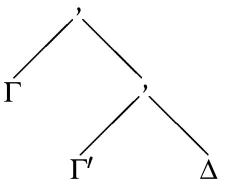
For example, if Γ and Γ' are additive and canonical and if Δ is canonical, then $(\Gamma, \Gamma', \Delta)$ is canonical. Our operational semantics, in Chapter 4 assumes that bunches are in canonical form.

Given associativity of the two bunch-forming operators, the two bunches $\Gamma, (\Delta, \Theta)$ and $(\Gamma, \Delta), \Theta$ are equivalent. As a first step towards getting rid of this redundancy, we define *canonical bunches*:

Definition 3.2.1 *A* canonical bunch *is a bunch with*

- *its left branch either a proposition, one of the units or a canonical bunch of the opposite kind than its parent*
- *its right branch a canonical bunch*

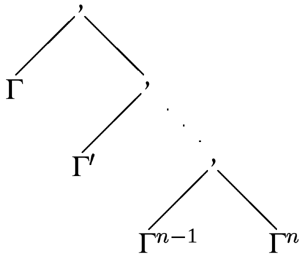
Provided Γ and Γ' are additive and canonical and Δ canonical the following is an example of a multiplicative canonical bunch



Lemma 3.2.2 (Canonical bunch) *Every bunch can be transformed into an equivalent canonical bunch.*

Proof: If the bunch itself is one of the base cases we are done. Otherwise, without loss of generality, lets Γ be a multiplicative bunch. Then it is of the form (Γ', Γ''). If Γ' is additive we can make it canonical by recursion; if it is a proposition or one of the units it is already canonical. And we can recurse on Γ''. Else Γ is of the form $((\Delta, \Delta'), \Gamma'')$. But because of associativity of the monoid operation "," this bunch is equivalent to $(\Delta, (\Delta', \Gamma'))$ and we can apply the transformation recursively to this bunch. Because Δ is strictly smaller than Γ' the recursion is guaranteed to terminate. The same transformation can be used for additive bunches since ";" is also a monoid operation.

We are left with a tree with the following shape:



where $\Gamma \ldots \Gamma^n$ are either propositions, units or additive bunches, to which, by induction hypothesis, the same transformation can be applied

□

Note that the position of the subbunches has not changed with respect to a left traversal of the tree. This fact is important if we are considering bunches of clauses, and the order in which the clauses are considered is relevant, as is usually the case in logic programming.

Canonical bunches let us represent bunches in a convenient way. Now, instead of writing a bunch like $\Gamma_1, (\Gamma_2, (\ldots, (\Gamma_{n-1}, \Gamma_n)))$ we can write it $\Gamma_1, \Gamma_2, \ldots, \Gamma_{n-1}, \Gamma_n$ with no loss of information.

3.3 Notation

Often we need to say explicitly whether a context is multiplicative or additive. The following notation is then useful: $\overset{,}{\Gamma}$ means that Γ is multiplicative and $\overset{;}{\Gamma}$ means that it is additive.

We will use the notation $\Gamma \in \Delta$ to mean that Δ is of the form $\Delta'(\Gamma)$. An extreme case is that $\Gamma \in \Gamma$. In the following discussion "," and ";" are considered to be multary operators, connecting two or more subbunches. In this way, not only associativity issues are canonized away but also we get that $\Gamma \in \Delta$ only if Γ collects all clauses or subbunches at its level. As a consequence, for example $a \notin a;b \notin a;b;c$.

Also we will define the relation $\Gamma \subseteq \Delta$. Clauses are considered to be additive bunches. If $\overset{;}{\Gamma}$ then $\overset{;}{\Gamma} \subseteq \Delta$ iff $\overset{;}{\Gamma} \in \Delta$. If $\overset{,}{\Gamma}$ then it is of the form $\overset{;}{\Gamma}_1, \cdots, \overset{;}{\Gamma}_n$ and $\overset{,}{\Gamma} \subseteq \Delta$ iff $\overset{;}{\Gamma}_i \in \Delta \; \forall 1 \leq i \leq n$.

However, this definition is not enough to rule out some cases in which we don't want the relation to hold. For example we don't want $(a;b),(h;i)$ to be a subbunch of $((a;b),c);(d,(h;i))$ because this would mean that we were using clauses from two different multiplicative subbunches, without doing a weakening first. I think that the following extra condition works:

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, *i.e.*, in which all instances of implicational left rules are of the following, essentially unary, forms (I is the unit of $*$):

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

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):

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

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

3.4 Uniform Proofs

The first requirement for a logic that is going to be used as a basis for a programming language, is that the amount of non-determinism is kept to a minimum.

Definition 3.4.1 (Hereditare Harrop Formulæ) *The Hereditary Harrop fragment of* **BI** *is defined by the following grammar*

$$\begin{array}{llll}
\text{Definite formulæ} & D & ::= & A \mid D \wedge D \mid G \to A \mid D * D \mid G \mathrel{-\!\!*} A \\
 & & & \mid \forall x.D \mid \forall_{\mathsf{new}} x.D \\
\text{Goal formulæ} & G & ::= & \top \mid A \mid G \wedge G \mid D \to G \mid G * G \mid D \mathrel{-\!\!*} G \\
 & & & \mid G \vee G \mid \exists x.G \mid \exists_{\mathsf{new}} x.G
\end{array}$$

At this point we have shown that in the Hereditary Harrop fragment of **BI** for each proof there is a uniform proof which is equivalent to it. But now we face the following problem: a uniform proof performs first all right rules down to an atom and at this point only a left rule can be used. But it is not clear which left rule should be applied first. We will show in this section that a goal-directed approach is complete. With this we mean that if we systematically choose to apply an axiom or a left implication where the consequent is the atom we are trying to prove, then we will always find a proof if there is one.

3.5 Weakening

We need to investigate the permuting properties of Weakening with respect to the other rules, specially the right rules.

Thinking of proofs as proof search, that is as happening upwards, it can be seen from a simple observation of the rules that it is possible to postpone the application of any weakenings until an axiom forces them, and in that case they can be done sidestepping any non-determinism. But there is an exception to this permutability: $*R$. Since this right rule has as a context a multiplicative bunch, in the case in which this context is additive there are two ways forward: first to try each of the multiplicative bunches constituting the additive bunch one by one, and in case of failure also a unit operation could be performed to transform an additive bunch Γ into the multiplicative I,Γ. We will see in the operational semantics that the context management system we use removes this source of non-determinism.

3.6 CutAxiom

The usual axiom rule $\alpha \vdash \alpha$ is not suitable for proof search, since there are cases where Cut cannot be eliminated without also losing goal directedness. An example will be analysed below, but before that we propose an alternative axiom rule which we will call CutAxiom:

Lemma 3.6.1 *The following rule, which we will call* CutAxiom*, is admissible.*

$$\dfrac{\Gamma \vdash I}{\Gamma, \alpha \vdash \alpha}\,\text{CutAxiom}$$

Proof: this rule encodes the following cut in a goal directed fashion:

$$\dfrac{\Gamma \vdash I \qquad \dfrac{\dfrac{}{\alpha \vdash \alpha}\,\text{Axiom}}{I, \alpha \vdash \alpha}\,\text{Unit}}{\Gamma, \alpha \vdash \alpha}\,\text{Cut}$$

□

The way to eliminate the cut in the proof above will depend on the structure of the proof $\Gamma \vdash I$ but typically it will involve finding a clause that defines I and performing a left rule on this clause. An example is

$$\dfrac{\dfrac{}{b \vdash b}\,\text{Axiom} \qquad \dfrac{\dfrac{}{a \vdash a}\,\text{Axiom}}{I, a \vdash a}\,\text{Unit}}{b, \boxed{b \mathrel{-\!\!*} I}, a \vdash a}\,{-\!\!*}L$$

In this example it is clear that the boxed clause, used in the ${-\!\!*}L$ rule, bears no relation whatsoever with the goal. With the CutAxiom, though, it becomes nicely goal directed:

$$\dfrac{\dfrac{\dfrac{}{b \vdash b}\,\text{Axiom} \qquad \dfrac{}{I \vdash I}\,\text{Axiom}}{b, b \mathrel{-\!\!*} I \vdash I}\,{-\!\!*}L}{b, b \mathrel{-\!\!*} I, a \vdash a}\,\text{CutAxiom}$$

Using the Unit L rule $\dfrac{}{\emptyset_m \vdash I}\,\text{Unit}$ and CutAxiom we can encode the normal axiom rule in the following way:

$$\dfrac{\dfrac{}{\emptyset_m \vdash I}\,\text{Unit}}{\alpha \vdash \alpha}\,\text{CutAxiom}$$

The CutAxiom rule won't be in the leaves of the proofs. Rather our proofs will have only the Unit L rule as leafs.

3.7 Resolution and Simple proofs

First we show some general permutation properties of the calculus. In the following sections α and β stand for atomic formulæ. Also we will annotate applications of $\to L$ rule as $\to L_\alpha$ to indicate that the principal formula in the rule is the one which defines α.

Lemma 3.7.1 *Given a proof with a left implication rule as the last rule of the right subproof and a left implication rule as the last rule, then these two rules can be permuted*

Proof: We analyse all possible combinations, the additive case on the right subproof first.

$\boxed{\to L}$ There is a special case to be considered, when both arrows are of the same type and belong to the same additive subbunch, in which case it is possible for α to go to the left branch of the right subproof, instead to the right branch. This case is considered first.

$$\dfrac{\Gamma_1 \vdash \phi \quad \dfrac{\Gamma_2;\alpha \vdash \psi \quad \Delta(\Gamma_3;\beta) \vdash \xi}{\Delta(\Gamma_2;\Gamma_3;\alpha;\psi \to \beta) \vdash \xi}\to L_\beta}{\Delta(\Gamma_1;\Gamma_2;\Gamma_3;\phi \to \alpha;\psi \to \beta) \vdash \xi}\to L_\alpha \quad \rightsquigarrow \quad \dfrac{\dfrac{\Gamma_1 \vdash \phi \quad \Gamma_2;\alpha \vdash \psi}{\Gamma_1;\Gamma_2;\phi \to \alpha \vdash \psi}\to L_\alpha \quad \Delta(\Gamma_3;\beta) \vdash \xi}{\Delta(\Gamma_1;\Gamma_2;\Gamma_3;\phi \to \alpha;\psi \to \beta) \vdash \xi}\to L_\beta$$

$$\dfrac{\Gamma_1 \vdash \phi \quad \dfrac{\Gamma_3 \vdash \psi \quad \Delta(\Gamma_2;\alpha)(\Gamma_4;\beta) \vdash \xi}{\Delta(\Gamma_2;\alpha)(\Gamma_3;\Gamma_4;\psi \to \beta) \vdash \xi}\to L_\beta}{\Delta(\Gamma_1;\Gamma_2;\phi \to \alpha)(\Gamma_3;\Gamma_4;\psi \to \beta) \vdash \xi}\to L_\alpha \quad \rightsquigarrow \quad \dfrac{\Gamma_3 \vdash \psi \quad \dfrac{\Gamma_1 \vdash \phi \quad \Delta(\Gamma_2;\alpha)(\Gamma_4;\beta) \vdash \xi}{\Delta(\Gamma_1;\Gamma_2;\phi \to \alpha)(\Gamma_4;\beta) \vdash \xi}\to L_\alpha}{\Delta(\Gamma_1;\Gamma_2;\phi \to \alpha)(\Gamma_3;\Gamma_4;\psi \to \beta) \vdash \xi}\to L_\beta$$

$$\dfrac{\Gamma_1 \vdash \phi \quad \dfrac{\Gamma_3 \vdash \psi \quad \Delta(\Gamma_2,\alpha)(\Gamma_4;\beta) \vdash \xi}{\Delta(\Gamma_2,\alpha)(\Gamma_3;\Gamma_4;\psi \to \beta) \vdash \xi}\to L_\beta}{\Delta(\Gamma_1,\Gamma_2,\phi -\!\!* \alpha)(\Gamma_3;\Gamma_4;\psi \to \beta) \vdash \xi} -\!\!* L_\alpha \quad \rightsquigarrow \quad \dfrac{\Gamma_3 \vdash \psi \quad \dfrac{\Gamma_1 \vdash \phi \quad \Delta(\Gamma_2,\alpha)(\Gamma_4;\beta) \vdash \xi}{\Delta(\Gamma_1,\Gamma_2,\phi -\!\!* \alpha)(\Gamma_4;\beta) \vdash \xi} -\!\!* L_\alpha}{\Delta(\Gamma_1,\Gamma_2,\phi -\!\!* \alpha)(\Gamma_3;\Gamma_4;\psi \to \beta) \vdash \xi}\to L_\beta$$

$\boxed{-\!\!* L}$ As in the previous case, when both arrows are multiplicative, there exist the possibility for α to go to the left branch of the right subproof and this case has to be treated separately.

$$\dfrac{\Gamma_1 \vdash \phi \quad \dfrac{\Gamma_2,\alpha \vdash \psi \quad \Delta(\Gamma_3,\beta) \vdash \xi}{\Delta(\Gamma_2,\Gamma_3,\alpha,\psi -\!\!* \beta) \vdash \xi} -\!\!* L_\beta}{\Delta(\Gamma_1,\Gamma_2,\Gamma_3,\phi -\!\!* \alpha,\psi -\!\!* \beta) \vdash \xi} -\!\!* L_\alpha \quad \rightsquigarrow \quad \dfrac{\dfrac{\Gamma_1 \vdash \phi \quad \Gamma_2,\alpha \vdash \psi}{\Gamma_1,\Gamma_2,\phi -\!\!* \alpha \vdash \psi} -\!\!* L_\alpha \quad \Delta(\Gamma_3,\beta) \vdash \xi}{\Delta(\Gamma_1,\Gamma_2,\Gamma_3,\phi -\!\!* \alpha,\psi -\!\!* \beta) \vdash \xi} -\!\!* L_\beta$$

$$\dfrac{\Gamma_1 \vdash \phi \qquad \dfrac{\Gamma_3 \vdash \psi \quad \Delta(\Gamma_2;\alpha)(\Gamma_4,\beta) \vdash \xi}{\Delta(\Gamma_2;\alpha)(\Gamma_3,\Gamma_4,\psi \mathrel{-\!\!*} \beta) \vdash \xi}\, {-\!\!*}L_\beta}{\Delta(\Gamma_1;\Gamma_2;\phi \to \alpha)(\Gamma_3,\Gamma_4,\psi \mathrel{-\!\!*} \beta) \vdash \xi}\, \to L_\alpha \quad \rightsquigarrow \quad \dfrac{\Gamma_3 \vdash \psi \qquad \dfrac{\Gamma_1 \vdash \phi \quad \Delta(\Gamma_2;\alpha)(\Gamma_4,\beta) \vdash \xi}{\Delta(\Gamma_1;\Gamma_2;\phi \to \alpha)(\Gamma_4,\beta) \vdash \xi}\, \to L_\alpha}{\Delta(\Gamma_1;\Gamma_2;\phi \to \alpha)(\Gamma_3,\Gamma_4,\psi \mathrel{-\!\!*} \beta) \vdash \xi}\, {-\!\!*}L_\beta$$

$$\dfrac{\Gamma_1 \vdash \phi \qquad \dfrac{\Gamma_3 \vdash \psi \quad \Delta(\Gamma_2,\alpha)(\Gamma_4,\beta) \vdash \xi}{\Delta(\Gamma_2,\alpha)(\Gamma_3,\Gamma_4,\psi \mathrel{-\!\!*} \beta) \vdash \xi}\, {-\!\!*}L_\beta}{\Delta(\Gamma_1,\Gamma_2,\phi \mathrel{-\!\!*} \alpha)(\Gamma_3,\Gamma_4,\psi \mathrel{-\!\!*} \beta) \vdash \xi}\, {-\!\!*}L_\alpha \quad \rightsquigarrow \quad \dfrac{\Gamma_3 \vdash \psi \qquad \dfrac{\Gamma_1 \vdash \phi \quad \Delta(\Gamma_2,\alpha)(\Gamma_4,\beta) \vdash \xi}{\Delta(\Gamma_1,\Gamma_2,\phi \mathrel{-\!\!*} \alpha)(\Gamma_4,\beta) \vdash \xi}\, {-\!\!*}L_\alpha}{\Delta(\Gamma_1,\Gamma_2,\phi \mathrel{-\!\!*} \alpha)(\Gamma_3,\Gamma_4,\psi \mathrel{-\!\!*} \beta) \vdash \xi}\, {-\!\!*}L_\beta$$

□

Definition 3.7.2 *Given a left implication rule, we will call it* unit-simple *if its right subproof consists of a series of weakenings followed immediately by a CutAxiom on the atom defined by the implication.*

Lemma 3.7.3 ((Unit-simple proofs)) *All occurrences of non-unit-simple rules can be removed by either eliminating the rule altogether or by permuting it upwards until it becomes a unit-simple rule.*

Proof Overview: We have to consider only instances of $\to L$ and ${-\!\!*}L$ where the atom defined by the principal clause is different than the atom that we are trying to prove, or it is the same but the axiom applied in the right subproof does not refer to the atom defined by the left implication rule.

The proof is based on the fact that any occurrence of a left implication rule which is non-unit-simple can be eliminated or permuted upwards until it becomes a unit-simple rule. There are two fundamental proof figures that we are concerned with:

$$\dfrac{\begin{matrix}\Xi_1\\ \vdots\\ \Gamma_1 \vdash \phi\end{matrix} \qquad \begin{matrix}\Xi_2\\ \vdots\\ \Delta(\Gamma_2;\alpha) \vdash \beta\end{matrix}}{\Delta(\Gamma_1;\Gamma_2;\phi \to \alpha) \vdash \beta}\, \to L \qquad \dfrac{\begin{matrix}\Xi_1\\ \vdots\\ \Gamma_1 \vdash \phi\end{matrix} \qquad \begin{matrix}\Xi_2\\ \vdots\\ \Delta(\Gamma_2,\alpha) \vdash \beta\end{matrix}}{\Delta(\Gamma_1,\Gamma_2,\phi \mathrel{-\!\!*} \alpha) \vdash \beta}\, {-\!\!*}L$$

In each case we assume that in Ξ_1 and Ξ_2 every occurrence of a left implication rule is unit-simple. The only case when the rule cannot be permuted upwards is when α is equal to β and the last rule applied in Ξ_2 (possibly after some weakenings) is a CutAxiom. But in this case the left implication rule is unit-simple.

Proof:

$\boxed{\text{CutAxiom}}$ When the last rule in the right branch is a CutAxiom there are three cases. If $\alpha = \beta$ and this instance of α is used in the CutAxiom, then it is a simple proof and there is nothing to do.

$$\dfrac{\Gamma_1 \vdash \phi \qquad \dfrac{\dfrac{\Delta' \vdash I}{\Delta', \alpha \vdash \alpha}\,\text{CutAxiom}}{\Delta(\Gamma_2;\alpha) \vdash \alpha}\,\text{Weakening}^*}{\Delta(\Gamma_1;\Gamma_2;\phi \to \alpha) \vdash \alpha}\,{\to}\, L \qquad \dfrac{\Gamma_1 \vdash \phi \qquad \dfrac{\dfrac{\Delta', \Gamma_2 \vdash I}{\Delta', \Gamma_2, \alpha \vdash \alpha}\,\text{CutAxiom}}{\Delta(\Gamma_2,\alpha) \vdash \alpha}\,\text{Weakening}^*}{\Delta(\Gamma_1,\Gamma_2,\phi {-\!\!*} \alpha) \vdash \alpha}\,{-\!\!*}L$$

In the next cases we don't need to assume that $\alpha \neq \beta$. The important fact is that the CutAxiom doesn't eliminate the atom that had just been created by the left implication rule.

If α is weakened away while bringing β to the top level then $\Gamma_1;\phi \to \alpha$ (or $\Gamma_1,\phi {-\!\!*} \alpha$) can be weakened before the application of the implication rule. Here $\Delta(\Gamma_2;\alpha)(\beta)$ means that Δ has two subbunches $\Gamma_2;\alpha$ and β and it includes the case $\Delta(\Gamma_2;\alpha;\beta)$.

$$\dfrac{\Gamma_1 \vdash \phi \quad \dfrac{\dfrac{\Delta' \vdash I}{\Delta', \beta \vdash \beta}\,\text{CutAxiom}}{\Delta(\Gamma_2;\alpha)(\beta) \vdash \beta}\,\text{Weakening}^*}{\Delta(\Gamma_1;\Gamma_2;\phi \to \alpha)(\beta) \vdash \beta}\,{\to}\, L \quad \rightsquigarrow \quad \dfrac{\dfrac{\Delta' \vdash I}{\Delta', \beta \vdash \beta}\,\text{CutAxiom}}{\Delta(\Gamma_1;\Gamma_2;\phi \to \alpha)(\beta) \vdash \beta}\,\text{Weakening}^*$$

In the case of $\Delta(\Gamma_2,\alpha)(\beta)$ the fact that Γ_2,α has been weakened away implies that it is a different subbunch than β, since "," does not admit weakening.

$$\dfrac{\Gamma_1 \vdash \phi \quad \dfrac{\dfrac{\Delta' \vdash I}{\Delta', \beta \vdash \beta}\,\text{CutAxiom}}{\Delta(\Gamma_2,\alpha)(\beta) \vdash \beta}\,\text{Weakening}^*}{\Delta(\Gamma_1,\Gamma_2,\phi {-\!\!*} \alpha)(\beta) \vdash \beta}\,{-\!\!*}L \quad \rightsquigarrow \quad \dfrac{\dfrac{\Delta' \vdash I}{\Delta', \beta \vdash \beta}\,\text{CutAxiom}}{\Delta(\Gamma_1,\Gamma_2,\phi {-\!\!*} \alpha)(\beta) \vdash \beta}\,\text{Weakening}^*$$

The simplified figures on the right are unit-simple by the Induction Hypothesis.

The third case is when $\Gamma_2;\alpha$ or Γ_2,α is not weakened away. In this case the implication rule can be permuted upwards.

$$\dfrac{\Gamma_1 \vdash \phi \quad \dfrac{\dfrac{\Delta'(\Gamma_2;\alpha) \vdash I}{\Delta'(\Gamma_2;\alpha),\beta \vdash \beta}\,\text{CutAxiom}}{\Delta(\Gamma_2;\alpha)(\beta) \vdash \beta}\,\text{Weakening}^*}{\Delta(\Gamma_1;\Gamma_2;\phi \to \alpha)(\beta) \vdash \beta}\to L \quad \rightsquigarrow \quad \dfrac{\dfrac{\dfrac{\Gamma_1 \vdash \phi \quad \Delta'(\Gamma_2;\alpha) \vdash I}{\Delta'(\Gamma_1;\Gamma_2;\phi \to \alpha) \vdash I}\to L}{\Delta'(\Gamma_1;\Gamma_2;\phi \to \alpha),\beta \vdash \beta}\,\text{CutAxiom}}{\Delta(\Gamma_1;\Gamma_2;\phi \to \alpha)(\beta) \vdash \beta}\,\text{Weakening}^*$$

$$\dfrac{\Gamma_1 \vdash \phi \quad \dfrac{\dfrac{\Delta'(\Gamma_2,\alpha) \vdash I}{\Delta'(\Gamma_2,\alpha),\beta \vdash \beta}\,\text{CutAxiom}}{\Delta(\Gamma_2,\alpha)(\beta) \vdash \beta}\,\text{Weakening}^*}{\Delta(\Gamma_1,\Gamma_2,\phi {-\!\!*} \alpha)(\beta) \vdash \beta}{-\!\!*}L \quad \rightsquigarrow \quad \dfrac{\dfrac{\dfrac{\Gamma_1 \vdash \phi \quad \Delta'(\Gamma_2,\alpha) \vdash I}{\Delta'(\Gamma_1,\Gamma_2,\phi {-\!\!*} \alpha) \vdash I}{-\!\!*}L}{\Delta'(\Gamma_1,\Gamma_2,\phi {-\!\!*} \alpha),\beta \vdash \beta}\,\text{CutAxiom}}{\Delta(\Gamma_1,\Gamma_2,\phi {-\!\!*} \alpha)(\beta) \vdash \beta}\,\text{Weakening}^*$$

$\boxed{{-\!\!*}L \textbf{ and} \to L}$ In these cases we use lemma 3.6.1 to permute the occurrence of the non-unit-simple rule upwards. However a special case arise when both implications are of the same type and at the same level. We will use ${-\!\!*}L$ to illustrate the problem and the solution.

$$\dfrac{\Gamma_1 \vdash \phi \quad \dfrac{\Gamma_2,\alpha \vdash \psi \quad \Delta(\Gamma_3,\beta) \vdash \beta}{\Delta(\Gamma_2,\Gamma_3,\alpha,\psi {-\!\!*} \beta) \vdash \beta}{-\!\!*}L}{\Delta(\Gamma_1,\Gamma_2,\Gamma_3,\phi {-\!\!*} \alpha,\psi {-\!\!*} \beta) \vdash \beta}{-\!\!*}L \quad \rightsquigarrow \quad \dfrac{\dfrac{\Gamma_1 \vdash \phi \quad \Gamma_2,\alpha \vdash \psi}{\Gamma_1,\Gamma_2,\phi {-\!\!*} \alpha \vdash \psi}{-\!\!*}L \quad \Delta(\Gamma_3,\beta) \vdash \beta}{\Delta(\Gamma_1,\Gamma_2,\Gamma_3,\phi {-\!\!*} \alpha,\psi {-\!\!*} \beta) \vdash \beta}{-\!\!*}L$$

The problem is that the permuted proof of $\Gamma_1,\Gamma_2,\phi {-\!\!*} \alpha \vdash \psi$ might no longer be uniform. However, if that was the case, we can apply the permutations given in the proof of uniformity and then apply the induction hypothesis.

□

To illustrate what happens in a specific case consider the proof of the sequent $d,c,d {-\!\!*} a,(a * c) {-\!\!*} b \vdash b$. For simplicity we will use the normal Axiom rule instead of CutAxiom. First consider the uniform but non-unit-simple proof

$$\dfrac{\dfrac{}{d \vdash d} \quad \dfrac{\dfrac{\dfrac{}{c \vdash c} \quad \dfrac{}{a \vdash a}}{c,a \vdash a * c}*R \quad \dfrac{}{b \vdash b}}{c,a,d {-\!\!*} a,(a * c) {-\!\!*} b \vdash b}{-\!\!*}L}{d,c,d {-\!\!*} a,(a * c) {-\!\!*} b \vdash b}{-\!\!*}L$$

After the transformation indicated above, this proof becomes

$$\dfrac{\dfrac{\dfrac{}{d\vdash d}\qquad \dfrac{\dfrac{}{c\vdash c}\quad \dfrac{}{a\vdash a}}{c,a\vdash a*c}\,*R}{d,c,d{-\!\!*}a\vdash a*c}\,{-\!\!*}L \qquad \dfrac{}{b\vdash b}}{d,c,d{-\!\!*}a,(a*c){-\!\!*}b\vdash b}\,{-\!\!*}L$$

which is not uniform, because there is a $-\!\!*L$ rule below a $*R$ rule. But this subproof can be transformed into the following uniform proof

$$\dfrac{\dfrac{\dfrac{\dfrac{}{d\vdash d}\quad \dfrac{}{a\vdash a}}{d,d{-\!\!*}a\vdash a}\,{-\!\!*}L \qquad \dfrac{}{c\vdash c}}{d,c,d{-\!\!*}a\vdash a*c}\,*R \qquad \dfrac{}{b\vdash b}}{d,c,d{-\!\!*}a,(a*c){-\!\!*}b\vdash b}\,{-\!\!*}L$$

Lemma 3.7.4 *The system resulting from replacing $-\!\!*L$ with the following rule, which we will call $-\!\!*L'$, is equivalent*

$$\dfrac{\Gamma_1\vdash\phi\quad \Gamma_2,\alpha\vdash\alpha}{\Gamma_1,\Gamma_2,\phi{-\!\!*}\alpha\vdash\alpha}\,{-\!\!*}L'$$

Proof: The crucial point is to note that all weakenings done to bring an atom to the top level (so the CutAxiom rule can be applied) could be done earlier to bring the implication clause to the top level and then applying the rule. At that point the surrounding context disappears.

First, using the last lemma we need to consider only simple proofs. Then the following transformation can be done.

$$\dfrac{\Gamma_1\vdash\phi\quad \dfrac{\dfrac{\Delta',\Gamma_2\vdash I}{\Delta',\Gamma_2,\alpha\vdash\alpha}\,\text{CutAxiom}}{\Delta(\Gamma_2,\alpha)\vdash\alpha}\,\text{Weakening}^*}{\Delta(\Gamma_1,\Gamma_2,\phi{-\!\!*}\alpha)\vdash\alpha}\,{-\!\!*}L \quad\rightsquigarrow\quad \dfrac{\dfrac{\Gamma_1\vdash\phi\quad \dfrac{\Delta',\Gamma_2\vdash I}{\Delta',\Gamma_2,\alpha\vdash\alpha}\,\text{CutAxiom}}{\Delta',\Gamma_1,\Gamma_2,\phi{-\!\!*}\alpha\vdash\alpha}\,{-\!\!*}L'}{\Delta(\Gamma_1,\Gamma_2,\phi{-\!\!*}\alpha)\vdash\alpha}\,\text{Weakening}^*$$

□

Notice that in the transformed rule we have decoupled the weakenings from the CutAxiom. This is harmless since the weakenings had to be done anyway, getting rid of the same clauses than before. Also the "goal directness" of the weakenings is not lost, because where before we were using α now we use $\phi{-\!\!*}\alpha$ with the same results.

3.8 Simple Proofs

These proofs have nice computational properties. In the case of $\mathrel{-\!\!*}L$ this means having a single thread of control and no context-management non-determinism. Again we will use the new axiom rule and move the side condition over to the other side by means of a cut.

Lemma 3.8.1 *The following* $\mathrel{-\!\!*}L$ *rule is admissible*

$$\frac{\Gamma \vdash \phi}{\Gamma, \phi \mathrel{-\!\!*} \alpha \vdash \alpha}$$

Proof: We transform the rule presented in the last lemma in the following way

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

and then the axiom becomes redundant.

□

The comment above respect to cut elimination applies here as well: that is, cut can always be eliminated but the way this is done will depend on the proof of $\Gamma_2 \vdash I$.

In the case of $\to L$ things are not so "simple". However it is possible to avoid a source of non-determinism by a trick shown in the following lemma. Note that in this case brackets are used only to force precedence of ";" over ",", and must not be confused with the sub-bunch notation.

Lemma 3.8.2 *The following rule is admissible and will be called* $\to L'$*. Moreover the replacement of* $\to L$ *with this rule doesn't affect the power of the system.*

$$\frac{\Gamma \vdash \phi \qquad \Delta \vdash I}{\Delta, (\Gamma; \phi \to \alpha) \vdash \alpha}\to L'$$

Proof: The proof is based in the fact that among the weakenings necessary to bring α to the top on the right subproof, there will be weakenings of any bit of context connected additively to it. These weakenings can be done on the left subproof.

$$
\begin{prooftree}
\AxiomC{$\Gamma_1 \vdash \phi$}
\AxiomC{$\Delta' \vdash I$}
\RightLabel{CutAxiom}
\UnaryInfC{$\Delta', \alpha \vdash \alpha$}
\doubleLine
\RightLabel{Weakening$^*_2$}
\UnaryInfC{$\Delta(\alpha) \vdash \alpha$}
\doubleLine
\RightLabel{Weakening$^*_1$}
\UnaryInfC{$\Delta(\Gamma_2;\alpha) \vdash \alpha$}
\RightLabel{$\to L$}
\BinaryInfC{$\Delta(\Gamma_1;\Gamma_2;\phi \to \alpha) \vdash \alpha$}
\end{prooftree}
\quad \rightsquigarrow \quad
\begin{prooftree}
\AxiomC{$\Gamma_1 \vdash \phi$}
\doubleLine
\RightLabel{Weakening$^*_1$}
\UnaryInfC{$\Gamma_1;\Gamma_2 \vdash \phi$}
\AxiomC{$\Delta' \vdash I$}
\RightLabel{CutAxiom}
\UnaryInfC{$\Delta', \alpha \vdash \alpha$}
\RightLabel{$\to L'$}
\BinaryInfC{$\Delta', (\Gamma_1;\Gamma_2;\phi \to \alpha) \vdash \alpha$}
\doubleLine
\RightLabel{Weakening$^*_2$}
\UnaryInfC{$\Delta(\Gamma_1;\Gamma_2;\phi \to \alpha) \vdash \alpha$}
\end{prooftree}
$$

Here the series of weakenings called Weakening$^*_1$ weakens the components of the subbunch Γ_2. The series of weakenings called Weakening$^*_2$ weakens the components of Δ necessary to bring α or the subbunch $\Gamma_1;\Gamma_2;\phi \to \alpha$ to the top level.

□

Finally we state and prove the most important lemma in this section:

Theorem 3.8.3 *Simple (resolution) proofs are complete for the hereditary Harrop fragment of* **BI**.

Proof: Given a sequent $\Gamma \vdash \phi$ where all components of Γ are definite formulæ and ϕ is a goal formula, we know from Section 3.10 that the goal is probable if and only if there is a normal proof.

Also from Lemma 3.7.3 we know that any proof of an atomic formula can be transformed into a resolution proof.

□

3.9 Resolution Proofs

We now present the system that will be used on the rest of this work. We will call it **LBI**$_R$ and the rules are shown in Table 3.1.

We assume that Γ and Δ have not got any additive or multiplicative conjunctions to insure uniformity of the proofs. This invariant has to be maintained when the context is increased via a right implication rule. Therefore we apply to the antecedent of the implications the following operation:

$$\frac{\Gamma \vdash_R I}{\Gamma, \alpha \vdash_R \alpha}\ \text{CutAxiom} \qquad \frac{\Gamma \vdash_R G}{\Gamma, G \mathrel{-\!\!*} \alpha \vdash_R \alpha}\ \mathrel{-\!\!*}\!Res \qquad \frac{\Gamma' \vdash_R G \quad \Gamma \vdash_R I}{\Gamma, (\Gamma'; G \to \alpha) \vdash_R \alpha} \to Res$$

$$\frac{\Gamma \vdash_R \phi \quad \Delta \vdash_R \psi}{\Gamma, \Delta \vdash_R \phi * \psi}\ {*R} \qquad \frac{\Gamma, [\phi] \vdash_R \psi}{\Gamma \vdash_R \phi \mathrel{-\!\!*} \psi}\ \mathrel{-\!\!*}\!R$$

$$\frac{\Gamma \vdash_R \phi \quad \Gamma \vdash_R \psi}{\Gamma \vdash_R \phi \wedge \psi}\ \wedge R \qquad \frac{\Gamma \vdash_R \phi_i}{\Gamma \vdash_R \phi_1 \vee \phi_2}\ (i = 1, 2)\ \vee R \qquad \frac{\Gamma; [\phi] \vdash_R \psi}{\Gamma \vdash_R \phi \to \psi} \to R$$

Table 3.1: Resolution proofs

Definition 3.9.1 (the clausal decomposition, [P])

$$\begin{array}{rcl} [\Gamma(\alpha)] & ::= & \Gamma(\alpha) \\ [\Gamma(\phi * \psi)] & ::= & \Gamma([\phi], [\psi]) \\ [\Gamma(\phi \wedge \psi)] & ::= & \Gamma([\phi]; [\psi]) \\ [\Gamma(\phi \mathrel{-\!\!*} \alpha)] & ::= & \Gamma(\phi \mathrel{-\!\!*} \alpha) \\ [\Gamma(\phi \to \alpha)] & ::= & \Gamma(\phi \to \alpha) \end{array}$$

3.10 Permutation of Rules in BI

As a general organizing principle in the following permutations, we have first treated the additive cases and then the multiplicative ones. There are four main subsections corresponding to the four left rules that have to be considered: $\to L$, $\mathrel{-\!\!*}L$, $\forall L$ and $\forall_{\mathsf{new}} L$. The subcases correspond to the right rules and are treated in the following order: implication, conjunction, disjunction (in the additive case only), universal and existential quantifiers. In the case of the right rules the order of treatment is again first the additives and then the multiplicatives.

3.10.1 Additive implication ($\to L$)

All applications of $\to L$ have the side condition that $Y \vdash \psi : \mathrm{Prop}$.

$\boxed{\to R}$

$$\dfrac{\{X\}\,\Gamma\vdash\phi \qquad \dfrac{\{Y'(Y)\}\,\Delta(\psi);\chi\vdash\xi}{\{Y'(Y)\}\,\Delta(\psi)\vdash\chi\to\xi}\to R}{\{Y'(X;Y)\}\,\Delta(\Gamma;\phi\to\psi)\vdash\chi\to\xi}\to L \qquad \rightsquigarrow$$

$$\dfrac{\dfrac{\{X\}\,\Gamma\vdash\phi \quad \{Y'(Y)\}\,\Delta(\psi);\chi\vdash\xi}{\{Y'(X;Y)\}\,\Delta(\Gamma;\phi\to\psi);\chi\vdash\xi}\to L}{\{Y'(X;Y)\}\,\Delta(\Gamma;\phi\to\psi)\vdash\chi\to\xi}\to R$$

$\boxed{\wedge R}$

$$\dfrac{\{X\}\,\Gamma\vdash\phi \qquad \dfrac{\{Y'(Y)\}\,\Delta(\psi)\vdash\chi \quad \{Y''\}\,\Delta'\vdash\xi}{\{Y'(Y);Y''\}\,\Delta(\psi);\Delta'\vdash\chi\wedge\xi}\wedge R}{\{Y'(X;Y);Y''\}\,\Delta(\Gamma;\phi\to\psi);\Delta'\vdash\chi\wedge\xi}\to L \qquad \rightsquigarrow$$

$$\dfrac{\dfrac{\{X\}\,\Gamma\vdash\phi \quad \{Y'(Y)\}\,\Delta(\psi)\vdash\chi}{\{Y'(X;Y)\}\,\Delta(\Gamma;\phi\to\psi)\vdash\chi}\to L \qquad \{Y''\}\,\Delta'\vdash\xi}{\{Y'(X;Y);Y''\}\,\Delta(\Gamma;\phi\to\psi);\Delta'\vdash\chi\wedge\xi}\wedge R$$

$\boxed{\vee R}$ For $i = 1,2$

$$\dfrac{\{X\}\,\Gamma\vdash\phi \qquad \dfrac{\{Y'(Y)\}\,\Delta(\psi)\vdash\chi_i}{\{Y'(Y)\}\,\Delta(\psi)\vdash\chi_1\vee\chi_2}\vee R}{\{Y'(X;Y)\}\,\Delta(\Gamma;\phi\to\psi)\vdash\chi_1\vee\chi_2}\to L \qquad \rightsquigarrow$$

$$\dfrac{\dfrac{\{X\}\,\Gamma\vdash\phi \quad \{Y'(Y)\}\,\Delta(\psi)\vdash\chi_i}{\{Y'(X;Y)\}\,\Delta(\Gamma;\phi\to\psi)\vdash\chi_i}\to L}{\{Y'(X;Y)\}\,\Delta(\Gamma;\phi\to\psi)\vdash\chi_1\vee\chi_2}\vee R$$

$\boxed{\forall R}$ Here a potential problem might arise if x is not free in Γ. Then a substitution should be done first.

$$\dfrac{\{X\}\,\Gamma\vdash\phi \qquad \dfrac{\{Y'(Y);v_1\}\,\Delta(\psi)\vdash\chi[v_1/x]}{\{Y'(Y)\}\,\Delta(\psi)\vdash\forall x.\chi(x)}(v\text{ new tag})\ \forall R}{\{Y'(X;Y)\}\,\Delta(\Gamma;\phi\to\psi)\vdash\forall x.\chi(x)}\to L \qquad \rightsquigarrow$$

$$\dfrac{\dfrac{\{X\}\,\Gamma\vdash\phi \quad \{Y'(Y);v_1\}\,\Delta(\psi)\vdash\chi[v_1/x]}{\{Y'(X;Y);v_1\}\,\Delta(\Gamma;\phi\to\psi)\vdash\chi[v_1/x]}\to L}{\{Y'(X;Y)\}\,\Delta(\Gamma;\phi\to\psi)\vdash\forall x.\chi(x)}(v\text{ new tag})\ \forall R$$

$\boxed{\exists R}$

$$\dfrac{\{X\}\,\Gamma \vdash \phi \qquad \dfrac{\{Y'(Y);Z\}\,\Delta(\psi) \vdash \chi[t/x]}{\{Y'(Y)\}\,\Delta(\psi) \vdash \exists x.\chi(x)}\,(Z \vdash t : \mathrm{Term})\ \exists R}{\{Y'(X;Y)\}\,\Delta(\Gamma;\phi \to \psi) \vdash \exists x.\chi(x)} \to L \qquad \rightsquigarrow$$

$$\dfrac{\dfrac{\{X\}\,\Gamma \vdash \phi \quad \{Y'(Y);Z\}\,\Delta(\psi) \vdash \chi[t/x]}{\{Y'(X;Y);Z\}\,\Delta(\Gamma;\phi \to \psi) \vdash \chi[t/x]} \to L}{\{Y'(X;Y)\}\,\Delta(\Gamma;\phi \to \psi) \vdash \exists x.\chi(x)}\,(Z \vdash t : \mathrm{Term})\ \exists R$$

$\boxed{\mathrel{-\!\!*} R}$

$$\dfrac{\{X\}\,\Gamma \vdash \phi \qquad \dfrac{\{Y'(Y)\}\,\Delta(\psi), \chi \vdash \xi}{\{Y'(Y)\}\,\Delta(\psi) \vdash \chi \mathrel{-\!\!*} \xi} \mathrel{-\!\!*} R}{\{Y'(X;Y)\}\,\Delta(\Gamma;\phi \to \psi) \vdash \chi \mathrel{-\!\!*} \xi} \to L \qquad \rightsquigarrow$$

$$\dfrac{\dfrac{\{X\}\,\Gamma \vdash \phi \quad \{Y'(Y)\}\,\Delta(\psi), \chi \vdash \xi}{\{Y'(X;Y)\}\,\Delta(\Gamma;\phi \to \psi), \chi \vdash \xi} \to L}{\{Y'(X;Y)\}\,\Delta(\Gamma;\phi \to \psi) \vdash \chi \mathrel{-\!\!*} \xi} \mathrel{-\!\!*} R$$

$\boxed{* R}$

$$\dfrac{\{X\}\,\Gamma \vdash \phi \qquad \dfrac{\{Y'(Y)\}\,\Delta(\psi) \vdash \chi \quad \{Y''\}\,\Delta' \vdash \xi}{\{Y'(Y), Y''\}\,\Delta(\psi), \Delta' \vdash \chi * \xi} * R}{\{Y'(X;Y), Y''\}\,\Delta(\Gamma;\phi \to \psi), \Delta' \vdash \chi * \xi} \to L \qquad \rightsquigarrow$$

$$\dfrac{\dfrac{\{X\}\,\Gamma \vdash \phi \quad \{Y'(Y)\}\,\Delta(\psi) \vdash \chi}{\{Y'(X;Y)\}\,\Delta(\Gamma;\phi \to \psi) \vdash \chi} \to L \qquad \{Y''\}\,\Delta' \vdash \xi}{\{Y'(X;Y), Y''\}\,\Delta(\Gamma;\phi \to \psi), \Delta' \vdash \chi * \xi} * R$$

$\boxed{\forall_{\mathsf{new}} R}$

$$\dfrac{\{X\}\,\Gamma \vdash \phi \qquad \dfrac{\{Y'(Y), v_1\}\,\Delta(\psi) \vdash \chi[v_1/x]}{\{Y'(Y)\}\,\Delta(\psi) \vdash \forall_{\mathsf{new}} x.\chi(x)}\,(v \text{ new tag})\ \forall_{\mathsf{new}} R}{\{Y'(X;Y)\}\,\Delta(\Gamma;\phi \to \psi) \vdash \forall_{\mathsf{new}} x.\chi(x)} \to L \qquad \rightsquigarrow$$

$$\dfrac{\dfrac{\{X\}\,\Gamma \vdash \phi \quad \{Y'(Y), v_1\}\,\Delta(\psi) \vdash \chi[v_1/x]}{\{Y'(X;Y), v_1\}\,\Delta(\Gamma;\phi \to \psi) \vdash \chi[v_1/x]} \to L}{\{Y'(X;Y)\}\,\Delta(\Gamma;\phi \to \psi) \vdash \forall_{\mathsf{new}} x.\chi(x)}\,(v \text{ new tag})\ \forall_{\mathsf{new}} R$$

$\boxed{\exists_{\mathsf{new}}R}$

$$\dfrac{\{X\}\,\Gamma\vdash\phi \qquad \dfrac{\{Y'(Y),Z\}\,\Delta(\psi)\vdash\chi[t/x]}{\{Y'(Y)\}\,\Delta(\psi)\vdash\exists_{\mathsf{new}}x.\chi(x)}\,(Z\vdash t:\mathrm{Term})\ \exists_{\mathsf{new}}R}{\{Y'(X;Y)\}\,\Delta(\Gamma;\phi\to\psi)\vdash\exists_{\mathsf{new}}x.\chi(x)}\,\to L \qquad \rightsquigarrow$$

$$\dfrac{\dfrac{\{X\}\,\Gamma\vdash\phi \quad \{Y'(Y),Z\}\,\Delta(\psi)\vdash\chi[t/x]}{\{Y'(X;Y),Z\}\,\Delta(\Gamma;\phi\to\psi)\vdash\chi[t/x]}\,\to L}{\{Y'(X;Y)\}\,\Delta(\Gamma;\phi\to\psi)\vdash\exists_{\mathsf{new}}x.\chi(x)}\,(Z\vdash t:\mathrm{Term})\ \exists_{\mathsf{new}}R$$

3.10.2 Multiplicative implication ($-\!\!*L$)

All applications of $-\!\!*L$ have the side condition that $Y\vdash\psi$: Prop.

$\boxed{\to R}$

$$\dfrac{\{X\}\,\Gamma\vdash\phi \qquad \dfrac{\{Y'(Y)\}\,\Delta(\psi);\chi\vdash\xi}{\{Y'(Y)\}\,\Delta(\psi)\vdash\chi\to\xi}\,\to R}{\{Y'(X,Y)\}\,\Delta(\Gamma,\phi-\!\!*\psi)\vdash\chi\to\xi}\,-\!\!*L \qquad \rightsquigarrow$$

$$\dfrac{\dfrac{\{X\}\,\Gamma\vdash\phi \quad \{Y'(Y)\}\,\Delta(\psi);\chi\vdash\xi}{\{Y'(X,Y)\}\,\Delta(\Gamma,\phi-\!\!*\psi);\chi\vdash\xi}\,-\!\!*L}{\{Y'(X,Y)\}\,\Delta(\Gamma,\phi-\!\!*\psi)\vdash\chi\to\xi}\,\to R$$

$\boxed{\wedge R}$

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

$$\dfrac{\dfrac{\{X\}\,\Gamma\vdash\phi \quad \{Y'(Y)\}\,\Delta(\psi)\vdash\chi}{\{Y'(X,Y)\}\,\Delta(\Gamma,\phi-\!\!*\psi)\vdash\chi}\,-\!\!*L \qquad \{Y''\}\,\Delta'\vdash\xi}{\{Y'(X,Y);Y''\}\,\Delta(\Gamma,\phi-\!\!*\psi);\Delta'\vdash\chi\wedge\xi}\,\wedge R$$

$\boxed{\vee R}$ For $i=1,2$

$$\dfrac{\{X\}\,\Gamma\vdash\phi \qquad \dfrac{\{Y'(Y)\}\,\Delta(\psi)\vdash\chi_i}{\{Y'(Y)\}\,\Delta(\psi)\vdash\chi_1\vee\chi_2}\,\vee R}{\{Y'(X,Y)\}\,\Delta(\Gamma,\phi-\!\!*\psi)\vdash\chi_1\vee\chi_2}\,-\!\!*L \qquad \rightsquigarrow$$

$$\dfrac{\dfrac{\{X\}\,\Gamma\vdash\phi \quad \{Y'(Y)\}\,\Delta(\psi)\vdash\chi_i}{\{Y'(X,Y)\}\,\Delta(\Gamma,\phi-\!\!*\psi)\vdash\chi_i}\,-\!\!*L}{\{Y'(X;Y)\}\,\Delta(\Gamma;\phi-\!\!*\psi)\vdash\chi_1\vee\chi_2}\,\vee R$$

$\boxed{\forall R}$

$$\dfrac{\{X\}\,\Gamma\vdash\phi \qquad \dfrac{\{Y'(Y);v_1\}\,\Delta(\psi)\vdash\chi[v_1/x]}{\{Y'(Y)\}\,\Delta(\psi)\vdash\forall x.\chi(x)}\,(v\text{ new tag})\ \forall R}{\{Y'(X,Y)\}\,\Delta(\Gamma,\phi\mathrel{-\!\!*}\psi)\vdash\forall x.\chi(x)}\,{-\!\!*}L \qquad \rightsquigarrow$$

$$\dfrac{\dfrac{\{X\}\,\Gamma\vdash\phi \quad \{Y'(Y);v_1\}\,\Delta(\psi)\vdash\chi[v_1/x]}{\{Y'(X,Y);v_1\}\,\Delta(\Gamma,\phi\mathrel{-\!\!*}\psi)\vdash\chi[v_1/x]}\,{-\!\!*}L}{\{Y'(X,Y)\}\,\Delta(\Gamma,\phi\mathrel{-\!\!*}\psi)\vdash\forall x.\chi(x)}\,(v\text{ new tag})\ \forall R$$

$\boxed{\exists R}$

$$\dfrac{\{X\}\,\Gamma\vdash\phi \qquad \dfrac{\{Y'(Y);Z\}\,\Delta(\psi)\vdash\chi[t/x]}{\{Y'(Y)\}\,\Delta(\psi)\vdash\exists x.\chi(x)}\,(Z\vdash t:\text{Term})\ \ \exists R}{\{Y'(X,Y)\}\,\Delta(\Gamma,\phi\mathrel{-\!\!*}\psi)\vdash\exists x.\chi(x)}\,{-\!\!*}L \qquad \rightsquigarrow$$

$$\dfrac{\dfrac{\{X\}\,\Gamma\vdash\phi \quad \{Y'(Y);Z\}\,\Delta(\psi)\vdash\chi[t/x]}{\{Y'(X,Y);Z\}\,\Delta(\Gamma,\phi\mathrel{-\!\!*}\psi)\vdash\chi[t/x]}\,{-\!\!*}L}{\{Y'(X,Y)\}\,\Delta(\Gamma,\phi\mathrel{-\!\!*}\psi)\vdash\exists x.\chi(x)}\,(Z\vdash t:\text{Term})\ \ \exists R$$

$\boxed{-\!\!*R}$

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

$$\dfrac{\dfrac{\{X\}\,\Gamma\vdash\phi \quad \{Y'(Y)\}\,\Delta(\psi),\chi\vdash\xi}{\{Y'(X,Y)\}\,\Delta(\Gamma,\phi\mathrel{-\!\!*}\psi),\chi\vdash\xi}\,{-\!\!*}L}{\{Y'(X,Y)\}\,\Delta(\Gamma,\phi\mathrel{-\!\!*}\psi)\vdash\chi\mathrel{-\!\!*}\xi}\,{-\!\!*}R$$

$\boxed{*R}$

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

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

$\boxed{\forall_{\mathsf{new}} R}$

$$\dfrac{\{X\}\,\Gamma \vdash \phi \qquad \dfrac{\{Y'(Y), v_1\}\,\Delta(\psi) \vdash \chi[v_1/x]}{\{Y'(Y)\}\,\Delta(\psi) \vdash \forall_{\mathsf{new}} x.\chi(x)}\,(v \text{ new tag})\ \forall_{\mathsf{new}} R}{\{Y'(X,Y)\}\,\Delta(\Gamma, \phi \mathrel{-\!\!*} \psi) \vdash \forall_{\mathsf{new}} x.\chi(x)}\,{-\!\!*}L \qquad \rightsquigarrow$$

$$\dfrac{\dfrac{\{X\}\,\Gamma \vdash \phi \quad \{Y'(Y), v_1\}\,\Delta(\psi) \vdash \chi[v_1/x]}{\{Y'(X,Y), v_1\}\,\Delta(\Gamma, \phi \mathrel{-\!\!*} \psi) \vdash \chi[v_1/x]}\,{-\!\!*}L}{\{Y'(X,Y)\}\,\Delta(\Gamma, \phi \mathrel{-\!\!*} \psi) \vdash \forall_{\mathsf{new}} x.\chi(x)}\,(v \text{ new tag})\ \forall_{\mathsf{new}} R$$

$\boxed{\exists_{\mathsf{new}} R}$

$$\dfrac{\{X\}\,\Gamma \vdash \phi \qquad \dfrac{\{Y'(Y), Z\}\,\Delta(\psi) \vdash \chi[t/x]}{\{Y'(Y)\}\,\Delta(\psi) \vdash \exists_{\mathsf{new}} x.\chi(x)}\,(Z \vdash t : \text{Term})\ \exists_{\mathsf{new}} R}{\{Y'(X,Y)\}\,\Delta(\Gamma, \phi \mathrel{-\!\!*} \psi) \vdash \exists_{\mathsf{new}} x.\chi(x)}\,{-\!\!*}L \qquad \rightsquigarrow$$

$$\dfrac{\dfrac{\{X\}\,\Gamma \vdash \phi \quad \{Y'(Y), Z\}\,\Delta(\psi) \vdash \chi[t/x]}{\{Y'(X,Y), Z\}\,\Delta(\Gamma, \phi \mathrel{-\!\!*} \psi) \vdash \chi[t/x]}\,{-\!\!*}L}{\{Y'(X,Y)\}\,\Delta(\Gamma, \phi \mathrel{-\!\!*} \psi) \vdash \exists_{\mathsf{new}} x.\chi(x)}\,(Z \vdash t : \text{Term})\ \exists_{\mathsf{new}} R$$

3.10.3 Additive universal quantifier ($\forall L$)

All applications of $\forall L$ have as side condition that $Y \vdash t : \text{Term}$

$\boxed{\to R}$

$$\dfrac{\dfrac{\{X;Y\}\,\Gamma(\phi(t));\psi \vdash \chi}{\{X;Y\}\,\Gamma(\phi(t)) \vdash \psi \to \chi}\,\to R}{\{X\}\,\Gamma(\forall x.\phi(x)) \vdash \psi \to \chi}\,\forall L \qquad \rightsquigarrow$$

$$\dfrac{\dfrac{\{X;Y\}\,\Gamma(\phi(t));\psi \vdash \chi}{\{X\}\,\Gamma(\forall x.\phi(x));\psi \vdash \chi}\,\forall L}{\{X\}\,\Gamma(\forall x.\phi(x)) \vdash \psi \to \chi}\,\to R$$

$\boxed{\wedge R}$

$$\dfrac{\dfrac{\{X;Y\}\,\Gamma(\phi(t)) \vdash \psi \quad \{Z\}\,\Delta \vdash \chi}{\{X;Y;Z\}\,\Gamma(\phi(t));\Delta \vdash \psi \wedge \chi}\,\wedge R}{\{X;Z\}\,\Gamma(\forall x.\phi(x));\Delta \vdash \psi \wedge \chi}\,\forall L \qquad \rightsquigarrow$$

$$\dfrac{\dfrac{\{X;Y\}\,\Gamma(\phi(t)) \vdash \psi}{\{X\}\,\Gamma(\forall x.\phi(x)) \vdash \psi}\,\forall L \qquad \{Z\}\,\Delta \vdash \chi}{\{X;Z\}\,\Gamma(\forall x.\phi(x));\Delta \vdash \psi \wedge \chi}\,\wedge R$$

$\boxed{\vee R}$ For $i = 1, 2$

$$\dfrac{\dfrac{\{X;Y\}\,\Gamma(\phi(t)) \vdash \psi_i}{\{X;Y\}\,\Gamma(\phi(t)) \vdash \psi_1 \vee \psi_2}\,\vee R}{\{X\}\,\Gamma(\forall x.\phi(x)) \vdash \psi_1 \vee \psi_2}\,\forall L \qquad \rightsquigarrow$$

$$\dfrac{\dfrac{\{X;Y\}\,\Gamma(\phi(t)) \vdash \psi_i}{\{X\}\,\Gamma(\forall x.\phi(x)) \vdash \psi_i}\,\forall L}{\{X\}\,\Gamma(\forall x.\phi(x)) \vdash \psi_1 \vee \psi_2}\,\vee R$$

$\boxed{\forall R}$

$$\dfrac{\dfrac{\{X;Y;v_1\}\,\Gamma(\phi(t)) \vdash \psi[v_1/y]}{\{X;Y\}\,\Gamma(\phi(t)) \vdash \forall_{\odot}\, y.\psi(y)}\,(v \text{ new tag})\ \forall R}{\{X\}\,\Gamma(\forall x.\phi(x)) \vdash \forall y.\psi(y)}\,\forall L \qquad \rightsquigarrow$$

$$\dfrac{\dfrac{\{X;Y;v_1\}\,\Gamma(\phi(t)) \vdash \psi[v_1/y]}{\{X;v_1\}\,\Gamma(\forall x.\phi(x)) \vdash \psi[v_1/y]}\,\forall L}{\{X\}\,\Gamma(\forall x.\phi(x)) \vdash \forall y.\psi(y)}\,(v \text{ new tag})\ \forall R$$

$\boxed{\exists R}$

$$\dfrac{\dfrac{\{X;Y;Z\}\,\Gamma(\phi(t)) \vdash \psi[u/x]}{\{X;Y\}\,\Gamma(\phi(t)) \vdash \exists x.\psi(x)}\,(Z \vdash u : \text{Term})\ \exists R}{\{X\}\,\Gamma(\forall x.\phi(x)) \vdash \exists x.\psi(x)}\,\forall L \qquad \rightsquigarrow$$

$$\dfrac{\dfrac{\{X;Y;Z\}\,\Gamma(\phi(t)) \vdash \psi[u/x]}{\{X;Z\}\,\Gamma(\forall x.\phi(x)) \vdash \psi[u/x]}\,\forall L}{\{X\}\,\Gamma(\forall x.\phi(x)) \vdash \exists x.\psi(x)}\,(Z \vdash u : \text{Term})\ \exists R$$

$\boxed{-\!\!*R}$

$$\dfrac{\dfrac{\{X;Y\}\,\Gamma(\phi(t)), \psi \vdash \chi}{\{X;Y\}\,\Gamma(\phi(t)) \vdash \psi -\!\!* \chi}\,-\!\!*R}{\{X\}\,\Gamma(\forall x.\phi(x)) \vdash \psi -\!\!* \chi}\,\forall L \qquad \rightsquigarrow$$

$$\dfrac{\dfrac{\{X;Y\}\,\Gamma(\phi(t)), \psi \vdash \chi}{\{X\}\,\Gamma(\forall x.\phi(x)), \psi \vdash \chi}\,\forall L}{\{X\}\,\Gamma(\forall x.\phi(x)) \vdash \psi -\!\!* \chi}\,-\!\!*R$$

$\boxed{*R}$

$$\dfrac{\dfrac{\{X;Y\}\,\Gamma(\phi(t)) \vdash \psi \quad \{Z\}\,\Delta \vdash \chi}{\{(X;Y),Z\}\,\Gamma(\phi(t)),\Delta \vdash \psi * \chi}\,{*R}}{\{X,Z\}\,\Gamma(\forall x.\phi(x)),\Delta \vdash \psi * \chi}\,\forall L \qquad \rightsquigarrow$$

$$\dfrac{\dfrac{\{X;Y\}\,\Gamma(\phi(t)) \vdash \psi}{\{X\}\,\Gamma(\forall x.\phi(x)) \vdash \psi}\,\forall L \qquad \{Z\}\,\Delta \vdash \chi}{\{X,Z\}\,\Gamma(\forall x.\phi(x)),\Delta \vdash \psi * \chi}\,{*R}$$

$\boxed{\forall_{\text{new}} R}$

$$\dfrac{\dfrac{\{(X;Y),v_1\}\,\Gamma(\phi(t)) \vdash \psi[v_1/y]}{\{X;Y\}\,\Gamma(\phi(t)) \vdash \forall_{\text{new}} y.\psi(y)}\,(v \text{ new tag})\ \forall_{\text{new}} R}{\{X\}\,\Gamma(\forall x.\phi(x)) \vdash \forall_{\text{new}} y.\psi(y)}\,\forall L \qquad \rightsquigarrow$$

$$\dfrac{\dfrac{\{(X;Y),v_1\}\,\Gamma(\phi(t)) \vdash \psi[v_1/y]}{\{X,v_1\}\,\Gamma(\forall x.\phi(x)) \vdash \psi[v_1/y]}\,\forall L}{\{X\}\,\Gamma(\forall x.\phi(x)) \vdash \forall_{\text{new}} y.\psi(y)}\,(v \text{ new tag})\ \forall_{\text{new}} R$$

$\boxed{\exists_{\text{new}} R}$

$$\dfrac{\dfrac{\{(X;Y),Z\}\,\Gamma(\phi(t)) \vdash \psi[u/x]}{\{X;Y\}\,\Gamma(\phi(t)) \vdash \exists_{\text{new}} x.\psi(x)}\,(Z \vdash u : \text{Term})\ \exists_{\text{new}} R}{\{X\}\,\Gamma(\forall x.\phi(x)) \vdash \exists_{\text{new}} x.\psi(x)}\,\forall L \qquad \rightsquigarrow$$

$$\dfrac{\dfrac{\{(X;Y),Z\}\,\Gamma(\phi(t)) \vdash \psi[u/x]}{\{X,Z\}\,\Gamma(\forall x.\phi(x)) \vdash \psi[u/x]}\,\forall L}{\{X\}\,\Gamma(\forall x.\phi(x)) \vdash \exists_{\text{new}} x.\psi(x)}\,(Z \vdash u : \text{Term})\ \exists_{\text{new}} R$$

3.10.4 Multiplicative universal quantifier ($\forall_{\text{new}} L$)

All applications of $\forall_{\text{new}} L$ have as side condition that $Y \vdash t : \text{Term}$

$\boxed{\to R}$

$$\dfrac{\dfrac{\{X,Y\}\,\Gamma(\phi(t));\psi \vdash \chi}{\{X,Y\}\,\Gamma(\phi(t)) \vdash \psi \to \chi}\,{\to R}}{\{X\}\,\Gamma(\forall_{\text{new}} x.\phi(x)) \vdash \psi \to \chi}\,\forall_{\text{new}} L \qquad \rightsquigarrow$$

$$\dfrac{\dfrac{\{X,Y\}\,\Gamma(\phi(t));\psi \vdash \chi}{\{X\}\,\Gamma(\forall_{\text{new}} x.\phi(x));\psi \vdash \chi}\,\forall_{\text{new}} L}{\{X\}\,\Gamma(\forall_{\text{new}} x.\phi(x)) \vdash \psi \to \chi}\,{\to R}$$

$\boxed{\wedge R}$

$$\dfrac{\dfrac{\{X,Y\}\,\Gamma(\phi(t)) \vdash \psi \quad \{Z\}\,\Delta \vdash \chi}{\{(X,Y);Z\}\,\Gamma(\phi(t));\Delta \vdash \psi \wedge \chi}\,\wedge R}{\{X;Z\}\,\Gamma(\forall_{\mathsf{new}} x.\phi(x));\Delta \vdash \psi \wedge \chi}\,\forall_{\mathsf{new}} L \qquad \rightsquigarrow$$

$$\dfrac{\dfrac{\{X,Y\}\,\Gamma(\phi(t)) \vdash \psi}{\{X\}\,\Gamma(\forall_{\mathsf{new}} x.\phi(x)) \vdash \psi}\,\forall_{\mathsf{new}} L \qquad \{Z\}\,\Delta \vdash \chi}{\{X;Z\}\,\Gamma(\forall_{\mathsf{new}} x.\phi(x));\Delta \vdash \psi \wedge \chi}\,\wedge R$$

$\boxed{\vee R}$ For $i = 1,2$

$$\dfrac{\dfrac{\{X,Y\}\,\Gamma(\phi(t)) \vdash \psi_i}{\{X,Y\}\,\Gamma(\phi(t)) \vdash \psi_1 \vee \psi_2}\,\vee R}{\{X\}\,\Gamma(\forall_{\mathsf{new}} x.\phi(x)) \vdash \psi_1 \vee \psi_2}\,\forall_{\mathsf{new}} L \qquad \rightsquigarrow$$

$$\dfrac{\dfrac{\{X,Y\}\,\Gamma(\phi(t)) \vdash \psi_i}{\{X\}\,\Gamma(\forall_{\mathsf{new}} x.\phi(x)) \vdash \psi_i}\,\forall_{\mathsf{new}} L}{\{X\}\,\Gamma(\forall_{\mathsf{new}} x.\phi(x)) \vdash \psi_1 \vee \psi_2}\,\vee R$$

$\boxed{\forall R}$

$$\dfrac{\dfrac{\{(X,Y);v_1\}\,\Gamma(\phi(t)) \vdash \psi[v_1/y]}{\{X,Y\}\,\Gamma(\phi(t)) \vdash \forall_{\mathsf{new}} y.\psi(y)}\,(v \text{ new tag})\ \forall R}{\{X\}\,\Gamma(\forall_{\mathsf{new}} x.\phi(x)) \vdash \forall_{\mathsf{new}} y.\psi(y)}\,\forall_{\mathsf{new}} L \qquad \rightsquigarrow$$

$$\dfrac{\dfrac{\{(X,Y);v_1\}\,\Gamma(\phi(t)) \vdash \psi[v_1/y]}{\{X;v_1\}\,\Gamma(\forall_{\mathsf{new}} x.\phi(x)) \vdash \psi[v_1/y]}\,\forall_{\mathsf{new}} L}{\{X\}\,\Gamma(\forall_{\mathsf{new}} x.\phi(x)) \vdash \forall_{\mathsf{new}} y.\psi(y)}\,(v \text{ new tag})\ \forall R$$

$\boxed{\exists R}$

$$\dfrac{\dfrac{\{(X,Y);Z\}\,\Gamma(\phi(t)) \vdash \psi[u/x]}{\{X,Y\}\,\Gamma(\phi(t)) \vdash \exists x.\psi(x)}\,(Z \vdash u : \text{Term})\ \exists R}{\{X\}\,\Gamma(\forall_{\mathsf{new}} x.\phi(x)) \vdash \exists x.\psi(x)}\,\forall_{\mathsf{new}} L \qquad \rightsquigarrow$$

$$\dfrac{\dfrac{\{(X,Y);Z\}\,\Gamma(\phi(t)) \vdash \psi[u/x]}{\{X;Z\}\,\Gamma(\forall_{\mathsf{new}} x.\phi(x)) \vdash \psi[u/x]}\,\forall_{\mathsf{new}} L}{\{X\}\,\Gamma(\forall_{\mathsf{new}} x.\phi(x)) \vdash \exists x.\psi(x)}\,(Z \vdash u : \text{Term})\ \exists R$$

$\boxed{-\!\!*R}$

$$\dfrac{\dfrac{\{X,Y\}\,\Gamma(\phi(t)),\psi \vdash \chi}{\{X,Y\}\,\Gamma(\phi(t)) \vdash \psi -\!\!*\chi}\,{-\!\!*}R}{\{X\}\,\Gamma(\forall_{\mathsf{new}}x.\phi(x)) \vdash \psi -\!\!*\chi}\,\forall_{\mathsf{new}}L \qquad \rightsquigarrow$$

$$\dfrac{\dfrac{\{X,Y\}\,\Gamma(\phi(t)),\psi \vdash \chi}{\{X\}\,\Gamma(\forall_{\mathsf{new}}x.\phi(x)),\psi \vdash \chi}\,\forall_{\mathsf{new}}L}{\{X\}\,\Gamma(\forall_{\mathsf{new}}x.\phi(x)) \vdash \psi -\!\!*\chi}\,{-\!\!*}R$$

$\boxed{*R}$

$$\dfrac{\dfrac{\{X,Y\}\,\Gamma(\phi(t)) \vdash \psi \quad \{Z\}\,\Delta \vdash \chi}{\{X,Y,Z\}\,\Gamma(\phi(t)),\Delta \vdash \psi * \chi}\,*R}{\{X,Z\}\,\Gamma(\forall_{\mathsf{new}}x.\phi(x)),\Delta \vdash \psi * \chi}\,\forall_{\mathsf{new}}L \qquad \rightsquigarrow$$

$$\dfrac{\dfrac{\{X,Y\}\,\Gamma(\phi(t)) \vdash \psi}{\{X\}\,\Gamma(\forall_{\mathsf{new}}x.\phi(x)) \vdash \psi}\,\forall_{\mathsf{new}}L \qquad \{Z\}\,\Delta \vdash \chi}{\{X,Z\}\,\Gamma(\forall_{\mathsf{new}}x.\phi(x)),\Delta \vdash \psi * \chi}\,*R$$

$\boxed{\forall_{\mathsf{new}}R}$

$$\dfrac{\dfrac{\{X,Y,v_1\}\,\Gamma(\phi(t)) \vdash \psi[v_1/y]}{\{X,Y\}\,\Gamma(\phi(t)) \vdash \forall_{\mathsf{new}}y.\psi(y)}\,(v \text{ new tag})\ \forall_{\mathsf{new}}R}{\{X\}\,\Gamma(\forall_{\mathsf{new}}x.\phi(x)) \vdash \forall_{\mathsf{new}}y.\psi(y)}\,\forall_{\mathsf{new}}L \qquad \rightsquigarrow$$

$$\dfrac{\dfrac{\{X,Y,v_1\}\,\Gamma(\phi(t)) \vdash \psi[v_1/y]}{\{X,v_1\}\,\Gamma(\forall_{\mathsf{new}}x.\phi(x)) \vdash \psi[v_1/y]}\,\forall_{\mathsf{new}}L}{\{X\}\,\Gamma(\forall_{\mathsf{new}}x.\phi(x)) \vdash \forall_{\mathsf{new}}y.\psi(y)}\,(v \text{ new tag})\ \forall_{\mathsf{new}}R$$

$\boxed{\exists_{\mathsf{new}}R}$

$$\dfrac{\dfrac{\{X,Y,Z\}\,\Gamma(\phi(t)) \vdash \psi[u/x]}{\{X,Y\}\,\Gamma(\phi(t)) \vdash \exists_{\mathsf{new}}x.\psi(x)}\,(Z \vdash u : \text{Term})\ \exists_{\mathsf{new}}R}{\{X\}\,\Gamma(\forall_{\mathsf{new}}x.\phi(x)) \vdash \exists_{\mathsf{new}}x.\psi(x)}\,\forall_{\mathsf{new}}L \qquad \rightsquigarrow$$

$$\dfrac{\dfrac{\{X,Y,Z\}\,\Gamma(\phi(t)) \vdash \psi[u/x]}{\{X,Z\}\,\Gamma(\forall_{\mathsf{new}}x.\phi(x)) \vdash \psi[u/x]}\,\forall_{\mathsf{new}}L}{\{X\}\,\Gamma(\forall_{\mathsf{new}}x.\phi(x)) \vdash \exists_{\mathsf{new}}x.\psi(x)}\,(Z \vdash u : \text{Term})\ \exists_{\mathsf{new}}R$$

3.11 A shorter version using pseudo-connectives

We will exploit the symmetries on the rules of bunched implications to cut down the number of figures analysed. To this end we introduce a special notation that can be instantiated to multiplicative or additive versions of the connectives. The symbols $\circ$ and $\odot$ are pseudo-operators for ’,’ or ’;’. The symbols ${-\Box}$ and ${-\boxdot}$ are pseudo-connectives for ${-\!\!*}$ and $\to$. The symbols $\Box$ and $\boxdot$ are pseudo-connectives for $*$ and $\wedge$. The symbols $\forall_\circ$ and $\forall_\odot$ are pseudo-connectives for $\forall_{\mathsf{new}}$ and $\forall$. The symbols $\exists_\circ$ and $\exists_\odot$ are pseudo-connectives for $\exists_{\mathsf{new}}$ and $\exists$.

$\circ$ has to match ${-\Box}$, $\Box$, $\forall_\circ$ and $\exists_\circ$. Conversely $\odot$ has to match ${-\boxdot}$, $\boxdot$, $\forall_\odot$ and $\exists_\odot$.

For example the sequent

$$\{Y'(X\circ Y)\odot Y''\}\,\Delta(\Gamma\circ\phi{-\Box}\psi)\odot\Delta'\vdash\chi\boxdot\xi$$

can have any of the following four instantiations

$$\{Y'(X;Y);Y''\}\,\Delta(\Gamma;\phi\to\psi);\Delta'\vdash\chi\wedge\xi$$
$$\{Y'(X;Y),Y''\}\,\Delta(\Gamma;\phi\to\psi),\Delta'\vdash\chi*\xi$$
$$\{Y'(X,Y);Y''\}\,\Delta(\Gamma,\phi{-\!\!*}\psi);\Delta'\vdash\chi\wedge\xi$$
$$\{Y'(X,Y),Y''\}\,\Delta(\Gamma,\phi{-\!\!*}\psi),\Delta'\vdash\chi*\xi$$

This way of presenting permutations is inspired by the work of [Kleene 52] in intuitionistic and classical logic.

3.11.1 Implication (${-\Box}L$)

All applications of ${-\Box}L$ have the side condition that $Y\vdash\psi:\mathrm{Prop}$.

$\boxed{{-\boxdot}R}$

$$\dfrac{\{X\}\,\Gamma\vdash\phi \quad \dfrac{\{Y'(Y)\}\,\Delta(\psi)\odot\chi\vdash\xi}{\{Y'(Y)\}\,\Delta(\psi)\vdash\chi{-\boxdot}\xi}\,{-\boxdot}R}{\{Y'(X\circ Y)\}\,\Delta(\Gamma\circ\phi{-\Box}\psi)\vdash\chi{-\boxdot}\xi}\,{-\Box}L \qquad \rightsquigarrow$$

$$\dfrac{\dfrac{\{X\}\,\Gamma\vdash\phi \quad \{Y'(Y)\}\,\Delta(\psi)\odot\chi\vdash\xi}{\{Y'(X\circ Y)\}\,\Delta(\Gamma\circ\phi{-\Box}\psi)\odot\chi\vdash\xi}\,{-\Box}L}{\{Y'(X\circ Y)\}\,\Delta(\Gamma\circ\phi{-\Box}\psi)\vdash\chi{-\boxdot}\xi}\,{-\boxdot}R$$

$\boxed{\boxdot R}$

$$\dfrac{\{X\}\,\Gamma\vdash\phi \quad \dfrac{\{Y'(Y)\}\,\Delta(\psi)\vdash\chi \quad \{Y''\}\,\Delta'\vdash\xi}{\{Y'(Y)\odot Y''\}\,\Delta(\psi)\odot\Delta'\vdash\chi\boxdot\xi}\,\boxdot R}{\{Y'(X\circ Y)\odot Y''\}\,\Delta(\Gamma\circ\phi{-\Box}\psi)\odot\Delta'\vdash\chi\boxdot\xi}\,{-\Box}L \qquad \rightsquigarrow$$

$$\dfrac{\dfrac{\{X\}\,\Gamma\vdash\phi \quad \{Y'(Y)\}\,\Delta(\psi)\vdash\chi}{\{Y'(X\circ Y)\}\,\Delta(\Gamma\circ\phi{-\Box}\psi)\vdash\chi}\,{-\Box}L \quad \{Y''\}\,\Delta'\vdash\xi}{\{Y'(X\circ Y)\odot Y''\}\,\Delta(\Gamma\circ\phi{-\Box}\psi)\odot\Delta'\vdash\chi\boxdot\xi}\,\boxdot R$$

$\boxed{\vee R}$ For $i = 1, 2$

$$\dfrac{\{X\}\,\Gamma \vdash \phi \quad \dfrac{\{Y'(Y)\}\,\Delta(\psi) \vdash \chi_i}{\{Y'(Y)\}\,\Delta(\psi) \vdash \chi_1 \vee \chi_2}\,\vee R}{\{Y'(X \circ Y)\}\,\Delta(\Gamma \circ \phi {-}\Box \psi) \vdash \chi_1 \vee \chi_2}\,{-}\Box L \qquad \rightsquigarrow$$

$$\dfrac{\dfrac{\{X\}\,\Gamma \vdash \phi \quad \{Y'(Y)\}\,\Delta(\psi) \vdash \chi_i}{\{Y'(X \circ Y)\}\,\Delta(\Gamma \circ \phi {-}\Box \psi) \vdash \chi_i}\,{-}\Box L}{\{Y'(X \circ Y)\}\,\Delta(\Gamma \circ \phi {-}\Box \psi) \vdash \chi_1 \vee \chi_2}\,\vee R$$

$\boxed{\forall_{\odot} R}$

$$\dfrac{\{X\}\,\Gamma \vdash \phi \quad \dfrac{\{Y'(Y) \odot v_1\}\,\Delta(\psi) \vdash \chi[v_1/x]}{\{Y'(Y)\}\,\Delta(\psi) \vdash \forall_{\odot} x.\chi(x)}\,(v \text{ new tag})\ \forall_{\odot} R}{\{Y'(X \circ Y)\}\,\Delta(\Gamma \circ \phi {-}\Box \psi) \vdash \forall_{\odot} x.\chi(x)}\,{-}\Box L \qquad \rightsquigarrow$$

$$\dfrac{\dfrac{\{X\}\,\Gamma \vdash \phi \quad \{Y'(Y) \odot v_1\}\,\Delta(\psi) \vdash \chi[v_1/x]}{\{Y'(X \circ Y) \odot v_1\}\,\Delta(\Gamma \circ \phi {-}\Box \psi) \vdash \chi[v_1/x]}\,{-}\Box L}{\{Y'(X \circ Y)\}\,\Delta(\Gamma \circ \phi {-}\Box \psi) \vdash \forall_{\odot} x.\chi(x)}\,(v \text{ new tag})\ \forall_{\odot} R$$

$\boxed{\exists_{\odot} R}$

$$\dfrac{\{X\}\,\Gamma \vdash \phi \quad \dfrac{\{Y'(Y) \odot Z\}\,\Delta(\psi) \vdash \chi[t/x]}{\{Y'(Y)\}\,\Delta(\psi) \vdash \exists_{\odot} x.\chi(x)}\,(Z \vdash t : \text{Term})\ \exists_{\odot} R}{\{Y'(X \circ Y)\}\,\Delta(\Gamma \circ \phi {-}\Box \psi) \vdash \exists_{\odot} x.\chi(x)}\,{-}\Box L \qquad \rightsquigarrow$$

$$\dfrac{\dfrac{\{X\}\,\Gamma \vdash \phi \quad \{Y'(Y) \odot Z\}\,\Delta(\psi) \vdash \chi[t/x]}{\{Y'(X \circ Y) \odot Z\}\,\Delta(\Gamma \circ \phi {-}\Box \psi) \vdash \chi[t/x]}\,{-}\Box L}{\{Y'(X \circ Y)\}\,\Delta(\Gamma \circ \phi {-}\Box \psi) \vdash \exists_{\odot} x.\chi(x)}\,(Z \vdash t : \text{Term})\ \exists_{\odot} R$$

3.11.2 Universal quantifier ($\forall_{\circ} L$)

All applications of $\forall_{\circ} L$ have as side condition that $Y \vdash t : \text{Term}$

$\boxed{{-}\boxdot R}$

$$\dfrac{\dfrac{\{X \circ Y\}\,\Gamma(\phi(t)) \odot \psi \vdash \chi}{\{X \circ Y\}\,\Gamma(\phi(t)) \vdash \psi {-}\boxdot \chi}\,{-}\boxdot R}{\{X\}\,\Gamma(\forall_{\circ} x.\phi(x)) \vdash \psi {-}\boxdot \chi}\,\forall_{\circ} L \qquad \rightsquigarrow$$

$$\dfrac{\dfrac{\{X \circ Y\}\,\Gamma(\phi(t)) \odot \psi \vdash \chi}{\{X\}\,\Gamma(\forall_{\circ} x.\phi(x)) \odot \psi \vdash \chi}\,\forall_{\circ} L}{\{X\}\,\Gamma(\forall_{\circ} x.\phi(x)) \vdash \psi {-}\boxdot \chi}\,{-}\boxdot R$$

$\boxed{\boxdot R}$

$$\dfrac{\dfrac{\{X\circ Y\}\,\Gamma(\phi(t))\vdash\psi \quad \{Z\}\,\Delta\vdash\chi}{\{(X\circ Y)\odot Z\}\,\Gamma(\phi(t))\odot\Delta\vdash\psi\boxdot\chi}\,\boxdot R}{\{X\odot Z\}\,\Gamma(\forall_{\bigcirc}x.\phi(x))\odot\Delta\vdash\psi\boxdot\chi}\,\forall_{\bigcirc}L \qquad \leadsto$$

$$\dfrac{\dfrac{\{X\circ Y\}\,\Gamma(\phi(t))\vdash\psi}{\{X\}\,\Gamma(\forall_{\bigcirc}x.\phi(x))\vdash\psi}\,\forall_{\bigcirc}L \qquad \{Z\}\,\Delta\vdash\chi}{\{X\odot Z\}\,\Gamma(\forall_{\bigcirc}x.\phi(x))\odot\Delta\vdash\psi\boxdot\chi}\,\boxdot R$$

$\boxed{\vee R}$ For $i=1,2$

$$\dfrac{\dfrac{\{X\circ Y\}\,\Gamma(\phi(t))\vdash\psi_i}{\{X\circ Y\}\,\Gamma(\phi(t))\vdash\psi_1\vee\psi_2}\,\vee R}{\{X\}\,\Gamma(\forall_{\bigcirc}x.\phi(x))\vdash\psi_1\vee\psi_2}\,\forall_{\bigcirc}L \qquad \leadsto$$

$$\dfrac{\dfrac{\{X\circ Y\}\,\Gamma(\phi(t))\vdash\psi_i}{\{X\}\,\Gamma(\forall_{\bigcirc}x.\phi(x))\vdash\psi_i}\,\forall_{\bigcirc}L}{\{X\}\,\Gamma(\forall_{\bigcirc}x.\phi(x))\vdash\psi_1\vee\psi_2}\,\vee R$$

$\boxed{\forall_{\odot}R}$

$$\dfrac{\dfrac{\{(X\circ Y)\odot v_1\}\,\Gamma(\phi(t))\vdash\psi[v_1/y]}{\{X\circ Y\}\,\Gamma(\phi(t))\vdash\forall_{\odot}y.\psi(y)}\,(v \text{ new tag})\ \forall_{\odot}R}{\{X\}\,\Gamma(\forall_{\bigcirc}x.\phi(x))\vdash\forall_{\odot}y.\psi(y)}\,\forall_{\bigcirc}L \qquad \leadsto$$

$$\dfrac{\dfrac{\{(X\circ Y)\odot v_1\}\,\Gamma(\phi(t))\vdash\psi[v_1/y]}{\{X\odot v_1\}\,\Gamma(\forall_{\bigcirc}x.\phi(x))\vdash\psi[v_1/y]}\,\forall_{\bigcirc}L}{\{X\}\,\Gamma(\forall_{\bigcirc}x.\phi(x))\vdash\forall_{\odot}y.\psi(y)}\,(v \text{ new tag})\ \forall_{\odot}R$$

$\boxed{\exists_{\odot}R}$

$$\dfrac{\dfrac{\{(X\circ Y)\odot Z\}\,\Gamma(\phi(t))\vdash\psi[u/x]}{\{X\circ Y\}\,\Gamma(\phi(t))\vdash\exists_{\odot}x.\psi(x)}\,(Z\vdash u:\text{Term})\ \exists_{\odot}R}{\{X\}\,\Gamma(\forall_{\bigcirc}x.\phi(x))\vdash\exists_{\odot}x.\psi(x)}\,\forall_{\bigcirc}L \qquad \leadsto$$

$$\dfrac{\dfrac{\{(X\circ Y)\odot Z\}\,\Gamma(\phi(t))\vdash\psi[u/x]}{\{X\odot Z\}\,\Gamma(\forall_{\bigcirc}x.\phi(x))\vdash\psi[u/x]}\,\forall_{\bigcirc}L}{\{X\}\,\Gamma(\forall_{\bigcirc}x.\phi(x))\vdash\exists_{\odot}x.\psi(x)}\,(Z\vdash u:\text{Term})\ \exists_{\odot}R$$

Chapter 4

Operational Semantics

4.1 Introduction

In this chapter we develope an operational semantics for a logic programming language based on **BI**. We use a structural approach as described in [Plotkin 81] and we provide a transition system in the shape of transition rules which closely match the sequent calculus described in the last chapter.

- Define a subtraction operation on bunches.
- Describe in detail an operational semantics for propositional **BI** logic programming.
- Prove soundness and completeness of the operational semantics with respect to **LBI**.
- Give an operational semantics for the quantifiers.

4.2 Subtraction Operation on Bunches

Definition 4.2.1 *We say that a clause ϕ is at* top level *in a bunch Γ if either Γ is ϕ itself or if Γ is a multiplicative bunch and ϕ is one of the additive bunches forming Γ.*

All clauses can be brought to the top level by performing weakenings in a unique way, as the following lemma shows:

Lemma 4.2.2 *Given a bunch Γ of clauses, and a clause ϕ belonging to it, there is exactly one way of performing the minimum number of Weakening reductions so as to bring ϕ to the top level.*

Proof: If ϕ is at top level already, there is nothing to do. Else there are two cases depending on whether Γ is additive or multiplicative. In the first case ϕ belongs to one of the multiplicative bunches forming Γ. It is clear that to put ϕ at top level all the other multiplicative bunches must be weakened away. Then we are either left with ϕ at top level, or we carry on with the second case.

In the second case ϕ belongs to one of the additive bunches forming $\Gamma = \overset{;}{\Gamma}_1, \cdots, \overset{;}{\Gamma}_n$, say Γ_i. Moreover Γ_i itself must be of the form $\overset{,}{\Delta}_1; \cdots; \overset{,}{\Delta}_n$, if ϕ was not at top level, and ϕ belongs to one of these, say Δ_j. To bring ϕ to the top level, all $\Gamma_k (k \neq j)$ must be weakened away and we are left with a new $\Gamma' = \Gamma_1, \cdots, \Delta_j, \cdots, \Gamma_n$ where Γ_i is replaced by Δ_j. We can repeat this process with Γ' which is strictly smaller than Γ, guaranteeing termination.

□

Consider Δ to be an additive bunch or a proposition. Then it is always possible to transform any bunch of the form $\Gamma(\Delta)$ into a bunch with the shape Γ', Δ using weakenings. Moreover, there is a unique way of doing this using the minimal number of weakenings possible.

The base case is when $\Gamma = \Delta$. No further processing is necessary.

If $\Gamma = \overset{,}{\Gamma}_1; \cdots; \overset{,}{\Gamma}_n$ then, since we are dealing with a case where $\Gamma \neq \Delta$, Δ has to belong to one of the Γ_i. Then all the other bunches at that level need to be weakened and the result is Γ'_i.

If $\Gamma = \overset{;}{\Gamma}_1, \cdots, \overset{;}{\Gamma}_n$ then Δ has to belong to one of the Γ_i and we continue the process with this bunch. If we call the resulting bunch Γ'_i then the final result will be $\Gamma_1, \cdots, \Gamma'_i, \cdots, \Gamma_n$.

This concept is easily extended to the case where Δ is a multiplicative bunch. We just perform the weakenings in turn on each additive subbunch.

Defined formally as an operation parameterised on Δ we have

Definition 4.2.3 (Minimum Number of Weakenings (MNW)) *The operation MNW_Δ is defined as*

1. $MNW_\Delta(\Delta) = \Delta$
2. $MNW_\Delta(\overset{,}{\Gamma}_1; \cdots; \overset{,}{\Gamma}_n) = MNW_\Delta(\overset{,}{\Gamma}_i)$
3. $MNW_\Delta(\overset{;}{\Gamma}_1, \cdots, \overset{;}{\Gamma}_n) = \overset{;}{\Gamma}_1, \cdots, MNW_\Delta(\overset{;}{\Gamma}_i), \cdots, \overset{;}{\Gamma}_n$
4. $MNW_{(\Delta_1,\Delta_2,\cdots,\Delta_n)}(\Gamma) = MNW_{\Delta_1}(MNW_{(\Delta_2,\cdots,\Delta_n)}(\Gamma))$

We need the *MNW* operation to define the notion of subbunch:

Definition 4.2.4 (subbunch) *A bunch Δ is said to be a* subbunch *of Γ, denoted $\Delta \subseteq \Gamma$ in the following cases:*

1. *when $\Delta = \varnothing_m$*
2. *when $\Delta = \overset{;}{\Delta}$ or Δ is a proposition and Γ is of the form $\Gamma'(\Delta)$*
3. *when $\Delta = \overset{;}{\Delta}_1, \cdots, \overset{;}{\Delta}_n$ and*
   - *$\Delta_1 \subseteq \Gamma$, and*
   - *if we let $(\Gamma', \Delta_1) = MNW_{\Delta_1}(\Gamma)$, then $(\overset{;}{\Delta}_2, \cdots, \overset{;}{\Delta}_n) \subseteq \Gamma'$*

Is it really necessary to define the “subbunch” relationship in such a complicated way? At first sight it might be thought that a simpler definition, as the one provided in [Armelin Pym 02] would be sufficient. However this definition is incorrect. For example it allows $(a;b) \subseteq (a,b)$

Lemma 4.2.5 *If $\Theta \subseteq \Delta \subseteq \Gamma$, then*

$$MNW_\Theta(MNW_\Delta(\Gamma)) = MNW_\Theta(\Gamma)$$

Proof: First we consider the case when Δ is an additive bunch or a clause. In this case we use induction on the shape of Γ. The proof is based on the fact that if Δ belongs to a subbunch of Γ, then Θ belongs to that same subbunch.

Case $\Gamma = \Delta$:

$$MNW_\Theta(MNW_\Delta(\Delta)) \quad = \quad MNW_\Theta(\Delta) \quad \text{(Definition 4.2.3)}$$

Case $\Gamma = \overset{;}{\Gamma}_1, \cdots, \overset{;}{\Gamma}_n$: Since Δ is an additive bunch or a clause, then $\Delta \subseteq \Gamma_i$ for some i. Then

$$\begin{aligned}
MNW_{\Theta}(MNW_{\Delta}(\overset{,}{\Gamma}_1; \cdots; \overset{,}{\Gamma}_n)) &= MNW_{\Theta}(MNW_{\Delta}(\overset{;}{\Gamma}_i)) \quad \text{(Definition 4.2.3)} \\
&= MNW_{\Theta}(\overset{;}{\Gamma}_i) \quad \text{(Induction Hypothesis)} \\
&= MNW_{\Theta}(\overset{,}{\Gamma}_1; \cdots; \overset{,}{\Gamma}_n) \quad \text{(Definition 4.2.3)}
\end{aligned}$$

Case $\Gamma = \overset{,}{\Gamma}_1; \cdots; \overset{,}{\Gamma}_n$: Again, given that Δ is an additive bunch or a clause, then $\Delta \subseteq \Gamma_i$ for some i. Then

$$\begin{aligned}
MNW_{\Theta}(MNW_{\Delta}(\overset{;}{\Gamma}_1, \cdots, \overset{;}{\Gamma}_n)) &= MNW_{\Theta}(\overset{;}{\Gamma}_1, \cdots, MNW_{\Delta}(\Gamma_i), \cdots, \overset{;}{\Gamma}_n) \quad \text{(Definition 4.2.3)} \\
&= \overset{;}{\Gamma}_1, \cdots, MNW_{\Theta}(MNW_{\Delta}(\overset{;}{\Gamma}_i)), \cdots, \overset{;}{\Gamma}_n \quad \text{(Definition 4.2.3)} \\
&= \overset{;}{\Gamma}_1, \cdots, MNW_{\Theta}(\overset{;}{\Gamma}_i), \cdots, \overset{;}{\Gamma}_n \quad \text{(Induction Hypothesis)} \\
&= MNW_{\Theta}(\overset{;}{\Gamma}_1, \cdots, \overset{;}{\Gamma}_n) \quad \text{(Definition 4.2.3)}
\end{aligned}$$

Finally, if Δ is multiplicative, we use induction on Δ itself:

$$\begin{aligned}
MNW_{\Theta}(MNW_{(\Delta_1, \cdots, \Delta_n)}(\Gamma)) &= MNW_{\Theta}(MNW_{\Delta_1}(MNW_{(\Delta_2, \cdots, \Delta_n)}(\Gamma))) \quad \text{(Definition 4.2.3)} \\
&= MNW_{\Theta}(MNW_{\Delta_1}(\Gamma)) \quad \text{(Case } \overset{;}{\Delta}\text{)} \\
&= MNW_{\Theta}(\Gamma) \quad \text{(Induction Hypothesis)}
\end{aligned}$$

□

Definition 4.2.6 (Subtraction of bunches) *Given two bunches* Γ *and* $\Delta \subseteq \Gamma$ *the* subtraction *operation between them, denoted* $\Gamma - \Delta$*, is defined by cases on* Δ

$$\Gamma - \varnothing_m = \Gamma \tag{4.1}$$

$$\Gamma - \overset{;}{\Delta} = \Gamma'^{\dagger} \tag{4.2}$$

$$\Gamma - (\overset{;}{\Delta}_1, \cdots, \overset{;}{\Delta}_n) = (\Gamma - \overset{;}{\Delta}_1) - (\Delta - \overset{;}{\Delta}_1) \tag{4.3}$$

† *where* $\Gamma', \overset{;}{\Delta} = MNW_{\Delta}(\Gamma)$

Now we prove a series of lemmas that will be used in the proofs of soundness and completeness of the operational semantics.

Lemma 4.2.7 *The following* Weakening *rule is admissible*

$$\dfrac{\Gamma - \Delta \vdash \phi}{(\Theta;\Gamma) - \Delta \vdash \phi}\,\text{Weakening}'$$

Proof: By cases on the shape of Δ. When $\Delta = \varnothing_m$ the rule just reduces to normal weakening:

$$\cfrac{\cfrac{\cfrac{\Gamma - \varnothing_m \vdash \phi}{\Gamma \vdash \phi}\,\text{Definition 4.2.6}}{\Theta;\Gamma \vdash \phi}\,\text{Weakening}}{(\Theta;\Gamma) - \varnothing_m \vdash \phi}\,\text{Definition 4.2.6}$$

Case $\overset{;}{\Delta}$: we notice that $MNW_{\Delta}(\Gamma) = MNW_{\Delta}(\Theta;\Gamma)$, since Θ has to be weakened away to get $\overset{;}{\Delta}$ to the top level.

$$\cfrac{\cfrac{\cfrac{\Gamma - \Delta \vdash \phi}{MNW_{\Delta}(\Gamma) - \Delta \vdash \phi}\,\text{Definition 4.2.6}}{MNW_{\Delta}(\Theta;\Gamma) - \Delta \vdash \phi}\,\text{Weakening}}{(\Theta;\Gamma) - \Delta \vdash \phi}\,\text{Definition 4.2.6}$$

Case $\overset{,}{\Delta} = \overset{;}{\Delta}_1, \cdots, \overset{;}{\Delta}_n$:

$$\cfrac{\cfrac{\cfrac{\cfrac{\cfrac{\Gamma - \Delta \vdash \phi}{(\Gamma - \Delta_1) - (\Delta - \Delta_1) \vdash \phi}\,\text{Definition 4.2.6}}{(MNW_{\Delta_1}(\Gamma) - \Delta_1) - (\Delta - \Delta_1) \vdash \phi}\,\text{Definition 4.2.6}}{(MNW_{\Delta_1}(\Theta;\Gamma) - \Delta_1) - (\Delta - \Delta_1) \vdash \phi}\,\text{Induction Hypothesis}}{((\Theta;\Gamma) - \Delta_1) - (\Delta - \Delta_1) \vdash \phi}\,\text{Definition 4.2.6}}{(\Theta;\Gamma) - \Delta \vdash \phi}\,\text{Definition 4.2.6}$$

□

Lemma 4.2.8 $(\Delta, \Gamma) - \Delta = \Gamma$

Proof: By cases on the shape of Δ

Case $\Delta = \varnothing_m$: $(\varnothing_m, \Gamma) - \varnothing_m = \varnothing_m, \Gamma = \Gamma$

Case $\overset{;}{\Delta}$: Given that $MNW_{\Delta}(\overset{;}{\Delta}, \Gamma) = \overset{;}{\Delta}, \Gamma$, we have that $(\overset{;}{\Delta}, \Gamma) - \overset{;}{\Delta} = \Gamma$

Case $\overset{,}{\Delta} = \overset{;}{\Delta}_1, \cdots, \overset{;}{\Delta}_n$:

$$\begin{aligned}
(\overset{,}{\Delta}, \Gamma) - \overset{,}{\Delta} &= (\overset{;}{\Delta}_1, \cdots, \overset{;}{\Delta}_n, \Gamma) - \overset{;}{\Delta}_1, \cdots, \overset{;}{\Delta}_n \;\; \text{(Case)} \\
&= ((\overset{;}{\Delta}_1, (\Delta - \overset{;}{\Delta}_1), \Gamma) - \overset{;}{\Delta}_1) - (\Delta - \overset{;}{\Delta}_1) \;\; \text{(Subtraction (4.2))} \\
&= ((\Delta - \overset{;}{\Delta}_1), \Gamma) - (\Delta - \overset{;}{\Delta}_1) \;\; \text{(Case } \overset{;}{\Delta}\text{)} \\
&= \Gamma \;\; \text{(Induction Hypothesis)}
\end{aligned}$$

□

Lemma 4.2.9 $\Gamma - (\Delta, \Theta) = (\Gamma - \Delta) - \Theta$

Proof: By induction on the shape of Δ.

Case $\Delta = \varnothing_m$: then $\Gamma - (\varnothing_m, \Theta) = \Gamma - \Theta = (\Gamma - \varnothing_m) - \Theta$

Case $\overset{;}{\Delta}$: by Lemma 4.2.8 we have that $(\overset{;}{\Delta}, \Theta) - \overset{;}{\Delta} = \Theta$ and so

$$\begin{aligned}
\Gamma - (\overset{;}{\Delta}, \Theta) &= (\Gamma - \overset{;}{\Delta}) - ((\overset{;}{\Delta}, \Theta) - \overset{;}{\Delta}) \;\; \text{(Subtraction (4.3))} \\
&= (\Gamma - \overset{;}{\Delta}) - \Theta \;\; \text{(Subtraction (4.2))}
\end{aligned}$$

Case $\overset{,}{\Delta} = \overset{;}{\Delta}_1, \cdots, \overset{;}{\Delta}_n$:

$$\begin{aligned}
\Gamma - (\Delta, \Theta) &= \Gamma - (\overset{;}{\Delta}_1, \cdots, \overset{;}{\Delta}_n, \Theta) \;\; \text{(Case)} \\
&= (((\Gamma - \overset{;}{\Delta}_1) - \cdots) - \overset{;}{\Delta}_n) - \Theta \;\; \text{(Subtraction (4.3))} \\
&= (\Gamma - (\overset{;}{\Delta}_1, \cdots, \overset{;}{\Delta}_n)) - \Theta \;\; \text{(Induction Hypothesis)} \\
&= (\Gamma - \Delta) - \Theta \;\; \text{(Case)}
\end{aligned}$$

□

Lemma 4.2.10 *If* $\Theta \subseteq \Gamma$ *then* $(\Gamma, \Delta) - \Theta = (\Gamma - \Theta), \Delta$

Proof: By induction on the structure of Θ.

Case $\Theta = \varnothing_m$: Then $(\Gamma, \Delta) - \varnothing_m = \Gamma, \Delta = (\Gamma - \varnothing_m), \Delta$

Case $\overset{;}{\Theta}$: then let $\Gamma' = \Gamma - \overset{;}{\Theta}$

$$\begin{aligned}
(\Gamma, \Delta) - \overset{;}{\Theta} &= (\Gamma', \Delta) \;\; \text{(Subtraction (4.2))} \\
&= (\Gamma - \overset{;}{\Theta}), \Delta \;\; \text{(Subtraction (4.2))}
\end{aligned}$$

Notice that in this case the weakenings necessary to bring Θ to the top level with respect to (Γ,Δ) are precisely those necessary to bring it to the top with respect to Γ alone.

Case $\overset{,}{\Theta} = \overset{;}{\Theta}_1, \cdots, \overset{;}{\Theta}_n$: The condition for $\Theta \subseteq \Gamma$ gives us that $\overset{;}{\Theta}_i \in \Gamma \; \forall \; 1 \leq i \leq n$. Then

$$\begin{aligned}
(\Gamma,\Delta) - \Theta &= (\Gamma,\Delta) - (\overset{;}{\Theta}_1, \cdots, \overset{;}{\Theta}_n) \;\; \text{(Case)} \\
&= ((\Gamma,\Delta) - \overset{;}{\Theta}_1) - (\Theta - \overset{;}{\Theta}_1) \;\; \text{(Subtraction (4.3))} \\
&= ((\Gamma - \overset{;}{\Theta}_1),\Delta) - (\Theta - \overset{;}{\Theta}_1) \;\; \text{(Case } \overset{;}{\Theta}\text{)} \\
&= ((\Gamma - \overset{;}{\Theta}_1) - (\Theta - \overset{;}{\Theta}_1)),\Delta \;\; \text{(Induction Hypothesis)} \\
&= (\Gamma - (\overset{;}{\Theta}_1, \cdots, \overset{;}{\Theta}_n)),\Delta \;\; \text{(Subtraction (4.3))} \\
&= (\Gamma - \Theta),\Delta \;\; \text{(Case)}
\end{aligned}$$

□

Note that the behaviour of the subtract operation can have different results if Γ and Δ are permuted. Of course, for this to be possible, we need not only $\Theta \subseteq \Gamma$ but also $\Theta \subseteq \Delta$. For an example of this abnormal behaviour take $\Gamma = a;(b,c)$ and $\Delta = \Theta = b$. Then we have that $(a;(b,c)),b - b = c,b$ but $b,(a;(b,c)) - b = a;(b,c)$. However, the lemma still holds. The only problem is that commutativity cannot be used.

Lemma 4.2.11 *If* $\Theta \subseteq \Delta \subseteq \Gamma$*, then* $(\Gamma,\Delta) - \Theta \sqsubseteq \Gamma,(\Delta - \Theta)$.

Proof: By induction on the structure of Θ.

Case $\Theta = \varnothing_m$:

$$\begin{aligned}
(\Gamma,\Delta) - \varnothing_m &= \Gamma,\Delta \;\; \text{(Subtraction (4.1))} \\
&= \Gamma,(\Delta - \varnothing_m) \;\; \text{(Subtraction (4.1))}
\end{aligned}$$

Case $\overset{;}{\Theta}$: Let $\Delta' = (\Delta - \overset{;}{\Theta})$

$$\begin{aligned}
(\Gamma, \Delta) - \overset{;}{\Theta} &= (\Gamma - \overset{;}{\Theta}), \Delta \quad \text{(Lemma 4.2.10)} \\
&= (MNW_{\Theta}(\Gamma) - \overset{;}{\Theta}), \Delta \quad \text{(Definition 4.2.6)} \\
&= (MNW_{\Theta}(MNW_{\Delta}(\Gamma)) - \overset{;}{\Theta}), \Delta \quad \text{(Lemma 4.2.5)} \\
&= \Gamma', \Delta', \Delta \quad \text{(Definitions)} \\
&= \Gamma', \Delta, \Delta' \quad \text{(Commutativity)} \\
&\sqsubseteq \Gamma, \Delta' \quad \text{(Weakenings)} \\
&= \Gamma, (\Delta - \Theta) \quad \text{(Definition)}
\end{aligned}$$

Case $\overset{,}{\Theta} = \overset{;}{\Theta}_1, \cdots, \overset{;}{\Theta}_n$: The condition for $\Theta \subseteq \Delta$ gives us that $\forall\ 1 \leq i \leq n.\overset{;}{\Theta}_i \in \Delta$. Then $\forall\ 1 \leq i \leq n.\overset{;}{\Theta}_i \notin \Gamma$. Then

$$\begin{aligned}
(\Gamma, \Delta) - \Theta &= (\Gamma, \Delta) - (\overset{;}{\Theta}_1, \cdots, \overset{;}{\Theta}_n) \quad \text{(Case)} \\
&= ((\Gamma, \Delta) - \overset{;}{\Theta}_1) - (\Theta - \overset{;}{\Theta}_1) \quad \text{(Subtract)} \\
&\sqsubseteq (\Gamma, (\Delta - \overset{;}{\Theta}_1)) - (\Theta - \overset{;}{\Theta}_1) \quad \text{(Case } \overset{;}{\Theta}\text{)} \\
&= \Gamma, ((\Delta - \overset{;}{\Theta}_1) - (\Theta - \overset{;}{\Theta}_1)) \quad \text{(Induction Hypothesis)} \\
&= \Gamma, (\Delta - (\overset{;}{\Theta}_1, \cdots, \overset{;}{\Theta}_n)) \quad \text{(Subtract)} \\
&= \Gamma, (\Delta - \Theta) \quad \text{(Case)}
\end{aligned}$$

□

Lemma 4.2.12 *Given* $\Theta \subseteq \Delta \subseteq \Gamma$,

$$(\Gamma - \Theta) - (\Delta - \Theta) = \Gamma - \Delta$$

Proof:

$$\begin{aligned}
(\Gamma - \Theta) - (\Delta - \Theta) &= \Gamma - (\Theta, (\Delta - \Theta)) \quad \text{(Lemma 4.2.9)} \\
&= \Gamma - ((\Theta, \Delta) - \Theta)) \quad \text{(Lemma 4.2.11)} \\
&= \Gamma - \Delta \quad \text{(Lemma 4.2.8)}
\end{aligned}$$

□

Lemma 4.2.13

$$(\Gamma(\alpha) - \Delta) - \alpha = (\Gamma(\alpha) - \alpha) - \Delta$$

Proof:

$$\begin{aligned}
(\Gamma(\alpha) - \Delta) - \alpha &= \Gamma(\alpha) - (\Delta, \alpha) \ \ \text{(Lemma 4.2.9)} \\
&= (\Gamma(\alpha) - \alpha) - ((\Delta, \alpha) - \alpha) \ \ \text{(Lemma 4.2.12)} \\
&= (\Gamma(\alpha) - \alpha) - (\Delta, (\alpha - \alpha)) \ \ \text{(Lemma 4.2.11)} \\
&= (\Gamma(\alpha) - \alpha) - \Delta \ \ \text{(Subtraction and unit operation)}
\end{aligned}$$

□

Lemma 4.2.14 *Given* $\Theta \subseteq \Gamma$ *and* $\Delta \subseteq (\Gamma - \Theta)$

$$((\Gamma - \Theta) - \Delta), \Theta = \Gamma - \Delta$$

Proof:

$$\begin{aligned}
((\Gamma - \Theta) - \Delta), \Theta &= ((\Gamma - \Theta), \Theta) - \Delta \ \ \text{(Lemma 4.2.10)} \\
&= ((\Gamma, \Theta) - \Theta) - \Delta \ \ \text{(Lemma 4.2.10)} \\
&= (\Gamma, (\Theta - \Theta)) - \Delta \ \ \text{(Lemma 4.2.11)} \\
&= (\Gamma, \varnothing_m) - \Delta \ \ \text{(Subtraction)} \\
&= \Gamma - \Delta \ \ \text{(Unit operation)}
\end{aligned}$$

□

Lemma 4.2.15 *If* $\Delta \subseteq \Gamma$*, then* $(\Gamma - \Delta), \Delta \sqsubseteq \Gamma$

Proof: By cases on Δ

$\Delta = \varnothing_m$: $(\Gamma - \varnothing_m), \varnothing_m = \Gamma$

$\Delta = \overset{;}{\Delta}$: According to Definition 4.2.6, $\Gamma - \overset{;}{\Delta} = \Gamma'$, where $\Gamma', \overset{;}{\Delta} = MNW_{\Delta}(\Gamma)$. Therefore $(\Gamma - \Delta), \Delta = MNW_{\Delta}(\Gamma)$ and since by the definition of *MNW* it is possible to reach Γ using a series of weakenings, then $(\Gamma - \Delta), \Delta \sqsubseteq \Gamma$

$\Delta = \overset{,}{\Delta}_1; \cdots ; \overset{,}{\Delta}_n$:

$$\begin{aligned} (\Gamma - \Delta), \Delta &= (\Gamma - (\overset{;}{\Delta}_1, \cdots, \overset{;}{\Delta}_n)), \overset{;}{\Delta}_1, \cdots, \overset{;}{\Delta}_n \text{ (Case)} \\ &= ((\Gamma - \overset{;}{\Delta}_1) - (\Delta - \overset{;}{\Delta}_1)), (\Delta - \overset{;}{\Delta}_1), \overset{;}{\Delta}_1 \text{ (Subtract)} \\ &\sqsubseteq (\Gamma - \overset{;}{\Delta}_1), \overset{;}{\Delta}_1 \text{ (Induction Hypothesis)} \\ &\sqsubseteq \Gamma \text{ (Case } \overset{;}{\Delta}) \end{aligned}$$

□

Lemma 4.2.16 *If* $\Theta \subseteq \Delta \subseteq \Gamma$*, then*

$$(\Gamma - \Delta), (\Delta - \Theta) \sqsubseteq \Gamma - \Theta$$

Proof:

$$\begin{aligned} (\Gamma - \Delta), (\Delta - \Theta) &= ((\Gamma - \Theta) - (\Delta - \Theta)), (\Delta - \Theta) \text{ (Lemma 4.2.8)} \\ &\sqsubseteq \Gamma - \Theta \text{ (Lemma 4.2.15)} \end{aligned}$$

□

4.3 Judgements for the operational semantics

We now introduce the judgements used to specify the operational semantics.

From the discussion in §1.4.4 we have concluded that there are unwanted interactions between the $*R$ rule and the rule of Weakening. The solution we found to this problem is based in the following idea: keep track of any additive extension of the program using a stack of boxes. Basically the idea is to use boxes to protect any additive part of the program that was added dynamically from unwanted weakenings, without compromising the soundness of the operational semantics.

In the example we used in §1.4.4, specially the first proposed solution which turned out to be unsound, the problem arised because of the equal status of two propositions, one of which was added dynamically through a $\to R$ rule. We will quickly revisit this example to motivate the use of boxes. Looking for a proof of $\tau; (\phi, \psi) \vdash (\chi \to (\psi \wedge \tau)) * \phi$ we need first to apply the rules $*R$ and $\to R$ and we reach the following unprovable sequent: $(\chi; \tau; (\phi, \psi)) \backslash \phi \vdash \psi \wedge \tau$. The crucial problem here is that χ and τ, despite having different origins, are living side by side, and the distinction between them is lost.

The solution we found is to put χ inside a box if one is available. This allows us to distinguish between χ and τ: one of them is inside a box, the other one is not. Note that if there was no box available, it would mean that there was no previous $*R$ rule, and the problem would not arise. Now it is possible to use Weakening without loss of information, because any clause that could still be needed in later stages can be recovered from the box.

Now we are ready to introduce the new box datatype which is defined as follows:

Definition 4.3.1 *A* box *can be*

- locked *We will use* ⊠ *as a notation for locked boxes*
- open *notated* $\boxed{\chi}$, *where* χ *is a bunch*
- full *which will be notated* $\{\Gamma, \chi\}$ *where* Γ *and* χ *are bunches*

In addition open boxes can be flagged with the theorem flag. The notation used for flagged open boxes will be $\boxed{\chi}^{\varepsilon}$ and sometimes we will use the notation $\boxed{\chi}^{-}$ to mean that it is unimportant whether the box is flagged or not. The meaning of the theorem flag is explained in detail below.

For the rest of the chapter we will work with pairs $\langle\Gamma|s\rangle$ where Γ is a bunch and s a stack (implemented as a list) of boxes.

The judgements used for the operational semantics will have the following form

$$\langle\Gamma|s\rangle_n \backslash \langle\Delta|s'\rangle_{n'} \vdash_o \phi$$

where Γ and Δ are bunches, s and s' are stacks of boxes, n and n' are integers and ϕ is a proposition. We will call the left pair ($\langle\Gamma|s\rangle$) the *upwards* pair, and the right pair ($\langle\Delta|s'\rangle$) the *downwards* pair. The meaning of the judgement is:

> Given a program Γ with a stack s as an input, after proving ϕ we are left with a program Δ and stack s' for the rest of the computation.

If Δ is ε (for εmpty flag) it means that it is known beforehand that Δ must be empty. If $n > 0$ then $\top$ has been found previously in a goal position. This means that

Γ actually refers to a subbunch of itself. If $n' > 0$ then $\top$ is in a goal position in ϕ and therefore Δ refers to a subbunch of itself. In these cases we are working with a logic which is locally affine, since any left over can be "pumped back" to where one of the $\top$ rules occurred.

Given a judgement, and using this interpretation, we sometimes are interested in finding out what are the resources necessary to prove ϕ. To do this we define an operation on these pairs which given two pairs will return a bunch. The operation is defined as follows

Definition 4.3.2 *The operation* pair subtraction *is defined by cases in the following way:*

1. *Remainders are not allowed and a full box is on the top of the downward stack:* $\langle\xi;\Gamma|\boxed{\xi}^{\varepsilon}{::}s\rangle\backslash\!\backslash\langle\varepsilon|\{\Delta,\xi\}{::}s\rangle = f(\xi);(\Gamma-\Delta)$, *where* $f(\xi)$ *is* ξ *if* ϕ *is an additive conjunction or implication, and is* $\top$ *otherwise.*

2. *Remainders are not allowed and there is something other than a full box on top of the downward stack:* $\langle\Gamma|s\rangle\backslash\!\backslash\langle\varepsilon|s\rangle = \Gamma \quad (= \Gamma - \emptyset_m)$.

3. *Finally, in case neither (1) nor (2):* $\langle\Gamma|s\rangle\backslash\!\backslash\langle\Delta|s\rangle = \Gamma - \Delta$.

It should be noted that the second case can be considered a special case of the third case if we consider ε as special notation for $\varnothing_m$. It was added to the definition because technically ε is not a bunch.

4.4 Operational semantics overview

At each occurrence of a $*R$ rule, a box is created. Initially, the box is locked, and is put at the top of the upward stack.

When an $\to R$ operator is used, the box is unlocked and the antecedent of the implication $\phi \to \psi$ is put into it: so we get a stack of the form $\boxed{\phi}^{\varepsilon}{::}s$. If the box is 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 $\boxed{\chi;\phi}^{\varepsilon}{::}s$. Also, the empty flag is reset, thereby allowing no remainder to pass except through the box. At this stage the open box carries the "theorem flag" ε which indicates that $\phi \to \psi$ is regarded as possibly an intuitionistic theorem, failure of which will be detected at a CutAxiom rule. This procedure is necessary because theorems behave like $\top$ with respect to multiplicative resources and the counters n and n' have to be set appropriately.

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

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, reducing $\Gamma(\alpha) \vdash \alpha$ to $\Gamma', \alpha \vdash \alpha$, 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 one of the Unit reductions, we can be more specific 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. If the top of the stack is an open box containing ξ, it is necessary to check whether α is in ξ. If not, then the theorem flag of the open box must be removed because it means that the $\phi \rightarrow \psi$ in a previous $\rightarrow R$ which created the open box is not a theorem.

$-\!\!*L$: 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'. At this point, we can try to prove 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. If a $\top$ goal or an intuitionistic theorem was found, the counter will be greater than zero and the computation will succeed even if ε is set and there is a non-empty remainder.

$\wedge R$s : There are five subcases. In the first two, 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.

The second case, $\wedge R^i$ takes care of the case in which $\top$ is found as a goal in both conjuncts. Of course, the $\wedge R$ rules do not propagate the presence of $\top$ from the left branch to the right branch. An increased counter should be passed on to the rest of the computation since any left over found later could have been given to the $\phi \wedge \psi$ subproof and weakened away at this point.

In the other three, handled by $\wedge R^{ii}$, $\wedge R^{iii}$ and $\wedge R^{iv}$, no remainder is permitted. Here we have three subcases, depending on whether the stack is full on return of the left, right or neither branch. An increased counter will be passed to the rest of the computation only if $\top$ is found in *both* left and right subproofs. An example which includes one of these cases follows after the description of the operators.

$\to R$s : There are four subcases. All four share the property that the antecedent *of the* *implication* leaves no remainder. Note that we assume that any occurrences of $\wedge$ or $*$ in the antecedents ϕ in $\phi \to \psi$ which are (inductively) principal connectives are immediately removed using operators corresponding to the operationally trivial $\wedge L$ or $*L$ rules. An example which illustrates $\to R$ follows after the description of the operators. The last rule is used when the implication is an intuitionistic theorem, when it should behave in similar ways to $\top$.

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

A worked example, as mentioned above, will clarify these complex constructions Thus we revisit the problematic example presented in Chapter 1 showing how the use of boxes manages the interaction between the multiplicatives and additives, specially $*R$, $\to R$ and Weakening. (here, “CA” denotes CutAxiom and the counters are 0 throughout):

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{\langle\phi|[\boxed{\chi}\,]\rangle\backslash\langle\varepsilon|[\{\phi,\chi\}]\rangle \vdash_o I}\,\text{Unit}^{ii}}{\langle\chi;(\phi,\psi)|[\,\boxed{\chi}^{\varepsilon}\,]\rangle\backslash\langle\varepsilon|[\{\phi,\chi\}]\rangle \vdash_o \psi}\,\text{CA}^{i} \qquad \dfrac{\dfrac{}{\langle\varnothing_m|[]\rangle\backslash\langle\varepsilon|[]\rangle \vdash_o I}\,\text{Unit}}{\langle\chi;\psi|[]\rangle\backslash\langle\varepsilon|[]\rangle \vdash_o \chi}\,\text{CA}}{\langle\chi;(\phi,\psi)|[\,\boxed{\chi}^{\varepsilon}\,]\rangle\backslash\langle\varepsilon|[\{\phi,\chi\}]\rangle \vdash_o \psi\wedge\chi}\,\wedge R^{iv}}{\langle(\phi,\psi)|[\boxtimes]\rangle\backslash\langle\phi|[\boxtimes]\rangle \vdash_o \chi \to (\psi\wedge\chi)}\,\to R^{i} \qquad \dfrac{\dfrac{}{\langle\varnothing_m|[]\rangle\backslash\langle\varepsilon|[]\rangle \vdash_o I}\,\text{Unit}}{\langle\phi|[]\rangle\backslash\langle\varepsilon|[]\rangle \vdash_o \phi}\,\text{CA}}{\langle(\phi,\psi)|[]\rangle\backslash\langle\varepsilon|[]\rangle \vdash_o (\chi\to(\psi\wedge\chi)) * \phi}\,*R$$

We are looking for a proof of $\phi,\psi \vdash ((\chi \to (\psi\wedge\chi)) * \phi$. So we set up the initial lists as empty, and we explicitly say that we require no left-overs from the computation. The operational version of the sequent is, then,

$$\langle(\phi,\psi)|[]\rangle\backslash\langle\varepsilon|[]\rangle \vdash_o (\chi \to (\psi\wedge\chi)) * \phi$$

We know from the previous chapter that if there is a proof, then there is a uniform proof. Therefore the first step is to apply the $*R$ rule. Then the left branch has the following shape:

$$\langle(\phi,\psi)|[\boxtimes]\rangle\backslash\langle\Delta|[\boxtimes]\rangle \vdash_o \chi \to (\psi\wedge\chi)$$

This rule put a locked box in the stack and carries on with the proof of the left branch. The value of Δ has to be determined later.

Since the goal is not yet atomic, we apply $\to R^i$. The other variants of $\to R$ cannot be applied: $\to R$ needs an empty stack; $\to R^{ii}$ needs an open box as the top element of the stack. The distinction between $\rightarrow R^{i}$ and $\rightarrow R^{iii}$ is irrelevant at this stage (they will be distinguished by the content of the stack on return).

$$\langle \chi;(\phi,\psi) | [\, \boxed{\chi}^{\varepsilon}] \rangle \backslash \langle \varepsilon | [??] \rangle \ \vdash_o \psi \wedge \chi$$

Looking at the goal, we see that we need to apply one of the $\wedge R$ rules. From the five subrules, three do not apply, because we have an ε as a condition. Again, the distinction between rules $\rightarrow R^{iii}$ and $\rightarrow R^{iv}$ will be made later on inspection of the stack.

We continue the computation with the left branch, and then arrive to this sequent:

$$\langle \chi;(\phi,\psi) | [\, \boxed{\chi}^{\varepsilon}] \rangle \backslash \langle \varepsilon | [??] \rangle \ \vdash_o \psi$$

At this point, the goal being atomic, we need to apply a left rule. In this case it can be seen that the rule CutAxiom^{i} is needed, because ψ is a subbunch of (ϕ,ψ). In this way we show that $\chi \rightarrow (\psi \wedge \chi)$ is not a theorem in intuitionistic logic, and therefore we don't need to increase the counter.

Then we need to find a proof of the following sequent

$$\langle \phi | [\, \boxed{\chi}\,] \rangle \backslash \langle \varepsilon | [??] \rangle \ \vdash_o I$$

Since the counter is zero, and the ε is present, the only Unit rule available is Unit^{ii}. Applying this rule, we find the value of [??]

$$\langle \phi | [\, \boxed{\chi}\,] \rangle \backslash \langle \varepsilon | [\{\phi,\chi\}] \rangle \ \vdash_o I$$

and start coming back, via the success continuation, to the branches which were left unresolved, namely the right subproof of $\wedge R$ first, and the right subproof of $*R$ afterwards.

Since there is a full box at the top of the stack, we know at this point that the rule we need to use is $\wedge R^{iv}$. We have to perform the appropriate subtraction operation to work out what is the corresponding program. Doing this we find that $\chi;((\phi,\psi) - \phi) = \chi;\psi$ and then we have as right subproof

$$\langle \chi;\psi | [] \rangle \backslash \langle \varepsilon | [] \rangle \ \vdash_o \chi$$

Now there is a straightforward application of a CutAxiom rule and a Unit rule, and this solves this subproof.

On return to the $\rightarrow R$ rule, we find out that it was $\rightarrow R^i$ that was required. This rule "unpacks" the content of the full box, sending the correct residue to the rest of the computation. It is at this point that we discover that $\Delta = \phi$ and therefore the right branch of the $*R$ rule is

$$\langle \phi | [] \rangle \backslash \langle \varepsilon | [] \rangle \vdash_o \phi$$

Again there are easy applications of the rules CutAxiom and Unit which finalise the proof.

4.5 Soundness and Completeness

In this section we first prove that the operational semantics shown in Table 4.1 is sound. This means that for each successful computation there is a corresponding proof in **LBI**.

To show completeness, however, it is better to prove first Lemma 4.5.2. Completeness then follows immediately.

Theorem 4.5.1 Soundness: If $\langle \Gamma | s \rangle_n \backslash \langle \Delta | s' \rangle_{n'} \vdash_o \phi$ then $\langle \Gamma | s \rangle \backslash\backslash \langle \Delta | s' \rangle \vdash \phi$

Proof: By induction on the structure of the proof. For the remainder of this proof we will consider the value of n to be 0, and exceptions and special cases will be treated explicitly. This affects the proof only in the following way: instead of talking about Γ in each case, when $n > 0$ we are talking about a subbunch of Γ.

In each case we present the proposed rule for the operational semantics, and we show that there is a corresponding proof in **BI**.

The proofs use Definition 4.3.2 of pair subtraction on page 70.

$\boxed{\top\text{Unit}}$

$$\frac{}{\langle \Gamma | s \rangle_n \backslash \langle \Gamma | s \rangle_{n+1} \vdash_o \top} \top\text{Unit}$$

When we find $\top$ as a goal we should succeed immediately, because *any* Γ will prove $\top$. However, if the rest of the computation leaves a remainder, this remainder could have been used in the proof of $\top$ getting rid of it in the first place. Because it is not possible to know in advance which subbunch should

$$\frac{}{\langle\Gamma|s\rangle_n\backslash\langle\Gamma|s\rangle_{n+1}\vdash_o\top}\;\top\text{Unit}\qquad\frac{}{\langle\Gamma|s\rangle_0\backslash\langle\Gamma|s\rangle_0\vdash_o I}\;\text{Unit}$$

$$\frac{}{\langle\Gamma|s\rangle_n\backslash\langle\boldsymbol{\varepsilon}|s\rangle_0\vdash_o I}\;\text{Unit}^i(n>0)\qquad\frac{}{\langle\phi;\Gamma|\boxed{\phi}^{\varepsilon}::s\rangle_n\backslash\langle\boldsymbol{\varepsilon}|\{\Gamma,\phi\}::s\rangle_n\vdash_o I}\;\text{Unit}^{ii}$$

$$\frac{\langle\Gamma'|s\rangle_n\backslash\langle\Delta|s'\rangle_{n'}\vdash_o I}{\langle\Gamma(\alpha)|s\rangle_n\backslash\langle\Delta|s'\rangle_{n'}\vdash_o\alpha}\;\text{CutAxiom}^{\dagger}\qquad\frac{\langle\Gamma'|\boxed{\xi}::s\rangle_n\backslash\langle\Delta|s'\rangle_{n'}\vdash_o I}{\langle\xi;\Gamma(\alpha)|\boxed{\xi}^{\varepsilon}::s\rangle_n\backslash\langle\Delta|s'\rangle_{n'}\vdash_o\alpha}\;\text{CutAxiom}^{i\dagger}$$

$$\frac{\langle\Gamma'|s\rangle_n\backslash\langle\Delta|s'\rangle_{n'}\vdash_o\phi}{\langle\Gamma(\phi\mathrel{-\!\!*}\alpha)|s\rangle_n\backslash\langle\Delta|s'\rangle_{n'}\vdash_o\alpha}\mathrel{-\!\!*}L^{\dagger}\qquad\frac{\langle\Theta;\phi\to\alpha|[]\rangle_0\backslash\langle\boldsymbol{\varepsilon}|[]\rangle_m\vdash_o\phi\qquad\langle\Gamma'|s\rangle_n\backslash\langle\Delta|s'\rangle_{n'}\vdash_o I}{\langle\Gamma(\Theta;\phi\to\alpha)|s\rangle_n\backslash\langle\Delta|s'\rangle_{n'}\vdash_o\alpha}\to L^{\dagger}$$

$$\frac{\langle\Gamma|\boxtimes::s\rangle_n\backslash\langle\Gamma'|\boxtimes::s\rangle_{n'}\vdash_o\phi\quad\langle\Gamma'|s\rangle_{n'}\backslash\langle\Delta|s'\rangle_{n''}\vdash_o\psi}{\langle\Gamma|s\rangle_n\backslash\langle\Delta|s'\rangle_{n''}\vdash_o\phi*\psi}\;*R\qquad\frac{\langle\phi,\Gamma|s\rangle_n\backslash\langle\Delta|s'\rangle_{n'}\vdash_o\psi}{\langle\Gamma|s\rangle_n\backslash\langle\Delta'|s'\rangle_{n'}\vdash_o\phi\mathrel{-\!\!*}\psi}\mathrel{-\!\!*}R^{\ddagger}$$

$$\frac{\langle\Gamma|s\rangle_n\backslash\langle\Delta|s\rangle_n\vdash_o\phi\quad\langle\Gamma-\Delta|[]\rangle_0\backslash\langle\boldsymbol{\varepsilon}|[]\rangle_m\vdash_o\psi}{\langle\Gamma|s\rangle_n\backslash\langle\Delta|s\rangle_n\vdash_o\phi\wedge\psi}\;\wedge R$$

$$\frac{\langle\Gamma|s\rangle_n\backslash\langle\Delta|s\rangle_{n'}\vdash_o\phi\quad\langle\Gamma|s\rangle_n\backslash\langle\Delta'|s\rangle_n\vdash_o\psi}{\langle\Gamma|s\rangle_n\backslash\langle\Delta'|s\rangle_n\vdash_o\phi\wedge\psi}\;\wedge R^i(n'>n\text{ and }\Delta'\subseteq\Delta)$$

$$\frac{\langle\Gamma|s\rangle_n\backslash\langle\Delta|s\rangle_{n'}\vdash_o\phi\quad\langle\Gamma|s\rangle_n\backslash\langle\Delta'|s\rangle_{n''}\vdash_o\psi}{\langle\Gamma|s\rangle_n\backslash\langle\Delta\cap\Delta'|s\rangle_{n'}\vdash_o\phi\wedge\psi}\;\wedge R^{ii}(n',n''>n)$$

$$\frac{\langle\Gamma|s\rangle_n\backslash\langle\boldsymbol{\varepsilon}|s\rangle_{n'}\vdash_o\phi\quad\langle\Gamma|s\rangle_n\backslash\langle\boldsymbol{\varepsilon}|s'\rangle_{n''}\vdash_o\psi}{\langle\Gamma|s\rangle_n\backslash\langle\boldsymbol{\varepsilon}|s'\rangle_{n'''}\vdash_o\phi\wedge\psi}\;\wedge R^{iii\sharp}$$

$$\frac{\langle\xi;\Gamma|\boxed{\xi}^{\varepsilon}::s\rangle_n\backslash\langle\boldsymbol{\varepsilon}|\{\Delta,\xi\}::s\rangle_{n'}\vdash_o\phi\quad\langle\xi;(\Gamma-\Delta)|[]\rangle_0\backslash\langle\boldsymbol{\varepsilon}|[]\rangle_{n''}\vdash_o\psi}{\langle\xi;\Gamma|\boxed{\xi}^{\varepsilon}::s\rangle_n\backslash\langle\boldsymbol{\varepsilon}|\{\Delta,\xi\}::s\rangle_{n'''}\vdash_o\phi\wedge\psi}\;\wedge R^{iv\sharp}$$

$$\frac{\langle\phi;\Gamma|[]\rangle_n\backslash\langle\boldsymbol{\varepsilon}|[]\rangle_{n'}\vdash_o\psi}{\langle\Gamma|[]\rangle_n\backslash\langle\boldsymbol{\varepsilon}|[]\rangle_{n'}\vdash_o\phi\to\psi}\to R\qquad\frac{\langle\phi;\Gamma|\boxed{\phi}^{\varepsilon}::s\rangle_n\backslash\langle\boldsymbol{\varepsilon}|\{\Delta,\phi\}::s\rangle_{n'}\vdash_o\psi}{\langle\Gamma|\boxtimes::s\rangle_n\backslash\langle\Delta|\boxtimes::s\rangle_{n'}\vdash_o\phi\to\psi}\to R^i$$

$$\frac{\langle\phi;\xi;\Gamma|\boxed{\xi;\phi}^{-}::s\rangle_n\backslash\langle\boldsymbol{\varepsilon}|\{\Delta,(\xi;\phi)\}::s\rangle_{n'}\vdash_o\psi}{\langle\xi;\Gamma|\boxed{\xi}^{-}::s\rangle_n\backslash\langle\boldsymbol{\varepsilon}|\{\Delta,\xi\}::s\rangle_{n'}\vdash_o\phi\to\psi}\to R^{ii\natural}\qquad\frac{\langle\phi;\Gamma|\boxed{\phi}^{\varepsilon}::s\rangle_n\backslash\langle\boldsymbol{\varepsilon}|\boxed{\phi}^{\varepsilon}::s\rangle_{n'}\vdash_o\psi}{\langle\Gamma|\boxtimes::s\rangle_n\backslash\langle\Gamma|\boxtimes::s\rangle_{n+1}\vdash_o\phi\to\psi}\to R^{iii}$$

$$\frac{\langle\Gamma|s\rangle_n\backslash\langle\Delta|s'\rangle_{n'}\vdash_o\phi}{\langle\Gamma|s\rangle_n\backslash\langle\Delta|s'\rangle_{n'}\vdash_o\phi\vee\psi}\;\vee R\qquad\frac{\langle\Gamma|s\rangle_n\backslash\langle\Delta|s'\rangle_{n'}\vdash_o\psi}{\langle\Gamma|s\rangle_n\backslash\langle\Delta|s'\rangle_{n'}\vdash_o\phi\vee\psi}\;\vee R$$

† Γ' is obtained uniquely by performing the minimal number of Weakenings required to bring α, $\phi\mathrel{-\!\!*}\alpha$ or $(\Theta;\phi\to\alpha)$ to top-level.

‡ If $n'>n$ then $\Delta'=\Delta-\phi$, else $\Delta'=\Delta$.

♯ If $n',n''>n$, then $n'''=n+1$, else $n'''=n$.

♮ Here $\boxed{}^{-}::s$ indicates that here the theorem flag may be present or not.

Table 4.1: Operational Semantics of BLP

be used in this manner, the standard way of dealing with this without incurring in unacceptable performance deterioration by use of backtracking, is setting a flag marking the fact that remainders are allowed and can be "pumped back" to the earlier occurrence of $\top$Unit. The need for a counter instead of a flag is discussed in detail when considering the $-\!\!*R$ rule.

Given $\langle\Gamma|s\rangle_n\backslash\langle\Gamma|s\rangle_{n+1}\vdash_o\top$, and $\Delta\subseteq\Gamma$ we have to show that $\langle\Gamma|s\rangle\backslash\!\!\backslash\langle\Delta|s\rangle\vdash\top$ that is $\Gamma-\Delta\vdash\top$. This is always the case in **BI** using the $\top R$ rule.

$\boxed{\text{Unit}}$

$$\frac{}{\langle\Gamma|s\rangle_0\backslash\langle\Gamma|s\rangle_0\vdash_o I}\ \text{Unit}$$

Since the flag is 0 we know that $\top$ was not found up to this point. We need to show that $\langle\Gamma|s\rangle\backslash\!\!\backslash\langle\Gamma|s\rangle\vdash I$.

$$\dfrac{\dfrac{\dfrac{}{\varnothing_m\vdash I}\ IR}{\Gamma-\Gamma\vdash I}\ \text{Subtraction}}{\langle\Gamma|s\rangle\backslash\!\!\backslash\langle\Gamma|s\rangle\vdash I}\ \text{Definition 4.3.2}$$

$\boxed{\text{Unit}^i}$

$$\frac{}{\langle\Gamma|s\rangle_n\backslash\langle\varepsilon|s\rangle_0\vdash_o I}\ \text{Unit}^i(n>0)$$

We have $\langle\Gamma|s\rangle_n\backslash\langle\varepsilon|s\rangle_0\vdash_o I$. The presence of ε indicates that no remainder is allowed to be passed on. But since $n>0$ we are concerned with a subbunch of Γ, call it Δ, such that $\langle\Delta|s\rangle\backslash\!\!\backslash\langle\varepsilon|s\rangle\vdash I$. $\varnothing_m$ is such a Δ. It is at this point that we discover that any proof that we are trying to find should have used Γ in the $\top R$ rule (the fact that the counter is greater than 0 indicates that there was effectively such a rule earlier in the computation). The counter should be reset, since we are making use of the $\top R$ rule at this point.

$\boxed{\text{Unit}^{ii}}$

$$\frac{}{\langle\phi;\Gamma|\boxed{\phi}^{\varepsilon}::s\rangle_n\backslash\langle\varepsilon|\{\Gamma,\phi\}::s\rangle_n\vdash_o I}\ \text{Unit}^{ii}$$

We have to show that $\langle\phi;\Gamma|\boxed{\phi}^{\varepsilon}::s\rangle\backslash\backslash\langle\varepsilon|\{\Gamma,\phi\}::s\rangle\vdash I$. Using Definition 4.3.2, and given that in this case $f(\phi)=\top$, $\langle\Gamma\|\boxed{\phi}^{\varepsilon}::s\rangle\backslash\backslash\langle\varepsilon|\{\Gamma,\phi\}::s\rangle = f(\phi);(\Gamma-\Gamma)=\varnothing_m$, and we have again that $\dfrac{}{\varnothing_m\vdash I}$

$\boxed{\text{CutAxiom}}$

$$\frac{\langle\Gamma'|s\rangle_n\backslash\langle\Delta|s'\rangle_{n'}\vdash_o I}{\langle\Gamma(\alpha)|s\rangle_n\backslash\langle\Delta|s'\rangle_{n'}\vdash_o \alpha}\,\text{CutAxiom}$$

Here $\Gamma',\alpha = MNW_\alpha(\Gamma(\alpha))$, or equivalently $\Gamma'=\Gamma(\alpha)-\alpha$.

We assume that $\langle\Gamma'|s\rangle\backslash\backslash\langle\Delta|s'\rangle\vdash I$ and we need to prove that $\langle\Gamma(\alpha)|s\rangle\backslash\backslash\langle\Delta|s'\rangle\vdash\alpha$.

There are many cases on how I can be proved. It can be done using one of the Unit rules, or it can be done by I being considered a normal atom (for example in $a, a{-\!\!*}I\vdash I$) via one of CutAxiom, $\to L$ or ${-\!\!*}L$. We are not concerned with this distinction, though. All cases of the induction hypothesis can be represented by the sequent $\Gamma'-\Delta\vdash I$.

$$\begin{array}{cl}
\Gamma'-\Delta\vdash I & \\ \cline{1-1}
(\Gamma(\alpha)-\alpha)-\Delta\vdash I & \\ \cline{1-1}
((\Gamma(\alpha)-\alpha)-\Delta),\alpha\vdash\alpha & \text{CutAxiom} \\ \cline{1-1}
\Gamma(\alpha)-\Delta\vdash\alpha & \text{Lemma 4.2.14} \\ \cline{1-1}
\langle\Gamma(\alpha)|s\rangle\backslash\backslash\langle\Delta|s'\rangle\vdash\alpha & \text{Definition 4.3.2}
\end{array}$$

$\boxed{\text{CutAxiom}^i}$

$$\frac{\langle\Gamma'\|\boxed{\xi}::s\rangle_n\backslash\langle\Delta|s'\rangle_{n'}\vdash_o I}{\langle\xi;\Gamma(\alpha)\|\boxed{\xi}^{\varepsilon}::s\rangle_n\backslash\langle\Delta|s'\rangle_{n'}\vdash_o \alpha}\,\text{CutAxiom}^i$$

If the top of the stack is an open box, and α cannot be proved from ξ, then we know that the $\to R$ rule that opened the box is not a theorem, and the theorem flag should be removed. Otherwise this rule behaves like CutAxiom. The induction hypothesis is $\langle\Gamma'\|\boxed{\xi}::s\rangle\backslash\backslash\langle\Delta|s'\rangle\vdash I$. We need to prove $\langle\xi;\Gamma(\alpha)\|\boxed{\xi}^{\varepsilon}::s\rangle\backslash\backslash\langle\Delta|s'\rangle\vdash\alpha$. From Definition 4.3.2 we know that $\langle\Gamma'\|\boxed{\xi}::s\rangle\backslash\backslash\langle\Delta|s'\rangle=\Gamma'-\Delta$ and then

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\Gamma' - \Delta \vdash I}{(\Gamma(\alpha) - \alpha) - \Delta \vdash I}}{((\Gamma(\alpha) - \alpha) - \Delta), \alpha \vdash \alpha}\text{ CutAxiom}}{\Gamma(\alpha) - \Delta \vdash \alpha}\text{ Lemma 4.2.14}}{(\xi;\Gamma(\alpha)) - \Delta \vdash \alpha}\text{ Lemma 4.2.7}}{\langle \xi;\Gamma(\alpha)|\boxed{\xi}::s\rangle\backslash\backslash\langle\Delta|s'\rangle \vdash \alpha}\text{ Definition 4.3.2}$$

$\boxed{\mathrel{-\!\!*}L}$

$$\dfrac{\langle\Gamma'|s\rangle_n\backslash\langle\Delta|s'\rangle_{n'} \vdash_o \phi}{\langle\Gamma(\phi\mathrel{-\!\!*}\alpha)|s\rangle_n\backslash\langle\Delta|s'\rangle_{n'} \vdash_o \alpha}\mathrel{-\!\!*}L$$

The assumption is $\langle\Gamma'|s\rangle\backslash\backslash\langle\Delta|s'\rangle \vdash \phi$, where $\Gamma' = \Gamma(\phi\mathrel{-\!\!*}\alpha) - \phi\mathrel{-\!\!*}\alpha$. We need to show that $\langle\Gamma(\phi\mathrel{-\!\!*}\alpha)|s\rangle\backslash\backslash\langle\Delta|s'\rangle \vdash \alpha$.

$$\dfrac{\begin{array}{c}\dfrac{(\Gamma(\phi\mathrel{-\!\!*}\alpha) - \phi\mathrel{-\!\!*}\alpha) - \Delta \vdash \phi}{((\Gamma(\phi\mathrel{-\!\!*}\alpha) - \phi\mathrel{-\!\!*}\alpha) - \Delta), \phi\mathrel{-\!\!*}\alpha \vdash \alpha}\mathrel{-\!\!*}L \\ \hline\hline \Gamma(\phi\mathrel{-\!\!*}\alpha) - \Delta \vdash \alpha\end{array}\text{ Lemma}\alpha\text{ 4.2.14}}{\langle\Gamma(\phi\mathrel{-\!\!*}\alpha)|s\rangle\backslash\backslash\langle\Delta|s'\rangle \vdash \alpha}\text{ Definition 4.3.2}$$

$\boxed{\rightarrow L}$

$$\dfrac{\langle\Theta;\phi \rightarrow \alpha|[]\rangle_0\backslash\langle\varepsilon|[]\rangle_m \vdash_o \phi \qquad \langle\Gamma'|s\rangle_n\backslash\langle\Delta|s'\rangle_{n'} \vdash_o I}{\langle\Gamma(\Theta;\phi \rightarrow \alpha)|s\rangle_n\backslash\langle\Delta|s'\rangle_{n'} \vdash_o \alpha}\rightarrow L$$

Here $\Gamma' = \Gamma(\Theta;\phi \rightarrow \alpha) - (\Theta;\phi \rightarrow \alpha)$.

The induction hypotheses are $\langle\Theta;\phi \rightarrow \alpha|[]\rangle\backslash\backslash\langle\varepsilon|[]\rangle \vdash \phi$ and $\langle\Gamma'|s\rangle\backslash\backslash\langle\Delta|s'\rangle \vdash I$. We need to show that $\langle\Gamma(\Theta;\phi \rightarrow \alpha)|s\rangle\backslash\backslash\langle\Delta|s'\rangle \vdash \alpha$.

First notice that $\langle\Theta;\phi \rightarrow \alpha|[]\rangle\backslash\backslash\langle\varepsilon|[]\rangle = \Theta;\phi \rightarrow \alpha$ and that $\langle\Gamma'|s\rangle\backslash\backslash\langle\Delta|s'\rangle = \Gamma' - \Delta$. For convenience we will use the abbreviation Θ^+ to stand for $\Theta;\phi \rightarrow \alpha$.

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\Theta^+ \vdash \phi \qquad \Gamma' - \Delta \vdash I}{(\Gamma' - \Delta), \Theta^+ \vdash \alpha}\rightarrow L}{(\Gamma', \Theta^+) - \Delta \vdash \alpha}\text{ Lemma 4.2.10}}{((\Gamma(\Theta^+) - \Theta^+), \Theta^+) - \Delta \vdash \alpha}\text{ Def. of }\Gamma'}{((\Gamma(\Theta^+), (\Theta^+ - \Theta^+)) - \Delta \vdash \alpha}\text{ Lemma 4.2.11}}{\Gamma(\Theta^+) - \Delta \vdash \alpha}\text{ Subtraction and Unit operation}}{\langle\Gamma(\Theta^+)|s\rangle\backslash\backslash\langle\Delta|s'\rangle \vdash \alpha}\text{ Definition 4.3.2}$$

$\boxed{*R}$ Suppose that

$$\dfrac{\langle\Gamma|\boxtimes{::}s\rangle_n\backslash\langle\Gamma'|\boxtimes{::}s\rangle_{n'}\vdash_o\phi \quad \langle\Gamma'|s\rangle_{n'}\backslash\langle\Delta|s'\rangle_{n''}\vdash_o\psi}{\langle\Gamma|s\rangle_n\backslash\langle\Delta|s'\rangle_{n''}\vdash_o\phi*\psi}\,*R$$

As induction hypothesis we have that $\langle\Gamma|\boxtimes{::}s\rangle\backslash\backslash\langle\Gamma'|\boxtimes{::}s\rangle\vdash\phi$ and $\langle\Gamma'|s\rangle\backslash\backslash\langle\Delta|s'\rangle\vdash\psi$. We need to show that $\langle\Gamma|s\rangle\backslash\backslash\langle\Delta|s'\rangle\vdash\phi*\psi$.

From Definition 4.3.2 we get that $\langle\Gamma|\boxtimes{::}s\rangle\backslash\backslash\langle\Gamma'|\boxtimes{::}s\rangle=\Gamma-\Gamma'$.

We note that ε behaves like $\varnothing_m$ with respect to the subtraction operation. Thus we can pack the different cases into one in which $\langle\Gamma'|s\rangle\backslash\backslash\langle\Delta|s'\rangle=\Gamma'-\Delta$ Then we use the following figure.

$$\dfrac{\dfrac{\Gamma-\Gamma'\vdash\phi \quad \Gamma'-\Delta\vdash\psi}{(\Gamma-\Gamma'),(\Gamma'-\Delta)\vdash\phi*\psi}\,*R}{\Gamma-\Delta\vdash\phi*\psi}\,\text{Weakening}^*\text{(Lemma 4.2.16)}$$

$\boxed{-\!\!*R}$

$$\dfrac{\langle\phi,\Gamma|s\rangle_n\backslash\langle\Delta|s\rangle_{n'}\vdash_o\psi}{\langle\Gamma|s\rangle_n\backslash\langle\Delta'|s\rangle_{n'}\vdash_o\phi-\!\!*\psi}\,-\!\!*R$$

Here Δ needs to be a subbunch of ϕ,Γ and Δ' needs to be a subbunch of Γ. This makes a difference if some residue of ϕ remains in Γ. If $n'>n$ it means that $\top$ was found as a goal while trying to prove ψ and then it is permissible to get rid of this part, effectively performing weakening of any left over from ϕ. It is not allowed to do that otherwise, even if $n>0$.

Once we have clarified this, we revert to the treatment of the rule as if $n=0$.

As induction hypothesis we have that $\langle(\phi,\Gamma)|s\rangle\backslash\backslash\langle\Delta|s\rangle\vdash\psi$. We need to show that $\langle\Gamma|s\rangle\backslash\backslash\langle\Delta|s\rangle\vdash\phi-\!\!*\psi$

Using Definition 4.3.2 we get that $\langle\phi,\Gamma'|s\rangle\backslash\backslash\langle\Delta|s\rangle=(\phi,\Gamma')-\Delta$, so

$$\dfrac{\dfrac{\dfrac{(\phi,\Gamma)-\Delta\vdash\psi}{(\Gamma-\Delta),\phi\vdash\psi}\,\text{Lemma 4.2.10}}{\Gamma-\Delta\vdash\phi-\!\!*\psi}\,-\!\!*R}{\langle\Gamma|s\rangle\backslash\backslash\langle\Delta|s\rangle\vdash\phi-\!\!*\psi}\,\text{Definition 4.3.2}$$

$\boxed{\wedge R}$

$$\dfrac{\langle\Gamma|s\rangle_n\backslash\langle\Delta|s\rangle_n \vdash_o \phi \quad \langle\Gamma-\Delta|[]\rangle_0\backslash\langle\varepsilon|[]\rangle_m \vdash_o \psi}{\langle\Gamma|s\rangle_n\backslash\langle\Delta|s\rangle_n \vdash_o \phi\wedge\psi}\wedge R$$

Note that the counter n is unchanged in the left branch. So as inductive hypotheses we have $\langle\Gamma|s\rangle\backslash\backslash\langle\Delta|s\rangle \vdash \phi$ and $\langle\Gamma-\Delta|[]\rangle\backslash\backslash\langle\varepsilon|[]\rangle \vdash \psi$. We need to show that $\langle\Gamma|s\rangle\backslash\backslash\langle\Delta|s\rangle \vdash \phi\wedge\psi$

From Definition 4.3.2 we have that $\langle\Gamma|s\rangle\backslash\backslash\langle\Delta|s\rangle = \Gamma-\Delta$ and $\langle\Gamma-\Delta|[]\rangle\backslash\backslash\langle\varepsilon|[]\rangle = \Gamma-\Delta$. Then

$$\dfrac{\dfrac{\Gamma-\Delta\vdash\phi \quad \Gamma-\Delta\vdash\psi}{\Gamma-\Delta\vdash\phi\wedge\psi}\wedge R}{\langle\Gamma|s\rangle\backslash\backslash\langle\Delta|s\rangle \vdash \phi\wedge\psi}\text{Definition 4.3.2}$$

$\boxed{\wedge R^i}$

$$\dfrac{\langle\Gamma|s\rangle_n\backslash\langle\Delta|s\rangle_{n'} \vdash_o \phi \quad \langle\Gamma|s\rangle_n\backslash\langle\Delta'|s\rangle_n \vdash_o \psi}{\langle\Gamma|s\rangle_n\backslash\langle\Delta'|s\rangle_n \vdash_o \phi\wedge\psi}\wedge R^i(n'>n \text{ and } \Delta'\subseteq\Delta)$$

As induction hypotheses we have that there is a $\Delta''\subseteq\Delta$ such that $\langle\Gamma|s\rangle\backslash\backslash\langle\Delta''|s\rangle\vdash\phi$ and that $\langle\Gamma|s\rangle\backslash\backslash\langle\Delta'|s\rangle\vdash\psi$

The behaviour of the counters imply that $\top$ is found during the proof of ϕ and that $\top$ is not found during the proof of ψ. Since $\top$ was found in the left subproof, we are concerned not with Δ but with a subbunch of it. We can decide a posteriori which subbunch of Δ we need, and that will be done when we return from the proof of ψ.

We need to prove that $\langle\Gamma|s\rangle\backslash\backslash\langle\Delta'|s\rangle\vdash\phi\wedge\psi$

$$\dfrac{\dfrac{\Gamma-\Delta'\vdash\phi \quad \Gamma-\Delta'\vdash\psi}{\Gamma-\Delta'\vdash\phi\wedge\psi}\wedge R}{\langle\Gamma|s\rangle\backslash\backslash\langle\Delta'|s\rangle \vdash \phi\wedge\psi}\text{Definition 4.3.2}$$

$\boxed{\wedge R^{ii}}$

$$\dfrac{\langle\Gamma|s\rangle_n\backslash\langle\Delta|s\rangle_{n'} \vdash_o \phi \quad \langle\Gamma|s\rangle_n\backslash\langle\Delta'|s\rangle_{n''} \vdash_o \psi}{\langle\Gamma|s\rangle_n\backslash\langle\Delta\cap\Delta'|s\rangle_{n+1} \vdash_o \phi\wedge\psi}\wedge R^{ii}(n',n''>n)$$

The behaviour of the counters imply that $\top$ is found in both branches of the conjunction. Therefore as induction hypotheses we have that there are $\Delta'' \subseteq \Delta$ and $\Delta''' \subseteq \Delta'$ such that $\langle\Gamma|s\rangle\backslash\backslash\langle\Delta''|s\rangle \vdash \phi$ and that $\langle\Gamma|s\rangle\backslash\backslash\langle\Delta'''|s\rangle \vdash \psi$.

We need to prove that $\langle\Gamma|s\rangle\backslash\backslash\langle\Delta\cap\Delta'|s\rangle \vdash \phi\wedge\psi$. So, making $\Delta'' = \Delta''' = \Delta\cap\Delta'$ we can use the following figure

$$\dfrac{\dfrac{\Gamma-(\Delta\cap\Delta')\vdash\phi \quad \Gamma-(\Delta\cap\Delta')\vdash\psi}{\Gamma-(\Delta\cap\Delta')\vdash\phi*\psi}\,\wedge R}{\langle\Gamma|s\rangle\backslash\backslash\langle\Delta\cap\Delta'|s\rangle\vdash\phi\wedge\psi}\,\text{Definition 4.3.2}$$

Note that $\Delta\cap\Delta'$ is the biggest subbunch that can be passed on to the rest of the computation without compromising soundness.

Notice also that this case produce a higher level of non-determinism than the others, since backtracking will be on the right subproof, trying to find a new proof of ψ and leaving a possibly different Δ'. This non-determinism is caused by the behaviour of $\top$ in this operational semantics. When the left subproof is allowed to leave a remainder and also contains $\top$ in a goal position, there is no way to know in advance what are the resources necessary for the right subproof. For example, in the proof of $a,b,c,d \vdash ((\top*a)\wedge(\top*b))*d$, the subbunch b,c is fed back to the proof of the first $\top$ and a,c to the proof of the second. But there are also proofs of $a,b,c,d \vdash ((\top*a)\wedge(\top*b*c))*d$ and of $a,b,c,d \vdash ((\top*a)\wedge(\top*a*b))*d$, changing every time the resources used by the second $\top$. This situation can only be solved by backtracking on the right subproof.

$\boxed{\wedge R^{iii}}$

$$\dfrac{\langle\Gamma|s\rangle_n\backslash\langle\varepsilon|s\rangle_{n'}\vdash_o\phi \quad \langle\Gamma|s\rangle_n\backslash\langle\varepsilon|s'\rangle_{n''}\vdash_o\psi}{\langle\Gamma|s\rangle_n\backslash\langle\varepsilon|s'\rangle_{n'''}\vdash_o\phi\wedge\psi}\,\wedge R^{iii}$$

Induction hypotheses $\langle\Gamma|s\rangle\backslash\backslash\langle\varepsilon|s\rangle\vdash\phi$ and $\langle\Gamma|s\rangle\backslash\backslash\langle\varepsilon|s\rangle\vdash\psi$.

We need to prove that $\langle\Gamma|s\rangle\backslash\backslash\langle\varepsilon|s\rangle\vdash\phi\wedge\psi$. Since $\langle\Gamma|s\rangle\backslash\backslash\langle\varepsilon|s\rangle = \Gamma$ the proof reduces to the figure

$$\dfrac{\Gamma\vdash\phi \quad \Gamma\vdash\psi}{\Gamma\vdash\phi\wedge\psi}\,\wedge R$$

$\boxed{\wedge R^{iv}}$

$$\dfrac{\langle \xi;\Gamma|\boxed{\xi}^{\varepsilon}::s\rangle_n\backslash\langle\varepsilon|\{\Delta,\xi\}::s\rangle_{n'}\vdash_o\phi \quad \langle\xi;(\Gamma-\Delta)|[]\rangle_0\backslash\langle\varepsilon|[]\rangle_{n''}\vdash_o\psi}{\langle \xi;\Gamma|\boxed{\xi}^{\varepsilon}::s\rangle_n\backslash\langle\varepsilon|\{\Delta,\xi\}::s\rangle_{n'}\vdash_o\phi\wedge\psi}\wedge R^{iv}$$

Induction hypotheses: $\langle \xi;\Gamma|\boxed{\xi}^{\varepsilon}::s\rangle\backslash\backslash\langle\varepsilon|\{\Delta,\xi\}::s\rangle\vdash\phi$ and $\langle\xi;(\Gamma-\Delta)|[]\rangle\backslash\backslash\langle\varepsilon|[]\rangle\vdash\psi$. We need to prove that $\langle \xi;\Gamma|\boxed{\xi}^{\varepsilon}::s\rangle\backslash\backslash\langle\varepsilon|\{\Delta,\xi\}::s\rangle\vdash\phi\wedge\psi$.

First from Definition 4.3.2 we get that

$$\langle \xi;\Gamma|\boxed{\xi}^{\varepsilon}::s\rangle\backslash\backslash\langle\varepsilon|\{\Delta,\xi\}::s\rangle=\xi;(\Gamma-\Delta)=\langle\xi;(\Gamma-\Delta)|[]\rangle\backslash\backslash\langle\varepsilon|[]\rangle$$

Then we can use the following figure:

$$\dfrac{\dfrac{\xi;(\Gamma-\Delta)\vdash\phi \quad \xi;(\Gamma-\Delta)\vdash\psi}{\xi;(\Gamma-\Delta)\vdash\phi\wedge\psi}\wedge R}{\langle \xi;\Gamma|\boxed{\xi}^{\varepsilon}::s\rangle\backslash\backslash\langle\varepsilon|\{\Delta,\xi\}::s\rangle\vdash\phi\wedge\psi}$$

$\boxed{\to R}$

$$\dfrac{\langle\phi;\Gamma|[]\rangle_n\backslash\langle\varepsilon|[]\rangle_{n'}\vdash_o\psi}{\langle\Gamma|[]\rangle_n\backslash\langle\varepsilon|[]\rangle_{n'}\vdash_o\phi\to\psi}\to R$$

As induction hypothesis we have that $\langle\phi;\Gamma|[]\rangle\backslash\backslash\langle\varepsilon|[]\rangle\vdash\psi$ and we need to prove that $\langle\Gamma|[]\rangle\backslash\backslash\langle\varepsilon|[]\rangle\vdash\phi\to\psi$

Given that $\langle\phi;\Gamma|[]\rangle\backslash\backslash\langle\varepsilon|[]\rangle=\phi;\Gamma$ then we use the $\to R$ rule

$$\dfrac{\dfrac{\phi;\Gamma\vdash\psi}{\Gamma\vdash\phi\to\psi}\to R}{\langle\Gamma|[]\rangle\backslash\backslash\langle\varepsilon|[]\rangle\vdash\phi\to\psi}\text{Definition 4.3.2}$$

$\boxed{\to R^i}$

$$\dfrac{\langle\phi;\Gamma|\boxed{\phi}^{\varepsilon}::s\rangle_n\backslash\langle\varepsilon|\{\Delta,\phi\}::s\rangle_{n'}\vdash_o\psi}{\langle\Gamma|\boxtimes::s\rangle_n\backslash\langle\Delta|\boxtimes::s\rangle_{n'}\vdash_o\phi\to\psi}\to R^i$$

As induction hypothesis we have that $\langle\phi;\Gamma|\boxed{\phi}^{\varepsilon}::s\rangle\backslash\backslash\langle\varepsilon|\{\Delta,\phi\}::s\rangle\vdash\psi$ and we need to prove that $\langle\Gamma|\boxtimes::s\rangle\backslash\backslash\langle\Delta|\boxtimes::s\rangle\vdash\phi\to\psi$

We use the fact that $\langle\phi;\Gamma|\boxed{\phi}^{\varepsilon}::s\rangle\backslash\backslash\langle\varepsilon|\{\Delta,\phi\}::s\rangle=\phi;(\Gamma-\Delta)$ which gives

$$\dfrac{\phi;(\Gamma-\Delta)\vdash\psi}{\Gamma-\Delta\vdash\phi\to\psi}\to R$$

and $\Gamma-\Delta=\langle\Gamma|\boxtimes::s\rangle\backslash\backslash\langle\Delta|\boxtimes::s\rangle$

$\boxed{\to R^{ii}}$

$$\frac{\langle \phi;\xi;\Gamma|\boxed{\xi;\phi}^{-}{::}s\rangle_n\backslash\langle \varepsilon|\{\Delta,(\xi;\phi)\}{::}s\rangle_{n'} \vdash_o \psi}{\langle \xi;\Gamma|\boxed{\xi}^{-}{::}s\rangle_n\backslash\langle \varepsilon|\{\Delta,\xi\}{::}s\rangle_{n'} \vdash_o \phi \to \psi} \to R^{ii\natural}$$

The induction hypothesis in this case is $\langle \phi;\xi;\Gamma|\boxed{\xi;\phi}^{-}{::}s\rangle\backslash\!\backslash\langle \varepsilon|\{\Delta,(\xi;\phi)\}{::}s\rangle \vdash \psi$ and we need to prove that $\langle \xi;\Gamma|\boxed{\xi}^{-}{::}s\rangle\backslash\!\backslash\langle \varepsilon|\{\Delta,\xi\}{::}s\rangle \vdash \phi \to \psi$.

We use the fact that $\langle \phi;\xi;\Gamma|\boxed{\xi;\phi}^{-}{::}s\rangle\backslash\!\backslash\langle \varepsilon|\{\Delta,(\xi;\phi)\}{::}s\rangle = \phi;\xi;(\Gamma-\Delta)$ and then

$$\frac{\phi;\xi;(\Gamma-\Delta) \vdash \psi}{\xi;(\Gamma-\Delta) \vdash \phi \to \psi} \to R$$

and $\xi;(\Gamma-\Delta) = \langle \xi;\Gamma|\boxed{\xi}^{-}{::}s\rangle\backslash\!\backslash\langle \varepsilon|\{\Delta,\xi\}{::}s\rangle$

$\boxed{\to R^{iii}}$

$$\frac{\langle \phi;\Gamma|\boxed{\phi}^{\varepsilon}{::}s\rangle_n\backslash\langle \varepsilon|\boxed{\phi}^{\varepsilon}{::}s\rangle_{n'} \vdash_o \psi}{\langle \Gamma|\boxtimes{::}s\rangle_n\backslash\langle \Gamma|\boxtimes{::}s\rangle_{n+1} \vdash_o \phi \to \psi} \to R^{iii}$$

The induction hypothesis is that $\langle \phi;\Gamma|\boxed{\phi}^{\varepsilon}{::}s\rangle\backslash\!\backslash\langle \varepsilon|\boxed{\phi}^{\varepsilon}{::}s\rangle \vdash \psi$ and we need to prove that $\langle \Gamma|\boxtimes{::}s\rangle\backslash\!\backslash\langle \Gamma|\boxtimes{::}s\rangle \vdash \phi \to \psi$. There is extra information in the fact that since the open box returned with the “theorem” flag still on, then ϕ is not necessary to prove ψ. Then we can use the following figure

$$\frac{\phi;\xi;(\Gamma-\Delta) \vdash \psi}{\xi;(\Gamma-\Delta) \vdash \phi \to \psi} \to R$$

$\boxed{\vee R}$

$$\frac{\langle \Gamma|s\rangle_n\backslash\langle \Delta|s'\rangle_{n'} \vdash_o \phi}{\langle \Gamma|s\rangle_n\backslash\langle \Delta|s'\rangle_{n'} \vdash_o \phi \vee \psi} \vee R \qquad \frac{\langle \Gamma|s\rangle_n\backslash\langle \Delta|s'\rangle_{n'} \vdash_o \psi}{\langle \Gamma|s\rangle_n\backslash\langle \Delta|s'\rangle_{n'} \vdash_o \phi \vee \psi} \vee R$$

This case is trivial. The induction hypothesis is that $\langle \Gamma|s\rangle\backslash\!\backslash\langle \Delta|s'\rangle \vdash \phi$ (or ψ) and we need to prove that $\langle \Gamma|s\rangle\backslash\!\backslash\langle \Delta|s'\rangle \vdash \phi \vee \psi$. This is straightforward using $\vee R$

□

Lemma 4.5.2 (Completeness) *If*

$$\langle\Gamma, s\rangle\backslash\backslash\langle\Delta, s'\rangle \vdash \phi$$

then

$$\langle\Gamma, s\rangle_n\backslash\langle\Gamma', s'\rangle_{n'} \vdash_o \phi$$

Proof: We will use induction on the structure of the proof $\langle\Gamma, s\rangle\backslash\backslash\langle\Delta, s'\rangle \vdash \phi$. If necessary there will be cases depending on the values of n and n', or the shape of $\langle\Gamma, s\rangle\backslash\backslash\langle\Delta, s'\rangle$.

$\boxed{\top R}$

$$\frac{}{\langle\Gamma, s\rangle\backslash\backslash\langle\Delta, s'\rangle \vdash \top}\ \top R$$

Here Δ is the subbunch used by other parts of the proof to the right of the current subproof.

The bunch represented by $\langle\Gamma, s\rangle\backslash\backslash\langle\Delta, s'\rangle$ can be Γ itself (if $\Delta = \varnothing_m$ or $\Delta = \varepsilon$), $\varnothing_m$ (if $\Delta = \Gamma$) or $\Gamma' \subseteq \Gamma$ (in all other cases). The rule $\top$Unit in the operational semantics will postpone the decision of which subbunch of Γ we need until the rest of the computation has consumed everything it needs. At that point any left over will be considered as consumed by this rule. But the fact that $\top$ was found has to be signalled to the rest of the computation. This is done by incrementing a counter. The reason why a simple flag is not sufficient will be discussed in the $-\!\!*R$ rule.

$$\frac{}{\langle\Gamma|s\rangle_n\backslash\langle\Gamma|s\rangle_{n+1} \vdash_o \top}\ \top\text{Unit}$$

$\boxed{IR}$

$$\frac{}{\langle\Gamma, s\rangle\backslash\backslash\langle\Delta, s'\rangle \vdash I}\ IR$$

For this rule to be valid $\langle\Gamma, s\rangle\backslash\backslash\langle\Delta, s'\rangle$ must be equal to $\varnothing_m$.

There are three ways that $\varnothing_m$ can be produced according to Definition 4.3.2:

$\langle \phi ; \Gamma | \boxed{\phi}^{\varepsilon} :: s \rangle \backslash\backslash \langle \varepsilon | \{\Delta, \phi\} :: s \rangle = \Gamma - \Delta$. This case will produce $\varnothing_m$ only when $\Delta = \Gamma$, and then we can use the rule Unit^{ii}.

$$\frac{}{\langle \phi ; \Gamma | \boxed{\phi}^{\varepsilon} :: s \rangle_n \backslash \langle \varepsilon | \{\Gamma, \phi\} :: s \rangle_n \vdash_o I}\;\text{Unit}^{ii}$$

$\langle \Gamma | s \rangle \backslash\backslash \langle \varepsilon | s \rangle = \Gamma$. This case will produce $\varnothing_m$ in two cases: when $\Gamma = \varnothing_m$ and when $\top$ was found in a goal position (which is signalled with $n > 0$). The first case can use the Unit rule (see next case). When Γ is not $\varnothing_m$ then the rule Unit^{i} can be used. The flag must be reset to indicate that the $\top R$ rule was made use of.

$$\frac{}{\langle \Gamma | s \rangle_n \backslash \langle \varepsilon | s \rangle_0 \vdash_o I}\;\text{Unit}^{i}(n > 0)$$

$\langle \Gamma | s \rangle \backslash\backslash \langle \Delta | s \rangle = \Gamma - \Delta$ which will be $\varnothing_m$ only when $\Delta = \Gamma$. In this case rule Unit can be used

$$\frac{}{\langle \Gamma | s \rangle_0 \backslash \langle \Gamma | s \rangle_0 \vdash_o I}\;\text{Unit}$$

$\boxed{\text{CutAxiom}}$

$$\frac{\Gamma \vdash I}{\Gamma, \alpha \vdash \alpha}\;\text{CutAxiom}$$

We are given that $\langle \Gamma' | s \rangle \backslash\backslash \langle \Delta | s' \rangle \vdash I$ where $\Gamma' = \Gamma(\alpha) - \alpha$, and as induction hypothesis we have that $\langle \Gamma' | s \rangle_n \backslash \langle \Delta | s' \rangle_{n'} \vdash_o I$. We need to prove that $\langle \Gamma(\alpha) | s \rangle_n \backslash \langle \Delta | s' \rangle_{n'} \vdash_o \alpha$. First, using Definition 4.3.2 we have that $\langle \Gamma' | s \rangle \backslash\backslash \langle \Delta | s' \rangle = \Gamma' - \Delta = (\Gamma(\alpha) - \alpha) - \Delta$. Then we can make the following transformations

$$\dfrac{\dfrac{\dfrac{(\Gamma(\alpha) - \alpha) - \Delta \vdash I}{((\Gamma(\alpha) - \alpha) - \Delta), \alpha \vdash \alpha}\;\text{CutAxiom}}{\Gamma(\alpha) - \Delta \vdash \alpha}\;\text{Lemma 4.2.14}}{\langle \Gamma(\alpha) | s \rangle \backslash\backslash \langle \Delta | s' \rangle \vdash \alpha}\;\text{Definition 4.3.2}$$

and any proof with this shape will be found by the following rule:

$$\frac{\langle \Gamma' | s \rangle_n \backslash \langle \Delta | s' \rangle_{n'} \vdash_o I}{\langle \Gamma(\alpha) | s \rangle_n \backslash \langle \Delta | s' \rangle_{n'} \vdash_o \alpha}\;\text{CutAxiom}$$

In the special case where $s = \boxed{\xi}{::}s''$ and α is not a subbunch of ξ then we have to take care of removing the flag that said that the $\rightarrow R$ rule which created the open box was possibly a theorem, because at this stage we know it cannot be, since the resources in ξ are not enough to prove the goal.

$$\frac{\langle \Gamma' | \boxed{\xi}{::}s\rangle_n \backslash \langle \Delta | s' \rangle_{n'} \vdash_o I}{\langle \xi ; \Gamma(\alpha) | \boxed{\xi}^{\varepsilon}{::}s\rangle_n \backslash \langle \Delta | s' \rangle_{n'} \vdash_o \alpha}\ \text{CutAxiom}^i$$

$\boxed{-\!\!*L}$

$$\frac{\Gamma \vdash \phi}{\Gamma, \phi -\!\!* \alpha \vdash \alpha}\ -\!\!*L$$

Since the stacks are not tampered with, subcases analysis are unnecessary.

We are given that $\langle \Gamma' | s \rangle \backslash\backslash \langle \Delta | s' \rangle \vdash \phi$ where $\Gamma' = \Gamma(\phi -\!\!* \alpha) - \phi -\!\!* \alpha$

The induction hypothesis in this case is $\langle \Gamma' | s \rangle_n \backslash \langle \Delta | s' \rangle_{n'} \vdash_o \phi$ and we need to prove that $\langle \Gamma(\phi -\!\!* \alpha) | s \rangle_n \backslash \langle \Delta | s' \rangle_{n'} \vdash_o \alpha$. First we show that from $\langle \Gamma' | s \rangle \backslash\backslash \langle \Delta | s' \rangle \vdash \phi$ there always is a proof of $\langle \Gamma(\phi -\!\!* \alpha) | s \rangle \backslash\backslash \langle \Delta | s \rangle \vdash \alpha$:

$$\cfrac{\cfrac{\cfrac{\cfrac{\langle \Gamma' | s \rangle \backslash\backslash \langle \Delta | s' \rangle \vdash \phi}{(\Gamma(\phi -\!\!* \alpha) - \phi -\!\!* \alpha) - \Delta \vdash \phi}\ \text{Definition 4.2.6}}{((\Gamma(\phi -\!\!* \alpha) - \phi -\!\!* \alpha) - \Delta), \phi -\!\!* \alpha \vdash \alpha}\ -\!\!*L}{\Gamma(\phi -\!\!* \alpha) - \Delta \vdash \alpha}\ \text{Lemma 4.2.14}}{\langle \Gamma(\phi -\!\!* \alpha) | s \rangle \backslash\backslash \langle \Delta | s \rangle \vdash \alpha}\ \text{Definition 4.2.6}$$

This can be done using $-\!\!*L$ to get

$$\frac{\langle \Gamma' | s \rangle_n \backslash \langle \Delta | s' \rangle_{n'} \vdash_o \phi}{\langle \Gamma(\phi -\!\!* \alpha) | s \rangle_n \backslash \langle \Delta | s' \rangle_{n'} \vdash_o \alpha}\ -\!\!*L$$

$\boxed{\rightarrow L}$

$$\frac{\Delta ; \phi \rightarrow \alpha \vdash \phi \quad \Gamma' \vdash I}{\Gamma(\Delta ; \phi \rightarrow \alpha) \vdash \alpha} \rightarrow L$$

In this rule the stacks are not used, so it is not necessary to go into the subcases.

We are given that $\langle \Theta ; \phi \rightarrow \alpha | [] \rangle \backslash\backslash \langle \varepsilon | [] \rangle \vdash \phi$ and $\langle \Gamma' | s \rangle \backslash\backslash \langle \Delta | s' \rangle I$

As induction hypotheses we have $\langle\Theta;\phi\to\alpha|[]\rangle_0\backslash\langle\varepsilon|[]\rangle_m\vdash_o\phi$ and $\langle\Gamma'|s\rangle_n\backslash\langle\Delta|s'\rangle_{n'}\vdash_o I$. We need to show that $\langle\Gamma(\Theta;\phi\to\alpha)|s\rangle_n\backslash\langle\Delta|s'\rangle_{n'}\vdash_o\alpha$.

For convenience we will use the abbreviation Θ^+ to stand for $\Theta;\phi\to\alpha$.

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\Theta^+\vdash\phi \qquad \Gamma'-\Delta\vdash I}{(\Gamma'-\Delta),\Theta^+\vdash\alpha}\to L}{(\Gamma',\Theta^+)-\Delta\vdash\alpha}\text{Lemma 4.2.10}}{((\Gamma(\Theta^+)-\Theta^+),\Theta^+)-\Delta\vdash\alpha}\text{Def. of }\Gamma'}{((\Gamma(\Theta^+),(\Theta^+-\Theta^+))-\Delta\vdash\alpha}\text{Lemma 4.2.11}}{\Gamma(\Theta^+)-\Delta\vdash\alpha}\text{Subtraction and Unit operation}}{\langle\Gamma(\Theta^+)|s\rangle\backslash\backslash\langle\Delta|s'\rangle\vdash\alpha}\text{Definition 4.3.2}$$

This proof will be found by the following rule:

$$\dfrac{\langle\Theta;\phi\to\alpha|[]\rangle_0\backslash\langle\varepsilon|[]\rangle_m\vdash_o\phi \qquad \langle\Gamma'|s\rangle_n\backslash\langle\Delta|s'\rangle_{n'}\vdash_o I}{\langle\Gamma(\Theta;\phi\to\alpha)|s\rangle_n\backslash\langle\Delta|s'\rangle_{n'}\vdash_o\alpha}\to L$$

$\boxed{*R}$ We are given $\langle\Gamma|\boxtimes::s\rangle\backslash\backslash\langle\Gamma'|\boxtimes::s\rangle\vdash\phi$ and $\langle\Gamma'|s\rangle\backslash\backslash\langle\Delta|s'\rangle\vdash\psi$. From Definition 4.3.2 we have $\langle\Gamma|\boxtimes::s\rangle\backslash\backslash\langle\Gamma'|\boxtimes::s\rangle=\Gamma-\Gamma'$ and $\langle\Gamma'|s\rangle\backslash\backslash\langle\Delta|s'\rangle=\Gamma'-\Delta$. We can perform the following transformations:

$$\dfrac{\dfrac{\dfrac{\Gamma-\Gamma'\vdash\phi \qquad \Gamma'-\Delta\vdash\psi}{(\Gamma-\Gamma'),(\Gamma'-\Delta)\vdash\phi*\psi}*R}{\Gamma-\Delta\vdash\phi*\psi}\text{Weakening* (Lemma 4.2.16)}}{\langle\Gamma|s\rangle\backslash\backslash\langle\Delta|s'\rangle\vdash\phi*\psi}$$

The induction hypotheses are $\langle\Gamma|\boxtimes::s\rangle_n\backslash\langle\Gamma'|\boxtimes::s\rangle_{n'}\vdash_o\phi$ and $\langle\Gamma'|s\rangle_{n'}\backslash\langle\Delta|s'\rangle_{n''}\vdash_o\psi$. We need to prove that $\langle\Gamma|s\rangle_n\backslash\langle\Delta|s'\rangle_{n''}\vdash_o\phi*\psi$. This is straightforward from $*R$

$$\dfrac{\langle\Gamma|\boxtimes::s\rangle_n\backslash\langle\Gamma'|\boxtimes::s\rangle_{n'}\vdash_o\phi \quad \langle\Gamma'|s\rangle_{n'}\backslash\langle\Delta|s'\rangle_{n''}\vdash_o\psi}{\langle\Gamma|s\rangle_n\backslash\langle\Delta|s'\rangle_{n''}\vdash_o\phi*\psi}*R$$

Note that there is no need to analyse subcases with respect to the values of n, n' or n''. It is enough to ensure that any information about occurrences of $\top$ in a goal position to be passed on correctly. This means passing the n' received from the left subproof as the initial counter for the right subproof.

$\boxed{-\!\!*R}$

$$\dfrac{\phi,\Gamma\vdash\psi}{\Gamma\vdash\phi-\!\!*\psi}\,-\!\!*R$$

Given $\langle\phi,\Gamma|s\rangle\backslash\!\!\backslash\langle\Delta|s'\rangle\vdash\psi$ we can make the following proof:

$$\dfrac{\dfrac{\dfrac{(\phi,\Gamma)-\Delta\vdash\psi}{(\Gamma-\Delta),\phi\vdash\psi}\,\text{Lemma 4.2.10}}{\Gamma-\Delta\vdash\phi-\!\!*\psi}\,-\!\!*R}{\langle\Gamma|s\rangle\backslash\!\!\backslash\langle\Delta|s\rangle\vdash\phi-\!\!*\psi}\,\text{Definition 4.3.2}$$

As induction hypothesis we have that $\langle\phi,\Gamma|s\rangle_n\backslash\langle\Delta|s'\rangle_{n'}\vdash_o\psi$ and we need to show that $\langle\Gamma|s\rangle_n\backslash\langle\Delta'|s'\rangle_{n'}\vdash_o\phi-\!\!*\psi$. This can be done immediately from the proof figure above, using the $-\!\!*R$ rule of the operational semantics.

$$\dfrac{\langle\phi,\Gamma|s\rangle_n\backslash\langle\Delta|s'\rangle_{n'}\vdash_o\psi}{\langle\Gamma|s\rangle_n\backslash\langle\Delta'|s'\rangle_{n'}\vdash_o\phi-\!\!*\psi}\,-\!\!*R$$

$\boxed{\wedge R}$

$$\dfrac{\Gamma\vdash\psi\quad\Gamma\vdash\phi}{\Gamma\vdash\psi\wedge\phi}\,\wedge R$$

Given $\langle\xi;\Gamma|\boxed{\xi}^{\varepsilon}::s\rangle\backslash\!\!\backslash\langle\varepsilon|\{\Delta,\xi\}::s\rangle\vdash\phi$ and $\langle\xi;(\Gamma-\Delta)|[]\rangle\backslash\!\!\backslash\langle\varepsilon|[]\rangle\vdash\psi$ and as induction hypotheses $\langle\xi;\Gamma|\boxed{\xi}^{\varepsilon}::s\rangle_n\backslash\langle\varepsilon|\{\Delta,\xi\}::s\rangle_{n'}\vdash_o\phi$ and $\langle\xi;(\Gamma-\Delta)|[]\rangle_0\backslash\langle\varepsilon|[]\rangle_{n''}\vdash_o\psi$ we need to prove that $\langle\xi;\Gamma|\boxed{\xi}^{\varepsilon}::s\rangle_n\backslash\langle\varepsilon|\{\Delta,\xi\}::s\rangle_{n'''}\vdash_o\phi\wedge\psi$.

$$\dfrac{\langle\xi;\Gamma|\boxed{\xi}^{\varepsilon}::s\rangle_n\backslash\langle\varepsilon|\{\Delta,\xi\}::s\rangle_{n'}\vdash_o\phi\quad\langle\xi;(\Gamma-\Delta)|[]\rangle_0\backslash\langle\varepsilon|[]\rangle_{n''}\vdash_o\psi}{\langle\xi;\Gamma|\boxed{\xi}^{\varepsilon}::s\rangle_n\backslash\langle\varepsilon|\{\Delta,\xi\}::s\rangle_{n'''}\vdash_o\phi\wedge\psi}\,\wedge R^{iv}$$

Given

$$\dfrac{\langle\Gamma|s\rangle\backslash\!\!\backslash\langle\Delta|s\rangle\vdash\phi\quad\langle\Gamma-\Delta|[]\rangle\backslash\!\!\backslash\langle\varepsilon|[]\rangle\vdash\psi}{\langle\Gamma|s\rangle\backslash\!\!\backslash\langle\Delta|s\rangle\vdash\phi\wedge\psi}\,\wedge R$$

and assuming as inductive hypotheses $\langle\Gamma|s\rangle_n\backslash\langle\Delta|s\rangle_n\vdash_o\phi$ and $\langle\Gamma-\Delta|[]\rangle_0\backslash\langle\varepsilon|[]\rangle_m\vdash_o\psi$ we need to prove that $\langle\Gamma|s\rangle_n\backslash\langle\Delta|s\rangle_n\vdash_o\phi\wedge\psi$. It can be done with $\wedge R$.

$$\dfrac{\langle\Gamma|s\rangle_n\backslash\langle\Delta|s\rangle_n\vdash_o\phi\quad\langle\Gamma-\Delta|[]\rangle_0\backslash\langle\varepsilon|[]\rangle_m\vdash_o\psi}{\langle\Gamma|s\rangle_n\backslash\langle\Delta|s\rangle_n\vdash_o\phi\wedge\psi}\,\wedge R$$

Given

$$\dfrac{\langle\Gamma|s\rangle\backslash\!\!\backslash\langle\Delta|s\rangle\vdash\phi\quad\langle\Gamma|s\rangle\backslash\!\!\backslash\langle\Delta'|s\rangle\vdash\psi}{\langle\Gamma|s\rangle\backslash\!\!\backslash\langle\Delta'|s\rangle\vdash\phi\wedge\psi}\,\wedge R^{i}(\Delta'\subseteq\Delta)$$

and induction hypotheses $\langle\Gamma|s\rangle_n\backslash\langle\Delta|s\rangle_{n'}\vdash_o\phi$ and $\langle\Gamma|s\rangle_n\backslash\langle\Delta'|s\rangle_n\vdash_o\psi$, we need to prove that $\langle\Gamma|s\rangle_n\backslash\langle\Delta'|s\rangle_n\vdash_o\phi\wedge\psi$. This can be done with $\wedge R^i$.

$$\frac{\langle\Gamma|s\rangle_n\backslash\langle\Delta|s\rangle_{n'}\vdash_o\phi \quad \langle\Gamma|s\rangle_n\backslash\langle\Delta'|s\rangle_n\vdash_o\psi}{\langle\Gamma|s\rangle_n\backslash\langle\Delta'|s\rangle_n\vdash_o\phi\wedge\psi}\wedge R^i(n'>n\text{ and }\Delta'\subseteq\Delta)$$

Given

$$\frac{\langle\Gamma|s\rangle\backslash\backslash\langle\Delta|s\rangle\vdash\phi \quad \langle\Gamma|s\rangle\backslash\backslash\langle\Delta'|s\rangle\vdash\psi}{\langle\Gamma|s\rangle\backslash\backslash\langle\Delta\cap\Delta'|s\rangle\vdash\phi\wedge\psi}\wedge R^{ii}$$

and induction hypotheses $\langle\Gamma|s\rangle_n\backslash\langle\Delta|s\rangle_{n'}\vdash_o\phi$ and $\langle\Gamma|s\rangle_n\backslash\langle\Delta'|s\rangle_{n''}\vdash_o\psi$, we need to show that $\langle\Gamma|s\rangle_n\backslash\langle\Delta\cap\Delta'|s\rangle_{n'}\vdash_o\phi\wedge\psi$. This is done with $\wedge R^{ii}$

$$\frac{\langle\Gamma|s\rangle_n\backslash\langle\Delta|s\rangle_{n'}\vdash_o\phi \quad \langle\Gamma|s\rangle_n\backslash\langle\Delta'|s\rangle_{n''}\vdash_o\psi}{\langle\Gamma|s\rangle_n\backslash\langle\Delta\cap\Delta'|s\rangle_{n'}\vdash_o\phi\wedge\psi}\wedge R^{ii}(n',n''>n)$$

Given

$$\frac{\langle\Gamma|s\rangle\backslash\backslash\langle\varepsilon|s\rangle\vdash\phi \quad \langle\Gamma|s\rangle\backslash\backslash\langle\varepsilon|s\rangle\vdash\psi}{\langle\Gamma|s\rangle\backslash\backslash\langle\varepsilon|s\rangle\vdash\phi\wedge\psi}\wedge R$$

and $\langle\Gamma|s\rangle_n\backslash\langle\varepsilon|s\rangle_{n'}\vdash_o\phi$ and $\langle\Gamma|s\rangle_n\backslash\langle\varepsilon|s\rangle_{n''}\vdash_o\psi$ as induction hypotheses, we need to prove $\langle\Gamma|s\rangle_n\backslash\langle\varepsilon|s\rangle_{n'''}\vdash_o\phi\wedge\psi$. This can be done using $\wedge R^{iii}$.

$$\frac{\langle\Gamma|s\rangle_n\backslash\langle\varepsilon|s\rangle_{n'}\vdash_o\phi \quad \langle\Gamma|s\rangle_n\backslash\langle\varepsilon|s'\rangle_{n''}\vdash_o\psi}{\langle\Gamma|s\rangle_n\backslash\langle\varepsilon|s'\rangle_{n'''}\vdash_o\phi\wedge\psi}\wedge R^{iii}$$

$\boxed{\to R}$

$$\frac{\phi;\Gamma\vdash\psi}{\Gamma\vdash\phi\to\phi}\to R$$

Given $\langle\phi;\Gamma|[]\rangle\backslash\backslash\langle\varepsilon|[]\rangle\vdash\psi$ and $\langle\phi;\Gamma|[]\rangle_n\backslash\langle\varepsilon|[]\rangle_{n'}\vdash_o\psi$ as inductive hypothesis, we need to show that $\langle\Gamma|[]\rangle_n\backslash\langle\varepsilon|[]\rangle_{n'}\vdash_o\phi\to\psi$

This is straightforwards given that $\langle\phi;\Gamma|[]\rangle\backslash\backslash\langle\varepsilon|[]\rangle=\phi;\Gamma$ and $\langle\Gamma|[]\rangle\backslash\backslash\langle\varepsilon|[]\rangle=\Gamma$

$$\frac{\dfrac{\dfrac{\langle\phi;\Gamma|[]\rangle\backslash\backslash\langle\varepsilon|[]\rangle\vdash\psi}{\phi;\Gamma\vdash\psi}\text{Definition 4.3.2}}{\Gamma\vdash\phi\to\psi}\to R}{\langle\Gamma|[]\rangle\backslash\backslash\langle\varepsilon|[]\rangle\vdash\phi\to\psi}\text{Definition 4.3.2}$$

This proof will be found by the following rule

$$\frac{\langle \phi;\Gamma|[]\rangle_n\backslash\langle \varepsilon|[]\rangle_{n'} \vdash_o \psi}{\langle \Gamma|[]\rangle_n\backslash\langle \varepsilon|[]\rangle_{n'} \vdash_o \phi \to \psi} \to R$$

Given $\langle \phi;\Gamma|\boxed{\phi}^{\varepsilon}{::}s\rangle\backslash\backslash\langle \varepsilon|\{\Delta,\phi\}{::}s\rangle \vdash \psi$ and $\langle \phi;\Gamma|\boxed{\phi}^{\varepsilon}{::}s\rangle_n\backslash\langle \varepsilon|\{\Delta,\phi\}{::}s\rangle_{n'} \vdash_o \psi$ as induction hypothesis, we need to prove that $\langle \Gamma|\boxtimes{::}s\rangle_n\backslash\langle \Delta|\boxtimes{::}s\rangle_{n'} \vdash_o \phi \to \psi$.

$$\frac{\langle \phi;\Gamma|\boxed{\phi}^{\varepsilon}{::}s\rangle_n\backslash\langle \varepsilon|\{\Delta,\phi\}{::}s\rangle_{n'} \vdash_o \psi}{\langle \Gamma|\boxtimes{::}s\rangle_n\backslash\langle \Delta|\boxtimes{::}s\rangle_{n'} \vdash_o \phi \to \psi} \to R^{i}$$

Given

$$\frac{\langle \phi;\xi;\Gamma|\boxed{\xi;\phi}^{-}{::}s\rangle\backslash\backslash\langle \varepsilon|\{\Delta,(\xi;\phi)\}{::}s\rangle \vdash \psi}{\langle \xi;\Gamma|\boxed{\xi}^{-}{::}s\rangle\backslash\backslash\langle \varepsilon|\{\Delta,\xi\}{::}s\rangle \vdash \phi \to \psi} \to R$$

and induction hypothesis $\langle \phi;\xi;\Gamma|\boxed{\xi;\phi}^{-}{::}s\rangle_n\backslash\langle \varepsilon|\{\Delta,(\xi;\phi)\}{::}s\rangle_{n'} \vdash_o \psi$ we need to prove $\langle \xi;\Gamma|\boxed{\xi}^{-}{::}s\rangle_n\backslash\langle \varepsilon|\{\Delta,\xi\}{::}s\rangle_{n'} \vdash_o \phi \to \psi$. We can do this using $\to R^{ii}$.

$$\frac{\langle \phi;\xi;\Gamma|\boxed{\xi;\phi}^{-}{::}s\rangle_n\backslash\langle \varepsilon|\{\Delta,(\xi;\phi)\}{::}s\rangle_{n'} \vdash_o \psi}{\langle \xi;\Gamma|\boxed{\xi}^{-}{::}s\rangle_n\backslash\langle \varepsilon|\{\Delta,\xi\}{::}s\rangle_{n'} \vdash_o \phi \to \psi} \to R^{ii}$$

Given

$$\frac{\langle \phi;\Gamma|\boxed{\phi}^{\varepsilon}{::}s\rangle\backslash\backslash\langle \varepsilon|\boxed{\phi}^{\varepsilon}{::}s\rangle \vdash \psi}{\langle \Gamma|\boxtimes{::}s\rangle\backslash\backslash\langle \Gamma|\boxtimes{::}s\rangle \vdash \phi \to \psi} \to R$$

and as induction hypothesis $\langle \phi;\Gamma|\boxed{\phi}^{\varepsilon}{::}s\rangle_n\backslash\langle \varepsilon|\boxed{\phi}^{\varepsilon}{::}s\rangle_{n'} \vdash_o \psi$ we want to show that $\langle \Gamma|\boxtimes{::}s\rangle_n\backslash\langle \Gamma|\boxtimes{::}s\rangle_{n+1} \vdash_o \phi \to \psi$. This is done with the $\to R^{iii}$ rule.

$$\frac{\langle \phi;\Gamma|\boxed{\phi}^{\varepsilon}{::}s\rangle_n\backslash\langle \varepsilon|\boxed{\phi}^{\varepsilon}{::}s\rangle_{n'} \vdash_o \psi}{\langle \Gamma|\boxtimes{::}s\rangle_n\backslash\langle \Gamma|\boxtimes{::}s\rangle_{n+1} \vdash_o \phi \to \psi} \to R^{iii}$$

$\boxed{\vee R}$

$$\frac{\Gamma \vdash \phi_i}{\Gamma \vdash \phi_1 \vee \phi_2} \vee R (i = 1,2)$$

Given

$$\frac{\langle \Gamma|s\rangle\backslash\backslash\langle \Delta|s'\rangle \vdash \phi \text{ (or } \psi)}{\langle \Gamma|s\rangle\backslash\backslash\langle \Delta|s'\rangle \vdash \phi \vee \psi} \vee R$$

and induction hypothesis either $\langle \Gamma|s\rangle_n\backslash\langle \Delta|s'\rangle_{n'} \vdash_o \phi$ or $\langle \Gamma|s\rangle_n\backslash\langle \Delta|s'\rangle_{n'} \vdash_o \psi$, we need to prove $\langle \Gamma|s\rangle_n\backslash\langle \Delta|s'\rangle_{n'} \vdash_o \phi \vee \psi$, which can be done using one of the $\vee R$ rules.

$$\frac{\langle \Gamma|s\rangle_n\backslash\langle \Delta|s'\rangle_{n'} \vdash_o \phi}{\langle \Gamma|s\rangle_n\backslash\langle \Delta|s'\rangle_{n'} \vdash_o \phi \vee \psi} \vee R \qquad \frac{\langle \Gamma|s\rangle_n\backslash\langle \Delta|s'\rangle_{n'} \vdash_o \psi}{\langle \Gamma|s\rangle_n\backslash\langle \Delta|s'\rangle_{n'} \vdash_o \phi \vee \psi} \vee R$$

□

4.6 Predicate BLP

We have discussed in § 2.5 the possibility of having bunched variables. Also all the analysis of permutations presented in Chapter 3 use bunched variables showing that it is perfectly feasible to have simple resolution proofs using the full predicate **BI**. However, the implementation described in Chapter 8 uses only additive maintenance of variables.

In the next section we will describe the first steps towards implementing a version of BLP with fully bunched variables.

4.6.1 Predicate rules for LBI

The predicate rules that we are going to utilise are a straightforward simplification from the ones given earlier in Definition 2.6.1. The main difference is that there is no need to tag explicitly the variables to mark them as additive, since all of them are. Also where before we used a bunched notation for the $\forall L$ rule, now a flat representation is enough.

$$\frac{\{X;Y\}\,\Gamma \vdash \phi[t/x]}{\{X\}\,\Gamma \vdash \exists x.\phi(x)}\,(Y \vdash t : \mathrm{Term})\ \exists R$$

$$\frac{\{X;Y\}\,\Gamma(\phi(t)) \vdash \psi}{\{X\}\,\Gamma(\forall x.\phi(x)) \vdash \psi}\,(Y \vdash t : \mathrm{Term})\ \forall L$$

4.6.2 Operational semantics for predicate BLP

The operational semantics for these rules are as we should expect:

$$\frac{\langle\{X;Y\}\,\Gamma|s\rangle_n\backslash\langle\{Z\}\,\Delta|s'\rangle_{n'} \vdash_o \phi[t/x]}{\langle\{X\}\,\Gamma|s\rangle_n\backslash\langle\{Z\}\,\Delta|s'\rangle_{n'} \vdash_o \exists x.\phi(x)}\,\exists R$$

$$\frac{\langle\{X;Y\}\,\Gamma(\phi(t))|s\rangle_n\backslash\langle\{Z\}\,\Delta|s'\rangle_{n'} \vdash_o \psi}{\langle\{X\}\,\Gamma(\forall x.\phi(x))|s\rangle_n\backslash\langle\{Z\}\,\Delta|s'\rangle_{n'} \vdash_o \psi}\,\forall L$$

or, more traditionally, using σ to denote a substitution (list of pairs of terms),

$$\frac{\langle\Gamma|s\rangle_n\backslash\langle\Delta|s'\rangle_{n'} \vdash_o^{Y::\sigma} \phi[t/x]}{\langle\Gamma|s\rangle_n\backslash\langle\Delta|s'\rangle_{n'} \vdash_o^{\sigma} \exists x.\phi(x)}\,\exists R$$

$$\frac{\langle\Gamma(\phi(t))|s\rangle_n\backslash\langle\Delta|s'\rangle_{n'} \vdash_o^{Y::\sigma} \psi}{\langle\Gamma(\forall x.\phi(x))|s\rangle_n\backslash\langle\Delta|s'\rangle_{n'} \vdash_o^{\sigma} \psi}\,\forall L$$

Soundness and completeness of these rules are trivial.

4.7 Predicate BI with bunched variables

The predicate sequent calculus presented in § 2.6 has some properties that makes it unsuitable for logic programming. The main problem is the $\mathrm{Axiom(x,y)}$ relation that needs to be maintained throughout the proof. This relation has got a global character which means that it cannot be determined at one particular step on the proof. It could be done using a system of constraints, but we prefer a simpler solution, which is to present an alternative system where this global constraints do not arise.

4.7.1 Alternative sequent calculus system: LBI′

An alternative system with the same power than **LBI** is given by the following modifications. First we will consider families of variables which will be α-convertible. We will identify the families by tags $(v,w,\ldots)$ and adorn each of them with integer subscripts to make them unique.

We change the axiom and the substitution rules as follows

Identity and Structure

$$\frac{X_1 \vdash \phi(X_1) : \mathbf{Prop}}{\{X_1 \circledcirc X_2\}\ \phi(X_1) \vdash \phi(X_2)}\ (X_1 \simeq X_2)\ \text{Axiom(where } \circledcirc =; \text{or,)}$$

$$\frac{\{X(v_n)\}\ \Gamma \vdash \phi \quad Y \vdash t : \mathrm{Term}}{\{X(Y)\}\ \Gamma[t/v_n] \vdash \phi[t/v_n]}\ \text{Substitution}$$

where $X_1 \simeq X_2$ iff all variables in X_1 have a matching variable in X_2 with the same tag, and Y is a bunch of variables using just the tag v

Multiplicative quantifiers

$$\frac{\{X,v_1\}\ \Gamma \vdash \phi(v_1)}{\{X\}\ \Gamma \vdash \forall_{\mathsf{new}} x.\phi[x/v_1]}\ \forall_{\mathsf{new}} R\ (\text{new } v)$$

$$\frac{\{X(Y)\}\ \Gamma(\phi(t)) \vdash \psi}{\{X(I)\}\ \Gamma(\forall_{\mathsf{new}} x.\phi[x/t]) \vdash \psi}\ (Y \vdash t : \mathrm{Term})\ \forall_{\mathsf{new}} L$$

$$\frac{\{X,Y\}\ \Gamma \vdash \phi[t/x]}{\{X\}\ \Gamma \vdash \exists_{\mathsf{new}} x.\phi(x)}\ (Y \vdash t : \mathrm{Term})\ \ \exists_{\mathsf{new}} R$$

$$\frac{\{X(v_1)\}\ \Gamma(\phi(v_1)) \vdash \psi}{\{X(I)\}\ \Gamma(\exists_{\mathsf{new}} x.\phi[x/v_1]) \vdash \psi}\ \exists_{\mathsf{new}} L\ (\text{new } v)$$

Additive quantifiers

$$\frac{\{X;v_1!\}\ \Gamma \vdash \phi(v_1)}{\{X\}\ \Gamma \vdash \forall x.\phi[x/v_1]}\ \forall R\ (\text{new } v)$$

$$\frac{\{X(Y!)\}\ \Gamma(\phi(t)) \vdash \psi}{\{X(1)\}\ \Gamma(\forall x.\phi[x/t]) \vdash \psi}\ (Y! \vdash t : \text{Term})\ \forall L$$

$$\frac{\{X;Y!\}\ \Gamma \vdash \phi[t/x]}{\{X\}\ \Gamma \vdash \exists x.\phi(x)}\ (Y! \vdash t : \text{Term})\ \ \exists R$$

$$\frac{\{X(v_1!)\}\ \Gamma(\phi(v_1)) \vdash \psi}{\{X(1)\}\ \Gamma(\exists x.\phi[x/v_1]) \vdash \psi}\ \exists L\ \ (\text{new } v)$$

Looking at the rules as going upwards, the linearity condition is preserved by creating always new variables. The logic restriction for $\forall_{\mathsf{new}}R$, $\forall R$, $\exists_{\mathsf{new}}L$ and $\exists L$ is now quite simple: v has to be a new tag.

4.7.2 Translation from LBI′ to LBI

We first provide a translation from **LBI′**to **LBI**to show that this system is sound with respect to the other.

- Every occurrence of an Axiom rule in LBI’ has to be changed to name explicitly the variables connected by this axiom.

$$\frac{X_1 \vdash \phi(X_2) : \textbf{Prop} \quad X_1 \simeq X_2}{\{X_1 \circledcirc X_2\}\ \phi(X_1) \vdash \phi(X_2)}\ \text{Axiom} \quad \rightsquigarrow$$

$$\frac{X_1 \vdash \phi(X_1) : \textbf{Prop} \quad X_2 \vdash \phi(X_2) : \textbf{Prop}}{\{X_1 \circledcirc X_2\}\ \phi(X_1) \vdash \phi(X_2)}\ \text{Axiom}(\mathrm{X_1},\mathrm{X_2})$$

- In every substitution LBI adds axioms, so they have to be added in the translation

$$\frac{\{X(v_n)\}\ \Gamma \vdash \phi \quad Y \vdash t : \text{Term}}{\{X(Y)\}\ \Gamma[t/v_n] \vdash \phi[t/v_n]}\ \text{Substitution} \quad \rightsquigarrow$$

$$\frac{\{X(v_n)\}\ \Gamma \vdash \phi \quad Y \vdash t : \text{Term}}{\{X(Y)\}\ \Gamma[t/v_n] \vdash \phi[t/v_n]}\ \text{Axiom}(Y,x')\text{Substitution}$$

for all x' s.t. Axiom(x,x')

- The right universal rules and the left existential rules need an α-conversion to match the syntax of the rules.

In short, the translation reduces to connecting bunches of variables via axioms. In principle it could happen that by adding these axioms some of the quantifier rules break down, in the sense that the restrictions don't hold any more. The linearity condition is preserved because LBI' uses new variables all the time. But we need to check the extra conditions.

First of all note that the axioms connect variables with the same tag. Then suppose we have a proof ending with the application of $\forall R$ or $\forall_{\mathsf{new}} R$ (I'll use the symbol $\forall_{\odot}$ to stand indistinctly for the two universal quantifiers)

$$\dfrac{\begin{array}{c}\Xi\\ \vdots\\ \{X \odot v_1\}\, \Gamma \vdash \phi(v_1)\end{array}}{\{X\}\, \Gamma \vdash \forall_{\odot} v_1.\phi(v_1)}\; \forall_{\odot} R$$

We need to check that it is still the case that

$$v_1 \notin \{y | \text{Axiom}(y,z) \text{ and } z \in \text{FV}(\Gamma)\}$$

and indeed, the set

$$\{y | \text{Axiom}(y,z) \text{ and } z \in \text{FV}(\Gamma)\}$$

contains only variables with tags different from v.

Likewise suppose the proof ends with an $\exists L$ or $\exists_{\mathsf{new}} L$ rule. Then after translation we will have a proof like

$$\dfrac{\begin{array}{c}\Xi\\ \vdots\\ \{X(v_1)\}\, \Gamma(\phi(v_1)) \vdash \psi\end{array}}{\{X(u)\}\, \Gamma(\exists_{\odot} v_1.\phi(v_1)) \vdash \psi}\; \exists_{\odot} L$$

The condition now is that

$$v_1 \notin \{y | \text{Axiom}(y,z) \text{ and } z \in \text{FV}(\psi) \text{ or } \text{FV}(\Gamma)\}$$

and because of the same reasons than before, this is the case.

4.7.3 Translation from LBI to LBI$'$

The variables are partitioned by α-convertibility. This is done by closing the relation $\text{Axiom}(\text{x},\text{y})$ under transitivity. We assign a different tag to each class. The translation is done by replacing each variable with the tag of its class subscripted by an unused integer.

Could it happen that in a rule that requires a new tag this procedure produces a tag already in use by some other variable? If that was the case then the two variables would be α-convertible which means that it would violate the corresponding extra condition in LBI.

Here there is a proof of the intuitionistic sequent $\forall_{\mathsf{new}} x.\forall y.P(x,y) \vdash \forall z.\forall_{\mathsf{new}} w.P(w,z)$ in LBI and its translation into LBI'

$$\begin{array}{cl}
\hline \{(y!;x),z!,w\}\ P(x,y) \vdash P(w,z) & \text{Axiom}(x,w)\ \text{Axiom}(y!,z!) \\
\hline \{(1;x),z!,w\}\ \forall y.P(x,y) \vdash P(w,z) & \forall L \\
\hline \{x,z!,w\}\ \forall y.P(x,y) \vdash P(w,z) & \text{Unit} \\
\hline \{I,z!,w\}\ \forall_{\mathsf{new}} x.\forall y.P(x,y) \vdash P(w,z) & \forall_{\mathsf{new}} L \\
\hline \{z!,w\}\ \forall_{\mathsf{new}} x.\forall y.P(x,y) \vdash P(w,z) & \text{Unit} \\
\hline \{z!\}\ \forall_{\mathsf{new}} x.\forall y.P(x,y) \vdash \forall_{\mathsf{new}} w.P(w,z) & \forall_{\mathsf{new}} R \\
\hline \{1;z!\}\ \forall_{\mathsf{new}} x.\forall y.P(x,y) \vdash \forall z.\forall_{\mathsf{new}} w.P(w,z) & \text{Unit} \\
\hline \{1\}\ \forall_{\mathsf{new}} x.\forall y.P(x,y) \vdash \forall z.\forall_{\mathsf{new}} w.P(w,z) & \forall R
\end{array}$$

There are two sets of α-convertible variables: $\{x,w\}$ and $\{y,z\}$. Assigning tags v and u respectively and doing some renaming we convert this proof into

$$\begin{array}{cl}
\hline \{(u!_2;v_2),u!_1,v_1\}\ P(v_2,u_2) \vdash P(v_1,u_1) & \text{Axiom} \\
\hline \{(1;v_2),u!_1,v_1\}\ \forall y.P(v_2,y) \vdash P(v_1,u_1) & \forall L \\
\hline \{v_2,u!_1,v_1\}\ \forall y.P(v_2,y) \vdash P(v_1,u_1) & \text{Unit} \\
\hline \{I,u!_1,v_1\}\ \forall_{\mathsf{new}} x.\forall y.P(x,y) \vdash P(v_1,u_1) & \forall_{\mathsf{new}} L \\
\hline \{u!_1,v_1\}\ \forall_{\mathsf{new}} x.\forall y.P(x,y) \vdash P(v_1,u_1) & \text{Unit} \\
\hline \{u!_1\}\ \forall_{\mathsf{new}} x.\forall y.P(x,y) \vdash \forall_{\mathsf{new}} x.P(x,u_1) & \forall_{\mathsf{new}} R \\
\hline \{1;u!_1\}\ \forall_{\mathsf{new}} x.\forall y.P(x,y) \vdash \forall y.\forall_{\mathsf{new}} x.P(x,y) & \text{Unit} \\
\hline \{1\}\ \forall_{\mathsf{new}} x.\forall y.P(x,y) \vdash \forall y.\forall_{\mathsf{new}} x.P(x,y) & \forall R
\end{array}$$

When searching for a proof of a sequent, we don't know in advance which ones are going to be the axioms. In many cases we could make good guesses at them but in general this is an issue that has to be left open until the last minute, when the proof is finished. From the point of view of proof search, the extra conditions that have to be satisfied for a proof to be valid can be seen as constraints on the possible axioms available. The system proposed as alternative have the constraints built-in in such a way that all decisions can be made locally preserving soundness and completeness.

4.8 Soundness and Completeness of LBI′

If we denote $\Gamma \vdash_{\mathbf{LBI}} \phi$ proofs done in **LBI** and likewise $\Gamma \vdash_{\mathbf{LBI'}} \phi$ proofs done in **LBI′**, then we can state the following

---

Lemma 4.8.1

$$\Gamma \vdash_{\mathbf{LBI}} \phi \quad \text{iff} \quad \Gamma \vdash_{\mathbf{LBI'}} \phi$$

Proof: There are easy forwards and backwards translations described in 4.7.2 and 4.7.3.

□

4.9 Operational semantics for full predicate BI

When considering the operational semantics for predicate BI, the first step is to provide operational rules for the quantifiers. In addition, we need to add a context into the rules, and a way of keeping track of the substitutions already performed.

The rules for existential quantifiers for **BI** are

$$\frac{\{X,Y\}\ \Gamma \vdash \phi[t/x]}{\{X\}\ \Gamma \vdash \exists_{\mathsf{new}} x.\phi(x)}\ (Y! \vdash t : \text{Term})\ \ \exists_{\mathsf{new}} R \qquad \frac{\{X;Y!\}\ \Gamma \vdash \phi[t/x]}{\{X\}\ \Gamma \vdash \exists x.\phi(x)}\ (Y! \vdash t : \text{Term})\ \ \exists R$$

However these rules turn out to be very restrictive. As an illustration of the issues arising, consider the program consisting only of the atom $p(a;(b,c))$. When trying to prove a predicate $p(x;(y,z))$ where x,y and z are variables, there is no combination of existential quantifiers that would allow a proof to be done. As an example we show a failed attempt using $\exists$ for x and $\exists_{\mathsf{new}}$ for y and z

$$\cfrac{\cfrac{\cfrac{\{(a;(b,c);d),e,f\}\ p(a;(b,c)) \vdash p(d;(e,f))}{\{(a;(b,c);d),e\}\ p(a;(b,c)) \vdash \exists_{\mathsf{new}} z.p(d;(e,z))}\ \exists_{\mathsf{new}} R}{\{a;(b,c);d\}\ p(a;(b,c)) \vdash \exists_{\mathsf{new}} y,z.p(d;(y,z))}\ \exists_{\mathsf{new}} R}{\{a;(b,c)\}\ p(a;(b,c)) \vdash \exists x.\exists_{\mathsf{new}} y,z.p(x;(y,z))}\ \exists R$$

and the context $\{(a;(b,c);d),e,f\}$ cannot be split in an appropriate way because it has the wrong shape. Trying other combinations of quantifiers ($\exists x.\exists_{\mathsf{new}} y \exists z.p(x;(y,z))$, $\exists_{\mathsf{new}} x,y.\exists z.p(x;(y,z))$) will not work either, and always for the same reason: the resulting bunch of variables is of the wrong shape.

One way of solving this problem is to restrict the shape that the bunch of terms can take. For example, it can be required that different kind of bunches cannot be embedded into each other, which effectively amounts to bunches which have a flat structure.

But it is possible to change slightly the rules in such a way that this problem doesn't arise. Consider the variant existential rules

$$\frac{\{X\circ(X';Y)\}\,\Gamma\vdash\phi[t/x]}{\{X\circ X'\}\,\Gamma\vdash\exists x.\phi(x)}\,\exists R \qquad \frac{\{X\circ(X',Y)\}\,\Gamma\vdash\phi[t/x]}{\{X\circ X'\}\,\Gamma\vdash\exists_{\mathsf{new}}x.\phi(x)}\,\exists_{\mathsf{new}}R$$

with the side conditions that $X'\vdash\exists_{\circ}x.\phi(x)$: Term and $Y\vdash t$: Term

Cut elimination for these rules in their new formulation can be done as follows: a proof figure like

$$\frac{\dfrac{\{X\circ(X';Z)\}\,\Delta\vdash\phi[t/x]}{\{X\circ X'\}\,\Delta\vdash\exists x.\phi(x)}\,\exists R \qquad \dfrac{\{Y(X';x)\}\,\Gamma(\phi(x))\vdash\psi}{\{Y(X')\}\,\Gamma(\exists x.\phi(x))\vdash\psi}\,\exists L}{\{Y(X)\}\,\Gamma(\Delta)\vdash\psi}\,\mathrm{Cut}$$

translates to

$$\frac{\{X\circ(X';Z)\}\,\Delta\vdash\phi(t) \qquad \{Y(X';Z)\}\,\Gamma(\phi(t))\vdash\psi}{\{Y(X)\}\,\Gamma(\Delta)\vdash\psi}\,\mathrm{Cut}$$

And for $\exists_{\mathsf{new}}$, a proof figure like

$$\frac{\dfrac{\{X\circ(X',Z)\}\,\Delta\vdash\phi[t/x]}{\{X\circ X'\}\,\Delta\vdash\exists_{\mathsf{new}}x.\phi(x)}\,\exists_{\mathsf{new}}R \qquad \dfrac{\{Y(X',x)\}\,\Gamma(\phi(x))\vdash\psi}{\{Y(X')\}\,\Gamma(\exists_{\mathsf{new}}x.\phi(x))\vdash\psi}\,\exists_{\mathsf{new}}L}{\{Y(X)\}\,\Gamma(\Delta)\vdash\psi}\,\mathrm{Cut}$$

translates to

$$\frac{\{X\circ(X',Z)\}\,\Delta\vdash\phi(t) \qquad \{Y(X',Z)\}\,\Gamma(\phi(t))\vdash\psi}{\{Y(X)\}\,\Gamma(\Delta)\vdash\psi}\,\mathrm{Cut}$$

The operational semantics for these rules is

$$\frac{\langle\{X\circ(X';Y)\}\,\Gamma|s\rangle_n\backslash\langle\{Z\}\,\Delta|s'\rangle_{n'}\vdash_o\phi[t/x]}{\langle\{X\circ X'\}\,\Gamma|s\rangle_n\backslash\langle\{Z\}\,\Delta|s'\rangle_{n'}\vdash_o\exists x.\phi(x)}\,\exists R$$

$$\frac{\langle\{X\circ(X',Y)\}\,\Gamma|s\rangle_n\backslash\langle\{Z\}\,\Delta|s'\rangle_{n'}\vdash_o\phi[t/x]}{\langle\{X\circ X'\}\,\Gamma|s\rangle_n\backslash\langle\{Z\}\,\Delta|s'\rangle_{n'}\vdash_o\exists_{\mathsf{new}}x.\phi(x)}\,\exists_{\mathsf{new}}R$$

with the side conditions $Y\vdash t$: Term and $X'\vdash\exists_{\circ}\phi(x)$: Term.

Now turning to the universal quantifiers, the **BI** rules are

$$\frac{\{X(Y;Z)\}\,\Gamma(\phi(t)) \vdash \psi}{\{X(Y)\}\,\Gamma(\forall x.\phi(x)) \vdash \psi}\,(Z \vdash t : \text{Term})\ \forall L$$

$$\frac{\{X(Y,Z)\}\,\Gamma(\phi(t)) \vdash \psi}{\{X(Y)\}\,\Gamma(\forall_{\mathsf{new}} x.\phi(x)) \vdash \psi}\,(Z \vdash t : \text{Term})\ \forall_{\mathsf{new}} L$$

But there is some extra information that might be used to simplify these rules. Since the proofs that we are using are uniform, we know that the conclusion of these rules has to be atomic. It is also convenient for the translation to separate the three cases that could arise.

Also we have some knowledge about the structure of the context of variables/constants. It is useful first to split the cases of the left universal quantifiers depending on the three possible resolution instances

First, the clause selected for resolution is an atom and the rule applicable after unification is a CutAxiom.

$$\cfrac{\cfrac{\{(X',(Y';Z))\circ Y\}\,\Gamma' \vdash I}{\{(X',(Y';Z))\circ Y\}\,\Gamma',\alpha(t) \vdash \alpha(t)}\,\text{CutAxiom}}{\{X(Y')\circ Y\}\,\Gamma(\forall x.\alpha(x)) \vdash \alpha(t)}\,(Z \vdash t : \text{Term})\ \forall L$$

The proposed operational semantics for CutAxiom$_\forall$ is

$$\frac{\langle\{X\}\,\Gamma|s\rangle_n\backslash\langle\{Z\}\,\Delta|s'\rangle_{n'} \vdash_o I}{\langle\{X,X'\circ Y\}\,\Gamma,\forall x.\alpha(x)|s\rangle_n\backslash\langle\{Z\}\,\Delta|s'\rangle_{n'} \vdash_o \alpha(\vec{t})}\,\text{CutAxiom}_\forall$$

with the side conditions ($Y \vdash \vec{t}$: Term and $X' \vdash \forall x.\alpha(x)$: Term). In addition we require that $(X';u) \approx Y$. Notice that Γ' has no free terms on it.

$$\frac{\langle\{X,Y\}\,\Gamma'|s\rangle_n\backslash\langle\{Z\}\,\Delta|s'\rangle_{n'} \vdash_o \phi(\vec{u})}{\langle\{(X,X');W\}\,\Gamma,\forall\vec{z}.(\phi(\vec{y}) \mathrel{-\!\!*} \alpha(\vec{x}))|s\rangle_n\backslash\langle\{Z\}\,\Delta|s'\rangle_{n'} \vdash_o \alpha(\vec{t})}\,\forall L$$

For this rule the side conditions are $W \vdash t$: Term, $X' \vdash \forall x.\phi(x)$: Term, and $Y \vdash u$: Term.

$$\frac{\langle\{X',Y\}\,\Theta;\forall x.(\phi(x) \to \alpha(x))|s\rangle_0\backslash\langle\{Z\}\,\Delta|s'\rangle_m \vdash_o \phi(t) \quad \langle\{X\}\,\Gamma'|s\rangle_n\backslash\langle\{Z\}\,\Delta|s'\rangle_{n'} \vdash_o I}{\langle\{(X,X')\}\,\Gamma,(\Theta;\forall\vec{z}.(\phi(\vec{y}) \to \alpha(\vec{x})))|s\rangle_n\backslash\langle\{Z\}\,\Delta|s'\rangle_{n'} \vdash_o \alpha(\vec{t})}\,\forall L$$

$Y \vdash t$: Term and $X' \vdash \forall x.\phi(x)$: Term

Chapter 5

Examples

5.1 Introduction

The examples developed in this chapter are based in the ideas of conflict or other interactions arising from memebership to different groups. The logic of **BI**, specially the affine version, seems to be well suited for dealing with these issues. In this chapter we

- Present two motivating examples: fights between political parties and family relationships.
- Discuss the advantages of `BLP` over Prolog for these kind of programs.
- Discuss problems arising from linear recursion.
- Present a major example: the implementation of an $\alpha\lambda$-calculus type inference system.
- Discuss issues related to the use of modules.

5.2 Fights Between Rival Factions

This logic seems well suited to model hooligan behaviour, as well as political parties. As an example consider the bunch `(p(a1);p(a2)),(p(b1);p(b2))`. In these examples p(x) means that x is a person. The bunch structure shows that `a1` and `a2` belong to the same group, and that `a1` and `b1` belong to belligerent groups. To say that two individuals are in a possible fighting relation we say simply $\forall x,y.p(x) * p(y) \mathrel{-\!\!*} fight(x,y)$, which is to say that x and y can fight if they are people which belong to different groups. A complete program would be

```
(p(a1);p(a2)),
(p(b1);p(b2)),


[x,y] fight(x,y) *- p(x) * p(y)
```

If more than two groups are involved, then the program has to be modified, because the relation holds even when some group is not considered. This can be done modifying the definition of fight to `[x,y] fight(x,y) *- p(x)* p(y)*T`, to be read: once `x` and `y` have been found, disregard any left-over. Alternatively each group could be decorated 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(c1);p(c2);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.
```

When we are talking of political parties, sometimes they split into rival factions, but each faction in turn might want to keep the former allies. See Figure 2 for an example of this. 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` been 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,_)
p(b1,t2,_).
   p(b21,t2,t1).p(b22,t2,t1).
   p(b23,t2,t2).p(b24,t2,t2).


fight(X,Y) :- p(X,T,_),p(Y,U,_),T ≠ U.
fight(X,Y) :- p(X,T,V), p(Y,U,W), T = U, V ≠ W.
```

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 can be dreamed up, like using a list of tags as a second argument:

```
p(a1,[t1]). p(a2,[t1]).
p(b1,[t2]).
  p(b21,[t2,t1]). p(b22,[t2,t1]).
  p(b23,[t2,t2]). p(b24,[t2,t2]).


fight(X,Y) :- p(X,U), p(Y,S), mismatch(U,S).
mismatch([H1|_],[H2|_]) :- H1 ≠ H2.
mismatch([H1|T1],[H2|T2]) :- H1 = H2, mismatch(T1,T2).
```

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

5.3 Family Matters

Another example that is well suited to be represented with bunches is family ties. Members of the same family are represented with additive bunches, and different families are multiplicatively connected. In the following examples m(x) and w(x) mean that x is a man or a woman respectively. In this context, if we want to express the old fashion practice that x and y can marry if they are of different sexes and different families, it could be done simply with the predicate [x,y] can_marry(x,y) *- (m(x)* w(y) | w(x)* m(y))* T. The addition of * T to the antecedent is, again, to take care of the families not considered in the relation. So a BLP program would look like

```
(m(john) ; w(kate)),
(m(patrick) ; w(isabelle)),
(m(abelard) ; w(eloise)),

[x,y]can_marry(x,y) *- ((m(x) * w(y)) | (w(x) * m(y))) * T.
```

Again Prolog needs the addition of extra-logical tags to reflect the structure of the family relations:

```
m(john,a).
w(kate,a).
m(patrick,b).
m(isabelle,b).
m(abelard,c).
w(eloise,c).

can_marry(X,Y) :- m(X,T), w(Y,U), T ≠ U.
can_marry(X,Y) :- w(X,T), m(Y,U), T ≠ U.
```

Hopefully it is clear by now that for this type of problems a programming language based on **BI** will provide a simple way of expressing the fundamental features, while Prolog will need at least some control structures foreign to the problem.

In the sequel to the chapter, we will explore other examples from a `BLP` perspective. We will not refer to Prolog again.

5.4 Encapsulation

In this family setting, complications might arise. If a Don Juan just arrived in town, would it be possible for him to marry somebody living there? We can answer this question by asking `BLP` whether the goal `<w> m(don_juan)-*can_marry(don_juan,w)` has got any answer substitution. Then we will see, to our dismay, that this outsider will be able to marry every woman available! As an example we show a simplified proof that Don Juan can marry Kate:

$$
\dfrac{\dfrac{\dfrac{\dfrac{m(dj) \vdash m(dj) \qquad m(j);w(k) \vdash w(k) \qquad m(a);w(e) \vdash \top}{m(dj),(m(j);w(k)),(m(a);w(e)) \vdash m(dj) * w(k) * \top}\;{*R}}{m(dj),(m(j);w(k)),(m(a);w(e)),\forall x,y.m(x) * w(y) * \top \mathrel{-\!\!*} cm(x,y) \vdash cm(dj,k)}\;{-\!\!*L}}{m(dj),(m(j);w(k)),(m(a);w(e)),\forall x,y.m(x) * w(y) * \top \mathrel{-\!\!*} cm(x,y) \vdash \exists w.cm(dj,w)}\;{\exists R}}{(m(j);w(k)),(m(a);w(e)),\forall x,y.m(x) * w(y) * \top \mathrel{-\!\!*} cm(x,y) \vdash m(dj) \mathrel{-\!\!*} \exists w.cm(dj,w)}\;{-\!\!*R}
$$

Of course we don't want that! With a simple modification of the program it is possible to avoid this happening. The trick consists on changing the multiplicative implication by an additive one. We also need to take care of connecting this clause additively to the rest of the program.

```
((m(john) ; w(kate)),
 (m(abelard) ; w(eloise))
);


[x,y]can_marry(x,y) <- ((m(x) * w(y)) | (w(x) * m(y))) * T.
```

With these changes it is not possible for outsiders to take advantage of the situation. Any tentative of adding `m(don_juan)` to the program will fail to hold a proof of `<w> can_marry(don_juan,w)`.

$$\dfrac{\dfrac{\dfrac{\dfrac{\vdots}{(m(j);w(k)),(m(a);w(e)) \vdash m(dj)*w(k)*\top}\,\text{Fails} \qquad \dfrac{\vdots}{m(dj) \vdash I}\,\text{Fails}}{m(dj),(((m(j);w(k)),(m(a);w(e)));\forall x,y.m(x)*w(y)*\top \to cm(x,y)) \vdash cm(dj,k)}\,\to L}{m(dj),(((m(j);w(k)),(m(a);w(e)));\forall x,y.m(x)*w(y)*\top \to cm(x,y)) \vdash \exists w.cm(dj,w)}\,\exists R}{((m(j);w(k)),(m(a);w(e)));\forall x,y.m(x)*w(y)*\top \to cm(x,y) \vdash m(dj) \mathrel{-\!\!*} \exists w.cm(dj,w)}\,\mathrel{-\!\!*}R$$

Trying to add `m(dj)` *multiplicatively* to the program fails because the left rule for additive implication requires to find a proof with the additive subbunch connected to the implication, leaving the proposition `m(dj)` outside its scope.

Lets see what happens if we try to avoid this problem by adding `m(dj)` using an additive implication:

$$\dfrac{\dfrac{\dfrac{\dfrac{\vdots}{m(dj);((m(j);w(k)),(m(a);w(e))) \vdash m(dj)*w(k)*\top}\,\text{Fails}}{m(dj);((m(j);w(k)),(m(a);w(e)));\forall x,y.m(x)*w(y)*\top \to cm(x,y) \vdash cm(dj,k)}\,\to L}{m(dj);((m(j);w(k)),(m(a);w(e)));\forall x,y.m(x)*w(y)*\top \to cm(x,y) \vdash \exists w.cm(dj,w)}\,\exists R}{((m(j);w(k)),(m(a);w(e)));\forall x,y.m(x)*w(y)*\top \to cm(x,y) \vdash m(dj) \to \exists w.cm(dj,w)}\,\to R$$

Here the proof fails because the definition of `can_marry` requires the two propositions to be multiplicatively apart, and in this case `m(dj)` is additively apart from everyone else.

5.5 Linear Recursion

Linear recursion is problematic since the use of a linear implication consumes it and then it is not available for use later on. This can be overcome by repeating the implication in the program as many times as could be foreseen, if it can be foreseen at all. But operationally this solution opens up another problem, which is that on backtracking, each repetition of the clause can be chosen as a new match, giving a polynomial number of repeated answers. For example, the following program permutes lists of three elements.

```
[L]distribute(nil,L) *- collect(L),
[X,L1,L2]distribute(c(X,L1),L2) *- (item(X)-*distribute(L1,L2)),
[X,L1,L2]distribute(c(X,L1),L2) *- (item(X)-*distribute(L1,L2)),
[X,L1,L2]distribute(c(X,L1),L2) *- (item(X)-*distribute(L1,L2)),

collect(nil),
[X,L]collect(c(X,L)) *- item(X) * collect(L),
[X,L]collect(c(X,L)) *- item(X) * collect(L),
[X,L]collect(c(X,L)) *- item(X) * collect(L),

[L,K]permute(L,K) *- distribute(L,K).
```

But each solution is going to come up 36 times because of the different ways of picking up the repeated clauses. So we have implemented the following feature: after a linear implication the programmer can write a number, and this will mean that the clause can be used precisely that number of times, but matched only once on backtracking. So this same program could be written

```
[L]distribute(nil,L) *- collect(L),
[X,L1,L2]distribute(c(X,L1),L2) *-3 (item(X)-*distribute(L1,L2)),

collect(nil),
[X,L]collect(c(X,L)) *-3 item(X) * collect(L),

[L,K]permute(L,K) *- distribute(L,K).
```

and each solution will be found only once.

If we wanted to permute lists of length *up to* 3, we could do it by changing the program slightly, only adding an additive unit to the definition of `permute`:

```
[L]distribute(nil,L) *- collect(L),
[X,L1,L2]distribute(c(X,L1),L2) *-3 (item(X)-*distribute(L1,L2)),

collect(nil),
[X,L]collect(c(X,L)) *-3 item(X) * collect(L),


[L,K]permute(L,K) *- distribute(L,K) * T.
```

Unfortunately, the encapsulation of this behaviour is very poor. In particular, the same behaviour can be achieved using the previous program by adding the additive unit to the query, *i.e.*, changing the query from

```
  <x> permute(c(A,c(B,nil)),x)
```

which would fail, to

```
  <x> permute(c(A,c(B,nil)),x) * T
```

which will succeed, something that the programmer probably did not intend. In particular, if there are other sections of the code which rely on the fact that lists are of length precisely 3, this raises the possibility of breaking the program at these points.

A solution to this problem, which is similar to the solution found early for the marriage problem, is to change the definition of `permute` from a multiplicative implication to an additive implication. Therefore the definitive version of the program that permutes lists of length exactly 3 is

```
( [L]distribute(nil,L) *- collect(L),
  [X,L1,L2]distribute(c(X,L1),L2) *-3 (item(X)-*distribute(L1,L2)),


  collect(nil),
  [X,L]collect(c(X,L)) *-3 item(X) * collect(L)
);


[L,K]permute(L,K) <- distribute(L,K).
```

Now a query with the form `<x> permute(c(A,c(B,nil)),x)*T` will fail. The reason that this query fails for this program can be seen schematically from the fact that while it is possible to transform $P {-\!*}_3 Q, Q {-\!*} R$ into $P {-\!*}_2 Q, Q {-\!*} R, P {-\!*} Q$, with the possibility of sending the last clause to an eventual proof of $\top$, the same transformation would be invalid in the case of $P{-\!\!*}_3 Q;Q \rightarrow R$, which is not equivalent to $(P{-\!\!*}_2 Q;Q \rightarrow R), P{-\!\!*}Q$.

Most of the time it is not known in advance how many times an implication is going to be needed, for example when working with lists of arbitrary length. As an ad-hoc solution to this problem we have added the following facility: when the number written in a multiplicative implication is 0 it is taken to mean an infinite number of times. The suggested notation is as in the following program which permutes a list of arbitrary length:

```
( [L]distribute(nil,L) *- collect(L),
  [X,L1,L2]distribute(c(X,L1),L2) *-00 (item(X)-*distribute(L1,L2)),


  collect(nil),
  [X,L]collect(c(X,L)) *-00 item(X) * collect(L)
);


[L,K]permute(L,K) <- distribute(L,K) * T.
```

Notice that we need to put an additive unit in the definition of `permute` since there will always be clauses that are not consumed.

5.6 Other Idioms

Going back to the example about political parties, consider the following variation. Each group might have a boss which behaves as a person with respect to fights, but with the difference that bosses don't fight among themselves. This kind of restriction is expressed in BLPin a very simple way by adding a clause that can be used just once. The program becomes

```
[x]p(x) *- boss(x),
(p(A1) ; boss(A2) ; p(A3)),
(p(B1) ; boss(B2) ; p(B3)),
[x,y]fight(x,y) *- p(x)*p(y)*T.
```

Given this program, `A1` will fight with `B1`, `B2` and `B3`, but `A2` will fight only with `B1` and `B3`. The fact that `A2` will not fight with `B2` is enforced by the clause `[x]p(x) *- boss(x)`, which can be used only once, and the fact that `A2` and `B2` are bosses.

A further complication could be that one of the tribes has got a chief instead of a boss, even though it should behave in the same way than a boss with respect to fights. A slight modification of the program will accomplish this behaviour

```
[x]p(x)*- boss(x) | chief(x),
(p(A1);boss(A2);p(A3)),
(p(B1);chief(B2);p(B3)),

[x,y]fight(x,y)*- p(x)*p(y)*T
```

and `A2` and `B2` will not fight for the same reasons than before.

Consider a multiplicative bunch of additively connected clauses. If we want to express a collection of items, one from each region, we can do it in this way

```
(p(A1);p(A2)),
(p(B1);p(B2)),
(p(C1);p(C2)),

[x,tl]oneofeach(c(x,tl)) *-3 p(x)*oneofeach(tl),
oneofeach(nil).
```

Here we need a 3 following the implication because there are three groups. In case of more groups this number should be increased accordingly. The result is collected in a list.

A slight modification allows us to represent the case where the elements of the bunches are no tagged uniformly

```
(p(A1);q(A2)),
(q(B1);p(B2)),
(p(C1);q(C2)),


[x,tl]oneofeach(c(x,tl)) *-3 p(x)*oneofeach(tl),
oneofeach(nil).
```

Since the definition of `oneofeach` speaks explicitly of `p(x)`, then `A2`, `B1` and `C2` are not going to be picked up. But this behaviour can be changed by adding to the goal an extra clause which allows a `q` to become a `p`. This is done like this: `<x>([y]q(y)-*p(y))-*oneofeach(x)`. If we allow two `q`s to become `p`s then it can be achieved with `<x>([y]q(y)-*2p(y))-*oneofeach(x)`.

Also if the program data consisted of many different tags, different individuals can be selected at run time using implications in the goal. For example

```
(p(A1);q(A2);r(A3)),
(r(B1);p(B2);q(B3)),
(q(C1);r(C2);p(C3)),


[x,tl]oneofeach(c(x,tl)) *-3 t(x)*oneofeach(tl),
oneofeach(nil).
```

We could use the goal

```
<x>([y]p(y)-*3t(y)) -* oneofeach(x)
```

if we wanted to talk about the `p`s,

```
<x>([y]q(y)-*3t(y)) -* oneofeach(x)
```

if we want to refer to the `q`s or

```
<x>([z]p(z)-*t(z)) -* ([z]r(z)-*2t(z)) -* oneofeach(x)
```

if we are talking about two `r`s and one `p`, etc.

5.7 The $\alpha\lambda$-calculus

The basic system as presented in [O'Hearn Pym 99] and [O'Hearn 99] consists of the following rules:

$$\frac{}{x:A\vdash x:A}\,Id$$

$$\frac{\Gamma;x:A\vdash M:B}{\Gamma\vdash \alpha x.M:A\to B}\to I \qquad \frac{\Gamma\vdash M:A\to B \quad \Delta\vdash N:A}{\Gamma;\Delta\vdash M@N:B}\to E$$

$$\frac{\Gamma,x:A\vdash M:B}{\Gamma\vdash \lambda x.M:A\mathrel{-\!\!*}B}\mathrel{-\!\!*}I \qquad \frac{\Gamma\vdash M:A\mathrel{-\!\!*}B \quad \Delta\vdash N:A}{\Gamma,\Delta\vdash MN:B}\mathrel{-\!\!*}E$$

However we will work with an implicit version of $\to E$ and Id as described later in the paper

$$\frac{}{\Gamma;x:A\vdash x:A}\,Id \qquad \frac{\Gamma\vdash M:A\to B \quad \Gamma\vdash N:A}{\Gamma\vdash M@N:B}\to E$$

The problem mentioned in the paper does not arise because `BLP` takes care of the weakenings automatically.

To build a type checking program for the $\alpha\lambda$-calculus we will represent the judgements $\Gamma\vdash M:A$ in the following way: judgements will be of the form $\Gamma\vdash M:A$ where Γ is the collection of clauses forming the program. The variables will be represented by clauses of the form `v(x,A)`, read “`x` is a variable of type `A`”. The typed terms of the calculus will be represented by clauses of the form `alt(M,A)` and are read “`M` is an alpha-lambda-term of type `A`”.

First we will show two failed attempts to build a type-checker for the $\alpha\lambda$-calculus. This will serve to show some features of `BLP` and also some of its limitations.

5.7.1 A first failed attempt

The first and most immediate attempt is to define the rules using the additive implication

```
[x,A] alt(x,A) <- v(x,A);                                    # Id


[x,A,M,B] alt(alpha(x,M),aimp(A,B)) <- v(x,A) -> alt(M,B);    # -> I


[x,A,M,B] alt(lambda(x,M),mwand(A,B)) <-  v(x,A) -* alt(M,B); # -* I
```

```
[M,A,B,N] alt(att(M,N),B) <- alt(M,aimp(A,B)) & alt(N,A);     # -> E


[M,A,B,N] alt(app(M,N),B) <- alt(M,mwand(A,B)) * alt(N,A)     # -* E
```

We now explain the rules in turn. The first rule says that, for any x and A, $x : A$ is an $\alpha\lambda$-term if there is a variable x of type A, or, equivalently, if we can prove `v(x,A)`. This correspond to the Id rule.

The next rule, corresponding to $\to I$, says that if by augmenting *additively* the current context with an extra variable $x : A$ we manage to prove that $M : B$ is a valid $\alpha\lambda$-term, then we are allowed to conclude that in the current context $\alpha x.M : A \to B$ is also a valid $\alpha\lambda$-term.

The next rule corresponds to $\mathrel{-\!\!*}I$. It is similar to the previous one, but says that if we have the variable $x : A$ connected *multiplicatively* to the context then we can conclude that $\lambda x.M : A \mathrel{-\!\!*} B$ is a valid $\alpha\lambda$-term.

The rule for $\to E$ says that if we know that M is of type $A \to B$ and N is of type A we can conclude that $M@N$ is of type B. Similarly in the case of $\mathrel{-\!\!*}E$.

This program will fail for the multiplicative connectives. The reason why this is so can be seen from the following proof skeleton when we try to prove $\lambda y.y : P \mathrel{-\!\!*} P$. Here we use Γ^- as a shorthand for the rest of the program, *i.e.*, the program without the clause that appears explicitly. Also x, A, M and B are universally quantified throughout.

$$\dfrac{\dfrac{\dfrac{\dfrac{}{\Gamma \vdash v(y,P)}\ \text{Not provable} \qquad \dfrac{}{v(y,P) \vdash I}\ \text{Not provable}}{v(y,P),(\Gamma^-;v(x,A) \to x:A) \vdash y:P}\to L}{\Gamma \vdash v(y,P) \mathrel{-\!\!*} y:P}\mathrel{-\!\!*}R}{\Gamma^-;(v(x,A) \mathrel{-\!\!*} M:B) \to l(x,M):mw(A,B) \vdash l(y,y):mw(P,P)}\to L\ \text{(and unification)}$$

The problem in this version arises from the structure of the left additive implication rule

$$\dfrac{\Gamma;\phi \to \alpha \vdash \phi \qquad \Delta \vdash I}{\Delta(\Gamma;\phi \to \alpha) \vdash \alpha}\to L$$

which sends the subbunch where the additive implication is located, and the rest of the bunch to two different subproofs.

5.7.2 A second failed attempt

We could try changing the definition of the rules using multiplicative implications instead of additive ones. This would give as a result the following program:

```
[x,A] alt(x,A) *-OO v(x,A),                                      # Id


[x,A,M,B] alt(alpha(x,M),aimp(A,B)) *-OO v(x,A) -> alt(M,B),     # -> I


[x,A,M,B] alt(lambda(x,M),mwand(A,B)) *-OO  v(x,A) -* alt(M,B),  # -* I


[M,A,B,N] alt(att(M,N),B) *-OO alt(M,aimp(A,B)) & alt(N,A),      # -> E


[M,A,B,N] alt(app(M,N),B) *-OO alt(M,mwand(A,B)) * alt(N,A).     # -* E
```

But this time the additive connectives are the ones that cause trouble. When we try to prove $\alpha y.y : P \to P$ we get the following proof skeleton

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{\Gamma \vdash v(y:P)}\;\text{Not provable}}{\Gamma^{-}, v(x,A) \mathrel{-\!\!*}_{\infty} x : A \vdash y : P}\;{-\!\!*}L}{v(y,P);(\Gamma^{-}, v(x,A) \mathrel{-\!\!*}_{\infty} x : A) \vdash y : P}\;\text{Weakening}}{\Gamma \vdash v(y,P) \to y : P}\;\to R}{\Gamma^{-}, (v(x,A) \to M : B) \mathrel{-\!\!*}_{\infty} a(x,M) : mw(A,B) \vdash a(y,y) : mw(P,P)}\;{-\!\!*}L\ \text{(and unification)}$$

Of course, the problem now arises from the weakening of $v(y,P)$. But, considering the proof search from the bottom up, we are forced to perform the weakening if we are going to use the ${-\!\!*}L$ rule, which expects the implication to be at the top level.

5.7.3 The working version

The problem with the previous versions is that the context where the program lives cannot be used as a context to keep the additive or multiplicative relation between the variables.

$$\dfrac{\Gamma \vdash I}{\Gamma, v(x,A) \vdash alt(x,A)}\;Id$$

$$\dfrac{\Gamma ; v(x,A) \vdash alt(M,B)}{\Gamma \vdash alt(\alpha x.M, A \to B)}\to I \qquad \dfrac{\Gamma \vdash alt(M, A \to B) \quad \Gamma \vdash alt(N,A)}{\Gamma \vdash alt(M@N, B)}\to E$$

$$\frac{\Gamma, v(x,A) \vdash alt(M,B)}{\Gamma \vdash alt(\lambda x.M, A -\!\!* B)} -\!\!*I \qquad \frac{\Gamma \vdash alt(M, A -\!\!* B) \quad \Delta \vdash alt(N,A)}{\Gamma, \Delta \vdash alt(MN, B)} -\!\!*E$$

A basic program that performs type checking or type inference of terms in the $\alpha\lambda$-calculus can be implemented using BLP as follows:

```
[x,A] alt(x,A) *-OO v(x,A)*T,                                            # Id


[x,A,M,B] alt(alpha(x,M),aimp(A,B)) *-OO (v(x,A) *-OO I) -* alt(M,B),  # -> I


[x,A,M,B] alt(lambda(x,M),mwand(A,B)) *-OO  v(x,A) -* alt(M,B),        # -* I


[M,A,B,N] alt(att(M,N),B) *-OO alt(M,aimp(A,B)) & alt(N,A),            # -> E


[M,A,B,N] alt(app(M,N),B) *-OO alt(M,mwand(A,B)) * alt(N,A).           # -* E
```

If we ask to type the term $\lambda x.\alpha y.y$ we get the run

```
Γ ?- <type> alt(lambda(x,alpha(y,y)),type)


type <- A -* (B -> B)
```

and if we try to type the term $\lambda x.\alpha f.(f@x)@x$ we get

```
Γ ?- <type> alt(lambda(x,alpha(f,att(att(f,x),x))),type)


type <- A -* ((A -> (A -> B)) -> B)
```

If we ask the program to find *term* and *type* such that `alt(app(term,t),type)` is well typed we are answered with

```
Γ ?- <term,type> alt(app(term,t),type)


term <- λx.αt.t
type <- B -> B
```

```
; # backtracking
term <- λx.αy.αt.t
type <- B -> (C -> C)
...
```

5.8 Modules

Suppose we have a **BI** program that solves the Towers of Hanoi problem in a file called Hanoi.bi

```
(number).


hanoi(number) *- move(number,left,center,right),
(
[a,b,c]move(o,a,b,c);
[x,a,b,c]move(s(x),a,b,c) <- move(x,a,c,b) & write(a,goes,to,b) & nl
        &  move(x,c,b,a)
).
```

This code can be used from another program in the following way

```
[x] start(x) *- (Hanoi(x)->hanoi(x)).
```

When presented with the goal `start(s(s(s(o))))` the goal

```
Hanoi(s(s(s(o))))->hanoi(s(s(s(o))))
```

will be tried. Now, since the antecedent is capitalised, it will try to load the program Hanoi.bi, if it exists in the current directory. After this, the goal `hanoi(s(s(s(o))))` will be solved in the new program created by replacing all occurrences of `number` by `s(s(s(o)))`.

Chapter 6

Denotational Semantics

6.1 Introduction

The traditional way of providing a denotational semantics for logic programming is giving a least Herbrand model, the set of all atoms provable from a particular program [Lloyd 87] which can be characterised by a least fixpoint [vanEmden Kowalski]

However, since this logic programming language admits a dynamic growth of programs via $\to R$ and $-\!\!*R$ rules, we provide a denotational semantics in the spirit of [Miller 81], where programs are interpreted by Kripke models. We need to point out that the model provided does not completely capture the operational behaviour of programs (*i.e.*, is not a fully abstract model). In particular the usual issues related to permutation of clauses and their possibly different behaviour are ignored by this semantics.

In this chapter we

- Define Herbrand interpretations for programs.
- Provide a fix point semantics for BLP.
- Show soundness and completeness for this semantics.

6.2 Hereditary Harrop formulæ for BI

Recall Definition 3.4.1:

$$
\begin{array}{llcl}
\text{Definite formulæ} & D & ::= & A \mid I \mid D * D \mid D \wedge D \mid G \mathrel{-\!\!*} A \mid G \to A \\
 & & & \mid \forall x.D \\[2ex]
\text{Goal formulæ} & G & ::= & \top \mid I \mid A \mid G * G \mid G \wedge G \mid G \vee G \mid D \mathrel{-\!\!*} G \mid D \to G \\
 & & & \mid \exists x.G
\end{array}
$$

This class of formulæ is not quite the largest we could consider. We could add at least the following without compromising completeness of resolution proofs:

- Inconsistency, $\bot$, to allow, for example, definite clauses of the form $G \to \bot$ and goals of the form $\bot$;
- Definite clauses of the form $\forall_{\mathsf{new}} x.D$ and goal clauses of the form $\exists_{\mathsf{new}} x.G$;
- Goal clauses of the form $\forall x.G$ and $\forall_{\mathsf{new}} x.G$.

We also conjecture that a variety of extensions to the implicational definite formulæ, admitting formulæ more complex than atoms in the positive position, are possible.

Although programs contain additively universally quantified clauses and goals contain additively existentially quantified "logical variables", we restrict our analysis here to the propositional structure. This restriction represents no loss of generality the first-order structure being handled operationally by unification, but considerably simplifies the presentation of the semantics. If we were to include not only additively quantified program clauses and goals but also their multiplicative counterparts, then it would be necessary to handle the quantifiers explicitly, probably within an infinitary extension of Definition 3.9.1 (*cf.* [Miller 81]).

Recall that the definition of clausal decomposition (Definition 3.9.1) is

$$
\begin{array}{rcl}
[P(A)] & ::= & P(A) \\
[P(D * D)] & ::= & P([D],[D]) \\
[P(D \wedge D)] & ::= & P([D];[D]) \\
[P(G \mathrel{-\!\!*} A)] & ::= & P(G \mathrel{-\!\!*} A) \\
[P(G \to A)] & ::= & P(G \to A)
\end{array}
$$

and that a *resolution proof* in **BI** is defined by

$$\frac{P \vdash_R I}{P, A \vdash_R A}\,\text{CutAxiom} \qquad \frac{P \vdash_R G}{P, G \mathrel{-\!\!*} A \vdash_R A} \mathrel{-\!\!*} Res \qquad \frac{P' \vdash_R G \quad P \vdash_R I}{P, (P'; G \to A) \vdash_R A} \to Res$$

$$\frac{P_1 \vdash_R G_1 \quad P_2 \vdash_R G_2}{P_1, P_2 \vdash_R G_1 * G_2} * R \qquad \frac{P, [D] \vdash_R G}{P \vdash_R D \mathrel{-\!\!*} G} \mathrel{-\!\!*} R$$

$$\frac{P \vdash_R G_1 \quad P \vdash_R G_2}{P \vdash_R G_1 \wedge G_2} \wedge R \qquad \frac{P \vdash_R G_i}{P \vdash_R G_1 \vee G_2}\,(i = 1, 2)\ \vee R \qquad \frac{P; [D] \vdash_R G}{P \vdash_R D \to G} \to R$$

6.3 A Denotational Semantics for BI

Definition 6.3.1 *Let $(\mathcal{P}, \cdot, e, \sqsubseteq)$ be the commutative monoid of programs in which $\cdot$ is given by ",", e is given by $\varnothing_m$ and $P \sqsubseteq P'$ just in case there is a P'' such that $P \equiv P'; P''$. We consider models in the category $[\mathcal{P}^{op}, \mathbf{Set}]$ with the interpretation of propositions* $\mathrm{Prop}(\mathcal{L})$ *over a signature of propositional atoms $\mathcal{L}$ given by a partial function*

$$[\![-]\!] : \mathrm{Prop}(\mathcal{L}) \rightharpoonup obj([\mathcal{P}^{op}, \mathbf{Set}])$$

such that $[\![G]\!](P) \subseteq \{\mathcal{R} | \mathcal{R} : P \vdash_R G\}$.

Definition 6.3.2 *$P \models G$ iff $[\![G]\!](P) \neq \emptyset$*

Lemma 6.3.3 *The forcing relation, $\models$, between interpretation programs (worlds), P, and goal formulæ, G, satisfies the following:*

$$\begin{array}{rcl}
P \models \top & & \text{always} \\
P \models I & \text{iff} & P \sqsubseteq \varnothing_m \\
P \models A & \text{iff} & [\![A]\!](P) \neq \emptyset \\
P \models G_1 \wedge G_2 & \text{iff} & P \models G_1 \textit{ and } P \models G_2 \\
P \models D \to G & \text{iff} & \textit{for all } P' \sqsubseteq P, \textit{ if } P' \models D \textit{ then } P' \models G \\
P \models G_1 \vee G_2 & \text{iff} & P \models G_1 \textit{ or } P \models G_2 \\
P \models G_1 * G_2 & \text{iff} & \textit{there are } P_1 \textit{ and } P_2, \textit{ where the relation } P \sqsubseteq P_1, P_2 \\
 & & \textit{holds, such that } P_1 \models G_1 \textit{ and } P_2 \models G_2 \\
P \models D \mathrel{-\!\!*} G & \text{iff} & \textit{for all } P', \textit{ if } P' \models D, \textit{ then } P, P' \models G
\end{array}$$

Proof: By induction on the structure of the goal formulæ. We show some cases:

$\top$ Any P produces a proof $P \vdash_R \top$, and therefore $[\![\top]\!](P)$ is always non-empty.

I By observation of the rule Unit, the only P that can prove I is $\varnothing_m$. So, for any $\mathcal{R} : P \vdash_R I$, it must be the case that $P \sqsubseteq \varnothing_m$.

A Immediate from Definition 6.3.2

$G_1 \wedge G_2$ Given that $[\![G_1 \wedge G_2]\!](P) \neq \emptyset$, then there exists $\mathcal{R} : P \vdash_R G_1 \wedge G_2$. Then there are $\mathcal{R}_1$ and $\mathcal{R}_2$ such that $\mathcal{R}_1 : P \vdash_R G_1$ and $\mathcal{R}_2 : P \vdash_R G_2$. This shows that $[\![G_1]\!](P) \neq \emptyset$ and $[\![G_2]\!](P) \neq \emptyset$, and therefore $P {\models} G_1$ and $P {\models} G_2$. Here we are using the $\wedge I$ rule with implicit contraction, which is admissible by Lemma 2.6.2.

$G_1 * G_2$ Given that $[\![G_1 * G_2]\!](P) \neq \emptyset$, then there exists $\mathcal{R} : P \vdash_R G_1 * G_2$

$D {-\!\!*} G$ We have that there exists a proof $\mathcal{R} : P \vdash_R D {-\!\!*} G$. For any P_1 that forces D, we have that $\mathcal{R}' : P_1 \vdash_R D$. Putting these two proofs together using ${-\!\!*}E$ we find a proof $\mathcal{R}'' : P, P_1 \vdash_R G$, which shows that $P, P_1 {\models} G$.

The remaining cases are similar.

□

Lemma 6.3.4 (deduction-theorem) *The forcing clauses for the two implications may be expressed as follows:*

1. *$P {\models} D \to G$ iff $P; [D] {\models} G$;*

2. *$P {\models} D {-\!\!*} G$ iff $P, [D] {\models} G$.*

Proof:

1.

$$\begin{array}{lll} P {\models} D \to G & \text{iff} & [\![D \to G]\!](P) \neq \emptyset \\ & \text{iff} & \text{there exists } \mathcal{R} : P \vdash_R D \to G \\ & \text{iff} & \text{there exists } \mathcal{R} : P; [D] \vdash G \\ & \text{iff} & [\![G]\!](P; [D]) \neq \emptyset \\ & \text{iff} & P; [D] {\models} G \end{array}$$

2.

$$\begin{array}{rll} P{\models}D{-\!\!*}G & \text{iff} & [\![D{-\!\!*}G]\!](P) \neq \emptyset \\ & \text{iff} & \text{there exists } \mathcal{R} : P \vdash_R D{-\!\!*}G \\ & \text{iff} & \text{there exists } \mathcal{R} : P, [D] \vdash G \\ & \text{iff} & [\![G]\!](P, [D]) \neq \emptyset \\ & \text{iff} & P, [D]{\models}G \end{array}$$

□

Definition 6.3.5 (Herbrand interpretation) *We define a Herbrand interpretation of programs*

$$\{\!\{-\}\!\}_H : \mathcal{P}^{op} \longrightarrow \textbf{Set},$$

by $\{\!\{P\}\!\}_H = \bigcup_{A \in G} [\![A]\!](P)$*, where* A *is atomic.*

The interpretation $\{\!\{-\}\!\}$ gives the meaning of a program P with respect to a given goal G: $\{\!\{G\}\!\}_P$

We seek a semantics for programs which is independent of any particular choice of goal. Accordingly, we consider the meaning of a program as being that which is determined by all possible (atomic) goals.

Thus, a Herbrand interpretation of a program P is given in the framework of **BI**'s model theory as follows:

Definition 6.3.6 (forcing) *There is a forcing relation* $\models$*, between Herbrand interpretations* $\{\!\{-\}\!\}_H$*, programs (worlds),* P*, and goal formulæ,* G*, which satisfies the follow-*

ing:

$$\begin{array}{llll}
\{\!\{-\}\!\}_H & , P \VDash \top & & \text{always} \\
\{\!\{-\}\!\}_H & , P \VDash I & \text{iff} & P \sqsubseteq \varnothing_m \\
\{\!\{-\}\!\}_H & , P \VDash A & \text{iff} & \textit{there is some } \mathcal{R} : P \vdash_R A \in \{\!\{P\}\!\}_H \\
\{\!\{-\}\!\}_H & , P \VDash G_1 \wedge G_2 & \text{iff} & \{\!\{-\}\!\}_H, P \VDash G_1 \textit{ and } \{\!\{-\}\!\}_H, P \VDash G_2 \\
\{\!\{-\}\!\}_H & , P \VDash D \to G & \text{iff} & \{\!\{-\}\!\}_H, P; [D] \VDash G \\
\{\!\{-\}\!\}_H & , P \VDash G_1 \vee G_2 & \text{iff} & \{\!\{-\}\!\}_H, P \VDash G_1 \textit{ or } \{\!\{-\}\!\}_H, P \VDash G_2 \\
\{\!\{-\}\!\}_H & , P \VDash G_1 * G_2 & \text{iff} & \textit{there are } P_1 \textit{ and } P_2\textit{, where the relation } P \sqsubseteq \\
 & & & P_1, P_2 \textit{ holds, such that } \{\!\{-\}\!\}_H, P_1 \VDash G_1 \textit{ and} \\
 & & & \{\!\{-\}\!\}_H, P_2 \VDash G_2 \\
\{\!\{-\}\!\}_H & , P \VDash D \mathrel{-\!\!*} G & \text{iff} & \{\!\{-\}\!\}_H \ , \ (P, [D]) \VDash G
\end{array}$$

Let $\mathcal{H}$ denote the set of all Herbrand interpretations. Given two Herbrand interpretations, H_1 and H_2, we can define the following operations:

- $\{\!\{-\}\!\}_{H_1} \sqsubseteq \{\!\{-\}\!\}_{H_2}$ iff for all programs P, $\{\!\{P\}\!\}_{H_1} \subseteq \{\!\{P\}\!\}_{H_2}$;
- $\{\!\{P\}\!\}_{H_1 \sqcup H_2} ::= \{\!\{P\}\!\}_{H_1} \cup \{\!\{P\}\!\}_{H_2}$;
- $\{\!\{P\}\!\}_{H_1 \sqcap H_2} ::= \{\!\{P\}\!\}_{H_1} \cap \{\!\{P\}\!\}_{H_2}$;
- $\{\!\{P\}\!\}_{H_\perp} = \emptyset$, for all P.

Lemma 6.3.7 (lattice) *With the definitions above, $\mathcal{H}$ is a complete lattice.*

Proof: Each $\{\!\{-\}\!\}_H$ is a map from programs (worlds) to the powerset of the set of proofs of sequents of the form $P \vdash_R A$, where P is a program and A is atomic, which is a complete lattice.

□

We now give a sequence of lemmas which explains the relationship between Herbrand interpretations ($\{\!\{-\}\!\}_H$) and forcing ($\VDash$)

Lemma 6.3.8 ($\sqsubseteq$ respects $\VDash$) *If $\{\!\{-\}\!\}_{H_1} \sqsubseteq \{\!\{-\}\!\}_{H_2}$, then, for all programs P and goals G, $\{\!\{-\}\!\}_{H_1}, P \VDash G$ implies $\{\!\{-\}\!\}_{H_2}, P \VDash G$*

Proof: By induction on the structure of G. In all cases we use the ordering defined above, that is $\{\{-\}\}_{H_1} \sqsubseteq \{\{-\}\}_{H_2}$ means that for all programs $\{\{P\}\}_{H_1} \subseteq \{\{P\}\}_{H_2}$. We show some cases:

G **atomic:** From Definition 6.3.6 $P \Vdash G$ under $\{\{P\}\}_{H_1}$ if and only if there is some proof $\mathcal{R} : P \vdash_R G$ in $\{\{P\}\}_{H_1}$. Therefore this proof is also in $\{\{P\}\}_{H_2}$, which means that $\{\{-\}\}_{H_2}, P \Vdash G$.

$G = D \to G$**:** We have that $\{\{-\}\}_{H_1}, P \Vdash D \to G$, which means that $\{\{-\}\}_{H_1}, P; [D] \Vdash G$. By induction hypothesis we know that $\{\{-\}\}_{H_2}, P; [D] \Vdash G$, which means that $\{\{-\}\}_{H_2}, P \Vdash D \to G$.

$G = D \mathrel{-\!\!*} G$**:** We have that $\{\{-\}\}_{H_1}, P \Vdash D \mathrel{-\!\!*} G$, which means that $\{\{-\}\}_{H_1}, (P, [D]) \Vdash G$. By induction hypothesis we know that $\{\{-\}\}_{H_2}, (P, [D]) \Vdash G$, which means that $\{\{-\}\}_{H_2}, P \Vdash D \mathrel{-\!\!*} G$.

$G = G_1 * G_2$**:** We have that $\{\{-\}\}_{H_1}, P \Vdash G_1 * G_2$, which means that there are P_1 and P_2 such that $P \sqsubseteq P1, P2$ and such that $\{\{-\}\}_{H_1}, P_1 \Vdash G_1$ and $\{\{-\}\}_{H_1}, P_2 \Vdash G_2$. By induction hypothesis we know that $\{\{-\}\}_{H_2}, P_1 \Vdash G_1$ and $\{\{-\}\}_{H_2}, P_2 \Vdash G_2$, and therefore $\{\{-\}\}_{H_2}, P \Vdash G_1 * G_2$

The remaining cases are similar.

□

Lemma 6.3.9 ($\Vdash$ at finite height) *Let $\{\{-\}\}_{H_1} \sqsubseteq \{\{-\}\}_{H_2} \sqsubseteq \ldots$ be an ω-chain and suppose that, for some program P and goal G,*

$$\bigsqcup_{0 \leq i < \omega} \{\{-\}\}_{H_i}, P \Vdash G,$$

then there is a $k \geq 1$ such that $\{\{-\}\}_{H_k}, P \Vdash G$.

Proof: By induction on the structure of G. Suppose G is atomic. If we have $\bigsqcup_{0 \leq i < \omega} \{\{-\}\}_{H_i}, P \Vdash G$, then there is some $\mathcal{R} : P \vdash_R G \in \bigcup_{0 \leq i < \omega} H_i(P)$. Therefore there is a k such that $\mathcal{R} \in H_k(P)$, so that $\{\{-\}\}_{H_k}, P \Vdash G$, as required.

For the inductive step, we suppose that the lemma holds for all goal formulæ of a given bounded complexity and proceed by cases. We give just two examples.

$G \equiv G_1 * G_2$**:** If $\bigsqcup_{0 \leq i < \omega} \{\!\{-\}\!\}_{H_i}, P \Vdash G_1 * G_2$, then, by the definition of $\Vdash$, there are P_1 and P_2, where $P_1, P_2 \sqsubseteq P$, such that

$$\bigsqcup_{0 \leq i < \omega} \{\!\{-\}\!\}_{H_i}, P_1 \Vdash G_1 \quad \text{and} \quad \bigsqcup_{0 \leq i < \omega} \{\!\{-\}\!\}_{H_i}, P_2 \Vdash G_2.$$

But each of G_1 and G_2 is smaller that G, and so, by the induction hypothesis, there are j and j' such that $\{\!\{-\}\!\}_{H_j}, P_1 \Vdash G_1$ and $\{\!\{-\}\!\}_{H_{j'}}, P_2 \Vdash G_2$. By Lemma 6.3.8 we can let k be the greater of j and j'.

$G \equiv D \mathrel{-\!\!*} G'$**:** If $\bigsqcup_{0 \leq i < \omega} \{\!\{-\}\!\}_{H_i}, P \Vdash D \mathrel{-\!\!*} G'$, then, by the definition of $\Vdash$, it is be the case that $\bigsqcup_{0 \leq i < \omega} \{\!\{-\}\!\}_{H_i}, (P, [D]) \Vdash G'$. But G' is smaller than G. Then by the induction hypothesis, there is a k such that $\{\!\{-\}\!\}_{H_k}, (P, [D]) \Vdash G'$ and we can conclude that $\{\!\{-\}\!\}_{H_k}, P \Vdash D \mathrel{-\!\!*} G'$, as required.

The remaining cases are similar.

□

Now, given this notion of Herbrand model, which amounts to a special case of the general notion of model explained in [Pym 02, Pym et al. 02], we can pick out a computationally useful semantics for programs, P. Whereas the truth of a goal G, with respect to a program P, in a given *Herbrand interpretation*, H, defined below, of **BI** is determined by the existence or not of a a proof of G from P in the image of H, the *denotation* of a program must account for all of the possible computations which P can support. Consequently, we define the meaning of a program, in a given Herbrand interpretation, H, to be a subset of the collection of all possible computations, here represented by proofs, of which it is capable.

We can now define an operator, T, on Herbrand interpretations which iteratively constructs a model corresponding to the execution of BLP programs. There are three cases in the iteration. The first corresponds to instances of the CutAxiom rule,

$$\frac{P \vdash_R I}{P, A \vdash_R A}\ \text{CutAxiom}$$

The second corresponds to the $\mathrel{-\!\!*} Res$ rule,

$$\frac{P \vdash_R G}{P, G \mathrel{-\!\!*} A \vdash_R A}\ \mathrel{-\!\!*} Res$$

And the third corresponds to the $\to Res$ rule,

$$\frac{P' \vdash_R G \quad P \vdash_R I}{P, (P'; G \to A) \vdash_R A} \to Res$$

Definition 6.3.10 (the operator T**)** *We define an operator* $T : \mathcal{H} \longrightarrow \mathcal{H}$ *as follows:*

$$\begin{aligned} \{\!\{P\}\!\}_{T(H)} \quad = \quad & \{ \ \mathcal{R} \mid \mathcal{R} : P \vdash_R A \ and \ \{\!\{-\}\!\}_H, P' \Vdash I \ \} \ (P \equiv P', A) \\ & \cup \\ & \{ \ \mathcal{R} \mid \mathcal{R} : P \vdash_R A, \ G \multimap\!\!* A \in P \ \ and \ \{\!\{-\}\!\}_H, P' \Vdash G \ \} \\ & (P \equiv P', G \multimap\!\!* A) \\ & \cup \\ & \{ \ \mathcal{R} \mid \mathcal{R} : P \vdash_R A, \ G \to A \in P, \ \{\!\{-\}\!\}_H, P' \Vdash G \ \ and \\ & \{\!\{-\}\!\}_H, P'' \Vdash I \ \} \ (P \equiv P'', (P'; G \to A)) \end{aligned}$$

We now give a series of lemmas which explain the relation between Herbrand interpretations ($\{\!\{-\}\!\}$), forcing ($\Vdash$) and the T operator:

Lemma 6.3.11 (monotonicity) *Let* $\{\!\{-\}\!\}_{H_1} \sqsubseteq \{\!\{-\}\!\}_{H_2}$ *be Herbrand interpretations. Then* $\{\!\{-\}\!\}_{T(H_1)} \sqsubseteq \{\!\{-\}\!\}_{T(H_2)}$.

Proof: Suppose $\{\!\{-\}\!\}_{H_1} \sqsubseteq \{\!\{-\}\!\}_{H_2}$ and let $\mathcal{R} : P \vdash_R A \in \{\!\{P\}\!\}_{T(H_1)}$. We must be in one of three cases, corresponding to the three components of the definition of T

- We are in the *CutAxiom* case and must therefore also have that

  $$\{\!\{-\}\!\}_{T(H_1)}, P''' \Vdash I$$

  and so also $\{\!\{-\}\!\}_{T(H_2)}, P''' \Vdash I$, so that $\mathcal{R} : P \vdash_R A \in \{\!\{P\}\!\}_{T(H_2)}$. Then it follows that $\{\!\{-\}\!\}_{T(H_1)} \sqsubseteq \{\!\{-\}\!\}_{T(H_2)}$.

- There is some $G \multimap\!\!* A$ in P such that

  $$\{\!\{-\}\!\}_{H_1}, P' \Vdash G.$$

  By Lemma 6.3.8, we must also have that

  $$\{\!\{-\}\!\}_{H_2}, P' \Vdash G,$$

  so that $\mathcal{R} : P \vdash_R A \in \{\!\{P\}\!\}_{T(H_2)}$. Therefore $\{\!\{-\}\!\}_{T(H_1)} \sqsubseteq \{\!\{-\}\!\}_{T(H_2)}$.

- There is some $G \to A$ in P such that

  $$\{\!\{-\}\!\}_{H_1}, P' \Vdash G \quad \text{and} \quad \{\!\{-\}\!\}_{H_1}, P'' \Vdash I.$$

  By Lemma 6.3.8, we must also have that

  $$\{\!\{-\}\!\}_{H_2}, P' \Vdash G \quad \text{and} \quad \{\!\{-\}\!\}_{H_2}, P'' \Vdash I,$$

  so that $\mathcal{R} : P \vdash_R A \in \{\!\{P\}\!\}_{T(H_2)}$. Therefore $\{\!\{-\}\!\}_{T(H_1)} \sqsubseteq \{\!\{-\}\!\}_{T(H_2)}$.

It follows that $\{\!\{-\}\!\}_{T(H_1)} \sqsubseteq \{\!\{-\}\!\}_{T(H_2)}$.

□

Lemma 6.3.12 (continuity) *Let* $\{\!\{-\}\!\}_{H_0} \sqsubseteq \{\!\{-\}\!\}_{H_1} \sqsubseteq \{\!\{-\}\!\}_{H_2} \sqsubseteq \ldots$ *be an* ω*-chain of Herbrand interpretations. Then*

$$\{\!\{-\}\!\}_{T(\bigsqcup_{0 \leq i < \omega} H_i)} = \{\!\{-\}\!\}_{\bigsqcup_{0 \leq i < \omega} T(H_i)}$$

Proof: The required equality is established by proving inclusion each way round.

Suppose, for an arbitrary j, that $\{\!\{-\}\!\}_{H_j} \sqsubseteq \{\!\{-\}\!\}_{\bigsqcup_{0 \leq i < \omega} H_i}$. By Lemma 6.3.11, we get

$$\{\!\{-\}\!\}_{T(H_j)} \sqsubseteq \{\!\{-\}\!\}_{T(\bigsqcup_{0 \leq i < \omega} H_i)}$$

Since j is arbitrary, we get

$$\{\!\{-\}\!\}_{\bigsqcup_{0 \leq i < \omega} T(H_i)} \sqsubseteq \{\!\{-\}\!\}_{T(\bigsqcup_{0 \leq i < \omega} H_i)}.$$

Conversely, suppose $\mathcal{R} : P \vdash_R A \in \{\!\{P\}\!\}_{T(\bigsqcup_{0 \leq i < \omega} H_i)}$. We must be in one of three cases, corresponding to the three components of the definition of T.

- We are in the *CutAxiom* case and must therefore also have that

  $$\{\!\{-\}\!\}_{T(\bigsqcup_{0 \leq i < \omega} H_i)}, P''' \Vdash I.$$

  In this case, $\mathcal{R} \in \{\!\{P\}\!\}_{T(H_j)}$, for any $j \geq 1$. Therefore, $\mathcal{R} \in \{\!\{P\}\!\}_{\bigsqcup_{0 \leq i < \omega} T(H_i)}$.

- There is some $G {-\!\!*} A$ in P such that

  $$\{\!\{-\}\!\}_{\bigsqcup_{0 \leq i < \omega} H_i}, P' \Vdash G.$$

By Lemma 6.3.9, there is a k such that

$$\{\!\{-\}\!\}_{H_k}, P' \Vdash G.$$

Therefore, $\mathcal{R} \in \{\!\{P\}\!\}_{T(H_k)} \subseteq \{\!\{P\}\!\}_{\bigsqcup_{0 \leq i < \omega} T(H_i)}$.

- There is some $G \to A$ in P such that

$$\{\!\{-\}\!\}_{\bigsqcup_{0 \leq i < \omega} H_i}, P' \Vdash G \quad \text{and} \quad \{\!\{-\}\!\}_{\bigsqcup_{0 \leq i < \omega} H_i}, P'' \Vdash I.$$

By Lemma 6.3.9, there are k' and k'' such that

$$\{\!\{-\}\!\}_{H_{k'}}, P' \Vdash G \quad \text{and} \quad \{\!\{-\}\!\}_{H_{k''}}, P'' \Vdash I.$$

Taking k to be the greater of k' and k'', we get

$$\{\!\{-\}\!\}_{H_k}, P' \Vdash G \quad \text{and} \quad \{\!\{-\}\!\}_{H_k}, P'' \Vdash I.$$

So $\mathcal{R} \in \{\!\{P\}\!\}_{T(H_k)} \subseteq \{\!\{P\}\!\}_{\bigsqcup_{0 \leq i < \omega} T(H_i)}$.

It follows that

$$\{\!\{-\}\!\}_{T(\bigsqcup_{0 \leq i < \omega} H_i)} \sqsubseteq \{\!\{-\}\!\}_{\bigsqcup_{0 \leq i < \omega} T(H_i)},$$

as required.

□

So we have that T is a monotone, continuous operator on a complete lattice, so it follows, by the the Knaster-Tarski fixed point theorem [Tarski 56, Lloyd 87], that T has a least fixed point given by

$$\{\!\{-\}\!\}_{T^\omega(H_\perp)} = \bigsqcup_{0 \leq i < \omega} \{\!\{-\}\!\}_{T^i(H_\perp)}$$

Having constructed a least fixed point of the operator, we must show that we have indeed provided an interpretation of programs within a model of **BI**.

Lemma 6.3.13 ($\{\!\{-\}\!\}_{T^\omega(H_\perp)}$ interprets within a model) *The Herbrand interpretation $\{\!\{-\}\!\}_{T^\omega(H_\perp)}$ provides an interpretation of programs P within a model of the hereditary Harrop fragment of* **BI**.

Proof: A straightforward calculation. First, we recall that the collection $\mathcal{P}$ of programs is a preordered monoid, with the monoidal combination of P and P' given by P, P' and the ordering being defined by $P \sqsubseteq P'$ iff, for some P'', it is the case that $P \equiv P'; P''$, which may be considered a small monoidal category. The required model is based on the category of presheaves, $[\mathcal{P}^{op}, \mathbf{Set}]$.

Next, we must define the interpretation map, $\{\!\{-\}\!\}$, with respect to this category. We define

$$\{\!\{-\}\!\} : \mathrm{Prop}(\mathcal{L}) \rightharpoonup obj[\mathcal{P}^{op}, \mathbf{Set}]$$

pointwise by

$$\{\!\{\phi\}\!\}(P) = \begin{cases} \{\mathcal{R} | \mathcal{R} : P \vdash_R \phi\} & \text{if } \phi \text{ is atomic} \\ \uparrow & \text{otherwise} \end{cases}$$

It follows that, with the forcing relation $\Vdash$, the category of presheaves, $[\mathcal{P}^{op}, \mathbf{Set}]$ is a model of propositional **BI** (without $\perp$). Note that we haven't exploited Day's tensor product directly, since we have defined $\{\!\{-\}\!\}$ only on atoms, but we have exploited it in purely propositional form via $\Vdash$.

□

The integration of $\perp$ should, given the analysis in [Pym 02, Pym et al. 02], be relatively straightforward.

Lemma 6.3.14 (soundness) *If $P \vdash_R G$, then $\{\!\{-\}\!\}_{T^{\omega}(H_{\perp})}, P \Vdash G$.*

Proof: By induction on the structure of proofs.

- Suppose the last rule applied is the $\top$ rule,

$$\frac{}{P \vdash_R \top}$$

Then we have immediately that

$$\{\!\{-\}\!\}_{T^{\omega}(H_{\perp})}, P \Vdash \top,$$

since $\{\!\{-\}\!\}_{T^{\omega}(H_{\perp})}, P \Vdash \top$ always.

- Suppose the last rule applied is the I rule,

  $$\frac{}{\emptyset_m \vdash_R I}$$

  We have immediately that

  $$\{\!\{-\}\!\}_{T^\omega(H_\bot)}\,,\ \emptyset_m \VDash I,$$

  since $\emptyset_m \sqsubseteq \emptyset_m$.

- Suppose the last rule applied is CutAxiom,

  $$\frac{P''' \vdash_R I}{P \vdash_R A}\ P \equiv P''', A$$

  By the induction hypothesis, we have that $\{\!\{-\}\!\}_{T^\omega(H_\bot)}\,,\ \emptyset_m \VDash I$ and, by assumption, we have $P''', A \vdash_{\mathcal{R}} A$, so that we have $\{\!\{-\}\!\}_{T^\omega(H_\bot)}\,,\ P \VDash A$, as required.

- Suppose the last rule applied is $\rightarrow_{Res}$,

  $$\frac{P' \vdash_R G \quad P'' \vdash_R I}{P \vdash_R A} \rightarrow_{Res}$$

  where $P \equiv P'', (P'; G \rightarrow A)$.

  By the induction hypothesis, we have that

  $$\{\!\{-\}\!\}_{T^\omega(H_\bot)}\,,\ P' \VDash G$$

  and

  $$\{\!\{-\}\!\}_{T^\omega(H_\bot)}\,,\ P'' \VDash I$$

  Therefore, by the definition of T, there is a proof of $P \vdash_R A$ in $\{\!\{P\}\!\}_{T(T^\omega(H_\bot))}$. But $\{\!\{P\}\!\}_{T^\omega(H_\bot)}$ is a fixed point, so there is a proof of $P \vdash_R A$ in $\{\!\{P\}\!\}_{T^\omega(H_\bot)}$. Therefore

  $$\{\!\{-\}\!\}_{T^\omega(H_\bot)}\,,\ P \VDash A,$$

  as required.

- Suppose the last rule applied is $-\!\!*_{Res}$,

  $$\frac{P' \vdash_R G}{P \vdash_R A} -\!\!*_{Res}$$

  where $P \equiv P', G -\!\!* A$. By the induction hypothesis, we have that

  $$\{\!\{-\}\!\}_{T^\omega(H_\bot)}\,,\ P' \VDash G.$$

Therefore, by the definition of T, there is a proof of $P \vdash_R A$ in $\{\!\{P\}\!\}_{T(T^{\omega}(H_{\perp}))}$. However, $\{\!\{P\}\!\}_{T^{\omega}(H_{\perp})}$ is a fixed point, so there is a proof of $P \vdash_R A$ in $\{\!\{P\}\!\}_{T^{\omega}(H_{\perp})}$. Therefore

$$\{\!\{-\}\!\}_{T^{\omega}(H_{\perp})}\,,\, P \Vdash A,$$

as required.

- Suppose the last rule applied is $\wedge R$,

  $$\frac{P \vdash_R G_1 \quad P \vdash_R G_2}{P \vdash_R G_1 \wedge G_2}$$

  Then $\{\!\{-\}\!\}_{T^{\omega}(H_{\perp})}\,,\, P \Vdash G_1 \wedge G_2$ follows immediately from the induction hypothesis and the definition of $\Vdash$.

- Suppose the last rule applied is $\to R$,

  $$\frac{P;[D] \vdash G}{P \vdash D \to G}$$

  By the induction hypothesis, we have that

  $$\{\!\{-\}\!\}_{T^{\omega}(H_{\perp})}\,,\, (P;[D]) \Vdash G$$

  and so, by the definition of $\Vdash$, we immediately have that

  $$\{\!\{-\}\!\}_{T^{\omega}(H_{\perp})}\,,\, P \Vdash D \to G,$$

  as required.

- Suppose the last rule applied is $\vee R$,

  $$\frac{P \vdash_R G_i}{P \vdash_R G_1 \vee G_2} \ (i = 1,2)$$

  Then $\{\!\{-\}\!\}_{T^{\omega}(H_{\perp})}\,,\, P \Vdash G_1 \vee G_2$ follows immediately from the induction hypothesis and the definition of $\Vdash$.

- Suppose the last rule applied is $*R$,

  $$\frac{P_1 \vdash_R G_1 \quad P_2 \vdash_R G_2}{P_1, P_2 \vdash_R G_1 * G_2}$$

  By the induction hypothesis, we have that

  $$\{\!\{-\}\!\}_{T^{\omega}(H_{\perp})}\,,\, P_1 \Vdash G_1 \quad \text{and} \quad \{\!\{-\}\!\}_{T^{\omega}(H_{\perp})}\,,\, P_2 \Vdash G_2.$$

Since $P_1, P_2 \sqsubseteq P_1, P_2$, we have, by the definition of $\Vdash$, that

$$\{\!\{-\}\!\}_{T^\omega(H_\perp)}, P_1, P_2 \Vdash G_1 * G_2,$$

as required.

- Suppose the last rule applied is $-\!\!*R$,

  $$\frac{P, [D] \vdash G}{P \vdash D -\!\!* G}$$

  By the induction hypothesis, we have that

  $$\{\!\{-\}\!\}_{T^\omega(H_\perp)}, (P, [D]) \Vdash G$$

  and so, by the definition of $\Vdash$, we immediately have that

  $$\{\!\{-\}\!\}_{T^\omega(H_\perp)}, P \Vdash D -\!\!* G,$$

  as required.

□

Lemma 6.3.15 (completeness) *If* $\{\!\{-\}\!\}_{T^\omega(H_\perp)}, P \Vdash G$, *then* $P \vdash_R G$.

Proof: By induction on the structure of the model and the structure of the goal, adapting an argument of Miller [Miller 81].

By Lemma 6.3.9, it is sufficient to suppose that P and G are such that, for some $k \geq 0$,

$$\{\!\{-\}\!\}_{T^k(H_\perp)}, P \Vdash G,$$

and suppose this is the least such k.

If G has $n \geq 0$ occurrences of connectives and quantifiers, then define the ordinal measure

$$\mu(P, G) = \omega \cdot (k-1) + n,$$

where $\cdot$ is ordinal multiplication. We show, by induction on μ, that if there is an α such that $\mu(P, G) = \alpha$, then $P \vdash_R G$.

In the base case, we have $\mu(P, G) = 0 (= \omega \cdot 0 + 0)$, so G must be atomic and, by the definition of T, it must be that either

- $\{\!\{-\}\!\}_{T^1(H_\perp)}, P \Vdash G$, $\{\!\{-\}\!\}_{T^1(H_\perp)}, P''' \Vdash I$ and there is some $\mathcal{R} \in \{\!\{P\}\!\}_{T^1(H_\perp)}$ such that $\mathcal{R} : P \vdash_R G$; or

- there is some $G' \mathrel{-\!\!*} G$ in P such that $\{\!\{-\}\!\}_{T^1(H_\perp)}, P' \Vdash G$ and there is some $\mathcal{R} \in \{\!\{P\}\!\}_{T^1(H_\perp)}$ such that $\mathcal{R} : P \vdash_R G$; or

- there is some $G' \mathrel{-\!\!*} G$ in P such that

$$\{\!\{-\}\!\}_{T^1(H_\perp)}, P' \Vdash G, \quad \{\!\{-\}\!\}_{T^1(H_\perp)}, P'' \Vdash I$$

  and there is some $\mathcal{R} \in \{\!\{P\}\!\}_{T^1(H_\perp)}$ such that $\mathcal{R} : P \vdash_R G$.

In each of these three cases, we immediately have that $P \vdash_R G$.

In the inductive case, we have $\mu(P,G) = \omega \cdot \alpha + \beta > 0$. So there are two subcases: either $\mu(P,G)$ is a limit ordinal (*i.e.*, $\beta = 0$) or a successor (*i.e.*, $\beta > 0$).

Suppose $\beta = 0$. Then G is atomic and $\alpha > 0$. So we have

$$\{\!\{-\}\!\}_{T^{\alpha+1}(H_\perp)}, P \Vdash G.$$

By the definition of the T, either

- $\{\!\{-\}\!\}_{T^{\alpha+1}(H_\perp)}, P \Vdash G$, $\{\!\{-\}\!\}_{T^{\alpha+1}(H_\perp)}, P''' \Vdash I$ and there is some $\mathcal{R} \in \{\!\{P\}\!\}_{T^{\alpha+1}(H_\perp)}$ such that $\mathcal{R} : P \vdash_R G$; or

- there is some $G' \mathrel{-\!\!*} G$ in P such that $\{\!\{-\}\!\}_{T^{\alpha+1}(H_\perp)}, P' \Vdash G$ and there is some $\mathcal{R} \in \{\!\{P\}\!\}_{T^{\alpha+1}(H_\perp)}$ such that $\mathcal{R} : P \vdash_R G$; or

- there is some $G' \mathrel{-\!\!*} G$ in P such that

$$\{\!\{-\}\!\}_{T^{\alpha+1}(H_\perp)}, P' \Vdash G, \quad \{\!\{-\}\!\}_{T^{\alpha+1}(H_\perp)}, P'' \Vdash I$$

  and there is some $\mathcal{R} \in \{\!\{P\}\!\}_{T^{\alpha+1}(H_\perp)}$ such that $\mathcal{R} : P \vdash_R G$.

Again, in each of these three cases, we immediately have that $P \vdash_R G$.

Suppose $\beta > 0$. Then G is not atomic and we proceed by cases on the structure of G, *i.e.*, one case for each possible principal connective of G. The cases closely follow the corresponding clauses of Definition 6.3.6 and we omit the details of all but one example.

- If $G \equiv G_1 * G_2$, then we have, by assumption, that

  $$\{\!\{-\}\!\}_{T^{\alpha+1}(H_\perp)}\,,\, P \Vdash G_1 * G_2.$$

  Therefore, by Definition 6.3.6, we have that

  $$\{\!\{-\}\!\}_{T^{\alpha+1}(H_\perp)}\,,\, P_1 \Vdash G_1 \quad \text{and} \quad \{\!\{-\}\!\}_{T^{\alpha+1}(H_\perp)}\,,\, P_2 \Vdash G_2$$

  Now, each of G_1 and G_2 has fewer connectives than $G_1 * G_2$ and so, by the induction hypothesis, we have that

  $$P_1 \vdash_R G_1 \quad \text{and} \quad P_2 \vdash_R G_2,$$

  as required.

The remaining cases are similar.

□

6.4 Discussion

We briefly consider a comparison with the model-theoretic semantics of the linear logic programming language Lolli, given in [Hodas Miller 94].

The basic idea in [Hodas Miller 94] is that a semantics for Lolli, *i.e.*, the resolution or uniform proof implementation of the hereditary Harrop fragment of intuitionistic linear logic, is given by a family of Kripke-structures, K_r, index by the elements r of a commutative monoid $(R, +, 0)$ of "resources".

A forcing semantics based on a judgement of the form

$$K_r, w \models \phi,$$

where w is in some poset of worlds $(W, \leq)$ read as, "ϕ holds at world w in the rth Kripke structure" Thus the forcing is parametrised on both worlds and on the choice of index, thereby permitting the semantics of $\multimap$ (and also $\otimes$) to be given via the idea of a *phase shift* between structures, *i.e.*,

$$\begin{array}{lll} K_r, w \models B_1 \multimap B_2 & \text{iff} & \text{for all } r' \in R \text{ and all } w' \in W, \\ & & \text{if } w \leq w' \text{ and } K_{r'}, w' \models B_1, \text{ then } K_{r+r'}, w' \models B_2 \end{array}$$

A completeness theorem is provided not with respect to a model constructed via a fixed-point operator but rather via a term model in which

$$K_r(w) = \{A | A \text{ is atomic and } w;r \vdash A\},$$

where the proof-theoretic judgement $w;r \vdash A$ is determined by the calculus of uniform proofs for ILL in which the stoup, ";", separates the intuitionistic worlds, w, from the linear resources, r. Here $\mathcal{W}$ is the set of all sets of hereditary Harrop formulæ, $\leq$ is subset, R is the set of all multisets of hereditary Harrop formulæ, $+$ is multiset union and 0 is the empty multiset.

Note that [Hodas Miller 94] provides no fixed point construction.

Chapter 7

Conclusion and further work

7.1 Introduction

In this chapter we describe possible extensions to the idea of bunched contexts and which kind of logic may arise from them. We analyse connectives with weakening and no contraction, and conversely operators with contraction but not weakening.

7.2 Other Bunch-forming Operators

BI has two bunch-forming operators: an additive one,“;” , which allows Weakening and Contraction, and a multiplicative,“,” , which does not. We will study in this section the addition of two extra operators, “,,”, which allows Weakening but not Contraction, and “;;” which allows Contraction but not Weakening. We will assign the names “affine” and “relevant” respectively to these new bunch operators.

7.2.1 Units

One of the first questions with these new operators is which kind of units they have. The following lemma shows that all operations which allow Weakening have the same unit.

Lemma 7.2.1 *The unit corresponding to “,,” is the same than the unit for “;”, that is* $\top$. *Moreover, any operation that allows* Weakening *has to have* $\top$ *as its unit.*

Proof: Suppose that Π is the unit for “„”, or more generally, the unit for any operation which allows Weakening. Then we show that this unit is equivalent to $\top$

$$\dfrac{\dfrac{\dfrac{}{\Pi \vdash \Pi}\,\text{Axiom}}{\Pi,,\top \vdash \Pi}\,\text{Weakening}(,,)}{\top \vdash \Pi}\,\text{Unit}(,,) \qquad \dfrac{\dfrac{\dfrac{}{\top \vdash \top}\,\text{Axiom}}{\top;\Pi \vdash \top}\,\text{Weakening}(;)}{\Pi \vdash \top}\,\text{Unit}(;)$$

□

Next we show the admissibility of a rewriting rule from “;” to “„”:

Lemma 7.2.2 *The existence of Weakening for “„” and Contraction for “;” makes the following rule admissible*

$$\dfrac{\Gamma(\Delta;\Theta) \vdash \phi}{\Gamma(\Delta,,\Theta) \vdash \phi}\,(; \rightarrow ,,)$$

Proof:

$$\dfrac{\dfrac{\Gamma(\Delta;\Theta) \vdash \phi}{\Gamma((\Delta,,\Theta);(\Delta,,\Theta)) \vdash \phi}\,\text{Weakening}(,,)}{\Gamma(\Delta,,\Theta) \vdash \phi}\,\text{Contraction}(;)$$

□

Conversely,

Lemma 7.2.3 *The addition of the rule* $(; \rightarrow ,,)$ *implies Weakening for “„”.*

Proof:

$$\dfrac{\dfrac{\Gamma(\Delta) \vdash \phi}{\Gamma(\Delta;\Theta) \vdash \phi}\,\text{Weakening}}{\Gamma(\Delta,,\Theta) \vdash \phi}\,(; \rightarrow ,,)$$

□

and hence, for Lemma 7.2.1 the equivalence of their units.

Therefore the logic obtained using only “;” and “„” is the affine version of **BI** described in [O’Hearn 99].

In the same way, the existence of Weakening for “;” and Contraction for “;;” makes the following rule admissible

$$\dfrac{\Gamma(\Delta;;\Theta) \vdash \phi}{\Gamma(\Delta;\Theta) \vdash \phi}\,(;; \rightarrow ;)$$

And again, adding this rule to the system implies Contraction for “;;”.

7.2.2 Still more operators?

So we have that if $\Gamma(\Delta;;\Theta) \vdash \phi$ then $\Gamma(\Delta;\Theta) \vdash \phi$ and then $\Gamma(\Delta,,\Theta) \vdash \phi$, producing an ordering on the operators. But where does “,” stand with respect to these transformations? It seems that it has no connection with any of them. For example, if we add the rule

$$\frac{\Gamma(\Delta,\Theta) \vdash \phi}{\Gamma(\Delta;;\Theta) \vdash \phi}\ (, \to ;;)$$

then we would induce Contraction for “,” as seen in Lemma 7.2.3. The system then would have two different (inequivalent) connectives with Contraction.

Following this idea, one could think of an infinite hierarchy of bunch-forming operators $;;_n$ and a meta-rule

$$\frac{\Gamma(\Delta;;_{n+1}\Theta) \vdash \phi}{\Gamma(\Delta;;_n\Theta) \vdash \phi}\ (;;_{n+1} \to ;;_n)$$

which would induce Contraction in all of them. Then “,” could be thought of as $;;_\omega$.

An alternative meta-rule could be

$$\frac{\Gamma(\Delta;;_n\Theta) \vdash \phi}{\Gamma(\Delta;\Theta) \vdash \phi}\ (;;_n \to ;)$$

then we would have a flat or horizontal hierarchy of inequivalent bunch-forming operators with Contraction. However “,” again goes off the picture.

These two ways of creating new bunch-forming operators can be combined into one, with an operator $;;_{m,n}$ and meta-rules

$$\frac{\Gamma(\Delta;;_{(1,n)}\Theta) \vdash \phi}{\Gamma(\Delta;\Theta) \vdash \phi}\ (;;_{(1,n)} \to ;) \qquad \frac{\Gamma(\Delta;;_{(m+1,n)}\Theta) \vdash \phi}{\Gamma(\Delta;;_{(m,n)}\Theta) \vdash \phi}\ (;;_{(m+1,n)} \to ;;_{(m,n)})$$

In the sequel we will use only the first (vertical) hierarchy, but the extension of the meta-rules to the last case is easy.

This analysis can be applied to “„” which would produce the following meta-rules

$$\frac{\Gamma(\Delta;\Theta) \vdash \phi}{\Gamma(\Delta,,_{(1,n)}\Theta) \vdash \phi}\ (; \to ,,_{(1,n)}) \qquad \frac{\Gamma(\Delta,,_{(m,n)}\Theta) \vdash \phi}{\Gamma(\Delta,,_{(m+1,n)}\Theta) \vdash \phi}\ (,,_{(m,n)} \to ,,_{(m+1,n)})$$

with the following simplified meta-rule which corresponds to the vertical hierarchy only, and is the one used from now on:

$$\frac{\Gamma(\Delta,,_n\Theta) \vdash \phi}{\Gamma(\Delta,,_{n+1}\Theta) \vdash \phi}\ (,,_n \to ,,_{n+1})$$

7.2.3 New Units

As shown in Lemma 7.2.1, all affine operators share the same unit. The new relevant bunch-forming operators, however, bring a whole new hierarchy of units with them. The unit rules associated to them can be succinctly expressed in the following Unit meta-rules:

$$\dfrac{\Gamma \vdash \phi}{\Gamma ;\!;_n \top\!\!\top_m \vdash \phi}\,\text{Unit}(n \leq m) \qquad \dfrac{\Gamma ;\!;_n \top\!\!\top_m \vdash \phi}{\Gamma \vdash \phi}\,\text{Unit}(n \geq m)$$

7.2.4 New Axiom

As seen in Chapter 3, **BI** admits the following axiom rule

$$\dfrac{\Gamma \vdash I}{\Gamma, \alpha \vdash \alpha}\,\text{CutAxiom}$$

The problem with this rule is that it relies on the fact that A can always be put at the top level in a bunch. This is the case when only "," and ";" are used to form bunches. Now, however, there is no guarantee that this will be the case. But there are still some side conditions that allow manipulation with the units.

We could try a meta-rule for the axiom with the following shape:

$$\dfrac{\Gamma_1 \vdash \top\!\!\top_{n_1} \quad \ldots \quad \Gamma_{m-1} \vdash \top\!\!\top_{n_{m-1}} \quad \Gamma_m \vdash \top\!\!\top_{n_m}}{\Gamma_1 ;\!;_{n_1} (\ldots (\Gamma_{m-1} ;\!;_{n_{m-1}} (\Gamma_m ;\!;_{n_m} \alpha))) \vdash \alpha}\,\text{Axiom}$$

justified by the following proof figure

$$\dfrac{\dfrac{\Gamma_m \vdash \top\!\!\top_{n_m} \qquad \dfrac{\dfrac{}{\alpha \vdash \alpha}\,\text{Axiom}}{\top\!\!\top_{n_m} ;\!;_{n_m} \alpha \vdash \alpha}\,\text{Unit}}{\Gamma_m ;\!;_{n_m} \alpha \vdash \alpha}\,\text{Cut} \qquad \dfrac{\vdots}{\Gamma_1 ;\!;_{n_1} (\ldots (\Gamma_{m-1} ;\!;_{n_{m-1}} \alpha)) \vdash \alpha}\,\text{Induction}}{\Gamma_1 ;\!;_{n_1} (\ldots (\Gamma_{m-1} ;\!;_{n_{m-1}} (\Gamma_m ;\!;_{n_m} \alpha))) \vdash \alpha}\,\text{Cut}$$

It is also possible to provide another axiom rule making use of one of the rules presented in §7.2.2, rule $(;\!;_{n+1} \to ;\!;_n)$. First, using this rule, we push the rank of all bunch operators above α upwards until α is at top level, and then we use an axiom rule similar to CutAxiom:

$$\dfrac{\Gamma_1 ;\!;_\nu \ldots ;\!;_\nu \Gamma_m \vdash \top\!\!\top_\nu}{\Gamma_1 ;\!;_{n_1} (\ldots (\Gamma_m ;\!;_{n_m} \alpha)) \vdash \alpha}\,\text{CutAxiom}$$

where $\nu = \max(n_1, \ldots, n_m)$.

It would be nice if the first of these rules could be derived from this second, simpler, one. But this is not the case. For example the sequent $\top\!\!\top_4 ;\!;_4((a ;\!;_2 a \twoheadrightarrow_2 \top\!\!\top_3) ;\!;_3 c) \vdash c$ can be proved using the former:

$$\dfrac{\top\!\!\top_4 \vdash \top\!\!\top_4 \qquad a ;\!;_2 a \twoheadrightarrow_2 \top\!\!\top_3 \vdash \top\!\!\top_3}{\top\!\!\top_4 ;\!;_4((a ;\!;_2 a \twoheadrightarrow_2 \top\!\!\top_3) ;\!;_3 c) \vdash c}\,\text{Axiom}$$

but not the latter:

$$\dfrac{\dfrac{\dfrac{}{\top\!\!\top_4 ;\!;_4(a ;\!;_2 a \twoheadrightarrow_2 \top\!\!\top_3) \vdash \top\!\!\top_4}\,\text{Not provable}}{\top\!\!\top_4 ;\!;_4(a ;\!;_2 a \twoheadrightarrow_2 \top\!\!\top_3) ;\!;_4 c \vdash c}}{\top\!\!\top_4 ;\!;_4((a ;\!;_2 a \twoheadrightarrow_2 \top\!\!\top_3) ;\!;_3 c) \vdash c}\,(;\!;_3 \rightarrow ;\!;_4)$$

7.2.5 New Connectives

Together with the new bunch forming operators there are new connectives related to them. They are new variants of addition and implication. With the purpose of not straining our imagination, the notation for them will be $\ast\!\ast_n$, $\rightarrow\!\ast\!\ast_n$, $\wedge\!\!\wedge_n$ and $\twoheadrightarrow_n$.

Conjunctions

The left rules for the new conjunctions are straightforward

$$\dfrac{\Gamma(\phi ,\!,_n \psi) \vdash \chi}{\Gamma(\phi \ast\!\ast_n \psi) \vdash \chi}\,\ast\!\ast_n L \qquad \dfrac{\Gamma(\phi ;\!;_n \psi) \vdash \chi}{\Gamma(\phi \wedge\!\!\wedge_n \psi) \vdash \chi}\,\wedge\!\!\wedge_n L$$

The right rules also can be stated in the familiar pattern

$$\dfrac{\Gamma \vdash \phi \quad \Delta \vdash \psi}{\Gamma ,\!,_n \Delta \vdash \phi \ast\!\ast_n \psi}\,\ast\!\ast_n R \qquad \dfrac{\Gamma \vdash \phi \quad \Delta \vdash \psi}{\Gamma ;\!;_n \Delta \vdash \phi \wedge\!\!\wedge_n \psi}\,\wedge\!\!\wedge_n R$$

But perhaps their behaviour is revealed better if we take under account two things. First, contraction is allowed for $;\!;_n$ and weakening for $,\!,_n$. Second, the conversion rules allow us to change the "rank" of the bunches. So to allow for maximum generality the meta-rules could be stated like

$$\dfrac{\Gamma ;\!;_m \Theta \vdash \phi \quad \Delta ;\!;_m \Theta \vdash \psi}{\Gamma ;\!;_n \Delta ;\!;_{n'} \Theta \vdash \phi \wedge\!\!\wedge_m \psi}\,\wedge\!\!\wedge_m R(n, n' \leq m) \quad \dfrac{\Gamma \vdash \phi \quad \Delta \vdash \psi}{\Gamma ,\!,_n \Delta ,\!,_{n'} \Theta \vdash \phi \ast\!\ast_m \psi}\,\ast\!\ast_m R(n \geq m)$$

where Θ represents in the case of $\wedge\!\!\wedge_n R$ the fraction of the context which is needed by both branches of the proof, and therefore needs to be duplicated via contraction, and in the case of $\ast\!\ast_n R$ the fraction of the context which is not used an weakened away. Note that in the first case the rank of the bunches goes up. This is a price to pay, since for example $a ;\!;_m b \vdash a \wedge\!\!\wedge_n b$ if and only if $m \leq n$. In $\ast\!\ast_m R$ there are no restrictions on n' since whatever its rank it would always be possible to weaken Θ.

Implications

Now it's the turn for the right rules to be straightforward

$$\frac{\Gamma;\!;_n\phi\vdash\phi}{\Gamma\vdash\phi\twoheadrightarrow_n\psi}\twoheadrightarrow_n R \qquad \frac{\Gamma,\!,_n\phi\vdash\phi}{\Gamma\vdash\phi\mathrel{-\!\!**}_n\psi}\mathrel{-\!\!**}_n R$$

The left rules can be stated following the pattern of the normal version of **BI**

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

Again these rules can be transformed into a more revealing format

$$\frac{\Gamma;\!;_m\Theta\vdash\phi\quad\Delta(\Delta';\!;_m\Theta;\!;_m\psi)\vdash\chi}{\Delta(\Gamma;\!;_n\Delta';\!;_{n'}\Theta;\!;_{n''}\phi\twoheadrightarrow_m\psi)\vdash\chi}\twoheadrightarrow_m L(n,n',n''\leq m)$$

$$\frac{\Gamma\vdash\phi\quad\Delta(\Delta',\!,_m\psi)\vdash\chi}{\Delta(\Gamma,\!,_n\Delta',\!,_{n'}\Theta,\!,_{n''}\phi\mathrel{-\!\!**}_m\psi)\vdash\chi}\mathrel{-\!\!**}_m L(n,n'\geq m)$$

Note again the raising of the ranks in the premises of $\twoheadrightarrow_m L$ and the lack of restrictions on n'' in $\mathrel{-\!\!**}_m L$.

7.2.6 Computational content

Here we use a similar notation than the one used in Chapter 4: $\langle\Gamma\rangle_n\backslash\langle\Delta\rangle_{n'}\vdash_o\phi$ expresses that the context Γ proves ϕ leaving a residue Δ. Left rules for the conjunctions and right rules for the implications are straightforward and omitted for the sake of brevity.

However, before looking at the rules we need to see how the new rules interact with the old ones. In particular there are weird interactions between $*$ and $\wedge\!\!\!\wedge$ and between $-\!\!*$ and $\twoheadrightarrow$. This is because these two operators are related to bunch operators which don't allow weakening.

Interaction between Multiplicatives and Relevant operators

To make the discussion more concrete let's look at how a proof of $a,b\vdash a\wedge\!\!\!\wedge b$ should fail, paying special attention to the lazy model of resource consumption:

$$\cfrac{\cfrac{a,b\vdash a\quad a,b\vdash b}{(a,b);\!;(a,b)\vdash a\wedge\!\!\!\wedge b}\wedge\!\!\!\wedge R}{a,b\vdash a\wedge\!\!\!\wedge b}\text{Contraction}$$

This proof should fail immediately in the left branch, given that there is not proof of $a,b\vdash a$ because b is left over. However, when all the context is given to the left branch of a computation, it is intended that any part of the program left over should be given to the rest of the computation to be consumed later. There is not easy way to know in advance if any particular left-over is legal or not.

Also the following derivation would go through under the operational semantics of $-\!\!*$, despite the sequent being unprovable.

$$\frac{\overline{a \vdash a}}{a;;a-\!\!*b \vdash b}\,-\!\!*L$$

Therefore it is proposed that the multiplicative and relevant bunch-forming operations, together with their related operators, not to be used together in the same program. One way of enforcing this could be to use the same syntactic operators for both, and let the programmer specify in which way they are intended to be used.

Right rules for Conjunctions

For a clearer understanding of the right rules, we will use the second version. The rule for $**R$ is the same than $*R$, the only difference being that the flag used to signal that an additive unit has been found as a goal is set and remains so thereafter.

$$\frac{\langle\Gamma\rangle_n\backslash\langle\Gamma'\rangle_{n'} \vdash_o \phi \quad \langle\Gamma'\rangle_n\backslash\langle\Delta\rangle_{n'} \vdash_o \psi}{\langle\Gamma\rangle_n\backslash\langle\Delta\rangle_{n'} \vdash_o \phi{**}\psi}\,**R \qquad \frac{\langle\Gamma\rangle_n\backslash\langle\Delta_1\rangle_{n'} \vdash_o \phi \quad \langle\Gamma\rangle_n\backslash\langle\Delta_2\rangle_{n'} \vdash_o \psi}{\langle\Gamma\rangle_n\backslash\langle\Delta_1 \cap \Delta_2\rangle_{n'} \vdash_o \phi \wedge\!\!\wedge \psi}\,\wedge\!\!\wedge R$$

Given that $\wedge\!\!\wedge$ allows contraction, all the context is given to both subformulas and the residue is that part of the context that was used by *neither* of them. Note that this operational semantics would be unsound if Multiplicative connectives are being used at the same time for the reasons explained above.

Left rules for Implications

As in the case of **BI**, the fact that we are working with Hereditary Harrop Formulae, normal, and goal directed proofs helps to simplify the left rules for implications. The crucial factor is that for an atom to be probable there has to be either an atom in the program or an implication with the atom as its consequent. That is the rationale behind the following simplifications

$$\frac{\Gamma \vdash \phi \quad \Delta(\Delta',,A) \vdash A}{\Delta(\Gamma,,\Delta',,\phi -\!\!** A) \vdash A}\,-\!\!**L \quad \frac{\Gamma \vdash \phi \quad \Delta(\Delta';;A) \vdash A}{\Delta(\Gamma;;\Delta';;\phi \twoheadrightarrow A) \vdash A}\,\twoheadrightarrow L$$

Now for a proof to be possible at all Δ', which is the subbunch connected with the atom in the right branch, has to disappear, either by proving a unit or by weakening if this is feasible. In either case the same derivation can be done in the left branch, allowing us to assimilate Δ' into Γ and so simplify the rules even further

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

In the same way than in normal **BI**, the relevant implication rule can be simplified one step more, ending up with a unary rule reflecting the fact that the proofs are simple. Extra care has to be taken with the clause $\phi \twoheadrightarrow A$ itself, which should be contracted just in case, but once it has been used, it could be discarded if necessary.

$$\frac{\Gamma(\phi \twoheadrightarrow A) \vdash \phi}{\Gamma(\phi \twoheadrightarrow A) \vdash A} \twoheadrightarrow L$$

Finally the operational semantics for the implications are

$$\frac{\langle \Gamma \rangle_n \backslash \langle \emptyset \rangle_{n'} \vdash_o \phi \quad \langle \Delta(A) \rangle_n \backslash \langle \Delta' \rangle_{n'} \vdash_o A}{\langle \Delta(\Gamma ,, \phi \mathrel{-\!\!**} A) \rangle_n \backslash \langle \Delta' \rangle_{n'} \vdash_o A} \mathrel{-\!\!**} L \qquad \frac{\langle \Gamma(\phi \twoheadrightarrow A) \rangle_n \backslash \langle \Delta \rangle_{n'} \vdash_o \phi}{\langle \Gamma(\phi \twoheadrightarrow A) \rangle_n \backslash \langle \Delta \rangle_{n'} \vdash_o A} \twoheadrightarrow L$$

7.3 Applications

We conjecture that this hierarchy of operators can provide a purely logical foundation for computer security. For example an operating system can provide to each user a number which will control what level of access the user can have. Suppose that a particular secret program is allowed a level 3 access. This could be written like this:

$$\text{open} \twoheadrightarrow_3 \top\!\!\top_3 \;;;_3\; \text{secret}(\alpha)$$

Any user with access up to 3 will be able to "unlock" the relevant piece of code. For example a user with access level 2 could find the following proof:

$$\cfrac{\cfrac{\cfrac{\text{open} \;;;_3\; \text{open} \twoheadrightarrow_3 \top\!\!\top_3 \vdash \top\!\!\top_3}{\text{open} \;;;_3\; \text{open} \twoheadrightarrow_3 \top\!\!\top_3 \;;;_3\; \text{secret}(\alpha) \vdash \text{secret}(x)} \text{Axiom}}{\text{open} \;;;_2 (\text{open} \twoheadrightarrow_3 \top\!\!\top_3 \;;;_3\; \text{secret}(\alpha)) \vdash \text{secret}(x)} (;;_2 \to ;;_3)}{\text{open} \twoheadrightarrow_3 \top\!\!\top_3 \;;;_3\; \text{secret}(\alpha) \vdash \text{open} \twoheadrightarrow_2 \text{secret}(x)} \twoheadrightarrow_2 R$$

But if the user had an access level greater than 3 this could not be done:

$$\cfrac{\cfrac{\cfrac{\cfrac{}{\text{open} \;;;_4\; \text{open} \twoheadrightarrow_3 \top\!\!\top_3 \vdash \top\!\!\top_4} \text{Not provable}}{\text{open} \;;;_4\; \text{open} \twoheadrightarrow_3 \top\!\!\top_3 \;;;_4\; \text{secret}(\alpha) \vdash \text{secret}(x)} \text{Axiom}}{\text{open} \;;;_4 (\text{open} \twoheadrightarrow_3 \top\!\!\top_3 \;;;_3\; \text{secret}(\alpha)) \vdash \text{secret}(x)} (;;_3 \to ;;_4)}{\text{open} \twoheadrightarrow_3 \top\!\!\top_3 \;;;_3\; \text{secret}(\alpha) \vdash \text{open} \twoheadrightarrow_4 \text{secret}(x)} \twoheadrightarrow_4 R$$

With this arrangement anyone with access level 1 can play the role of super-user.

A different kind of restriction can be achieved with the connective $\wedge\!\!\wedge$. The program $\forall x Q(x) \twoheadrightarrow_7 P(x) ;\!;_3 Q(a) ;\!;_3 Q(b)$ will allow users to prove the clause $P(a) \wedge\!\!\wedge_n P(b)$ only if $3 \leq n \leq 7$. For example with $n = 4$ we have:

$$\dfrac{\dfrac{\dfrac{\dfrac{}{Q(a) \vdash Q(a)}\,\text{Axiom} \qquad \dfrac{}{P(a) \vdash P(a)}\,\text{Axiom}}{Q(a) \twoheadrightarrow_7 P(a) ;\!;_7 Q(a) \vdash P(a)}\,\twoheadrightarrow_7 L}{\forall x\; Q(x) \twoheadrightarrow_7 P(x) ;\!;_4 Q(a) \vdash P(a)}\,(;\!;_4 \to ;\!;_7) + \text{Res} \qquad \dfrac{\vdots}{\forall x\; Q(x) \twoheadrightarrow_7 P(x) ;\!;_4 Q(b) \vdash P(b)}}{\dfrac{\forall x\; Q(x) \twoheadrightarrow_7 P(x) ;\!;_4 Q(a) ;\!;_4 Q(b) \vdash P(a) \wedge\!\!\wedge_4 P(b)}{\forall x\; Q(x) \twoheadrightarrow_7 P(x) ;\!;_3 Q(a) ;\!;_3 Q(b) \vdash P(a) \wedge\!\!\wedge_4 P(b)}\,(;\!;_3 \to ;\!;_4)}\,\wedge\!\!\wedge_4 R$$

If $n < 3$ the proof will fail because there is no rule $(;\!;_3 \to ;\!;_n)$, and if $n > 7$ the rule $\twoheadrightarrow_7 L$ would not be applicable because the bunch will have a rank that is too high.

It is possible to produce the same effect in Prolog by attaching each clause with an integer and coding in explicitly the restrictions we want to enforce. The version described above is more elegant and simpler.

Chapter 8

Implementation for BLP

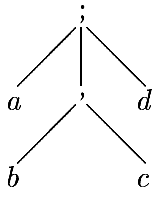
8.1 Introduction

In this chapter we describe a suitable datatype for bunches and describe in detail the two main modules of the OCaml [Cousineau Mauny 98] implementation *i.e.*, `bunch.ml` and `interpreter.ml`

This code is available from `http://www.dcs.qmul.ac.uk/~pablo/blp/`

8.2 A Data Structure for Bunches

Consider the following instance of a canonical additive bunch



It is composed by two additive clauses, a and d, and a multiplicative bunch. When defining a datatype for this structure, we need these three components to belong to the same type, so we can put them together in a list. Also it seems reasonable to consider the additive unit to be an additive bunch.

This construction suggests the following data structure:

```
type bunch = MulBunch of ab | AddBunch of mb

and ab = UnitA | A of clause | MN of mb list
```

```
    and mb = UnitM | M of clause | AN of ab list
```

The use of this data structure presents the following slightly paradoxical characteristic: clauses that are in an additive relation with each other should be called additive clauses. In addition, if a multiplicative bunch is in an additive relation with some other bunches, it should be of the same type than the clauses at that level. So, for example, if there is a bunch a ; (b , c), the subtree (b,c) should be of the same type than a, that is an additive bunch. So we get that an additive bunch could be the additive unit, an additive clause, or a multiplicative node. The final twist is that this bunch is additive, and it is an additive node of a list of additive bunches. So for the constructors of the general type bunch, the constructor MulBunch takes an additive bunch, and AddBunch a multiplicative one.

With this definition of the datatype the mutually recursive nature of additive and multiplicative bunches is enforced at the type level. However it also makes the code more complicated. So we have settled for a simpler datatype as explained in the next section.

8.3 Code for `bunch.ml`

In this section we describe the `bunch` datatype and its related operations. Also we show the implementation of some important operations described in the main body of the thesis.

8.3.1 The `bunch` datatype

We are concerned mainly with bunches of clauses, but, as mentioned in 2.5 **BI** allows bunched contexts of variables. For this reason it is better to work with a generic dependent type $Bunch(A)$, which represents bunches of elements of type A.

As a first step towards a simplified datatype, we will use only one unit: the multiplicative unit (I). The reason is that in the operational semantics

The `bunch` datatype is defined as

```
type 'a bunch =
    Empty
  | L of 'a
  | MB of 'a bunch list
  | AB of 'a bunch list
```

8.3.2 Joining bunches

```
let mjoin ba bb =
  let f a b =
    match (a,b) with
        (MB m , MB n) -> m@n
      | (MB m, _ ) -> m@[b]
      | ( _ , MB m) -> a::m
      | _  -> a::[b]
  in
      if ba = Empty then bb
      else if bb = Empty then ba
      else MB (f ba bb)

let ajoin ba bb =
  let f a b =
    match (a,b) with
        (AB m , AB n) -> m@n
      | (AB m, _ ) -> m@[b]
      | ( _ , AB m) -> a::m
      | _  -> a::[b]
  in
      if ba = Empty then bb
      else if bb = Empty then ba
      else AB (f ba bb)
```

8.3.3 The "minimum number of weakenings" operation

In Chapter 4 we defined an operation called *minimum number of weakenings* which will transform a bunch with the shape $\Gamma(\Delta)$ into a bunch with the shape Γ',Δ.

1. $MNW_{\Delta}(\Delta) = \Delta$

2. $MNW_{\Delta}(\overset{;}{\Gamma}_1,\cdots,\overset{;}{\Gamma}_n) = \overset{;}{\Gamma}_1,\cdots,MNW_{\Delta}(\overset{;}{\Gamma}_i),\cdots,\overset{;}{\Gamma}_n$

3. $MNW_{\Delta}(\overset{,}{\Gamma}_1;\cdots;\overset{,}{\Gamma}_n) = MNW_{\Delta}(\overset{,}{\Gamma}_i)$

Although this definition is made with respect to a general subbunch Δ, the implementation was done with respect to an individual clause. The reason is that in the cases where this operation is needed, doing it at the level of a clause is enough. In the case of the rules CutAxiom and $\multimap L$ this is obvious, but even in the case of $\to L$ the result of $MNW_{\Gamma;\phi\to\alpha}\Theta(\Gamma;\phi\to\alpha)$ is Θ' which is equal to $MNW_{\phi\to\alpha}\Theta(\Gamma;\phi\to\alpha)$.

The implementation of the operation is as follows:

```
let rec mnw a = function
   | MB (h::[]) -> if mem a h then mnw a h else h
   | MB (h::tl) -> if mem a h then mjoin (mnw a h) (MB tl)
                   else mjoin h (mnw a (MB tl))
   | AB (h::[]) -> mnw a h
   | AB (h::tl) -> if mem a h then mnw a h
                   else mnw a (AB tl)
   | cl -> cl
```

8.3.4 The subtract operation

Recall the definition of subtraction given in Definition 4.2.6:

$$\Gamma - \varnothing_m = \Gamma \tag{8.1}$$
$$\Gamma - \overset{;}{\Delta} = \Gamma'^{\dagger} \tag{8.2}$$
$$\Gamma - (\overset{;}{\Delta}_1,\cdots,\overset{;}{\Delta}_n) = (\Gamma - \overset{;}{\Delta}_1) - (\Delta - \overset{;}{\Delta}_1) \tag{8.3}$$

† where $\Gamma',\overset{;}{\Delta} = MNW_{\Delta}(\Gamma)$

The implementation is done as follows:

```
let rec subtract sub bunch =
  match sub with
      Empty -> bunch
    | L a -> axiom a bunch
    | MB (h::[]) ->
```

```
     subtract h bunch
  | MB (h::tl) ->
     subtract h (subtract (MB tl) bunch)
  | AB (h::tl) -> subtract h bunch
  | _ -> raise BunchErrorException
```

8.3.5 Other important bunch operations

Subbunch

Recall that the definition of subbunch (Definition 4.2.4) says that $\Delta \subseteq \Gamma$

1. when $\Delta = \varnothing_m$
2. when $\Delta = \overset{;}{\Delta}$ or Δ is a proposition and Γ is of the form $\Gamma'(\Delta)$
3. when $\Delta = \overset{;}{\Delta}_1, \cdots, \overset{;}{\Delta}_n$ and
   - $\Delta_1 \subseteq \Gamma$, and
   - if we let $(\Gamma', \Delta_1) = MNW_{\Delta_1}(\Gamma)$, then $(\overset{;}{\Delta}_2, \cdots, \overset{;}{\Delta}_n) \subseteq \Gamma'$

```
let subbunch a bunch =
 let f b = if not (mem b bunch) then raise Not_found else ()
 and l = ref true in
   begin try biter f a;()
   with Not_found -> l := false
   end;
   !l
```

Intersection between two bunches

This operation is needed in the application of the rule $\wedge R^{ii}$ described in 4.5.

```
let rec intersection a = function
  Empty -> Empty
 | (L c) as bunch -> if mem c a then bunch else Empty
 | MB (h::[]) -> intersection h a
 | MB (h::tl) -> mjoin (intersection h a) (intersection (MB tl) a)
```

```
  | AB (h::[]) -> intersection h a
  | AB (h::tl) -> ajoin (intersection h a) (intersection (AB tl) a)
  | _ -> raise BunchErrorException
```

Check

This operation checks whether ϕ was completely consumed in $\multimap R$:

$$\frac{\phi, \Gamma \vdash \psi}{\Gamma \vdash \phi \multimap \psi} \multimap R$$

```
let check sb bunch =
  let f b = if mem b bunch then raise Found else ()
  and l = ref true in
      begin try biter f sb;()
      with Found -> l := false
      end; !l
```

Get an additive bunch

After a $\to L$ rule the left branch proceeds with the additive subbunch at the same level than the additive implication. This code retrieves this additive bunch given a clause.

```
let rec getabunch a = function
    Empty -> raise (Failure "empty in getabunch")
  | (L c) as bunch -> if c == a then bunch
            else raise (Failure "-")
  | MB (h::[]) ->
        getabunch a h
  | MB (h::tl) ->
        if mem a h then getabunch a h
        else getabunch a (MB tl)
  | (AB list) as bunch ->
        let rec f = function
            L c -> c == a
          | _ -> false
        and g  = function
            (h::[]) -> getabunch a h
          | (h::tl) ->
                if mem a h then getabunch a h
                else g tl
          | _ -> raise BunchErrorException
        in
            if (List.fold_right (or) (List.map f list) false)
            then bunch
            else g list
  | _ -> raise BunchErrorException
```

8.4 Code for interpreter.ml

In general, given a rule of the operational semantics, there is a translation into OCaml using continuation passing style (CPS). We explain here how this is done.

For a rule with the shape

$$\frac{\langle\Gamma|s\rangle_n\backslash\langle\Delta|s'\rangle_{n'} \vdash_o \phi \qquad \langle\Delta|s'\rangle_{n'}\backslash\langle\Theta|s''\rangle_{n''} \vdash_o \psi}{\langle\Gamma|s\rangle_n\backslash\langle\Theta|s''\rangle_{n''} \vdash_o \psi \boxdot \psi}\ \boxdot R$$

we get the following CPS translation

```
let rec (|-) ((Γ,s) as program) goal epsilon n sigma sc =
  match goal with
     ...
     | φ⊡ψ ->
              (program ⊢ φ) epsilon n sigma
                  (fun ((Δ,s') as prog') epsilon' n' sigma' ->
                       (prog' ⊢ψ) epsilon' n' sigma'
                            (fun ((Θ,s'') as prog'') epsilon'' n'' sigma'' ->
                               sc prog'' epsilon'' n'' sigma''))
```

which can be simplified using η-reduction to

```
let rec (|-) ((Γ,s) as program) goal epsilon n sigma sc =
  match goal with
     ...
     | φ⊡ψ ->
              (program ⊢ φ) epsilon n sigma
                  (fun ((Δ,s') as prog) epsilon' n' sigma' ->
                       (prog ⊢ ψ) epsilon' n' sigma' sc

(* ... *)
(* ---------------------------------------------------------------- *)
(* Define some operators to improve readability *)
let (--) a b = B.subtract b a
let (&) = B.ajoin
let ( * ) = B.mjoin


(* ...  *)
let rec (|-) ((gamma,s)as program) goal epsilon n sigma sc =
  print gamma goal sigma;
  match goal with
(*
```

When an atom is found we can call a left rule, which is done in the function |-- below. We need to pass on the flags (εpsilon and n), the substitution (σigma) and the success continuation (sc).

```
 *)
    | Gatom alpha -> (program |-- alpha) epsilon n sigma sc
(*
```

If the goal is a command we execute it. The current substitution may be needed for side effects, so we pass it on.

```
 *)
    | Gcommand s -> executeCommand sigma s;
          sc program epsilon n sigma
(*
```

For $\wedge R$ there are five subcases. The first three arise when εpsilon $\neq \varnothing_m$ and the last two when εpsilon $= \varnothing_m$. However, notice that in all cases $\langle \Gamma | s \rangle$ is used in the left subproof. This means that it is possible to take common factor with respect to the left branch, and analyse cases after this has been done.

```
 *)
    | Gaand (phi, psi) ->
          (program |- phi) epsilon n sigma
(*
```

Note that it is all right to ignore the εpsilon flag, since it is not going to be changed.

```
 *)
              (fun (delta, s') _ n' sigma' ->
(*
```

First, if εpsilon $\neq \varepsilon$ then we are in one of the first three cases:

```
 *)
                  if epsilon <> Epsilon then (* ∧ R, ∧ Ri, ∧ Rii *)
                      if n' = n then (* ∧ R *)
(*
```

The rule for $\wedge R$ is

$$\frac{\langle \Gamma | s \rangle_n \backslash \langle \Delta | s \rangle_n \vdash_o \phi \quad \langle \Gamma - \Delta | [] \rangle_0 \backslash \langle \varepsilon | [] \rangle_m \vdash_o \psi}{\langle \Gamma | s \rangle_n \backslash \langle \Delta | s \rangle_n \vdash_o \phi \wedge \psi} \wedge R$$

In the continuation we can ignore the program (we know that it has to be $\langle \varnothing_m, [] \rangle$) and the flags εpsilon and m from the right branch.

```
 *)
                          ((gamma -- delta , []) |- psi) Epsilon 0 sigma'
                              (fun _ _ _ -> sc (delta,s') epsilon n)
(*
```

At this point we know that $n' \neq n$. It has to be greater, then, since the only way of reducing n is through the Unit^i rule, which is not applicable since εpsilon $\neq \varepsilon$. Now the distinction between $\wedge R^i$ and $\wedge R^{ii}$ is based on whether $\top$ appears in a goal position in ϕ or not. So we unconditionally do the left branch and decide which rule to use after testing the $\top$ counter.

```
 *)
                      else if n' > n then(* ∧ Ri, ∧ Rii *)
                          (program |- psi) epsilon n sigma'
(*
```

In the following continuation we deliberately ignore the stack returned, as well as the εpsilon flag.

```
 *)
                              (fun (delta',_) _ n'' ->
(*
```

The rule for $\wedge R^i$ is

$$\frac{\langle\Gamma|s\rangle_n\backslash\langle\Delta|s\rangle_{n'} \vdash_o \phi \quad \langle\Gamma|s\rangle_n\backslash\langle\Delta'|s\rangle_n \vdash_o \psi}{\langle\Gamma|s\rangle_n\backslash\langle\Delta'|s\rangle_n \vdash_o \phi\wedge\psi} \wedge R^i (n' > n \text{ and } \Delta' \subseteq \Delta)$$

For it to succeed, we need to test whether the condition $\Delta' \subseteq \Delta$ holds.

```
 *)
                                  if n'' = n && B.subbunch delta' delta (* ∧ Ri *)
                                  then  sc (delta',s') epsilon n
(*
```

The rule for $\wedge R^{ii}$ is

$$\frac{\langle\Gamma|s\rangle_n\backslash\langle\Delta|s\rangle_{n'} \vdash_o \phi \quad \langle\Gamma|s\rangle_n\backslash\langle\Delta'|s\rangle_{n''} \vdash_o \psi}{\langle\Gamma|s\rangle_n\backslash\langle\Delta\cap\Delta'|s\rangle_{n'} \vdash_o \phi\wedge\psi} \wedge R^{ii} (n', n'' > n)$$

In this case we succeed, but taking the intersection of both remainders. This produces backtracking on the right subproof, increasing the non-determinism of the program. For a more detailed discussion of this issue see §4.5 in Chapter 4

```
 *)
                                  else if n'' > n (* ∧ Rii *)
                                  then sc (B.intersection delta delta',s')
                                          epsilon n'
                                  else raise (Failure "strange n in ∧ Rii"))
                      else raise (Failure "strange n in ∧ R ")
(*
```

The other two cases, $\wedge R^{iii}$ and $\wedge R^{iv}$, arise when εpsilon $= \varepsilon$. We distinguish the cases by looking in the stack returned by the proof of ϕ. It is convenient to look first at the case $\wedge R^{iv}$.

```
 *)
         else match s' with (* εpsilon = ε *)
(*
```

The rule for $\wedge R^{iv}$ is

$$\frac{\langle \xi;\Gamma\|\boxed{\xi}^{\varepsilon}{::}s\rangle_n\backslash\langle \varepsilon|\{\Delta,\xi\}{::}s\rangle_{n'} \vdash_o \phi \quad \langle \xi;(\Gamma-\Delta)|[]\rangle_0\backslash\langle \varepsilon|[]\rangle_{n''} \vdash_o \psi}{\langle \xi;\Gamma\|\boxed{\xi}^{\varepsilon}{::}s\rangle_n\backslash\langle \varepsilon|\{\Delta,\xi\}{::}s\rangle_{n'''} \vdash_o \phi\wedge\psi}\wedge R^{iv}$$

```
 *)
           (Full(delta,xsi))::l -> (* ∧  Riv *)
             ((xsi&(gamma--delta),[]) |- psi) Epsilon 0
               sigma' (fun (empty, _ ) _ n''->
                 sc (empty,s') Epsilon
                   (if n' > n && n'' > n then n' else n))
(*
```

Finally, the rule for $\wedge R^{iii}$ is

$$\frac{\langle \Gamma|s\rangle_n\backslash\langle \varepsilon|s\rangle_{n'} \vdash_o \phi \quad \langle \Gamma|s\rangle_n\backslash\langle \varepsilon|s'\rangle_{n''} \vdash_o \psi}{\langle \Gamma|s\rangle_n\backslash\langle \varepsilon|s'\rangle_{n'''} \vdash_o \phi\wedge\psi}\wedge R^{iii}$$

and we use it if the stack is of any other shape.

```
 *)
          | _ ->   (* ∧ Riii *)
             (program |- psi) Epsilon n sigma'
               (fun np _ n'' ->
                 sc np Epsilon
                   (if n' > n && n'' > n then (n+1) else n)
                   ))
(*
```

The rule for $*R$ is

$$\frac{\langle \Gamma|\boxtimes{::}s\rangle_n\backslash\langle \Gamma'|\boxtimes{::}s\rangle_{n'} \vdash_o \phi \quad \langle \Gamma'|s\rangle_{n'}\backslash\langle \Delta|s'\rangle_{n''} \vdash_o \psi}{\langle \Gamma|s\rangle_n\backslash\langle \Delta|s'\rangle_{n''} \vdash_o \phi*\psi}*R$$

```
*)
    | Gmand(phi,psi) ->
          ((gamma, Locked::s) |- phi) Leavesome n sigma
              (fun (gamma',Locked::s) _ n' sigma' ->
                  ((gamma',s) |- psi) epsilon n' sigma' sc)
(*
```

The rules for $\vee R$ are

$$\frac{\langle\Gamma|s\rangle_n\backslash\langle\Delta|s'\rangle_{n'} \vdash_o \phi}{\langle\Gamma|s\rangle_n\backslash\langle\Delta|s'\rangle_{n'} \vdash_o \phi \vee \psi}\,\vee R \qquad \frac{\langle\Gamma|s\rangle_n\backslash\langle\Delta|s'\rangle_{n'} \vdash_o \psi}{\langle\Gamma|s\rangle_n\backslash\langle\Delta|s'\rangle_{n'} \vdash_o \phi \vee \psi}\,\vee R$$

so we first try ϕ and in case of failure (return) we try ψ

```
*)
    | Gaor(phi,psi) ->
          (program |- phi) epsilon n sigma sc;
          (program |- psi) epsilon n sigma sc
(*
```

The rule for $-\!\!*R$ is

$$\frac{\langle\phi,\Gamma|s\rangle_n\backslash\langle\Delta|s'\rangle_{n'} \vdash_o \psi}{\langle\Gamma|s\rangle_n\backslash\langle\Delta'|s'\rangle_{n'} \vdash_o \phi -\!\!* \psi}\,-\!\!*R$$

```
*)
    | Gmimp(phi,psi,f) ->
          let exphi = explode (load phi) in
  let newprog = if f then (gamma * exphi,s) else (exphi * gamma,s) in
              (newprog |- psi) epsilon n sigma
                  (fun (delta,s') epsilon' n' ns ->
                      if n' > n
                      then sc (delta--exphi,s') epsilon' n' ns
                      else if (B.check exphi delta)
                      then sc (delta,s') epsilon' n' ns)


(*
```

There are various rules for $\to R$.

```
 *)
```

```
      | Gaimp(phi, psi) ->
            let exphi = explode (load phi)
            in
            let lst = match s with
(*
```

$$\frac{\langle \phi ; \Gamma | [] \rangle_n \backslash \langle \boldsymbol{\varepsilon} | [] \rangle_{n'} \vdash_o \psi}{\langle \Gamma | [] \rangle_n \backslash \langle \varepsilon | [] \rangle_{n'} \vdash_o \phi \to \psi} \to R$$

```
 *)
               | [] -> []
(*

 *)

               | (Open (a,tick))::l -> Open (exphi & a,tick)::l
(*
```

$$\frac{\langle \phi ; \Gamma | \boxed{\phi}^{\varepsilon} :: s \rangle_n \backslash \langle \boldsymbol{\varepsilon} | \{\Delta, \phi\} :: s \rangle_{n'} \vdash_o \psi}{\langle \Gamma | \boxtimes :: s \rangle_n \backslash \langle \Delta | \boxtimes :: s \rangle_{n'} \vdash_o \phi \to \psi} \to R^i$$

```
 *)
               | Locked ::l -> (Open (exphi,On))::l
               | _ ::l -> raise (Failure "Weird s in Gaimp")
            in
                ((exphi & gamma,lst) |- psi) Epsilon n sigma
                    (fun (delta,l) na tf ->
                        let cont, nt = match l with
                            [] -> delta -- exphi, tf
                          | (Full (gamma',_ ))::l -> gamma' -- exphi, tf
                                | (Open (a,On)):: l -> gamma, tf + 1
                          | _ -> delta -- exphi, tf
                        in
```

```
             sc (cont,s) epsilon nt)
(*
```

```
 *)
  | Gex ((Varbind v as vr), g) ->
     let var = newAddvar(v)
     in let nc = (B.L (Const v,var)) & sigma in
       (program |- gsubst var vr g) epsilon n nc sc
(*
```

```
 *)
  | Gexnew ((Varbind v as vr), g) ->
     let var = newMulvar(v)
     in let nc = (B.L (Const v,var)) * sigma in
       (program |- gsubst var vr g) epsilon n nc sc
  | Gall (v, g) -> raise (Failure "Forall in goals not implemented yet")
  | Gallnew (v, g) -> raise (Failure "Allnew in goals not implemented yet")

(* and what used to be called match_atom    *)

and (|--) (gamma,s) alpha epsilon n sigma sc =
 let rec test c =
  let r t =
   match (t,Term.dereference alpha sigma) with
     (Const a, Const b) -> if a = b then c else emptyc
    | (Appl(a,_), Appl(b,_)) -> if a = b then c else emptyc
    | _ -> emptyc
  in
      match c with
      Catom t1  -> r t1
     | Cmimp(g,t1,n) -> r t1
     | Caimp(g,t1) -> r t1
     | Call (v,cl) ->
```

```
              if test cl = emptyc then emptyc else c
        | Callnew (v,cl) ->
              if test cl = emptyc then emptyc else c
        | Cmand(c1,c2) ->
              if test c1 = emptyc && test c2 = emptyc then emptyc else c
        | Caand(c1,c2) ->
              if test c1 = emptyc && test c2 = emptyc then emptyc else c
and resolve = function
    B.Empty -> emptyc
  | B.L c -> test c
  | B.MB (h::[]) ->
        resolve h
  | B.MB (h::tl) ->
        let g = resolve h in
            if g = emptyc then resolve (B.MB tl)
            else g
  | B.AB (h::[]) ->
        resolve h
  | B.AB (h::tl) ->
        let g = resolve h in
            if g = emptyc then resolve (B.AB tl)
            else g
  | _ -> raise B.BunchErrorException
in let c = ref (resolve gamma)
and b = ref gamma in
    while !c <> emptyc do
        let gamma' = axiom !c gamma in
        let tfound a =
          if a = atrue then 1 else 0
        in
        let clause =
      match !c with
           Call (_,_) | Callnew (_,_) -> strip !c
       | _ -> !c
     in
       begin match clause with
(*
```

There are two rules for CutAxiom. One of them applies just when the top of the stack has got an open box with the theorem flag and the atom considered is not a member of the content of the box (ξ)

$$\frac{\langle\Gamma'|s\rangle_n\backslash\langle\Delta|s'\rangle_{n'}\vdash_o I}{\langle\Gamma(\alpha)|s\rangle_n\backslash\langle\Delta|s'\rangle_{n'}\vdash_o \alpha}\,\text{CutAxiom}^{\dagger}\qquad\frac{\langle\Gamma'|\boxed{\xi}::s\rangle_n\backslash\langle\Delta|s'\rangle_{n'}\vdash_o I}{\langle\xi;\Gamma(\alpha)|\boxed{\xi}^{\varepsilon}::s\rangle_n\backslash\langle\Delta|s'\rangle_{n'}\vdash_o \alpha}\,\text{CutAxiom}^{i\dagger}$$

```
*)
        | Catom alpha' ->
           Term.unify alpha' alpha (fun sigma' ->
             print1 alpha alpha' sigma';
             let s' =
              match s with
                Open (xi,On)::l ->
                  if not (B.mem clause xi) then
                    Open (xi,Off)::l
                  else s
               | _ -> s
             in
             ((gamma',s') |- i) epsilon
               (n + (tfound alpha'))
               sigma' sc) sigma
(*
```

The rule for $-\!\!*L$ is

$$\frac{\langle\Gamma'|s\rangle_n\backslash\langle\Delta|s'\rangle_{n'}\vdash_o \phi}{\langle\Gamma(\phi-\!\!*\alpha)|s\rangle_n\backslash\langle\Delta|s'\rangle_{n'}\vdash_o \alpha}\,-\!\!*L^{\dagger}$$

```
*)
```

```
| Cmimp(phi, alpha', k) ->
   let gamma' =
    if k = 0 then B.mnw !c gamma
    else if k = 1 then gamma'
    else B.replace (B.L(reduce !c))
        !c (B.mnw !c gamma)
   in
   Term.unify alpha' alpha (fun sigma' ->
     print1 alpha alpha' sigma';
     ((gamma',s) |- phi) epsilon
       (n + (tfound alpha'))
       sigma' sc) sigma
```

(*

$$\frac{\langle\Theta;\phi\to\alpha|[]\rangle_0\backslash\langle\varepsilon|[]\rangle_m\vdash_o\phi \qquad \langle\Gamma'|s\rangle_n\backslash\langle\Delta|s'\rangle_{n'}\vdash_o I}{\langle\Gamma(\Theta;\phi\to\alpha)|s\rangle_n\backslash\langle\Delta|s'\rangle_{n'}\vdash_o\alpha}\to L^{\dagger}$$

*)

```
| Caimp(phi, alpha') ->
   let theta = B.getabunch !c gamma in
     Term.unify alpha' alpha
       (fun sigma' ->
         print1 alpha alpha' sigma';
         ((theta,[]) |- phi) Epsilon 0 sigma'
           (fun _ _ _ sigma'' ->
             ((gamma',s) |- i) epsilon n
               sigma'' sc))
       sigma
```

(*

We can still get a multiplicative or additive conjunction if it was wrapped in a quantifier

*)

```
| Cmand(_,_) | Caand(_,_) ->
```

```
                      let program = B.replace (explode (B.L clause)) !c gamma
                      in
                          ((program,s) |-- alpha) epsilon n sigma sc
                | _ -> backtrack ""
              end;
              b := B.remfirst !c !b;
              c := resolve !b
      done;
      match alpha with
(*
```

The rule for $\top$Unit is

$$\frac{}{\langle\Gamma|s\rangle_n\backslash\langle\Gamma|s\rangle_{n+1}\vdash_o\top}\,\top\text{Unit}$$

```
 *)
        | Const "T" -> sc (gamma,s) epsilon (n + 1) sigma
(*
```

```
 *)
        | Const "I" -> (* gmunit *)
              if epsilon = Epsilon && (gamma <> B.Empty)
(*
```

If $n = 0$ we use the rule for Unit^{ii}

$$\frac{}{\langle\phi;\Gamma|\boxed{\phi}^{\varepsilon}::s\rangle_n\backslash\langle\boldsymbol{\varepsilon}|\{\Gamma,\phi\}::s\rangle_n\vdash_o I}\,\text{Unit}^{ii}$$

```
 *)
              then if n = 0 then match s with
                | (Open (phi,tick))::l ->
                      sc (B.Empty,(Full (gamma,phi)::l)) Epsilon 0 sigma
                | _ -> ()
(*
```

Else, we use the rule Unit^i

$$\frac{}{\langle\Gamma|s\rangle_n\backslash\langle\varepsilon|s\rangle_0 \vdash_o I}\ \text{Unit}^i(n > 0)$$

```
 *)
              else (* n > 0 *)
                  sc (B.Empty,s) Epsilon 0 sigma
(*
```

Finally, if ε is not present or the bunch is empty, there is no reason not to succeed, and we use the rule Unit

$$\frac{}{\langle\Gamma|s\rangle_0\backslash\langle\Gamma|s\rangle_0 \vdash_o I}\ \text{Unit}$$

```
 *)
              else sc (gamma,s) epsilon n sigma
(*
```

If none of these cases applies we backtrack, but we should never reach this case

```
 *)
        | _ -> backtrack ""
and strip = function
    Call ((Varbind s) as v,cl) -> csubst (newAddvar(s)) v (strip cl)
  | Callnew ((Varbind s) as v,cl) -> csubst (newMulvar(s)) v (strip cl)
  | cl -> cl
and reduce = function
    Call (v,cl) -> Call (v,reduce cl)
  | Callnew (v,cl) -> Callnew (v,reduce cl)
  | Cmimp (phi, alpha, n)-> Cmimp(phi, alpha, n-1)
  | cl -> cl


(* ... *)
```

Appendix A

Uncommented code of main modules

For a fully commented version of this code, see Chapter 8.

A.1 bunch.ml

```
exception Found
exception BunchErrorException
exception BunchMatchException

type ’a node =
    Leaf of ’a | Add | Mul
  | End | OBracket | CBracket

type ’a bunch =
    Empty
  | L of ’a
  | MB of ’a bunch list
  | AB of ’a bunch list

let mjoin ba bb =
  let f a b =
    match (a,b) with
        (MB m , MB n) -> m@n
      | (MB m, _ ) -> m@[b]
      | ( _ , MB m) -> a::m
      | _  -> a::[b]
  in
      if ba = Empty then bb
      else if bb = Empty then ba
      else MB (f ba bb)

let ajoin ba bb =
  let f a b =
    match (a,b) with
        (AB m , AB n) -> m@n
      | (AB m, _ ) -> m@[b]
      | ( _ , AB m) -> a::m
      | _  -> a::[b]
  in
      if ba = Empty then bb
      else if bb = Empty then ba
      else AB (f ba bb)

let rec arity b =
  let f l = (List.length l)::List.concat (List.map arity l)
  in
      match b with
          Empty -> [0]
        | L a -> [1]
        | MB m -> 1::(f m)
        | AB m -> f m
```

```
let rec proc f =
  let g j c l = j (proc f (List.hd l))(proc f (c (List.tl l)))
  and fmb l = MB l
  and fab l = AB l
  in function
      Empty -> Empty
    | L a -> f a
    | MB (h::[]) -> proc f h
    | MB m -> g mjoin fmb m
    | AB (h::[]) -> proc f h
    | AB m -> g ajoin fab m

let rec bmap f =
  let g l = List.map (bmap f) l
  in function
      Empty -> Empty
    | L a -> L (f a)
    | MB m -> MB (g m)
    | AB m -> AB (g m)

let rec bzip b1 b2 =
  match (b1,b2) with
      (Empty,Empty) -> Empty
    | (L a , L b) -> L (a,b)
    | (MB (m1::[]), MB (m2::[])) -> bzip m1 m2
    | (MB (m1::tl1), MB (m2::tl2)) ->
          mjoin (bzip m1 m2) (bzip (MB tl1) (MB tl2))
    | (AB (m1::[]), AB (m2::[])) -> bzip m1 m2
    | (AB (m1::tl1), AB (m2::tl2)) ->
          ajoin (bzip m1 m2) (bzip (AB tl1) (AB tl2))
    | _ -> raise (Failure "Mismatch in bzip")
```

```
let rec biter f =
  let g l =  List.iter (biter f) l
  in function
      Empty -> ()
    | L a -> f a
    | MB m -> g m
    | AB m -> g m

let bexists f bunch =
  let g a = if f a then raise Found else ()
  and l = ref false in
      begin try biter g bunch;()
      with Found -> l := true
      end;
      !l

let mem a bunch =
  let f b = if a == b then raise Found else ()
  and l = ref false in
      begin try biter f bunch;()
      with Found -> l := true
      end;
      !l

let subbunch a bunch =
  let f b = if not (mem b bunch) then raise Not_found else ()
  and l = ref true in
      begin try biter f a;()
      with Not_found -> l := false
      end;
      !l

let rec intersection a = function
    Empty -> Empty
  | (L c) as bunch -> if mem c a then bunch else Empty
  | MB (h::[]) -> intersection h a
  | MB (h::tl) -> mjoin (intersection h a) (intersection (MB tl) a)
  | AB (h::[]) -> intersection h a
  | AB (h::tl) -> ajoin (intersection h a) (intersection (AB tl) a)
  | _ -> raise BunchErrorException

let check sb bunch =
  let f b = if mem b bunch then raise Found else ()
  and l = ref true in
      begin try biter f sb;()
      with Found -> l := false
      end; !l

let rec remfirst a bunch =
  let f b = if b == a then Empty else L b in
      proc f bunch

let rec axiom a = function
    Empty -> Empty
  | (L c) as bunch -> if c == a then Empty else bunch
  | MB (h::[]) ->
        axiom a h
  | MB (h::tl) ->
        let ntl =
          if List.length tl = 1 then List.hd tl
          else MB tl
        in
            if mem a h then
                mjoin (axiom a h) ntl
            else mjoin h (axiom a ntl)
  | AB (h::[]) ->
        axiom a h
  | (AB (h::tl)) as bunch ->
        if not (mem a bunch) then bunch else
        if mem a h then axiom a h
        else axiom a (AB tl)
  | _ -> raise BunchErrorException

let rec replace clause quantified = function
      Empty -> raise (Failure "empty in replace")
    | (L a) as c -> if a == quantified then clause else c
    | MB (h::[]) -> replace clause quantified h
    | MB (h::tl) ->
          if mem quantified h then MB (replace clause quantified h::tl)
          else mjoin h (replace clause quantified (MB tl))
    | AB (h::[]) -> replace clause quantified h
    | AB (h::tl) ->
          if mem quantified h then AB (replace clause quantified h::tl)
          else ajoin h (replace clause quantified (AB tl))
    | _ -> raise BunchErrorException

let rec subtract sub bunch =
  match sub with
      Empty -> bunch
    | L a -> axiom a bunch
    | MB (h::[]) ->
          subtract h bunch
    | MB (h::tl) ->
          subtract h (subtract (MB tl) bunch)
    | AB (h::tl) -> subtract h bunch
    | _ -> raise BunchErrorException

let rec getabunch a = function
    Empty -> raise (Failure "empty in getabunch")
  | (L c) as bunch -> if c == a then bunch
            else raise (Failure "-")
  | MB (h::[]) ->
        getabunch a h
  | MB (h::tl) ->
        if mem a h then getabunch a h
        else getabunch a (MB tl)
  | (AB list) as bunch ->
        let rec f = function
            L c -> c == a
          | _ -> false
        and g  = function
            (h::[]) -> getabunch a h
          | (h::tl) ->
                if mem a h then getabunch a h
                else g tl
          | _ -> raise BunchErrorException
        in
            if (List.fold_right (or) (List.map f list) false)
            then bunch
            else g list
  | _ -> raise BunchErrorException

let rec mnw a = function
```

```
   |  MB (h::[]) -> if mem a h then mnw a h else h
   |  MB (h::tl) -> if mem a h then mjoin (mnw a h) (MB tl)
                    else mjoin h (mnw a (MB tl))
   |  AB (h::[]) -> mnw a h
   |  AB (h::tl) -> if mem a h then mnw a h
                    else mnw a (AB tl)
   | cl -> cl

let rec build_bunch = parser
    [< e = build_mult; e1 = build_rest_add e >] -> e1
and build_rest_add e = parser
    [< 'Add; e1 = build_mult;
        e2 = build_rest_add (ajoin e e1) >] -> e2
  | [< >] -> e
and build_mult = parser
    [< e = build_leaf; e1 = build_rest_mul e >] -> e1
and build_rest_mul e = parser
    [< 'Mul; e1 = build_leaf;
        e2 = build_rest_mul (mjoin e e1) >] -> e2
  | [< >] -> e
and build_leaf = parser
    [< 'Leaf a >] -> L a
  | [< 'OBracket; e = build_bunch; 'CBracket >] -> e
  | [< 'End >] -> Empty


(* for printing; f: 'a -> string *)
let rec bunch_to_string f =
  let rec g c = function
      [] -> ""
    | (h::[]) -> bunch_to_string f h
  | (h::tl) -> bunch_to_string f h ^ c ^ g c tl
 in function
  Empty -> "Empty"
 | L a -> f a
 | MB l -> "("^(g "," l) ^ ")"
 | AB l -> "("^(g ";" l)^")"

let print_bunch f b = print_string(bunch_to_string f b)

let pib = print_bunch string_of_int
let psb = print_bunch (function x -> x)

let lexer = Genlex.make_lexer["(";")";",";";";":";"."]

let rec parse s = match s with parser
  [< ’Genlex.Kwd "(" >] -> [< ’OBracket ; parse s >]
 | [< ’Genlex.Kwd ")" >] -> [< ’CBracket ; parse s >]
 | [< ’Genlex.Kwd "," >] -> [< ’Mul ; parse s >]
 | [< ’Genlex.Kwd ";" >] -> [< ’Add ; parse s >]
 | [< ’Genlex.Ident i >] -> [< ’Leaf i ; parse s >]
 | [< ’Genlex.Int i >] -> [< ’Leaf (string_of_int i) ; parse s >]
 | [< >] -> [< ’End >]
```

A.2 interpreter.ml

```
module P = Parser
module B = Bunch

type tag = Term.tag = Add | Mul

type term = Term.term  =
  Bvar of string
  | Evar of tag*string*int*string
  | Const of string
  | Appl of string * (term Bunch.bunch)
type varbind = Term.varbind = Varbind of string
type goal = P.goal
type clause = P.clause

type tflag = int
type eflag = On | Off

type listused =
    Locked
  | Open of clause B.bunch * eflag
  | Full of clause B.bunch * clause B.bunch
type context = Epsilon | Leavesome

exception Finished
exception End
exception Failed of clause B.bunch

open P

let mtrue, atrue = Const "I", Const "T"

(* --------------------------------------------------------------------- *)
(* The only unit actually used is gmunit = multiplicative unit as a goal *)
let cmunit = Catom mtrue
let caunit = Catom atrue
let gmunit = Gatom mtrue
let gaunit = Gatom atrue
let emptyc = Catom (Const "")
let i = gmunit


(* ---------------------------------------------------------------- *)
(* Define some operators to improve readability *)
let (--) a b = B.subtract b a
let (&) = B.ajoin
let ( * ) = B.mjoin


(* ---------------------------------------------------------------- *)
(* Printing and debugging functions   *)


let debug = ref false


let setdebug t = debug := t


let print p g ctxt = if not !debug then () else
  let s = match g with
      Gatom a -> Term.printTerm ctxt a
    | _ -> printGoal ctxt g
  and pc = printClause ctxt
  in
      B.print_bunch pc p;
      print_string (" |- "^s^"∧")


let print_context ctxt =
  let f (x,y) = "("^(Term.printTerm1 ctxt x)
    ^ ","
    ^(Term.printTerm1 ctxt y)
    ^ ")"
  in B.print_bunch f ctxt; print_string "∧"
```

```
exception Found

let backtrack s = () (*; print_string (s^"∧")*)

(* ----------------------------------------------------------------- *)
(* * Left and & Left rules, to be made at the beginning    *)

let rec explode bunch =
  let f = function
      Cmand (c1,c2) ->
          (explode (B.L c1)) * (explode (B.L c2))
    | Caand (c1,c2) ->
          (explode (B.L c1)) & (explode (B.L c2))
    | c -> B.L c
  in
      B.proc f bunch

(* ----------------------------------------------------------------- *)
(* Create logic variables with a unique name    *)
let tag = ref ""

let next_tag () =
   let rec f n =
      let a = Char.code ’a’ in
      let range = Char.code ’z’ - a + 1
      in
      !tag.[n] <- Char.chr(((Char.code (!tag.[n])-a + 1) mod range) + a);
      if !tag.[n] = ’a’ then
      if n = 0 then (tag := !tag ^ "a")
      else f (n-1)
      else ()
   in let n = String.length !tag in
       if n = 0 then tag := "a" else f(n-1);
       String.copy !tag

let next_evar = function
    Evar(t,s,n,b) -> Evar(t,s,n+1,b)
  | _ -> raise (Failure "in next_evar()")



let newAddvar(s) = Evar(Add,next_tag(),1,s)
let newMulvar(s) = Evar(Mul,next_tag(),1,s)



(*-------------------------------------------------------------------*)
(* Substitutions and the like  *)

let shadow (Varbind v1) (Varbind v2) = (v1 = v2)

let formsubst t x =
  let rec gsb = function
      Gatom a -> Gatom (Term.subst t x a)
    | Gcommand a -> Gcommand (Term.subst t x a)
    | Gaand(g1,g2) -> Gaand(gsb g1, gsb g2)
    | Gmand(g1,g2) -> Gmand(gsb g1, gsb g2)
    | Gaor(g1,g2) -> Gaor(gsb g1, gsb g2)
    | Gmimp(c,g,f) -> Gmimp(csb c, gsb g,f)
    | Gaimp(c,g) -> Gaimp(csb c, gsb g)
    | Gex(v,g) -> Gex(v,if shadow x v then g else gsb g)
    | Gexnew(v,g) -> Gexnew(v,if shadow x v then g else gsb g)
    | Gall(v,g) -> Gall(v,if shadow x v then g else gsb g)
    | Gallnew(v,g) -> Gallnew(v,if shadow x v then g else gsb g)
  and csb = function
      Catom a -> Catom (Term.subst t x a)
    | Caand(c1,c2) -> Caand(csb c1, csb c2)
    | Cmand(c1,c2) -> Cmand(csb c1, csb c2)
    | Cmimp(g,a,n) -> Cmimp(gsb g, Term.subst t x a,n)
    | Caimp(g,a) -> Caimp(gsb g, Term.subst t x a)
    | Call(v,c) -> Call(v,if shadow x v then c else csb c)
    | Callnew(v,c) -> Callnew(v,if shadow x v then c else csb c)
  in
      (gsb, csb)

let gsubst t x = (fst (formsubst t x))

let csubst t x = (snd (formsubst t x))

(* ------------------------------------------------------------------- *)
(* Implementation of the CutAxiom rule   *)

let rec axiom a = function
    B.Empty -> B.Empty
  | (B.L c) as bunch -> if c == a then B.Empty else bunch
  | B.MB (h::[]) ->
        axiom a h
  | B.MB (h::tl) ->
        let ntl =
          match tl with
             [x] -> x
            | x1 -> B.MB x1
        in
        if B.mem a h then
            (axiom a h) * ntl
        else h * (axiom a ntl)
  | B.AB (h::[]) ->
        axiom a h
  | (B.AB (h::tl)) as bunch ->
        if not (B.mem a bunch) then bunch else
        if B.mem a h then axiom a h
        else axiom a (B.AB tl)
  | _ -> raise B.BunchErrorException


(* ----------------------------------------------------------------- *)

let printclause = P.printClause B.Empty

let readgoal = P.read_goal

let isModule =
  let test s = match s.[0] with
      'A'..'Z' -> true
    | _ -> false
  in function
      Bvar s -> false
    | Evar _ -> false
    | Const s -> test s
    | Appl(s,b) -> test s

let rec map f c = function
    B.Empty -> raise (Failure "empty in map (interpreter)")
  | B.L a -> f a c
  | B.MB (m::[]) -> map f c m
  | B.MB (m::tl) -> map f (map f c (B.MB tl)) m
  | B.AB (m::[]) -> map f c m
  | B.AB (m::tl) -> map f (map f c (B.AB tl)) m
  | _ -> raise (Failure "something strange in map (interpreter)")

let filesubst pars vars clauses=
  let b = B.bzip pars vars in
  let f (p,v) c = csubst p v c in
      B.bmap (fun c -> map (fun z -> f z) c b) clauses

let rec mklist_of_vars = function
    B.Empty -> []
  | B.L (Varbind a) -> [a]
  | B.MB list -> List.concat (List.map mklist_of_vars list)
  | B.AB list -> List.concat (List.map mklist_of_vars list)

let readfile s =
        let file = P.openFile s in
        let vars = P.readVariables file in
            P.readClauses (mklist_of_vars vars) file

let loadModule = function
    Bvar s -> raise (Failure "bound variable in load module")
  | Evar _ -> raise (Failure "logic variable in load module")
  | Const s -> readfile (s^".bi")
  | Appl(s,b) ->
        let file = P.openFile (s^".bi") in
        let vars = P.readVariables file in
        let clauses = P.readClauses (mklist_of_vars vars) file in
            filesubst b vars clauses

let rec write ctxt = function
```

```
    [] -> ()
  | (x::tl) -> let t = Term.findbyname x ctxt in
        print_string (x^":"^(Term.printTerm ctxt t))



let executeCommand ctxt = function
    Const "exit" -> raise End
  | Const "nl" ->  print_string "∧"
  | Const "sp" ->  print_string " "
  | Appl("write", b) ->
        let f t = print_string ((Term.printTerm ctxt t)^" ")
        in B.biter f b
  | s -> raise (Failure ("Unknown keyword "^(Term.printTerm ctxt s)))

let rec load = function
    Catom a as phi ->
        if isModule a then loadModule a
        else explode (B.L phi)
  | Caand(phi,psi) -> load phi & load psi
  | Cmand(phi,psi) -> load phi * load psi
  | s -> explode (B.L s)

(* -------------------------------------------------------------------- *)
(* This is what used to be called solve   *)

let rec (|-) ((gamma,s)as program) goal epsilon n sigma sc =
  print gamma goal sigma;
  match goal with
    | Gatom alpha -> (program |-- alpha) epsilon n sigma sc
    | Gcommand s -> executeCommand sigma s;
          sc program epsilon n sigma
    | Gaand (phi, psi) ->
          (program |- phi) epsilon n sigma
              (fun (delta, s’) _ n’ sigma’ ->
                  if epsilon <> Epsilon then (* / R, / Ri, / Rii *)
                      if n’ = n then (* / R *)
                          ((gamma -- delta , []) |- psi) Epsilon 0 sigma’
                              (fun _ _ _ -> sc (delta,s’) epsilon n)
                      else if n’ > n then(* / Ri, / Rii *)
                          (program |- psi) epsilon n sigma’
                              (fun (delta’,_) _ n’’ ->
                                  if n’’ = n && B.subbunch delta’ delta (* /
Ri *)
                                  then  sc (delta’,s’) epsilon n
                                  else if n’’ > n (* / Rii *)
                                  then sc (B.intersection delta delta’,s’)
                                          epsilon n’
                                  else raise (Failure "strange n in / Rii"))
                      else raise (Failure "strange n in / R ")
                  else match s’ with
                      (Full(delta,xsi))::l -> (* /  Riv *)
                          ((xsi&(gamma--delta),[]) |- psi) Epsilon 0
                              sigma’ (fun (empty, _ ) _ n’’->
                                  sc (empty,s’) Epsilon
                                      (if n’ > n && n’’ > n then n’ else n))
                    | _ ->   (* / Riii *)
                          (program |- psi) Epsilon n sigma’
                              (fun np _ n’’ ->
                                  sc np Epsilon
                                      (if n’ > n && n’’ > n then (n+1) else n)
                                      ))
    | Gmand(phi,psi) ->
```

```
      ((gamma, Locked::s) |- phi) Leavesome n sigma
          (fun (gamma',Locked::s) _ n' sigma' ->
              ((gamma',s) |- psi) epsilon n' sigma' sc)
| Gaor(phi,psi) ->
      (program |- phi) epsilon n sigma sc;
      (program |- psi) epsilon n sigma sc
| Gmimp(phi,psi,f) ->
      let exphi = explode (load phi) in
      let newprog = if f then (gamma * exphi,s) else (exphi * gamma,s) in
          (newprog |- psi) epsilon n sigma
              (fun (delta,s') epsilon' n' ns ->
                  if n' > n
                  then sc (delta--exphi,s') epsilon' n' ns
                  else if (B.check exphi delta)
                  then sc (delta,s') epsilon' n' ns)


| Gaimp(phi, psi) ->
      let exphi = explode (load phi)
      in
      let lst = match s with
        | [] -> []
        | (Open (a,tick))::l -> Open (exphi & a,tick)::l
        | Locked ::l -> (Open (exphi,On))::l
        | _ ::l -> raise (Failure "Weird s in Gaimp")
      in
          ((exphi & gamma,lst) |- psi) Epsilon n sigma
              (fun (delta,l) na tf ->
                  let cont, nt = match l with
                      [] -> delta -- exphi, tf
                    | (Full (gamma',_ ))::l -> gamma' -- exphi, tf
                          | (Open (a,On)):: l -> gamma, tf + 1
```

```
            | _ -> delta -- exphi, tf
           in
             sc (cont,s) epsilon nt)
  | Gex ((Varbind v as vr), g) ->
     let var = newAddvar(v)
     in let nc = (B.L (Const v,var)) & sigma in
       (program |- gsubst var vr g) epsilon n nc sc
  | Gexnew ((Varbind v as vr), g) ->
     let var = newMulvar(v)
     in let nc = (B.L (Const v,var)) * sigma in
       (program |- gsubst var vr g) epsilon n nc sc
  | Gall (v, g) -> raise (Failure "Forall in goals not implemented yet")
  | Gallnew (v, g) -> raise (Failure "Allnew in goals not implemented yet")

(* and what used to be called match_atom    *)

and (|--) (gamma,s) alpha epsilon n sigma sc =
 let rec test c =
  let r t =
   match (t,Term.dereference alpha sigma) with
     (Const a, Const b) -> if a = b then c else emptyc
    | (Appl(a,_), Appl(b,_)) -> if a = b then c else emptyc
    | _ -> emptyc
  in
      match c with
      Catom t1  -> r t1
     | Cmimp(g,t1,n) -> r t1
     | Caimp(g,t1) -> r t1
     | Call (v,cl) ->
         if test cl = emptyc then emptyc else c
     | Callnew (v,cl) ->
```

```
       if test cl = emptyc then emptyc else c
    | Cmand(c1,c2) ->
       if test c1 = emptyc && test c2 = emptyc then emptyc else c
    | Caand(c1,c2) ->
       if test c1 = emptyc && test c2 = emptyc then emptyc else c
and resolve = function
  B.Empty -> emptyc
 | B.L c -> test c
 | B.MB (h::[]) ->
    resolve h
 | B.MB (h::tl) ->
    let g = resolve h in
      if g = emptyc then resolve (B.MB tl)
      else g
 | B.AB (h::[]) ->
    resolve h
 | B.AB (h::tl) ->
    let g = resolve h in
      if g = emptyc then resolve (B.AB tl)
      else g
 | _ -> raise B.BunchErrorException
in let c = ref (resolve gamma)
and b = ref gamma in
  while !c <> emptyc do
    let gamma' = axiom !c gamma in
    let tfound a =
     if a = atrue then 1 else 0
    in
    let clause =
     match !c with
          Call (_,_) | Callnew (_,_) -> strip !c
    | _ -> !c
in
    begin match clause with
      | Catom alpha' ->
            Term.unify alpha' alpha (fun sigma' ->
                print1 alpha alpha' sigma';
                let s' =
                  match s with
                      Open (xi,On)::l ->
                          if not (B.mem clause xi) then
                              Open (xi,Off)::l
                          else s
                    | _ -> s
                in
                ((gamma',s') |- i) epsilon
                    (n + (tfound alpha'))
                    sigma' sc) sigma
      | Cmimp(phi, alpha', k) ->
            let gamma' =
              if k = 0 then B.mnw !c gamma
              else if k = 1 then gamma'
              else B.replace (B.L(reduce !c))
                      !c (B.mnw !c gamma)
            in
            Term.unify alpha' alpha (fun sigma' ->
                print1 alpha alpha' sigma';
                ((gamma',s) |- phi) epsilon
                    (n + (tfound alpha'))
                    sigma' sc) sigma
      | Caimp(phi, alpha') ->
            let theta = B.getabunch !c gamma in
```

```
                          Term.unify alpha' alpha
                              (fun sigma' ->
                                  print1 alpha alpha' sigma';
                                  ((theta,[]) |- phi) Epsilon 0 sigma'
                                      (fun _ _ _ sigma'' ->
                                          ((gamma',s) |- i) epsilon n
                                              sigma'' sc))
                              sigma
                | Cmand(_,_) | Caand(_,_) ->
                      let program = B.replace (explode (B.L clause)) !c gamma
                      in
                          ((program,s) |-- alpha) epsilon n sigma sc
                | _ -> backtrack ""
              end;
              b := B.remfirst !c !b;
              c := resolve !b
      done;
      match alpha with
        | Const "T" -> sc (gamma,s) epsilon (n + 1) sigma
        | Const "I" -> (* gmunit *)
              if epsilon = Epsilon && (gamma <> B.Empty)
              then if n = 0 then match s with
                | (Open (phi,tick))::l ->
                      sc (B.Empty,(Full (gamma,phi)::l)) Epsilon 0 sigma
                | _ -> ()
              else (* n > 0 *)
                  sc (B.Empty,s) Epsilon 0 sigma
              else sc (gamma,s) epsilon n sigma
        | _ -> backtrack ""
and strip = function
    Call ((Varbind s) as v,cl) -> csubst (newAddvar(s)) v (strip cl)
```

```
  | Callnew ((Varbind s) as v,cl) -> csubst (newMulvar(s)) v (strip cl)
  | cl -> cl
and reduce = function
    Call (v,cl) -> Call (v,reduce cl)
  | Callnew (v,cl) -> Callnew (v,reduce cl)
  | Cmimp (phi, alpha, n)-> Cmimp(phi, alpha, n-1)
  | cl -> cl

(* ------------------------------------------------------------------ *)
(* The external interface   *)

let rec getvars = function
    Gex (Varbind v, g) -> B.L (Term.Const v) & getvars g
  | Gexnew (Varbind v, g) ->  B.L (Term.Const v) * getvars g
  | _ -> B.Empty

let prove bc g =
  tag := "";
  ((explode bc,[])|- g) Epsilon 0 B.Empty
      (fun (leftover,s) epsilon n subst ->
          Term.printSubst (getvars g) subst;
          if epsilon = Epsilon && (leftover <> B.Empty) && n = 0 then
              begin print_string "failed with left over: ";
              B.print_bunch (P.printClause subst) leftover;
              print_string "\n**********\n";
              raise Finished
              end
          else
              begin
              print_string "yes."\n**********\n";
              let s = read_line() in
```

```
               if s = ";" then ()
               else raise Finished
 end);
print_string "no."\n**********\n";
raise Finished
```

Bibliography

[Aït-Kaci 91] H. Aït-Kaci. *Warren's Abstract Machine: A Tutorial Reconstruction.* MIT Press, Cambridge, MA, 1991.

[Alberti et al. 98] F. Alberti, N. Ghani, V. de Paiva, E. Ritter. *An efficient linear abstract machine with single-pointer property.* unpublished, 1998

[Armelin Pym 02] P. Armelín, D. Pym. *Bunched Logic Programming.* Proc. IJCAR 2001, LNAI 2083:289–304, 2001.

[Avron 92] A. Avron. *Whither relevance logic?* Journal of Philosophical Logic 21:243–281, 1992.

[Bollen 91] A.W. Bollen. *Relevant Logic Programming.* Journal of Automated Reasoning, 7:563–585, 1991.

[Bugliesi et al. 93] M. Bugliesi, E. Lamma, P. Mello. *Modularity in Logic Programmimg.* Journal of Logic Programming, Elsevier, 1993.

[Cervesato et al. 00] I. Cervesato, J. Hodas, F. Pfenning. *Efficient resource management for linear logic proof search.* Theoretical Computer Science, 232:133–163, 2000.

[Clocksin 97] W. Clocksin. *Clause and effect.* Springer-Verlag, 1997.

[Clocksin Mellish 94] W. Clocksin, C. Mellish. *Programming in Prolog.*. Springer-Verlag, 1994.

[Cousineau Mauny 98] G. Cousineau, M. Mauny. *The Functional Approach to Programming.* Cambridge University Press, 1998.

[Dyckhoff 92] R. Dyckhoff. *Contraction-free sequent calculi for intuitionistic logic.* Journal of Symbolic Logic, 57:3, 1992.

[Došen 92] K. Došen. *The first axiomatization of relevant logic.* Journal of Philosophical Logic 21:339–356, 1992.

[Elliot Pfenning 91] C. Elliot, F. Pfenning. *A Semi-Functional Implementation of a Higher-Order Logic Programming Language.* In P. Lee, editor, Topics in Advanced Language Implementation, 289–325, MIT Press, 1991.

[Gabbay 91] D. Gabbay, F. Kriwaczek. *A Family of Goal Directed Theorem Provers Based on Conjunction and Implication: Part 1.* Journal of Automated Reasoning, 7:511–536, 1991.

[Gabbay 92] D. Gabbay. *Extending the Curry-Howard Interpretation to linear, relevant and other resource logics.* Journal of Symbolic Logic, 1992.

[Galmiche Méry 01] D. Galmiche, D. Méry. *Proof-search and countermodel generation in propositional* **BI***Logic.* TACS 2001, 2001.

[Galmiche et al. 02] D. Galmiche, D. Méry, D. Pym. *Resource Tableaux (Extended Abstract).* Proceedings of CSL ’02, Edinburgh, 2002.

[Girard 87] J.-Y. Girard. *Linear logic.* Theoretical Computer Science, 1–102, 1987.

[Girard et al. 89] J.-Y. Girard, Y. Lafont, P. Taylor. *Proofs and Types.* Cambridge University Press, 1989.

[Gunter 92] C.A. Gunter. *Semantics of Programming Languages.* The MIT Press, 1992.

[Harland 92] J. Harland. *On Normal Forms and Equivalence for Logic Programs.* Proceedings of the Joint International Conference and Symposium on Logic Programming 146-160, Washington D.C, 1992.

[Harland 93] J. Harland. *Success and Failure for Hereditary Harrop Formulae.* Journal of Logic Programming, 17:1–29, 1993.

[Harland Pym 97] J. Harland, D. Pym. *Resource-distribution via Boolean constraints (extended abstract).* Proc. CADE-14, LNAI, 1249: 222–236, Springer, 1997.

[Harland et al. 96] J.A. Harland, D.J. Pym, M. Winikoff. *Programming in Lygon: an overview.* In M. Wirsing and M. Nivat, editors, LNCS 1101: 391–405, 1996.

[Haynes 87] C. Haynes. *Logic Continuations.* Journal of logic programming, Elsevier, 1987.

[Hodas 94] J.S. Hodas. *Logic Programming in Intuitionistic Linear Logic: Theory, Design, and Implementation.* PhD Thesis, 1994.

[Hodas Miller 94] J.S. Hodas, D. Miller. *Logic programming in a fragment of intuitionistic linear logic.* Information and Computation, 110(2):327–365, 1 May 1994.

[Ishtiaq Pym 98] S. Ishtiaq, D. Pym. *A Relevant Analysis of Natural Deduction.* J. Logic Computat. 6:809–838, 1998.

[Ishtiaq O'Hearn 01] S. Ishtiaq, P. O'Hearn. **BI***as an Assertion Language for Mutable Data Structures.* Proceedings of POPL, 2001.

[Kleene 52] S.C. Kleene. *Permutability of inferences in Gentzen's calculi LK and LJ.* Two papers on the predicate calculus. Memoirs of the American Mathematical Society, number 10, 1952.

[Kowalski 79] R. Kowalski. *Logic for Problem-solving.* North-Holland, Elsevier, 1979.

[Kripke 65] S.A. Kripke. *Semantical analysis of intuitionistic logic I.* In J. Crossley and M. Dummett, editors, Formal Systems and Recursive Functions, 92–130. North-Holland, Amsterdam, 1965.

[Lafont 88] Y. Lafont. *The Linear Abstract Machine.* Theoretical Computer Science, 59:157–180, 1988.

[Leroy 96] X. Leroy. *A modular module system.* Rapport de recherche 2866, INRIA, 1996.

[Lloyd 87] J.W. Lloyd. *Foundations of Logic Programming.* Springer-Verlag, 1987.

[Loveland Nadathur 94] D. Loveland, G. Nadathur. *Proof Procedures for Logic Programming.* in Handbook of Logic in AI and Logic Programming, D. Gabbay, C. Hogger and A. Robinson (eds.), Oxford University Press, 1994.

[Miller 81] D. Miller. *A logical analysis of modules in logic programming.* Journal of Logic Programming, 6(1& 2):431–483, 1981.

[Miller 92] D. Miller. *A Proposal for Modules in λProlog.* 1992.

[Miller et al. 91] D. Miller, G. Nadathur, F. Pfenning, A. Ščedrov. *Uniform proofs as a foundation for logic programming.* Annals of Pure and Applied Logic, 51:125–157, 1991.

[Nadathur 93] G. Nadathur. *A Proof Procedure for the Logic of Hereditary Harrop Formulas.* Journal of Automated Reasoning, 115–145, 1993.

[Nadathur Tong 99] G. Nadathur, G. Tong. *Realizing Modularity in λProlog.* Journal of Functional and Logic Programming, MIT, 1999.

[O'Hearn 99] P. O'Hearn. *Resource Interpretations, Bunched Implications and the $\alpha\lambda$-calculus*, 1999.

[O'Hearn 02] P. O'Hearn. *On Bunched Typing.* Journal of Functional Programming, 2002.

[O'Hearn Pym 99] P. O'Hearn, D.J. Pym. *The logic of bunched implications.* Bulletin of Symbolic Logic, 5(2):215–244, June 1999.

[Plotkin 81] G. Plotkin. *A structural approach to operational semantics.* Technical Report DAIMI FN-19, Aarhus University, 1981.

[Polakow Pfenning 98] J. Polakow, F. Pfenning. *Ordered linear logic programming.* Technical Report CMU-CS-98-183, Carnegie Mellon University, 1998.

[Pym 98] D. Pym. *Logic Programming with Bunched Implications.* 1998.

[Pym 99a] D. Pym. *On bunched predicate logic.* In Proc. LICS'99, 183–192. IEEE Computer Society Press, 1999.

[Pym 02] D. Pym. *The Semantics and Proof Theory of the Logic of Bunched Implications.* Applied Logic Series, Kluwer Academic Publishers, 2002.

[Pym et al. 02] D. Pym, P. O'Hearn, H. Yang. *Possible worlds and resources: The semantics of* **BI**. Submitted. Manuscript at http://www.dcs.qmw.ac.uk/~pym, 2002.

[Pym Harland 94] D. Pym, J. Harland. *A uniform proof-theoretic investigation of linear logic programming.* J. Logic Computat., 4:175–207, 1994.

[Read 88] S. Read. *Relevant Logic, A Philosophical Examination of Inference.* Basil Blackwell, 1988.

[Reynolds 93] J. Reynolds. *The Discoveries of Continuations.* Lisp and Symbolic Computation, 6, 233–247, 1993.

[Schmidt 86] D. Schmidt. *Denotational Semantics.* Allyn and Bacon, 1986.

[Tarski 56] A. Tarski. *Logic, Semantics, Methamatematics.* Oxford University Press, 1956.

[Tennant 92] N. Tennant. *Autologic.* Edinburgh University Press, 1992.

[Urquhart 72] A. Urquhart. *Semantics for relevant logics.* Journal of Symbolic Logic, 1059–1073, 1972.

[van Dalen 97] D. van Dalen. *Logic and Structure.* Springer, 1997.

[vanEmden Kowalski] M. van Emden, R. Kowalski. *The Semantics of Predicate Logic as a Programming Language.* J. ACM 23,4:733–742, 1976.

[Van Roy 94] P. Van Roy. *1983-1993: The Wonder Years of Sequential Prolog Implementation.* Journal of Logic Programming, Elsevier North-Holland, 1994.

[Vickers 1989] S. Vickers. *Topology Via Logic.* Cambridge University Press, 1989.

[Winikoff 97] M. Winikoff. *Logic Programming with Linear Logic.* PhD Thesis, 1997.

[Winikoff Harland 95] M. Winikoff, J. Harland. *Implementation and development Issues for the Linear Logic Programming Language Lygon.* Proceedings of the Eighteenth Australasian Computer Science Conference, 563–572, 1995.

[Zhang 91] G.-Q. Zhang. *Logic of Domains* Birkhäuser, 1991.

Corrections to Thesis “Programming with Bunched Implications”

December 16, 2002

Page 45: “lemma 3.6.1” should read “lemma 3.7.1”.

Page 72: the description of n and n' should say “non-negative integers” instead of just “integers”.

Page 74: in the case $-\!\!*L$ “to bring α to top-level” should read “to bring $\phi -\!\!* \alpha$ to top-level”.

Pages 81 and 82: the justification for the first step on each of the proofs should be “Definition 4.2.6”.

Page 84: in rule $\wedge R$ (and also in Table 4.1) all instances of $\langle \Delta | s \rangle$ should be changed to $\langle \Delta | s' \rangle$.

Page 85 (and Table 4.1): rule $\wedge R^{iii}$ should be changed to

$$\frac{\langle \Gamma | s \rangle_n \backslash\!\!\backslash \langle \varepsilon | s' \rangle_{n'} \vdash_o \phi \quad \langle \Gamma | s \rangle_n \backslash\!\!\backslash \langle \varepsilon | s'' \rangle_{n''} \vdash_o \psi}{\langle \Gamma | s \rangle_n \backslash\!\!\backslash \langle \varepsilon | s''' \rangle_{n'''} \vdash_o \phi \wedge \psi} \wedge R^{iii}$$

were s''' depends on s' and s'' in the following way:

```
match s', s'' with
  | $\boxed{\xi}::s$ , _    -> $\boxed{\xi}::s$
  | _    , $\boxed{\xi}::s$ -> $\boxed{\xi}::s$
  | any1,any2 -> any2
```

Notice the absence of the theorem flag in all open boxes.

Page 87: in rule $\rightarrow R^{iii}$ the proof figure should be

$$\frac{\phi; (\Gamma - \Delta) \vdash \psi}{\Gamma - \Delta \vdash \phi \rightarrow \psi} \rightarrow R$$