Agents via Mixed-mode Computation in Linear Logic: A Proposal

James Harland Michael Winikoff
School of Computer Science and Information Technology,
Royal Melbourne Institute of Technology,
GPO Box 2476V, Melbourne 3001,
Australia.
Email: {jah,winikoff}@cs.rmit.edu.au

August 31, 2001

Abstract

Agent systems based on the *Belief*, *Desire* and *Intention* model of Rao and Georgeff have been used for a number of successful applications. However, it is often difficult to learn how to apply such systems, due to the complexity of both the semantics of the system and the computational model. In addition, there is a gap between the semantics and the concepts that are presented to the programmer. In this paper we address these issues by re-casting the foundations of such systems into a logic programming framework. In particular we show how the integration of backward- and forward-chaining techniques for linear logic provides a natural starting point for this investigation. We discuss how the integrated system provides for the interaction between the proactive and reactive parts of the system, and we discuss several aspects of this interaction. In particular, one perhaps surprising outcome is the conflation of goals and plans, which may be thought of as *declarative* and *procedural* aspects of the same concept. We also discuss some possibilities (of which there seems to be a large number) for further work in this direction.

1 Introduction

An increasingly popular programming paradigm is that of *agent-oriented programming*. This paradigm, often described as a natural successor to object-oriented programming [20], is highly suited for applications which are embedded in complex dynamic environments, and is based on human concepts, such as beliefs, goals and plans. This allows a natural specification of sophisticated software systems in terms that are similar to human understanding, thus permitting programmers to concentrate on the critical properties of the application rather than getting absorbed in the intricate detail of a complicated environment. Agent technology has been used in areas for applications such as air traffic control, automated manufacturing, and maintenance tasks on the space shuttle [21].

There are many possible conceptions of agent-oriented programming. However a common theme [10, 41, 40] is that agent systems should include properties such as

- *pro-activeness:* the agent has an agenda to pursue and will persist in trying to achieve its aims
- *reactiveness:* the agent will notice and respond to changes in the environment
- *autonomy:* the agent will act without necessarily being instructed to take particular steps
- *situated:* the agent both influences and is influenced by the environment around it

Other possible attributes of agent systems include being *social*, (i.e. teaming up with other agents in order to achieve common goals), *learning* (i.e. taking note of previous actions and adjusting future actions accordingly), and *rationality*, (i.e. working to achieve its aims, and not working against them).

One of the most popular and successful technical realisations of such conceptions is the framework of Rao and Georgeff [35], in which the notions of *Belief*, *Desire* and *Intention* are central, and hence are often referred to as *BDI* agents. Roughly speaking, beliefs represent the agent's current knowledge about the world, including information about the current state of the environment inferred from perception devices (such as cameras or microphones) and messages from other agents, as well as internal information. Desires represent a state which the agent is trying to achieve, such as the safe landing of all planes currently circling the airport, or the achievement of a certain financial return on the stock market. Intentions are the chosen means to achieve the agent's desires, and are generally implemented as plans (which may be thought of as procedures which come with pre-conditions (to determine when a plan is applicable) and intended outcomes (to state what is achieved upon the successful completion of the plan)).

Rao and Georgeff gave both a logical system incorporating these concepts [35] (a version of temporal logic extended to include the appropriate notions of belief, desire and intension) and an architecture for the execution of programs following the BDI paradigm [36].

Whilst BDI-based systems have been successfully applied in a number of areas, there remain some foundational and design issues to be solved. One such issue is the "gap" between the BDI theory, which is based on branching-time temporal logic (in which there is only one past but many possible futures) and the BDI architectures on which systems such as dMARS[7], JACK[1] and JAM[19] are based. The BDI theory has an elegant description of the relationship between beliefs, desires and intentions via possible worlds, but the architectures tends to deal in beliefs, events and plans, and it is not altogether obvious how these relate to the given semantics (although the latter can clearly be seen as an inspiration and specification of the ideal behaviour). The closure of this gap is not helped by the development of purely model-theoretic approaches to the semantics of such systems, with a corresponding lack of emphasis on the proof-theoretical aspects (although [37] is a notable step in this direction).

A second difficulty, particularly when it comes to making agent-oriented programming accessible to a wide audience, is the complexity of the BDI semantics. Whilst there is no magical way to simplify an inherently complex system, it has generally been the case that successful applications of this technology has come from either the developers of the agent system or from groups who have relied on a significant amount of input from a BDI expert (who are generally in very short supply). Hence in order for BDI agents to become significantly more widespread, a simpler means of understanding the system is required.

In order to address these issues, we re-visit the foundations of the BDI framework in order to develop an alternative (and hopefully simpler) view of the semantics of an agent system. It should be noted that this is not intended as a replacement for the BDI semantics, but rather as an alternative, complementary view of the activities of the agent system. In particular, we hope to minimise the number of concepts necessary to the understanding of a simple agent system.

Our method for doing this is to re-cast the basic agent model into a logic programming framework. As noted by Kowalski and Sadri [24] the main technical question here is to determine how to incorporate actions and reactive behaviour into a logic programming environment (which has traditionally been very strong on backward-chaining methods, and thus is closely related to traditional planning techniques). Indeed, there has been a noticeable movement within logic programming to include abduction techniques into logic programming languages [22], which can be viewed as a means of determining how to make goals which fail into goals which succeed, i.e. a means of determining the actions necessary to achieve the goal.

Abductive methods have generally been studied in the context of classical logic, which may be thought of as determining what missing information would make the goal true. In agent systems, the requirement is somewhat stronger; not only does the agent need to know what needs to be achieved, but also it needs to be able to take appropriate action in order to ensure that the desired state of the world is achieved. Thus an agent system requires not just abductive reasoning, but also a suitable notion of action.

One way to provide such a notion is to use *linear logic* [12], a logic designed with bounded resources in mind. In particular, linear logic is not only a conservative extension of classical logic (so that classical reasoning, where appropriate, may be used), but also has been shown to be a natural way to model concurrency, database updates and state-based transitions[13, 17, 15]. In particular, it has been shown that actions, such as those required by the classic blocks world scenario, can be modelled simply and naturally in linear logic [25, 26]. Given also the existence of a number of logic programming languages based on linear logic (such as LO [5], Lolli [17], Forum [27], LLP [18] and Lygon [15]), it seems natural to explore the use of linear logic as a basis for BDI-style agent systems.

Some initial work in this direction was done by Abdullah-Al Amin [3]; however, that was limited to the use of a particular language (Lygon) and did not take reactive behaviour into account. In [16] it was shown how a notion of *forward-chaining* could be introduced into the standard sequent calculus for linear logic in order to provide such behaviour.

This paper thus attempts to re-cast the framework of BDI systems into a logic programming paradigm based on

Philosophy:	Belief	Desire	Intention
Theory:	Belief	Goal	Intention
Implementation:	Relational DB (or arbitrary object)	Event	Running Plan

Table 1: BDI: Philosophy vs. Theory vs. Implementation

linear logic. Whilst this may be somewhat cynically viewed as yet another exercise in making things look like nails when one only has a hammer, it seems that there is not only a potentially very natural fit between the requirements of a BDI-style system and the properties of linear logic, but also that this exercise in itself will help shed light on the essential properties of BDI systems. This, in turn, will hopefully lead to the simplification of the programming model for such systems.

This paper is organised as follows: in §2 we given an overview of BDI systems, linear logic and linear logic programming, and in §3 we discuss the issues in the design of BDI agent systems. In §4 we show how these issues are addressed by the combination of backward- and forward-chaining in linear logic and in §5 we discuss our conclusions and possibilities for further work.

2 Background

2.1 BDI Agents

The BDI model (Belief Desire Intention) of Rao and Georgeff [35, 36] is a popular model for intelligent agents which has its basis in philosophy [6] and offers a *logical theory* which defines the mental attitudes of Belief, Desire, and Intention using a modal logic; a *system architecture*; a *number of implementations of this architecture* (e.g. PRS, JAM, dMars, JACK); and *applications* demonstrating the viability of the model. The central concepts in the BDI model are

Beliefs: The agent's information about the environment;

Desires: Objectives to be accomplished, possibly with each objective's associated priority/payoff;

Intentions: The currently chosen course of action; and

Plans: Means of achieving certain future world states. Intuitively, plans are an abstract specification of both the means for achieving certain desires and the options available to the agent. Each plan has (i) a body describing the primitive actions or sub-goals that have to be achieved for plan execution to be successful; (ii) an invocation condition which specifies the triggering event, and (iii) a context condition which specifies the situation in which the plan is applicable.

The BDI model has developed over about 15 years and there are certainly strong relationships between the theoretical work and implemented systems. The paper [36] describes an abstract architecture which is instantiated in systems such as dMars and JACK and shows how that is related to the BDI logic. However, the concepts that have been found to be useful for development within these systems do not necessarily match the concepts most developed in the theoretical work. Neither are they necessarily exactly the concepts which have arisen within particular implemented systems such as JACK. An additional complication is confusion and small differences between similar concepts, such as Desires and Goals, which receive differing emphasis in different work at different times.

Desires are understood to be things the agent wants to achieve. They play an important role in the philosophical foundation, but the logical theory deals primarily with Goals. Goals in the logic are assumed to be a consistent set of Desires. Philosophically there is no requirement for desires to be internally consistent. At the implementation level the motivational concept is reduced to events – goals are implicit and the creation of a new goal is treated as an event which can trigger plans. In the theoretical model plans are simply beliefs or intentions. However in the implementations plans are a central concept. Events are ignored in the theoretical framework but play a key role in implementations, although they are not well distinguished from goals. Some key differences between the philosophy, theory, and implementation viewpoints of BDI are shown in table 1.

2.2 Linear Logic

Linear logic [12] is sometimes described as *resource-sensitive*, in that the notion of resource is a natural one in this logic. The traditional techniques of logic treat two copies of a formula as being equivalent to one copy (as mathematical truth is not dependent on the number of times a property is stated), and hence formulae can be arbitrarily copied. However, this does not fit well with some application areas, in which there is a finite amount of resources, such as money, computer memory, floor space or execution time. Resource-sensitive logics such as linear logic do not allow arbitrary copying; in linear logic, by default, each formula has to be used exactly once. This property means that linear logic is a natural way to study state changes, and so provides a more direct way to model resource-bounded applications than the traditional techniques. In particular, linear logic has been applied to concurrency problems [12, 13], database updates [17] and planning problems [25, 26].

Space prevents us from giving a general introduction to linear logic here; such introductions can be found in [13, 12, 38, 2], for example. The sequent calculus for linear logic is given in Appendix A.

2.3 Logic Programming in Linear Logic

The execution models on which these languages are based have generally been based on *backward-chaining*, i.e. given a number of statements (or *formulae*) which make up the program, the user then requests the system to determine whether or not a given formula (the *goal*) follows from the information in the program. The way that this is achieved is generally by working backwards, i.e. establishing premises which, if true, would establish the truth of the goal. This process is repeated until either an unconditional statement of truth is found (an *axiom*), in which case the original goal succeeds, or no such premises can be found, in which case the original goal fails. Hence backward-chaining consists of starting with a given conclusion and working our way back (hopefully) to the axioms.

Whilst this paradigm appears to be intuitive and natural for various applications, such as querying a database, or solving a particular set of constraints, it does not allow programs to react to an environment, as they must wait for a specific goal to be given. In applications such as a stock market monitor, it is generally desirable to have the program "watch" the environment, which may include large amounts of data, until a given set of circumstances is observed, such as a sharp fall in the price of a blue-chip stock. Then it would be expected to take the appropriate action, such as buying such stock, and selling it again once the price has recovered. Thus the key element is for the program to evaluate the current environment until certain trigger conditions are met.

Such *reactive* behaviour is more akin to *forward-chaining*, which is a method of reasoning which begins with the axioms, and applies any known rules to generate new results. In the context of linear logic, which is well-known to be a useful way to model and reason about state changes, a forward-chaining approach seems particularly suitable for reactive systems, as this provides a simple and natural way to express conditions which are dependent on the dynamics of the environment.

The techniques for backward-chaining (both classically and for linear logic) are well-known [23, 28, 34, 17]; the integration of forward-chaining techniques into such a system was investigated in [16]. In particular, this allows a combination of *don't know* nondeterminism (common in logic programming) via backward-chaining with *don't care* nondeterminism via forward-chaining.

Such an integrated system is thus able to both follow a planned sequence of instructions (backward-chaining) and react to the environment and make appropriate changes (forward-chaining). In particular, this may be thought of as an increased emphasis on *process-oriented* computation (such as the safe running of a power plant or operating system) rather than *result-oriented* computation (such as calculating a pay cheque or checking whether a given credit card number is valid). As argued in [39, 4], amongst others, the process view of computation is one that is becoming increasing important, and in which safety considerations are vital.

Kowalski and Sadri [24] developed extensions to the traditional logic programming paradigm to incorporate agent features. A key difference in our approach is the use of linear logic, which provides a better representation of dynamic information (such as actions and environmental changes) than classical logic.

The key technical point is to determine the appropriate inference rules for the forward-chaining part of the system. At first, this may seem rather trivial, in that we simply take the well-known rule of *modus ponens*, and use it to determine that B follows from A and $A \supset B$. However, there are some subtleties to this, particularly for resource-sensitive logics.

In the classical case, the formulae $A \wedge A \supset B$ and $A \wedge B$ are equivalent, which means that a forward-chaining system based on this rule has several strong properties. One of these is *monotonicity*, in that the set of conclusions reached can only increase. This property is exploited not only in the well-known T_P semantics for logic programs [8], but also by deductive database systems such as Aditi, for which the monotonicity property is the underlying reason behind the differential optimisation.

The corresponding analysis in linear logic is not as straightforward, though, as the above equivalence does not hold. In particular, given p and $p \multimap q$, it is possible to derive q, but p is "consumed" in this process. Hence the use of modus ponens in linear logic is more like a committed choice, in that once the inference rule is applied, p is no longer available, but q is, and so our analysis needs to proceed in a more subtle way than in the classical case.

Our general procedure is to integrate elements of forward-chaining into the standard sequent calculus for linear logic (which is given in an Appendix). The sequent calculus is well-known as an inference system which permits an appropriate analysis of backward-chaining, and whilst there are systems similarly suitable for forward-chaining (such as natural deduction and Hilbert-type systems), it is not clear how backward-chaining can be introduced into such systems.

It should be noted that backward-chaining techniques are generally applied to a program and a goal: given a program $\mathcal{P}$ and a goal $\mathcal{G}$, we proceed to search for a proof of the sequent $\mathcal{P} \vdash \mathcal{G}$ via some appropriate search strategy. Such proofs are generally *cut-free*, in that the cut rule is not used in the search, as it may introduce formulae with no known relationship to the original sequent, and thus result in a hopelessly infeasible search.

By contrast, forward-chaining techniques are applied to a program, and produce another program. Hence, the natural approach is to define a relation $\rightsquigarrow$ between programs, so that $\mathcal{P} \rightsquigarrow \mathcal{P}'$ denotes that $\mathcal{P}'$ can be derived from $\mathcal{P}$ (via forward-chaining techniques).

We then need to determine not only the rules for $\rightsquigarrow$, but also how these inference rules interact with the standard rules of the linear sequent calculus (and hence with backward-chaining methods). Our approach is to model the interaction between the two types of inference by a particular type of occurrence of the cut rule, known as *direct* or *analytic* cuts. In particular, given a forward-chaining inference $\mathcal{P} \rightsquigarrow \mathcal{P}'$ and a backward-chaining one $\mathcal{P}'' \vdash \mathcal{G}$, then these two inferences can synchronise when $\mathcal{P}' = \mathcal{P}''$. Thus we have that from $\mathcal{P} \rightsquigarrow \mathcal{P}'$ and $\mathcal{P}' \vdash \mathcal{G}$ we can deduce $\mathcal{P} \vdash \mathcal{G}$ (or, for that matter, that $\mathcal{P}, \mathcal{G} \multimap D \rightsquigarrow D$, as the two premises are sufficient to establish that $\mathcal{P} \vdash \mathcal{G}$), which is just an instance of the cut rule. The key point to note is that not only are $\mathcal{P}$ and $\mathcal{G}$ known at the outset, but also that we expect the inference rules for a conclusion such as $\mathcal{P} \rightsquigarrow \mathcal{P}'$ to be such that given $\mathcal{P}$, we can readily derive $\mathcal{P}'$ from an appropriate number of applications of the rule.

In this sense the rules for $\rightsquigarrow$ are reminiscent of *conditional rewriting rules*, or of Plotkin's *Structured Operational Semantics* [30], in that given a unary rule for $\rightsquigarrow$

$$\frac{\mathcal{P} \rightsquigarrow \mathcal{P}''}{\mathcal{P} \rightsquigarrow \mathcal{P}'} R$$

it will generally be the case that $\mathcal{P}$ is known but $\mathcal{P}'$ is not, and so we will use the premise (and any appropriate sub-proofs) to evaluate $\mathcal{P}$ to $\mathcal{P}''$, and then using the rule R we will then determine that $\mathcal{P} \rightsquigarrow \mathcal{P}'$.

Such a process is also reminiscent of [14], in which it was shown that given an (intuitionistic) uniform proof of $\mathcal{P} \vdash \mathcal{G}$, it is possible, via a simple algorithm, to find $\mathcal{P}'$ such that $\mathcal{P} \vdash (\otimes \mathcal{P}')$ and $\mathcal{P}' \vdash \mathcal{G}$. In our case case, $\mathcal{P}'$ is calculated from $\mathcal{P}$ by a more intricate set of inference rules, and may arise in the course of a computation involving both forward-chaining and backward-chaining techniques.

Hence we proceed by inserting cuts into the inference rules of the linear sequent calculus, and study the properties of the resulting rules. We have some preliminary results along these lines, which derive an appropriate set of inference rules for $\rightsquigarrow$ and examines their integration into the linear sequent calculus. These results were published in [16] and include permutation properties (which are fundamental to the issue of proof normalizations and hence proof-search strategies) and cut-elimination results. The rules for mixed-mode inference in linear logic can be found in appendix B.

3 Designing Agent Systems

In order to examine the complexities of the BDI model, it is instructive to ask what concepts comprise the *minimal necessities* for an agent system. In other words, what, intuitively, is the most fundamental difference between an agent-oriented programming paradigms and other programming paradigms? Naturally there are many possible answers to this question, but one reasonable answer would be "Goals (at least)". For example, consider a drink-waiter agent, who is instructed to produce the finest wine possible. This instruction (i.e. goal) may produce different results when given at three different times; the wine that was produced the first time may be out of stock by the second request, or a better wine may have come into stock between the requests, or by the third request, a still better wine may be in stock, but cannot be served in time as it requires a certain amount of chilling or airing before being served.

Whilst, as above, one can certainly add a seemingly arbitrarily long list of features as necessary to the function of an agent system, in order to simplify the conceptual model of agents, it seems reasonable to concentrate on this central notion before developing others. Hence we will focus on goals, rather than start off from a position of the threefold interaction of beliefs, desires and intentions.

In general, an agent may have several goals; indeed, one may think of such a system having a number of desires (which are unconstrained) and from these determining a number of goals, which have to be consistent, possible, not yet achieved, etc. In addition, as discussed by Padgham and Lambrix [29], it seems reasonable to consider only goals for which a feasible means of achievement is available. For example, there is little point in having a goal such as making it rain tomorrow (which may well be consistent, possible, not yet achieved etc.) but which the agent has no feasible means to achieve. As we shall see, this property will have an important effect in the design of such systems.

One of the fundamental technical issues in the design of BDI systems is then to determine how best to balance the competing needs of being proactive (i.e. pursuing one's goals *despite* the environment) and being reactive (i.e. adapting one's behaviour *in response to* the environment). At one extreme are classic planning systems such as STRIPS [9], in which plans are generated in great detail in advance, and only executed once all steps are known (which often comes at great computational expense). A classic criticism of such systems is that by the time that their plans are completed, they may have long since been rendered obsolete by changes in the environment. At the other extreme are purely reactive systems, in which there is no time to do anything but use pre-determined actions corresponding to the given inputs.

The architectural realisation of this tension is the need to both execute a given course of action, as well as to periodically evaluate whether this is indeed the most appropriate action to take, and, if necessary, suspend (or abort) the current plan in favour of another.

Hence it seems necessary for an agent system to have at least the specific functionalities below:

1. A means of decomposing a given goal G into subgoals
2. A means of determining a set of possible plans to achieve the subgoals
3. A means of monitoring environmental changes and accordingly evaluating the most appropriate plan to execute

For example, consider the problem of providing a drink to a thirsty person (as discussed in [36] and [24]). The main goal is to quench the drinker's thirst[1]. There are two ways to do this: one is to drink lemonade, the other to drink water. To drink lemonade, one needs to be holding a bottle of lemonade, a glass (presumably clean and empty) and then perform a pour action followed by a drink action. To be holding a bottle of lemonade, one has to open the fridge and take a bottle. To drink water, one needs to have a glass, fill it with water from the tap and then perform a drink action.

As discussed in [24] it is possible to represent the state information via clausal rules similar to those of Prolog, such as those below (taken from [24] with some minor variations):

holds(quench-thirst)	iff	*holds(drink-lemonade)* or *holds(drink-water)*
holds(drink-lemonade)	iff	*holds(have-glass)* and *holds(have-lemonade)* and *do(pour)* and *do(drink)*
holds(have-lemonade)	iff	*do(open-fridge)* and *do(get-lemonade)*
holds(drink-water)	iff	*holds(have-glass)* and *do(open-tap)* and *do(drink)*

The key point to note is that as the above rules define the conditions under which various pieces of state information hold *(holds(drink-lemonade), holds(have-lemonade)*, etc.), the bodies of the rules contain a mixture of states

[1] Here, for simplicity, we have ignored time constraints, such as achieving this goal within 10 units of time. As in [24] one could simulate such constraints by the judicious use of an extra argument and some equalites and inequalities.

assumed to have been achieved (e.g. *holds(have-lemonade)*) together with actions needed to achieve some new state (e.g. *do(open-fridge)*). Hence one can think of the above rules as defining sets of actions[2] which, when perfomed, will achieve the desired state. In particular, the ability to use "intermediate" states, such as *holds(have-lemonade)*, means that alternatives can be easily integrated. For example, if two new methods of getting a bottle of lemonade become available (say purchasing one from a nearby vending machine and ordering one over the Internet), then these can be simply added as extra cases to the rule for *holds(have-lemonade)* without changing any other parts of the code. This flexibility is one of the key features of agent systems, as it shows clearly the way in which alternative methods of achieving the same goal can be interleaved as required in response to environmental changes.

An alternative presentation of the above example is

to *achieve(quench-thirst)*	try	*achieve(drink-lemonade)* or *achieve(drink-water)*
to *achieve(drink-lemonade)*	try	*(achieve(have-glass)* and *achieve(have-lemonade)* then *do(pour)* then *do(drink)*
to *achieve(have-lemonade)*	try	*do(open-fridge)* then *do(get-lemonade)*
to *achieve(drink-water)*	try	*achieve(have-glass)* then *do(open-tap)* then *do(drink)*

where we distinguish between "and" which does not imply any sequencing, and "A then B" which implies that A must done before B. This presentation is more "honest" in that it emphasises that the link between a plan body and the head is partial: there can be no guarantee that executing a plan will actually achieve the goal which is the head of the clause. All that can be said is that the body is a reasonable way to try and and achieve the goal.

In order to achieve the top-level goal, it is generally necessary to consider a mixture of sub-goals, plans, and actions. In fact, as there is an understood nexus between goals and plans (such as in the JACK system, in which goals are represented by events, and every event must be handled by some plan, and every plan must have some event that it handles), it seems reasonable to consider them not as separate concepts which need to be somehow linked up, but as different aspects of the same underlying concept. As discussed above, a goal can reasonably be expected to require that there is some plan which will achieve it (perhaps as yet unknown, but able to be determined effectively and efficiently); conversely, plans come with pre-conditions and intended outcomes, which indicate information about the state of the world both before and after execution. In logic programming terms, both goals and plans have *declarative* and *procedural* content — the goal specifies *what* needs to be achieved, and the plan specifies *how* it is to be achieved. *Hence as goals need to be associated with some procedure to achieve them, and plans require a goal which they achieve, it seems sensible to think of the combination of goal and plan as a unified concept.* The unified goal/plan concept is the *clause*: a clause of the form $G \leftarrow P$ has a goal (the head) and a plan (the body). As discussed earlier, the link between them is of the form "this is a reasonable decomposition to try". The plan can contain other goals which are in turn decomposed. Hence the unification is achieved by the ability to interpret clauses both declaratively and procedurally, although in this case the procedural interpretation may include actions and state changes as well as proof-search.

As discussed in [28], logic programming can be thought of as a particular implementation of *goal-directed proof search*, in that computation is specified by a given goal G, and the way the computation proceeds is dependent primarily on G rather than the program itself. In order to achieve the above integration, it is clearly necessary to extend the notion of a goal to include actions (and hence plans) in some way. As mentioned above, this has happened to some extent via abduction methods (as well as in the work of Gabbay [11]) in order to specify what needs to be achieved in order to make the top-level goal succeed; below we will discuss how this can be done in linear logic by means of actions as well.

Now in order to achieve the three functionalities mentioned above, backchaining can be used to decompose a goal into subgoals (see point 1 above), i.e. standard logic programming techniques to decompose a goal into subgoals. For example, for the quench-thirst example above, it is simple enough to see how the goal *holds(quench-thirst)* can be reduced to *holds(have-glass) and do(open-fridge) and do(get-lemonade) and do(pour) and do(drink)* or alternatively to *holds(have-glass) and do(open-tap) and do(drink)*.

Clearly the standard logic programming techniques of resolution and backtracking can be used to perform this reduction. In general a goal G is reduced by this process to a set of sufficient sub-goals and actions $G_1, G_2, \ldots G_n$. If

[2] In fact, in [36] and [24] these are *sequences* of actions, with the sequencing obtained in the former case by the use of procedural programming techniques, and in the latter via an extra argument for time for each predicate.

any one of these sub-goals is defined by a rule in the above manner, then further such reduction can be performed. If not, then we need to be able to find a set of actions (i.e. a plan) which will achieve this goal.

From the point of view of planning, the relationship between the subgoals can vary — the G_i can be viewed disjunctively (i.e. the success of one sub-goal is sufficient for the success of the overall goal), conjunctively (i.e. the success of all sub-goals is required for the success of the overall goal) or as a sequence. Naturally the first two possibilities fit naturally into a logic programming context; the third would require a logical notion of sequence. This is generally a difficult problem for logic programming languages to solve cleanly. However, some recent work by Polakow[32, 31] on ordered linear logic (i.e. non-commutative linear logic) may be a useful solution to this problem.

Another point to note is that as the environment can change during the backchaining process, it may be necessary to verify that the achievement of the sub-goals, once attained, does actually achieve the overall goal. In doing so it would presumably be useful to keep track of which chains of reasoning will remain true despite changes in the environment (such as the above decomposition of quenching thirst) as distinct from those which depend either on the current state of the environment (which may change) or on the success of some action (which may fail).

Hence an agent arrives at a list of sub-goals it wishes to pursue, and hence generates a list of plans which would achieve them. It then selects a given plan to execute. In order to incorporate reactive behaviour as well, at some point during execution the environment is checked to determine what has changed. At this point it is possible that another one of the applicable plans is selected, and work on the original is suspended. If a plan fails, then it is discarded. If all plans fail, then the parent plan fails. Hence the proactive behaviour extends only to the level of plans; one doesn't investigate the cause of failure, but instead looks for an alternative.

In order to provide this functionality within the system, what is required is a means of re-evaluating the previous planning process. One way to do this is to simply re-try the original goal, and see what results. Either the same conclusion will be reached (i.e. that no changes (or at least none of significance) have occurred), or a new conclusion will be reached, presumably due to environmental changes.

Hence reactivity is not so much about firing off triggers and the like in response to a given signal, but more about re-evaluating whether the current plan is still the best option.

4 Agents via Mixed-mode Computation in Linear Logic

We have discussed how backward and forward chaining can be integrated within the one inference system. This section discusses how these two aspects of the unified inference system can naturally model agents. In particular backward chaining is used to model the *proactive* aspect of the agent (which involves finding ways to achieve goals), and forward chaining is used to model the *reactive* aspect of the agent (which involves integrating events/percepts). We also show below how actions can be performed via forward chaining.

An agent can be represented by the sequent

$$\mathcal{E}, \mathcal{A}, \mathcal{B}, !\mathcal{P} \vdash \mathcal{G}$$

where $\mathcal{B}$ is the beliefs of the agent (which are linear since they change), $\mathcal{P}$ is the program clauses (i.e. goal-plan decompositions), and $\mathcal{G}$ is the agent's goals[3]. We also allow events ($\mathcal{E}$) and actions ($\mathcal{A}$) to appear.

The following proof fragment illustrates the interaction of backwards chaining over goals (proactive) with actions. Events are also handled using forward chaining ($\rightsquigarrow$). Adding an action ($\mathrm{do}(A)$) to the context triggers forward chaining using a directed cut (labelled $Cut \vdash$ below):

$$\dfrac{\dfrac{\dfrac{\begin{matrix}\vdots & & \vdots \\ \mathcal{P}, \mathrm{do}(A) \rightsquigarrow \mathcal{P}' & & \mathcal{P}' \vdash G', \ldots, G_n\end{matrix}}{\mathcal{P}, \mathrm{do}(A) \vdash G', \ldots, G_n}\ Cut \vdash}{\mathcal{P} \vdash \mathrm{do}(A) \multimap G', \ldots, G_n}\ \multimap\text{-R}}{\begin{matrix}\vdots \\ \hline \mathcal{P} \vdash G_1, \ldots, G_n\end{matrix}}$$

[3]Actually since $\mathcal{G}$ includes executing plans it is closer to the intention structure

where $\mathcal{P}$ includes both beliefs (linear) and program clauses (non-linear) which include action descriptions such as

$$!(\text{ do(get\_lemonade)} \otimes \text{fridge(open)} \multimap \text{fridge(open)} \otimes \text{have\_lemonade})$$

$$!(\text{ do(open\_fridge)} \otimes \text{fridge(closed)} \multimap \text{fridge(open)})$$

One issue concerns the choice of rules: the inference rules do not constrain which rule is to be applied at any given point. However, in order for agents to respond to events in a timely fashion, and in order for actions to be executed when they are scheduled we would like to constrain the selection of rules. In particular, whenever there are events or actions in the left side (antecedent) of the sequent (i.e. $\Delta, \text{do(A)} \vdash \Gamma$ or $\Delta, \text{event(E)} \vdash \Gamma$) then forward chaining is performed. Permutabilities can be used to precisely characterise where rule selection policies affect completeness; note that completeness will need to be compromised in order to rule out backtracking over performed actions.

Another issue is that of actions failing. For example, in the above do(get-lemonade) action, a pre-condition is that the fridge door is open. If it is closed when this action is executed, then the agent could either attempt to open the door, or abandon the current action (and hence current plan) in favour of some other option. To represent the second scenario would require an extra rule for the do(get-lemonade) action along the lines of the one below.

$$!(\text{ do(get\_lemonade)} \otimes \text{fridge(closed)} \multimap \text{fails(get\_lemonade)})$$

In general when an action's pre-conditions are not satisfied, we can either fail, attempt to make the pre-conditions true, or delay (i.e. in order to try the action again at a later time). Current BDI systems make the action (and hence the current plan) fail, and is not re-tried. How to deal with such issues remains an interesting question.

Most of our discussion (and presented formalism) has focussed on a single agent. There is a natural extension to multiple agents which simply uses separate sequents to mark the context boundary between agents. In particular, an agent has access to its own beliefs only. We introduce the ability to have multiple agents by simply maintaining parallel sequents

$$\mathcal{B}_1, !\mathcal{P}_1 \vdash_1 \mathcal{G}_1 \qquad \ldots \qquad \mathcal{B}_n, !\mathcal{P}_n \vdash_n \mathcal{G}_n$$

and allow for communication by sending events to other agents. Sending an event is denoted by $E \overset{X}{\rightarrowtail} G'$ which sends the event E to agent X and then continues with G':

$$\frac{\mathcal{B}_1, !\mathcal{P}_1 \vdash_1 G', \Gamma \qquad E, \mathcal{B}_2, !\mathcal{P}_2 \vdash_2 \Delta}{\mathcal{B}_1, !\mathcal{P}_1 \vdash_1 E \overset{2}{\rightarrowtail} G', \Gamma \qquad \mathcal{B}_2, !\mathcal{P}_2 \vdash_2 \Delta}\ Send$$

The effects of actions can be captured by making the environment into a distinguished agent. This agent (the Environment Agent, EA) has no goals, it simply performs a cycle of receiving actions from agents, performing them, and sending percepts (based on the new state of the world) to agents.

5 Conclusions and Further Work

This paper presents a proposal for a framework for agents based on mixed mode computation in linear logic. As we have seen, linear logic can be used to provide a nature framework for state changes and actions, and the combination of backward- and forward-chaining enables us to separate reactive computation (percepts and events) from proactive computation (goals and plans). Reactive computation is captured by forward-chaining (which, due to the basis in linear logic, naturally incorporates state changes), and goal decomposition (or planning) takes place via backward-chaining. The interaction between the two systems then becomes a matter of judiciously chosen directed cuts[4].

We have also discussed how the logic programming framework allows the use of program clauses such as $G \leftarrow P$ to unify the notions of goal and plan into a single concept. Simplifications such as this one are an important pragmatic consideration for the designers of agent systems.

In this way this paper describes a broad agenda for future work aimed at providing a clean, elegant, natural, and powerful foundation for situated agents which are reactive, proactive, and autonomous. Whilst this is somewhat ambitious, our previous experience in implementation of linear logic programming languages [16, 15], together with the technical evidence, strongly suggest that this agenda is feasible.

[4] In the proof-theoretic sense, not in the sense of a pruning operator. Cuts without !'s as it were :-)

Naturally there remains a considerable amount of further work to be done. One such item is the further development of the mixed mode system. Whilst the inference rules appear satisfactory, there remains a considerable amount of non-determinism, and hence any computational model based on these rules will require an appropriate means for dealing with this. For example, given a formula A and two rules $A \multimap B$ and $A \multimap C$, we need some method for choosing which rule to apply. In addition, as the application of one forward-chaining rule may trigger other rules, we need to determine whether to execute such rules in a depth-first or breadth-first (or iterative deepening) manner. In short, development of an appropriate operational semantics is required. The extension of the mixed mode system to other logics (such as affine logic, light linear logic, **BI**[33], etc.) is also a topic of interest.

Another item is the further pursuit of this particular notion of agent-oriented programming. For example, it is not clear how the notion of intention fits into our framework (although it seems reasonable that the search strategy used to solve the goals/execute the plans may be the manifestation of this concept). Naturally, beliefs should be considered as well, although these appear to be less problematic. In addition, actions, even when the pre-conditions are satisfied, may still fail, and hence we need to be able to model this in our framework. One possibility is to add a rule such as $B \multimap C$ to the environment when outcome B is expected; when it arrives, we can easily perform the computation from $B, B \multimap C$ to C. If it does not arrive (or at least not within some reasonable period), the presence of $B \multimap C$ can be used to indicate the absence of the expected outcome.

Development of the properties of the mixed mode inference system and the agent framework will presumably be symbiotic in nature, in that developments in one will suggest possible developments in the other. In addition, implementation of the system and some sample applications will greatly assist this process.

This broad agenda may be characterised as an attempt to delineate a programming paradigm of *goal-directed computation*; in other words, the key aspect is to determine how to satisfy the given goal, whether it is a Prolog-style goal (i.e. test for truth, possibly determining some extra information when successful) or a planning-style goal (i.e. change the world to achieve this state), some other style (such as *maintaining* a given goal in the manner of integrity constraints, or endlessly processing requests as they are made to a server). In all cases, the ability to balance proactive and reactive behaviour is critical to the design of the system.

Acknowledgements. We thank Lin Padgham, Omar Rana, David Pym and the Agents group at RMIT for discussions related to this work.

References

[1] *JACK Intelligent Agents User Guide*, Agent Oriented Software (AOS), Carlton, 2000.

[2] V. Alexiev, Applications of Linear Logic to Computation: An Overview, *Bulletin of the IGPL* 2:1:77-107, 1994.

[3] Abdullah-Al Amin, *Agent-Oriented Programming in Linear Logic*, Honours Thesis, Department of Computer Science, RMIT, November, 1999.

[4] J.-M. Andreoli, Focussing and Proof Construction, *Annals of Pure and Applied Logic* 107:1:131-153, 2001.

[5] J.-M. Andreoli and R. Pareschi, Linear Objects: Logical Processes with Built-in Inheritance, *Proceedings of the International Conference on Logic Programming* 496-510, Jerusalem, June, 1990.

[6] Michael E. Bratman, *Intentions, Plans, and Practical Reason*, Harvard University Press, Cambridge, MA, 1987.

[7] *The dMARS V1.6.11 System Overview*, Technical Report, Australian Artificial Intelligence Institute (AAII), 1996.

[8] M.H. van Emden and R.A. Kowalski, The Semantics of Predicate Logic as a Programming Language, *Journal of the Association for Computing Machinery* 23:4:733-742, October, 1976.

[9] R. E. Fikes and N. J. Nilsson. STRIPS: A new approach to theorem proving in problem solving. *Artificial Intelligence*, 2:189–208, 1971.

[10] Stan Franklin and Art Graesser, *Is it an Agent or just a Program?: A Taxonomy for Intelligent Agents*, Proceedings of the Third International Workshop on Agent Theories, Architectures and Languages, Springer, 1996.

[11] Dov Gabbay, *Dynamics of Practical Reasoning: A Position Paper*, in K. Segerberg, M. Zakharyaschev, M. de Rijke, and H. Wansing, editors, *Advances in Modal Logic*, 2:197-242, CSLI Publications, 2000.

[12] J-Y. Girard, Linear Logic, *Theoretical Computer Science* 50, 1-102, 1987.

[13] J-Y. Girard, Y. L and L. Regnier, *Advances in Linear Logic*, London Mathematical Society Lecture Note Series 222, Cambridge University Press, 1995.

[14] J. Harland, A Proof-Theoretic Analysis of Goal-Directed Provability, *Journal of Logic and Computation* 4:1:69-88, January, 1994.

[15] J. Harland, D. Pym and M. Winikoff, *Programming in Lygon: An Overview*, Proceedings of the Fifth International Conference on Algebraic Methodology and Software Technology 391-405, Munich, July, 1996.

[16] J. Harland, D. Pym and M. Winikoff, *Forward and Backward Chaining in Linear Logic*, Proceedings of the CADE-17 Workhop on Proof-Search in Type-Theoretic Systems, Pittsburgh, June, 200.

[17] J. Hodas and D. Miller, Logic Programming in a Fragment of Intuitionistic Linear Logic, *Information and Computation* 110:2:327-365, 1994.

[18] J. Hodas, K. Watkins, N. Tamura and K-S. Kang, Efficient Implementation of a Linear Logic Programming Language, *Proceedings of the Joint International Conference and Symposium on Logic Programming* 145-159, June, Manchester, 1998.

[19] Marcus Huber, *JAM: A BDI-theoretic Mobile Agent Architecture*, Proceedings of the Third International Conference on Autonomous Agents (Agents'99) 236-243, Seattle, May, 1999.

[20] Nick Jennings, An agent-based approach for building complex software systems, *Communications of the ACM* 44:4:35-41, 2001.

[21] Nick Jennings and Michael Woolridge, Applications of Intelligent Agents, in *Agent Technology: Foundations, Applications, and Markets* 3-28, Nick Jennings and Michael Woolridge (eds.) Springer, 1998.

[22] A. Kakas, R. Kowalsk and F. Toni, Abductive Logic Programming, *Journal of Logic and Computation* 2:719-770, 1992.

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

[24] R. Kowalski and F. Sadri, From Logic Programming towards Multi-agent Systems, *Annals of Mathematics and Artificial Intelligence*, 1999.

[25] M. Masseron and C. Tollu and J. Vauzeilles, Generating Plans in Linear Logic I: Actions as proofs, *Theoretical Computer Science 113:349-371*, 1993.

[26] M. Masseron, Generating Plans in Linear Logic II: A geometry of conjunctive actions, *Theoretical Computer Science 113:371-375*, 1993.

[27] D.A. Miller, A Multiple-Conclusioned Meta-Logic, Proceedings of the Symposium on Logic in Computer Science 272-281, Paris, June, 1994.

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

[29] L. Padgham and P. Lambrix, Agent Capabilities: Extending BDI Theory, in *Proceedings of Seventeenth National Conference on Artificial Intelligence - AAAI 2000* 68-73, August, 2000.

[30] G. Plotkin. Structural Operational Semantics (lecture notes). Technical Report DAIMI FN-19, Aarhus University, 1981 (reprinted 1991).

[31] Jeff Polakow, *Ordered Linear Logic and Applications*, Ph.D. thesis, Department of Computer Science, Carnegie Mellon University, August 2001. Available as Technical Report CMU-CS-01-152 and from `http://www.cs.cmu.edu/~jpolakow`.

[32] Jeff Polakow, Linear Logic Programming with an Ordered Context, *2nd International Conference on Principles and Practice of Declarative Programming (PPDP 2000)*, Montreal, September 2000.

[33] D. Pym, On Bunched Predicate Logic, *Proceedings of the IEEE Symposium on Logic in Computer Science*, Trento, July, 1999.

[34] D. Pym and J. Harland, A Uniform Proof-Theoretic Investigation of Linear Logic Programming, *Journal of Logic and Computation* 4:2, April, 1994.

[35] Anand Rao and Michael Georgeff, Modelling Rational Agents within a BDI-Architecture, *Proceedings of the Second International Conference on Principles of Knowledge Representation and Reasoning* 473-484, R. Fikes and E. Sandewall (eds.), Cambridge (USA), 1991.

[36] Anand Rao and Michael Georgeff, An Abstract architecture for Rational Agents, *Proceedings of the third International Conference on Principles of Knowledge Representation and Reasoning* 439-449, C. Rich, W. Startout and B. Nebel (eds.), Boston, 1992.

[37] Anand Rao and Michael Georgeff, Decision Procedures for BDI Logics, *Proceedings of the third International Conference on Principles of Knowledge Representation and Reasoning* 439-449, C. Rich, W. Startout and B. Nebel (eds.), Boston, 1992.

[38] A. Scedrov, A Brief Guide to Linear Logic, in *Current Trends in Theoretical Computer Science*, G. Rozenberg and A. Salolmaa (eds.), World Scientific, 1993.

[39] Stein, L. A., Challenging the Computational Metaphor: Implications for How We Think, *Cybernetics and Systems*, 30(6), September 1999. Available from `http://www.ai.mit.edu/people/las/papers/`.

[40] Michael Wooldridge, *Reasoning about Rational Agents*, MIT Press, 2000.

[41] Michael Wooldridge and Nick Jennings, *Agent Theories, Architectures and Languages: A Survey*, in Michael Wooldridge and Nick Jennings (eds.), *Intelligent Agents* 1-22, Springer, Berlin, 1995.

A Sequent Calculus for Linear Logic

$$\frac{}{\phi \vdash \phi}\ \text{axiom} \qquad \frac{\Gamma \vdash \phi, \Delta \qquad \Gamma', \phi \vdash \Delta'}{\Gamma, \Gamma' \vdash \Delta, \Delta'}\ \text{cut}$$

$$\frac{\Gamma, \phi, \psi, \Gamma' \vdash \Delta}{\Gamma, \psi, \phi, \Gamma' \vdash \Delta}\ \text{X-L} \qquad \frac{\Gamma \vdash \Delta, \phi, \psi, \Delta'}{\Gamma \vdash \Delta, \psi, \phi, \Delta'}\ \text{X-R}$$

$$\frac{\Gamma \vdash \Delta}{\Gamma, \mathbf{1} \vdash \Delta}\ \mathbf{1}\text{-L} \qquad \frac{}{\vdash \mathbf{1}}\ \mathbf{1}\text{-R}$$

$$\frac{}{\bot \vdash}\ \text{L}^{\bot} \qquad \frac{\Gamma \vdash \Delta}{\Gamma \vdash \bot, \Delta}\ \text{R}^{\bot}$$

$$\frac{}{\Gamma, \mathbf{0} \vdash \Delta}\ \mathbf{0}\text{-L} \qquad \frac{}{\Gamma \vdash \top, \Delta}\ \top\text{-R}$$

$$\frac{\Gamma \vdash \phi, \Delta}{\Gamma, \phi^{\bot} \vdash \Delta}\ \bot\text{-L} \qquad \frac{\Gamma, \phi \vdash \Delta}{\Gamma \vdash \phi^{\bot}, \Delta}\ \bot\text{-R}$$

$$\frac{\Gamma, \phi, \psi \vdash \Delta}{\Gamma, \phi \otimes \psi \vdash \Delta}\ \otimes\text{-L} \qquad \frac{\Gamma \vdash \phi, \Delta \qquad \Gamma' \vdash \psi, \Delta'}{\Gamma, \Gamma' \vdash \phi \otimes \psi, \Delta, \Delta'}\ \otimes\text{-R}$$

$$\frac{\Gamma, \phi \vdash \Delta}{\Gamma, \phi \,\&\, \psi \vdash \Delta} \qquad \frac{\Gamma, \psi \vdash \Delta}{\Gamma, \phi \,\&\, \psi \vdash \Delta}\ \&\text{-L} \qquad \frac{\Gamma \vdash \phi, \Delta \qquad \Gamma \vdash \psi, \Delta}{\Gamma \vdash \phi \,\&\, \psi, \Delta}\ \&\text{-R}$$

$$\frac{\Gamma, \phi \vdash \Delta \qquad \Gamma, \psi \vdash \Delta}{\Gamma, \phi \oplus \psi \vdash \Delta} \oplus\text{-L} \qquad \frac{\Gamma \vdash \phi, \Delta}{\Gamma \vdash \phi \oplus \psi, \Delta} \qquad \frac{\Gamma \vdash \psi, \Delta}{\Gamma \vdash \phi \oplus \psi, \Delta} \oplus\text{-R}$$

$$\frac{\Gamma, \phi \vdash \Delta \qquad \Gamma', \psi \vdash \Delta'}{\Gamma, \Gamma', \phi \parr \psi \vdash \Delta, \Delta'} \parr\text{-L} \qquad \frac{\Gamma \vdash \phi, \psi, \Delta}{\Gamma \vdash \phi \parr \psi, \Delta} \parr\text{-R}$$

$$\frac{\Gamma \vdash \phi, \Delta \qquad \Gamma', \psi \vdash \Delta'}{\Gamma, \Gamma', \phi \multimap \psi \vdash \Delta, \Delta'} \multimap\text{-L} \qquad \frac{\Gamma, \phi \vdash \psi, \Delta}{\Gamma \vdash \phi \multimap \psi, \Delta} \multimap\text{-R}$$

$$\frac{\Gamma, \phi \vdash \Delta}{\Gamma, !\phi \vdash \Delta} \text{!-L} \qquad \frac{!\Gamma \vdash \phi, ?\Delta}{!\Gamma \vdash !\phi, ?\Delta} \text{!-R}$$

$$\frac{!\Gamma, \phi \vdash ?\Delta}{!\Gamma, ?\phi \vdash ?\Delta} \text{?-L} \qquad \frac{\Gamma \vdash \phi, \Delta}{\Gamma \vdash ?\phi, \Delta} \text{?-R}$$

$$\frac{\Gamma \vdash \Delta}{\Gamma, !\phi \vdash \Delta} W\text{!-L} \qquad \frac{\Gamma \vdash \Delta}{\Gamma \vdash ?\phi, \Delta} W\text{?-R}$$

$$\frac{\Gamma, !\phi, !\phi \vdash \Delta}{\Gamma, !\phi \vdash \Delta} C\text{!-L} \qquad \frac{\Gamma \vdash ?\phi, ?\phi, \Delta}{\Gamma \vdash ?\phi, \Delta} C\text{?-R}$$

$$\frac{\Gamma, \phi[t/x] \vdash \Delta}{\Gamma, \forall x \,.\, \phi \vdash \Delta} \forall\text{-L} \qquad \frac{\Gamma \vdash \phi[y/x], \Delta}{\Gamma \vdash \forall x \,.\, \phi, \Delta} \forall\text{-R}$$

$$\frac{\Gamma, \phi[y/x] \vdash \Delta}{\Gamma, \exists x \,.\, \phi \vdash \Delta} \exists\text{-L} \qquad \frac{\Gamma \vdash \phi[t/x], \Delta}{\Gamma \vdash \exists x \,.\, \phi, \Delta} \exists\text{-R}$$

where y is not free in Γ, Δ.

B Mixed-mode Inference Rules from [16]

$$\frac{}{\mathcal{P} \leadsto \mathcal{P}} \; Axiom \leadsto \qquad \frac{}{\mathcal{P}, A \vdash A} \; Axiom \vdash \qquad \frac{\mathcal{P} \leadsto \mathcal{P}'}{\mathcal{P}, D \leadsto \mathcal{P}'} \; W \leadsto \qquad \frac{\mathcal{P}, D, D \leadsto \mathcal{P}'}{\mathcal{P}, D \leadsto \mathcal{P}'} \; C \leadsto$$

$$\frac{\mathcal{P} \leadsto \mathcal{P}'' \quad \mathcal{P}'' \leadsto \mathcal{P}'}{\mathcal{P} \leadsto \mathcal{P}'} \; Cut \leadsto \qquad \frac{\mathcal{P} \leadsto \mathcal{P}' \quad \mathcal{P}' \vdash G}{\mathcal{P} \vdash G} \; Cut \vdash \qquad \frac{\mathcal{P} \leadsto \mathcal{P}'}{\mathcal{P}, \mathcal{P}'' \leadsto \mathcal{P}', \mathcal{P}''} \; Weak$$

$$\frac{\mathcal{P} \leadsto \mathcal{P}' \quad \mathcal{P}' \vdash G}{\mathcal{P}, G \supset D \leadsto \mathcal{P}, D} \supset\leadsto \qquad \frac{\mathcal{P}, D_1, D_2 \leadsto \mathcal{P}'}{\mathcal{P}, D_1 \wedge D_2 \leadsto \mathcal{P}'} \wedge \leadsto \qquad \frac{\mathcal{P} \vdash G_1 \quad \mathcal{P} \vdash G_2}{\mathcal{P} \vdash G_1 \wedge G_2} \wedge \vdash$$

$$\frac{\mathcal{P}, D[t/x] \leadsto \mathcal{P}'}{\mathcal{P}, \forall x D \leadsto \mathcal{P}'} \forall \leadsto \qquad \frac{\mathcal{P} \vdash G[y/x]}{\mathcal{P} \vdash \forall x G} \forall \vdash \qquad \frac{\mathcal{P} \vdash G_i}{\mathcal{P} \vdash G_1 \vee G_2} \vee \vdash \qquad \frac{\mathcal{P} \vdash G[t/x]}{\mathcal{P} \vdash \exists x G} \exists \vdash$$

The rule $\forall \vdash$ has the usual restriction that y is not free in $\mathcal{P}$ or G.