# Agent Negotiation as Proof Search in Linear Logic

James Harland
jah@cs.rmit.edu.au

Michael Winikoff
winikoff@cs.rmit.edu.au

RMIT University
Melbourne, AUSTRALIA

## Categories and Subject Descriptors

I.2.11 [**Artificial Intelligence**]: Distributed Artificial Intelligence—*Intelligent agents, Multiagent systems*; F.4.1 [**Mathematical Logic and Formal Languages**]: Mathematical Logic

## 1. INTRODUCTION

Negotiation is a key aspect of multi-agent systems, and one which clearly requires appropriate reasoning processes. Fisher [2] has introduced the notion of negotiation as theorem proving, in which the agents can reach a satisfactory arrangement iff there is proof in a given system. Fisher's system is based on a distributed theorem proving environment employing resolution techniques in classical logic. In this paper we show how a basis in *linear logic* enables a richer (and arguably more natural) basis for reasoning about negotiations. In particular (i) the representation of conditionals is direct and natural in linear logic; (ii) linear logic allows for consumables to be modelled; and (iii) linear logic allows varying types of choices to be modelled in a clearer way than using classical logic. This last is of particular importance – there is an important distinction between an agent that is willing to provide *clothing* or *food* where another agent makes the choice, and an agent that is willing to provide either *clothing* or *food* but where that agent chooses between them. Hence the contribution of this paper is to provide a more natural basis for the reasoning required. In this sense our focus is on the *outcome* of the negotiation, rather than the (possibly argumentative) *process*. In particular, we assume that each party specifies what it desires, and the negotiation process presented finds solutions that satisfy all parties.

## 2. LINEAR LOGIC

Linear logic [3] 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 [3, 4], database updates [5] and planning problems [6].

Linear logic contains two forms of conjunction: one which is "cumulative", i.e. for which $p \otimes p \not\equiv p$, and one which is not, i.e. $p \,\&\, p \equiv p$. Roughly speaking, the former is what allows linear logic to deal with resource issues, whilst the latter allows for these issues to be overlooked (or, more precisely, for an "internal" choice to be made between the resources used).

Consider the following menu from a restaurant:

fruit or seafood (in season)
main course
all the chips you can eat
tea or coffee

Note that the first choice, between fruit and seafood, is a classical disjunction: we know that one or the other of these will be served, but we cannot predict which one. This may be thought of as an "external" choice, in that someone else makes the decision. On the other hand, the choice between tea and coffee is an "internal" choice — the customer is free to choose which one shall be served. Note that the internal choice is a conjunction; in order to satisfy this, the restauranteur has to be able to supply *both* tea and coffee, and not just one of them. The chips course clearly involves a potentially infinite amount of resources, in that there is no limit on the amount of chips that the customer may order. We represent this situation by prefixing such formulae with a !. Note also that the meal consists of four components which we connect with $\otimes$. Hence we have the following representation of the menu:

$$(\texttt{fruit} \oplus \texttt{seafood}) \otimes \texttt{main} \otimes {!}\,\texttt{chips} \otimes (\texttt{tea} \,\&\, \texttt{coffee})$$

where we write $\oplus$ for the classical disjunction. Note that the use of ! makes it possible to recover classical reasoning, as formulae beginning with ! behave classically, in that such formulae may used arbitrarily many times, including 0, rather than exactly once. Hence `chips` corresponds to *exactly* one serving of chips, while ! `chips` corresponds to an arbitrary number of servings (including 0). In this way we may think of a formula $!F$ in linear logic as representing an unbounded resource, i.e. one that may be used as many times as we like. Thus classical logic may be seen as a particular fragment of linear logic, in that there is a class of linear formulae which precisely matches classical formulae. There is far more to linear logic than can be discussed here; for a more complete introduction see the papers [1, 3, 4], among others.

## 3. NEGOTIATIONS IN LINEAR LOGIC

Turning now to negotiation, the technical crux for us is to show how the notion of requirements and supply can be appropriately modelled in linear logic, and then how we can easily supplement this model, particularly where choices are concerned. The basic idea is that each agent has requirements and things it will supply, which can be simply and naturally modelled with a linear implication[1] $require \multimap supply$ where $require$ and $supply$ are formulae which specify the requirements and supply respectively. Well-known properties of linear implication can then be used to determine whether all the requirements can be satisfied or not. For example, consider the following scenario (from [2]):

- Agent A requires *clothing* and *food*, and if supplied (by another agent) with *jewels*, it will provide *money*.
- Agent B, if given *money* will provide *food* and if given *clothing*, will provide *jewels*.
- Agent C will provide *clothing* to any other agent.
- Agent D, if given *credit*, will provide both *clothing* and *food*

One solution, as discussed in [2] is for agent C to supply clothing to A and B, B then supplies jewels to A, who supplies money to B, who supplies food to A. This can be represented by the following sequent[2] in linear logic:
$!(jewels \multimap money), !(money \multimap food), !(clothing \multimap jewels)$
$!clothing, !(credit \multimap (clothing \otimes food)) \vdash clothing \otimes food$
where the formulae on the left of $\vdash$ are those with non-empty supply, and those on the right are requirements only (both from agent A, as it happens). Note that, as is common in applications of linear logic, the rules (e.g. $!(jewels \multimap money)$) that specify the conversion of resources are classical rather than linear, i.e. it is the requirements and the resources needed to fulfil them which will be finitary, rather than our ability to apply the rules themselves. However, in some situations this may not be the case: suppose that in the example above agent B only has a certain number of *jewels*. Modelling this in Fisher's framework would be difficult at best, but this can be simply modelled in linear logic.

Note also that it is simple to change the rules to reflect different rates of conversion. For example, if agent B requires two lots of clothing before it will supply jewels, then we can modify the above rule to $!((clothing \otimes clothing) \multimap jewels)$.

The connectives of linear logic allow two notions of choice. If an agent can supply **both** $food$ and $money$, then we can write this as $!(requirements \multimap (food \otimes money))$.

If an agent can only supply one of these, then there are two possible ways to make the choice — either the supplying agent chooses which, or the agent that requires the resource will make the choice. These two choices can be modelled by the connectives $\oplus$ and $\&$ respectively, as $!(requirements \multimap (food \oplus money))$ and $!(requirements \multimap (food \& money))$

In the first case, an agent who requires $money$ will not be able to use this supply, as it cannot be guaranteed that what will actually be supplied is money (as it could be $food$). In the second case, the supplying agent provides both possibilities, but the choice is left to the agent supplied. Hence, an agent who requires $money$ will be able to get some in the second case, but not the first.

Similar remarks apply to requirements; a requirement of both $food$ and $money$ is written as $food \otimes money$. A requirement of one of the two (and the power to choose which) can be modelled as $food \& money$ and the requirement of one of the two *without* the power to choose can be modelled as $food \oplus money$. Note that an agent that requires $food \oplus money$ will be satisfied by $food \& money$ but not vice-versa. Thus one strength of linear logic is its ability to clearly represent choices and where they are made. This ability is clearly important in a range of protocols. For example, the simple division method of "I cut, you choose" relies on two separate choices being made, and the identity of those making them.

The representation discussed above does not indicate which agents have which resources. This can be easily added by tagging resources with their owner and extending rules appropriately. For example, the possession of food by agent B is represented by the fact $food(B)$ and the rule that agent B will provide food if given money is represented by the rule $!\forall x.(money(x) \otimes food(B)) \multimap (money(B) \otimes food(x))$. This rule says that if agent B has food and another agent $x$ has money then if $x$ gives the money to B ($money(B)$) then B will give $x$ food ($food(x)$). This more detailed encoding also requires us to specify explicitly the initial resources. For example, if B is *always* able to provide food in exchange for money then it must have an inexhaustible food supply, i.e. $!food(B)$.

## 4. DEALING WITH NEGOTIATIONS

One question to ask in work such as this is what features are required of the negotiation process. Clearly some notion of search is one such thing, in that whilst there are many possibilities, the trick is to find a way to match up requirements and supply so that all are satisfied. In the case that there is no way to make an agreement, the basis in logic suggests that a failure should be accompanied by a countermodel, i.e. an example which illustrates a reason why negotiations broke down.

An example which we are working on is the trading of sports players (say footballers) between clubs. This may involve selling a player to another club, trading one or more players for another player, trading draft choices and various possibilities for three-way trades and so forth. Clearly there are rules and requirements to be elucidated here (such as tradeable and untradeable players, and team requirements such as a desire for a goal-scorer in a strong defensive team). We are working particularly on a set of example trading scenarios used in the Australian Football League (AFL).

**Acknowledgements:** The support of Agent Oriented Software Pty. Ltd. and of the Australian Research Council (under grant CO0106934) is gratefully acknowledged.

## 5. REFERENCES

[1] V. Alexiev. Applications of linear logic to computation: An overview. *Bulletin of the IGPL*, 2(1):77–107, 1994.

[2] M. Fisher. Characterising simple negotiation as distributed agent-based theorem proving — a preliminary report. In *Proceedings of the Fourth International Conference on Multi-Agent Systems*, Boston, July 2000. IEEE Press.

[3] J.-Y. Girard. Linear logic. *Theoretical Computer Science*, 50:1–102, 1987.

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

[5] J. Harland, D. Pym, and M. Winikoff. Programming in lygon: An overview. In *Proceedings of the International Conference on Algebraic Methodology and Software Technology*, pages 391–405. LNCS 1101, July 1996.

[6] M. Masseron, C. Tollu, and J. Vauzeilles. Generating plans in linear logic I: Actions as proofs. *Theoretical Computer Science*, 113:349–371, 1993.

[1]The formula $A \multimap B$ is a linear implication which specifies that $A$ is *consumed* in order to create $B$.
[2]Note that the sequent $A, B, C \vdash D$ is equivalent to the formula $(A \otimes B \otimes C) \multimap D$.