The Use of Typed Lambda Calculus for Requirements Capture in the Domain of Ecological Modelling

Alan Bundy and Mike Uschold

DAI Research Paper No.

April 26, 2004

Submitted to the Journal of Logic and Computation

Department of Artificial Intelligence
University of Edinburgh
80 South Bridge
Edinburgh EH1 1HN
Scotland

The Use of Typed Lambda Calculus for Requirements Capture in the Domain of Ecological Modelling *

Alan Bundy and Mike Uschold

Abstract

We will describe the use of order-sorted, typed lambda calculus to represent ecological simulation models. It will be used to represent both the qualitative specifications of ecological situations and the differential equations which describe these situations quantitatively. The numerical solution of these equations constitutes a simulation model of the situation. This logic-based formalism extends previous formalisms for ecological modelling.

Particular emphasis will be given to the automatic help given to users in formulating their goals, describing the ecological situation to be modelled and forming the differential equations which model it. The typed lambda calculus representation provides a simple grammar for constraining and guiding this process of requirements capture. Users develop their goals by a process of refinement of logical expressions. Each refinement step can also be understood in ecological terms. The refinement steps are necessarily not truth preserving because we are engaged in requirements capture rather than specification transformation.

1 Introduction

In the Eco project we are exploring ways to assist ecologists build ecological models, [Robertson *et al.* 87, Robertson *et al.* 88]. Our premise is that most ecologists could be assisted by the use of computer simulation techniques, but that this possibility is effectively denied to many of them because they lack the necessary mathematical and computational skills. We are investigating whether it is possible to offer such ecologists computational assistance to specify their model in ecological terms, and hence to have their specifications transformed into computer programs. We believe that by restricting the range of programs to be synthesised to a rich but well defined domain, that powerful computer assistance can be brought to the user to assist the requirements capture task.

In [Robertson *et al.* 89] we explored a variety of ways of doing this, eventually concluding that a wide spectrum logical language capable of representing both ecological relationships and mathematical equations, offered the best approach. This approach is being implemented in a series of computer programs which collectively form the *EcoLogic* system.

In this paper we develop this approach in more detail. In particular, we describe the use of a version of typed lambda calculus, [Church 40], to represent the various ecological entities, variables and equations on which ecological simulation

*The research reported in this paper was supported by SERC grant GR/E/00730 and an SERC Senior Fellowship to the first author. We would like to thank the other members of the Eco project, Dave Robertson, Mandy Haggith and Bob Muetzelfeldt, for many discussions about the work reported in this paper.

models are based. We have added to typed lambda calculus as necessary to get the representational power we require. In particular, we have added a rich type structure which draws on recent work in typed λ calculus, *e.g.* [Cardelli 88]. The ideas described here have been totally or partially implemented within the EcoLogic system.

2 Ecological Models

An ecological simulation model can be idealised as a program for the numerical solution of a set of differential equations describing mathematical relationships between ecological variables. For instance, a typical differential equation[1] might be:

$$\frac{dx}{dt} = r.x.(1 - \frac{x}{k}) \tag{1}$$

where x is the number of objects in some population at time t, k is the asymptote of these numbers and r is the rate of growth.

In order for EcoLogic to be able to assist the user to form such equations it is necessary to represent and reason about the relationships between ecological variables like x, k, t and r, in (1) above.

- Ecologists will want to use such relationships to specify the ecological situation they wish to have modelled.
- EcoLogic will require a grammar of legitimate ecological variables to guide users in the formation of their specifications. This grammar should permit the formation of an infinite variety of such variables from a small collection of primitives.
- Each step in the incremental specification of the ecological entities, variables and equations of the model must make ecological sense to the user.
- EcoLogic will need to access relevant mathematical formulae and must instantiate them appropriately using the user's specification. It may also need to infer further relationships from those specified in order to complete the model.

We choose to encode some of the relationships between the ecological variables by representing them with logical terms. Note that, under this encoding, ecological variables will *not* necessarily be logical variables, but more often be compound logical terms, the sub-terms of which encode relationships between the ecological variables. For more discussion of this point see appendix B. As we will see, the natural representation of ecological variables will use higher order logic with a rich type structure.

3 Representing Ecological Variables in Typed Lambda Calculus

The need for higher order functions follows immediately from our need to represent differential equations; differentiation is naturally represented as a second order function from one unary (first-order) function to another of the same type. We will use the second order function *rate* for this, *i.e.* rate has the type[2]:

$$\forall Sort : \varrho(entities), \forall Nums : \varrho(numbers). \quad rate : \quad (Sort \mapsto Nums) \mapsto (Sort \mapsto Nums)$$

[1] We are grateful to Bob Muetzelfeldt for suggesting the ecological models on which this and the other examples in this paper are based.

[2] We will adopt the Prolog convention that identifiers beginning with capital letters are logical variables and those starting with lower case letters are logical constants.

where $entities$ is the sort of all first order entities, $numbers$ is the sort of all numbers and $\varrho(Sort)$ is the kind of all sub-sorts of $Sort$[3], *e.g.* $times : \varrho(entities)$, and $reals : \varrho(numbers)$

Note that $rate$ is constrained to produce a unary function of the same type as it is given, *i.e.* $Sort$ and $Nums$ must be instantiated to the same sorts in both the argument and result of $rate$. In practice, $Sort$ will usually be $times$, but can be any first-order sort, provided its instances are totally ordered. $Nums$ will usually be $reals$ or $naturals$. Thus, this type declaration for $rate$ is polymorphic across different sorts. The types of all functions and constants defined in this paper are repeated, in alphabetic order, in appendix C.

The total order condition is required in order for differentiation to make sense. We will usually use a version of differentiation for a totally ordered, discrete sort, *e.g.* time instants, rather than the usual version of differentiation for real numbers, which are continuous. Some form of differentiation can be made to work for almost any totally ordered sort.

Thus our representation of equation (1) will require $rate$ to be applied to a unary function. x, the argument of the differentiation in (1), is the number of objects in some population at time t. Let *pop* be a logical constant representing this population. We will introduce a binary function, *number*, which takes a population and a time instant and returns the number of objects in the that population. Then x is represented by $number(pop, t)$. To make a unary function out of this that can serve as the argument to $rate$ we will make t into a variable T and apply lambda abstraction to it, yielding $\lambda T : times.number(pop, T)$.

To represent the type of *pop* we will need a sort $objects$ of ecological objects and a unary function *set* which takes a sort and returns the sort of sets of entities of that sort[4].

Formally the types of the functions and constants defined above are:

$$\begin{array}{ll} t: & times \\ set: & \varrho(entities) \mapsto \varrho(entities) \\ pop: & set(objects) \\ number: & set(objects) \times times \mapsto naturals \end{array}$$

Similarly, we will replace k by $asymptote(pop)$, where:

$$asymptote: \quad set(objects) \mapsto naturals$$

i.e. *asymptote* takes a set of objects, *e.g.* *pop*, and returns a natural number. This number is an upper limit on the number of objects that can be in the set. Note that having *pop* as an argument of both x and k provides a connection between them. We will exploit such connections in the process of eliciting model specifications from users. In this case, the rate of growth, r, is a constant, so does not require to be made dependent on anything else. We will leave it unchanged.

Our revised representation of equation (1) is now:

$$rate(\lambda T : times.number(pop, T))(t) \quad = \quad r.number(pop, t).(1 - \frac{number(pop, t)}{asymptote(pop)}) \tag{2}$$

Note that we are not claiming that equation (2) is more readable than equation (1). Indeed, it is not — and we would not present it to the ecologist user in this form. Rather, the claim is that it is much richer in information about the types of entity

[3] ϱ is the power operator, [Cardelli 88].

[4] *Cf.* $list(Type)$ in many type theory systems.

that are being related and their inter-relationships. We exploit this richness both during requirements capture and during program synthesis.

As a further example, consider the annual maximum, daily average biomass of a sheep. This is a typical example of an ecological variable with which ecologists want to form equations. Maximum and average are best viewed as second order functions from unary functions over some sets of values to some sort of numbers, *i.e.*

$$\begin{array}{lll} \forall Sort : \varrho(entities), \forall Nums : \varrho(numbers). & maximum : & (Sort \mapsto Nums) \times set(Sort) \mapsto Nums \\ \forall Sort : \varrho(entities), \forall Nums : \varrho(numbers). & average : & (Sort \mapsto Nums) \times set(Sort) \mapsto Nums \end{array}$$

$set(Sort)$ is the set of values of the unary function over which we want to calculate the maximum or average. Note that the use of the type variable, $Sort$, and the type function, set, enables us to ensure that the members of this range are correctly typed to be arguments of the unary function.

The annual maximum, daily average biomass of a sheep can then be represented as:

$$maximum(\lambda Dy : set(hours).average(\lambda Hr : hours.biomass(shp, Hr), Dy), yr) \tag{3}$$

where $hours$ are a sort of $times$, days are represented as a set of hours, years as a set of days, yr is a typical year, shp is a typical sheep and $biomass$ is a binary function from lifeforms and times to reals, *i.e.*

$$\begin{array}{ll} yr : & set(set(hours)) \\ shp : & sheep \\ biomass : & lifeforms \times times \mapsto reals \end{array}$$

Note the nesting of $average$ within $maximum$ and the way in which the λ variable Hr in $average$ ranges over values of the λ variable Dy in $maximum$ and Dy, in turn, ranges over yr.

In simulation modelling, we often invent idealised, fictitious entities, like the typical sheep, shp, and the typical year, yr, used above, and let these stand-in for a collection of real world entities. Thus shp will be endowed by the modeller with attributes that are common to, or the average of, those sheep in which s/he is interested. shp and yr are constants because they stand for particular entities in the model.

4 Representing the Sort Hierarchy

The examples in the previous section have implicitly identified a need for a hierarchy of sorts. For instance, we have defined $biomass$ over $lifeforms$ and $times$ but then used it over $sheep$ and $hours$. To make this work we will need to define $sheep$ as a sub-sort of $lifeforms$ and $hours$ as a sub-sort of $times$. This could be avoided if we had either defined $biomass$ more narrowly over $sheep$ and $hours$, or defined shp and Hr more broadly as of type $lifeforms$ and $times$, respectively.

Neither of these alternatives is attractive. As we will see below, it will be convenient to have a limited collection of general-purpose functions like $biomass$, $number$, $average$, *etc.*, since this will limit the choices to be presented to the user during requirements capture. Furthermore, it will be convenient to specify ecological models, in part, by assigning narrowly defined types to ecological entities.

The sort hierarchy can be defined using the sub-sort relation, '$\sqsubset$', between sorts and the instance relation ':' between sorts and their instances. All the sorts we have discussed are sub-sorts of $things$, the sort of all ecological individuals, *e.g.*

$$numbers \sqsubset things, \quad times \sqsubset things, \quad objects \sqsubset things$$

Hence they are all instances of $\varrho(things)$, *i.e.*

$$numbers : \varrho(things), \quad times : \varrho(things), \quad objects : \varrho(things)$$

$hours$ are a sub-sort of $times$. $reals$ and $naturals$ are sub-sorts of $numbers$.

$$hours \sqsubset times, \quad reals \sqsubset numbers, \quad naturals \sqsubset numbers$$

$lifeforms$ and $minerals$ are sub-sorts of $objects$, $animals$ and $plants$ are sub-sorts of $lifeforms$, $mammals$ and $reptiles$ are sub-sorts of $animals$ and $sheep$ and $wolves$ are sub-sorts of $mammals$.

$$\begin{array}{ll} lifeforms \sqsubset objects, & minerals \sqsubset objects, \\ animals \sqsubset lifeforms, & plants \sqsubset lifeforms, \\ mammals \sqsubset animals, & reptiles \sqsubset animals, \\ sheep \sqsubset mammals & wolves \sqsubset mammals \end{array}$$

We need the rule:

$$X : S_1 \wedge S_1 \sqsubset S_2 \quad \rightarrow \quad X : S_2 \tag{4}$$

to deduce $shp : lifeforms$ and $shp : objects$, *etc.* from $shp : sheep$.

The $\sqsubset$ relation defines tree structures on sorts in the standard way (see figure 1 (a)). However, our sort hierarchy is made more complex by the *set* function on sorts. Clearly, we need the rule:

$$S_1 \sqsubset S_2 \quad \rightarrow \quad set(S_1) \sqsubset set(S_2)$$

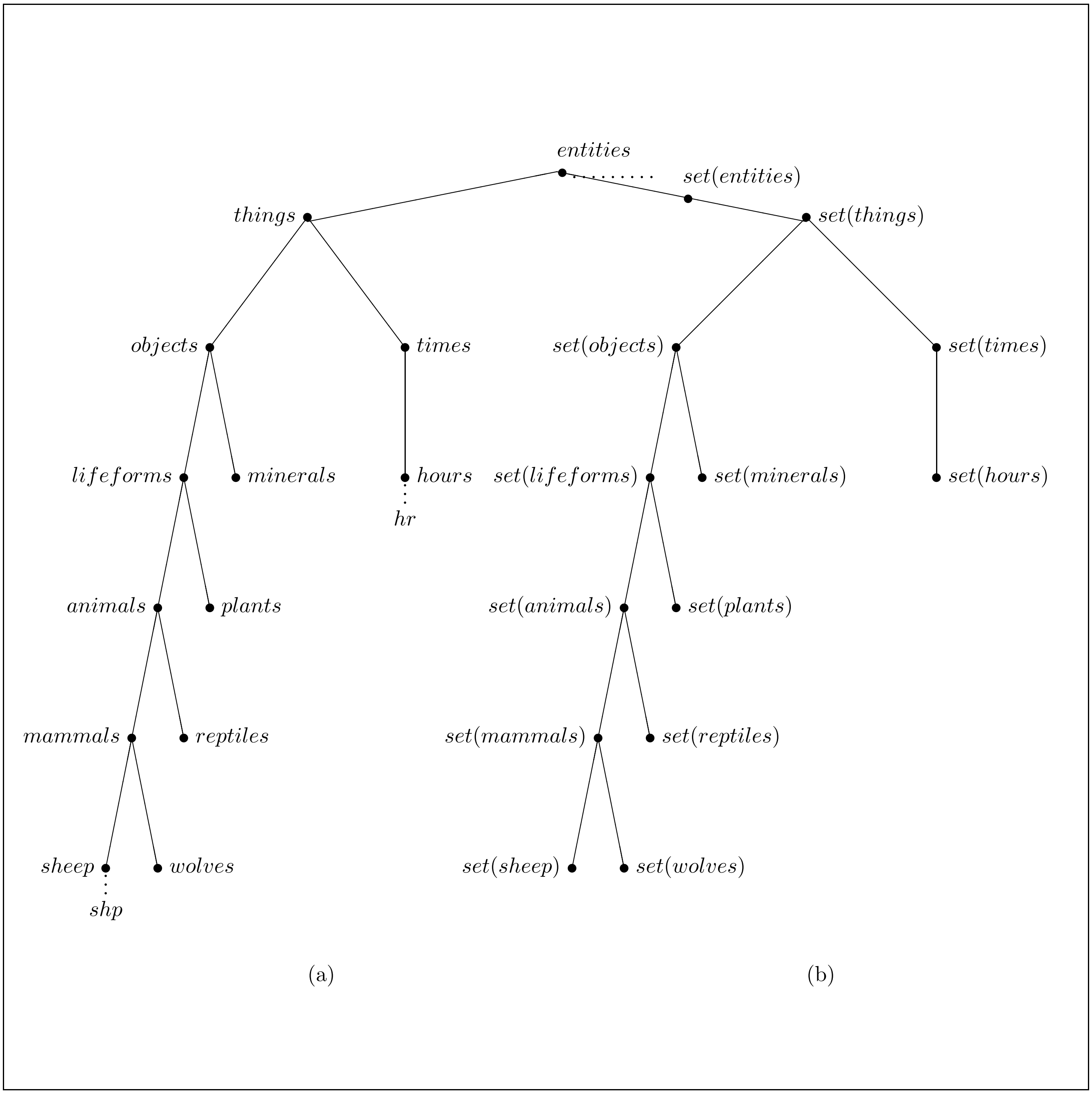
This induces another tree structure on sets of sorts, and this process recurses (see figure 1 (b)). Usually, we will assign sorts from the tips of these trees to the ecological entities, *e.g.* *shp* will be assigned the sort *sheep* and then will inherit higher sorts using rule (4).

Some of this sort hierarchy is general-purpose information which will be known to EcoLogic, but some of it is special to the particular ecological model. Sub-sort relations at the top of a sort tree will usually be general-purpose, *e.g.* $objects \sqsubset things$, whereas the assignment of sorts to ecological entities will usually be special-purpose, *e.g.* $shp : sheep$. In between there is a grey area. For instance, we will often want sub-sort relations at the bottom of a sort tree to be special-purpose. For instance, we cannot anticipate all the sorts of animals and plants that a user might want to model, and must allow users to add any new such sorts that they require, *e.g.* *sheep* as a new sort of *mammals*. So in EcoLogic we will provide an editor for assisting the user in building any problem specific parts of the sort hierarchy.

5 Representing Substructure

Up to now ecological entities have been represented as logical constants, *e.g.* *shp*, *t*, and used as the primitive base for defining ecological variables as compound logical terms. However, ecological entities often have a fine substructure. For instance, we might decide to model not just one kind of sheep, but to distinguish different sheep on the basis of their age, sex, location, breed, *etc.* and model each of these different kinds of sheep. The user will choose to distinguish on some dimension when some process to be modelled differs significantly along this dimension, *e.g.* if eating is to be modelled and if eating habits are significantly different for different breeds then the user will want to distinguish different kinds of sheep by breed. For instance, sheep might be distinguished along two orthogonal dimensions:

- by location into sheep in pasture, meadow and heath; and



Dot nodes represent sorts and the unbroken lines between them represent '$\sqsubset$' relations. The other nodes represent instances and the dotted lines linking them to dot nodes represent ':' relations. (a) is a sort sub-tree for some of the sorts we have introduced. (b) is the sort sub-tree induced by (a) and the set function. (b) and set then generate another tree, and so on. entities is a super sort of all these sorts.

Figure 1: Part of the Sort Hierarchy as a Tree.

- by breed into leicester sheep and shetland sheep.

This is called the *substructure* of the sheep.

We will represent this substructure by modifying the sort hierarchy. The obvious way to do it is to invent a new sub-sort for each kind of entity, *e.g.* *pasture_sheep*, *pasture_leicester_sheep*, *etc.* and create a new sheep instance for each new sub-sort, *e.g.* p_shp, p_l_shp, *etc.* Even in the simple example above this leads to 11 new sub-sorts, 17 new $\sqsubset$ relations and 11 new instances. Adding these to the sort hierarchy manually is both burdensome and error-prone. We can automate the additions, but the resulting sort hierarchy is messy and hard to read. It also turns the sort hierarchy from a collection of trees into a lattice, thus complicating the procedures for manipulating it. Furthermore, each of the differential equations involving *shp* must be duplicated 6 times: once for each instance of a leaf sub-sort.

Therefore, we have adopted the following, more compact, representation. We will not create any new sub-sorts of sheep. Instead we will replace the constant, *shp*, by a compound term with two free variables, $shp(F, B)$, where F ranges over the 3 locations and B over the 2 breeds. By instantiating these variables to particular values we can refer to particular sheep instances. By quantifying over these variables we can refer simultaneously to all of them. Note how the substructure of the sheep is directly reflected in the structure of the terms that represent them.

Let *locations* be the sort of locations of *objects* and *breeds* be the sort of breeds of $lifeforms$.

$$locations \sqsubset things, \quad breeds \sqsubset things$$

We declare the required values of our two dimensions to be instances of these two sorts, *i.e.*:

$$\begin{array}{lll} pasture : locations, & meadow : locations, & heath : locations, \\ leicester : breeds, & shetland : breeds & \end{array}$$

We now re-declare *shp* to be a binary function from *locations* and *breeds* to *sheep*, *i.e.*:

$$shp : \quad locations \times breeds \mapsto sheep$$

In practice, we will want to restrict *shp* to apply just to the locations and breed instances declared above. We will do this using bounded quantification over sets of these instances. To simplify our notation we will introduce names for these sets, namely:

$$\begin{array}{llrcl} fields : & set(locations) & fields & = & \{pasture, meadow, heath\} \\ shp\_brds : & set(breeds) & shp\_brds & = & \{leicester, shetland\} \end{array}$$

In this representation a typical leicester sheep in the pasture is represented by the term:

$$shp(pasture, leicester) : sheep$$

So ecological entities are no longer always represented by logical constants. If they have substructure then they are represented by compound logical terms.

Typical sheep in the pasture of either breed, can be referred to by the term:

$$shp(pasture, B) : sheep$$

where B is a free variable ranging over *shp_brds*.

$shp(pasture, leicester)$ will be endowed with just those attributes that are common to, or the average of, leicester sheep in the pasture. Similarly for the other 5 sheep instances.

Note that in this representation each field contains each breed of sheep. More complex arrangements are possible, *e.g.* each field containing only some of the breeds. Such arrangements require more complex term structures, *e.g.* making each field instance into a function from breeds in that field to the parts of the field containing that breed. So that if *pasture* only has *shetland* in its value space then there is an instance $shp(pasture(shetland))$ but no instance $shp(pasture(leicester))$.

We can make general statements about all elements of the substructure by the use of bounded quantification, for instance,

$$\forall F \in fields, B \in shp\_brds \ldots \quad biomass(shp(F,B), Hr) \quad \ldots$$

This use of universally quantified variables as the arguments to *shp* enables us to assert the formula simultaneously about all 6 instances of *sheep*. If we wanted to assert it only for the 2 instances of sheep in the pasture we could use the logical formula:

$$\forall B \in shp\_brds \ldots \quad biomass(shp(pasture, B), Hr) \quad \ldots$$

This representation of substructure is based on a purely syntactic technique for representing sortal information in logic programming (see, *e.g.*, [Bundy *et al.* 82]). For instance, to record that b_1 is a block it is represented as $block(b_1)$. Unfortunately, it is not clear what the function *block* means. It appears to map each block to itself, using two names for each block, *e.g.* b_1 and $block(b_1)$, but not $block(block(b_1))$, which seems equally meaningful semantically. Therefore, it is not clear that the such terms can be given a semantics. In our version we are able to attach types to the functions and constants used in the representation, and hence to give a semantics to the terms they form in the conventional way.

The ability to represent substructure and deal with the different elements of the substructure in a uniform way, *i.e.* by quantification across the value spaces, as above, extends previously known formalisms for ecological simulation models. For instance, neither systems dynamics, [Forrester 61], sets of differential equations nor previous logic-based formalisms, [Niven 82], provide a representation for substructure.

By a similar technique we can define the substructure of time needed for our sheep biomass example. Recall that we want one year divided into days, say 365, each of which is divided into hours, say 24. We can regard this as providing substructure for some time entity $hr : hours$. This substructure will have two dimensions: hour in the day and day in the week. Both the values spaces of these two dimensions are sets of *naturals*; the values of hours in the day being drawn from the numbers 1 to 24 and those of days in the year from 1 to 365. Thus by analogy with *shp* above, we want to declare *hours* as a new sub-sort of *times* and *hr* as a binary function from *naturals* to *hours*, *i.e.*

$$hours \sqsubset times, \quad hr : naturals \times naturals \mapsto hours$$

Just as *shp* will usually be restricted to quantification over a particular set of *locations* and a particular set of *breeds*, *hr* will usually be restricted to two particular sets of *naturals*, namely the numbers 1–24 and 1–365. We can assume that each of these numbers has already been declared as of sort *naturals*, so it only remains to name the sets. We will adopt the convention that int_n stands for the set of *naturals* from 1 to n. It will also be convenient to use $dy(J)$ to stand for the J_{th} day and yr for the year composed of these days.

$$\begin{array}{lllcl}
int_{24} : & set(naturals) & int_{24} & = & \{1, \ldots, 24\} \\
int_{365} : & set(naturals) & int_{365} & = & \{1, \ldots, 365\} \\
dy : & naturals \mapsto set(hours) & dy(J) & = & \{hr(I,J) | I \in int_{24}\} \\
yr : & set(set(hours)) & yr & = & \{dy(J) | J \in int_{365}\}
\end{array}$$

In order for differentiation with respect to time to make sense it is necessary to impose a total order on the time instants $hr(I, J)$. With discrete time instants, like $hr(I, J)$, we will require that there is a unary function, $next$, from instants to instants,

$$next: \quad times \mapsto times$$

which acts as a successor function. When $naturals$ are used as the substructure dimension, as with hr above, an appropriate total order is implied, namely:

$$next(hr(I, J)) = \begin{cases} hr(1, J+1) & \text{if } I = 24 \\ hr(I+1, J) & \text{otherwise} \end{cases}$$

6 Eliciting Ecological Variables

A common use of ecological models is to observe the effect of an independent variable on a dependent variable, *e.g.* by plotting a graph of the first against the second. Computationally, this means solving some differential equations numerically to give values of the dependent variable for different values of the independent variable. This may involve the calculation of values of some intermediate variables and processes and the provision of the values of various external variables, parameters and constants[5].

One way of specifying an ecological model is to start by formulating the independent and dependent variables, including their substructure, using this information to identify the equations to be used in their calculation, and then using this to suggest the other ecological variables that will be required. This is the approach we focus on in this paper.

We will show how our representation of ecological variables in typed lambda calculus serves as a basis for guiding the user in the formulation of the independent and dependent variables. Logical terms representing these ecological variables are built up in stages. Each partial representation is refined by applying logical functions to it. The provision of a few general purpose functions and a rich type structure restricts the choice of function at each stage of refinement, making the refinement process manageable. These choices can be presented to the user by asking an ecological questions. Those details of the underlying representation that users must know in order to answer these questions, must be presented in a way that is intelligible to them.

To illustrate the process consider how, using EcoLogic, the logical term:

$$maximum(\lambda Dy : set(hours).average(\lambda Hr : hours.biomass(shp(F, B), Hr), Dy), yr)$$

might be formulated as the dependent variable. Such a dependent variable might be used by a farmer trying to compare the slaughter weights of different groups of sheep. He will send the sheep to slaughter on the day they attain maximum biomass, but he cannot be accurate to the time of day, so assumes them to be at their average biomass for the day.

On entering EcoLogic, the user will be presented with a series of questions about the model. Each question is associated with one or more windows containing menus, buttons, *etc.* One such window is designed to elicit what the dependent variable is. On selecting this window the user will first be asked to identify the ecological entities that the model is about. Note that this is an ecological rather than mathematical question. This is done by browsing the sort hierarchy and using the sort editor to attach one or more new instances to one of its sorts. It may also be necessary to

[5] For the definition of these various kinds of ecological objects see appendix B.

add further sub-sorts. In our case the user might add *sheep* as a new sub-sort under *mammals* and then attach *shp* as an instance of a *sheep*, *i.e.* formally:

$$sheep \sqsubset mammals, \quad shp : sheep$$

These declarations will be presented to the user graphically as in figure (1).

shp : *sheep* is the initial representation of the dependent variable. The user will then be invited to refine this representation in one of two ways:

- either by specifying what ecological variable(s) of the sheep is/are to be modelled;
- or by defining the substructure of the sheep.

The first question is associated with a menu of possible refinement functions to apply to *shp*. The second question is associated with a menu of possible dimensions for elaborating the substructure of *shp*.

It is important that the number of items on each menu is small, so that the user will not become overwhelmed by the choices. This will be achieved in EcoLogic in three ways.

- Overall, EcoLogic will build its variable descriptions from a relatively small set of general-purpose functions and dimensions. This is made possible by the representation of variables by compound terms; the variety will be encoded in the nesting of the terms rather than in special-purpose function names. For instance, "annual maximum, daily average biomass" will be represented by the compound term (3), composed of general-purpose functions *maximum*, *average*, *biomass* and *set*, rather than some special-purpose constant, *max_aver_biomass*, say.
- In each menu the choices will be further restricted to those compatible with the current term. This is made possible by the use of strong typing of the functions and constants, which enables us to reject incompatible combinations. Compatibility is defined as follows: two types are *compatible* if and only if they have a common sub-type. For instance, *biomass* will be offered in the menu for specifying the ecological variable of *shp* because it has a first argument of type $lifeforms$ which is a super-sort of *sheep* and thus is compatible with it. *number* will not be offered because it requires arguments of type *set*(*objects*) and *times*, neither of which is compatible with *sheep*.
- Furthermore, type information can be used to order the choices; more compatible ones being offered earlier in the menu. This is made possible by the sort hierarchy, which provides a measure of 'more compatible'. Type S_1 is more compatible with type S than type S_2 if the minimum distance from S_1 to S via their common sub-type in the sort hierarchy is shorter than the minimum distance from S_2 to S. For instance, *biomass* will be offered earlier in the *shp* menu than *volume* because *biomass* is a function of $lifeforms$ whereas *volume* is a function of *objects* and $lifeforms$ is nearer to *sheep* in the sort hierarchy than *objects*.

Suppose the user chooses to apply function *biomass* to *shp* forming:

$$biomass(shp, T_1) \quad : \quad reals \tag{5}$$

where T_1 : *times* is a new free variable. This choice specifies the biomass of the sheep as the quantity to be modelled. The substructure menu for *shp* is still available, and a substructure menu for T_1 is now offered. However, suppose the user decides to further specify the dependent variable. This is done by choosing further functions to apply to the term (5). This term has type *reals*, but it also has a free variable, $T_1 : times$, which will suggest to EcoLogic that it not only try 1st order functions that take *reals* as arguments, *e.g.* $+, .$, but also 2nd order functions that take unary functions from *times* to *reals*, *e.g.* $rate, average, maximum$. If *shp* were a free variable rather than a constant, then the user would also have been offered a choice of forming the $rate, average, maximum$, *etc.* over a set of sheep.

Suppose the user chooses to consider the average biomass of the sheep over time. This is done by selecting *average* from the menu, forming:

$$average(\lambda T_1 : times.biomass(shp, T_1), T_2) \quad : \quad reals \tag{6}$$

where $T_2 : set(times)$ is a new free variable. Note that the average biomass of the sheep will be calculated over the range T_2 in terms of each of the time instants T_1. This means T_2 must be a set of T_1s. T_1 must have a substructure so that we can represent the elements of this set. The default dimension for this substructure is *naturals*, although other dimensions are conceivable. This partially determined substructure must be remembered by EcoLogic. In addition, a new substructure menu must be created for T_2.

The menu options for applying functions to (6) will be the same as those for (5). Suppose the user makes a similar decision and chooses *maximum*, forming:

$$maximum(\lambda T_2.average(\lambda T_1 : times.biomass(shp, T_1), T_2), T_3) \quad : \quad reals \tag{7}$$

where $T_3 : set(set(times))$ is a new free variable. This choice further specifies that it is the maximum of the previous averages over the T_2s in T_3 that is to be the dependent variable. Again, note that T_2 must have such substructure as will allow the representation of the elements of the set T_3. Thus if T_3 has n dimensions of substructure, then T_2 must have $n + 1$ dimensions and T_1 must have $n + 2$. A substructure menu must be created for T_3.

7 Eliciting Substructure

Suppose the user has now specified sufficiently the ecological variable to be modelled, and turns his/her attention to specifying the substructure of the ecological entities. Substructure menus are available for $shp : sheep$, $T_1 : times$, $T_2 : set(times)$ and $T_3 : set(set(times))$. Associated with each menu is a question asking the user to specify the substructure of the object in question. The menus can be dealt with in any order.

Suppose the user chooses $shp : sheep$. Possible dimensions for the substructure will be retrieved by inheritance from the sort *sheep* and presented in a menu. Possible dimensions will be associated with each sort in the sort hierarchy and inherited by their sub-sorts. For instance, *ages*, *sexes* and *breeds* will be associated with $lifeforms$, and *locations* would be associated with *objects*. The total number of possible dimensions inherited by each sort will be a few tens, and hence manageably small. In addition, users can add special-purpose dimensions of their own.

When a user chooses one of these dimensions (s)he is prompted for its value space. Again default value spaces for each dimension will be stored in the sort hierarchy and retrieved by inheritance. Several alternative defaults will be stored and retrieved. Defaults will be particularly useful for dimensions like *sexes* and *ages* and for the dimensions of *times*. Users will be able to override these defaults and supply their own value spaces.

Suppose the dimensions specified and their value spaces are as in §5, namely:

- location: pasture, meadow and heath; and

- breed: leicester and shetland.

EcoLogic will automatically add *pasture*, *meadow* and *heath* to the sort hierarchy as new instances of *locations*, and *leicester* and *shetland* as new instances of *breeds*, and it will redeclare the type of *shp* as $locations \times breeds \mapsto sheep$. This implicitly creates 6 new instances of sort *sheep*, *e.g.* $shp(meadow, shetland)$. When forming equations containing $shp(F, B)$, EcoLogic will bind F and B with bounded quantifiers ranging over sets of these instances, *i.e.* $fields$ and shp_brds, as defined in §5. This completes the representation of the substructure of *shp*.

Suppose the user now specifies the substructure of T_1. The first step is to instantiate the free variable to some name and choose its sort. The user will use the EcoLogic sort hierarchy editor to do this. Since T_1 is intended to range over hours it is natural to name it as *hr* and give it sort *hours*, a sub-sort of *times*. From the use of T_1 in the dependent variable, (7), we can already infer that it must have at least two dimensions of substructure, since it must have two more than T_3. The default for each of these two dimensions is *naturals*. If this default is chosen, EcoLogic will prompt for the value space, which is assumed to be a set of consecutive *naturals* from 1 to n. Suppose the user chooses 24 as the value for n on the first dimension and 365 as its value on the second. These choices induce 24.365 instances of *hours*, namely $hr(I, J)$ for $1 \leq I \leq 24$ and $1 \leq J \leq 365$. If the user decided to specify further substructure on T_1 then this would automatically add any extra dimensions to T_2 and T_3 as well. Suppose that the user is content with T_1 and now moves on to T_2 and T_3.

The choices for T_1 automatically induce one dimension of substructure to T_2, since T_2 must have one dimension fewer than T_1, *i.e.* if T_2 is instantiated to *dy* then *dy* will be a unary function from *naturals* to $set(hours)$, and $dy(J)$ for $1 \leq J \leq 365$ are all declared to be instances of $set(hours)$. Any further dimensions given to T_2 would be inherited by T_1 and T_3. Suppose the user is content with T_2 and moves on to T_3.

Suppose T_3 is instantiated to *yr*. The above substructure choices for T_1 and T_2 do not induce any dimensions of substructure on T_3, since T_3 must have one fewer dimension than T_2. But any further dimensions given to T_3 at this stage would be inherited by T_1 and T_2. For instance, if we decided to run the model over ten years then *yr* would become a unary function over int_{10}, *dy* would become binary and *hr* ternary. Suppose that the user is content with T_3 as it is. The substructure of time is now complete. Similar results would have been obtained if the time menus had been dealt with in a different order.

With the new substructure the term representing the dependent variable has been refined to:

$$maximum(\lambda Dy : set(hours).average(\lambda Hr : hours.biomass(shp(F, B), Hr), Dy), yr) \tag{8}$$

The independent variables are F and B. Note that it was not possible to identify these before we had defined the substructure of *shp*. This is an example of the need to interleave the elicitation of ecological variables with that of substructure. We might also want to interleave both with the elicitation of equations. For instance, the building of an equation might introduce a new intermediate variable which depends on a possible dimension of an ecological entity, thus motivating the introduction of this dimension into the substructure of the entity. EcoLogic will allow complete flexibility in the interleaving of these various tasks.

8 Eliciting Equations

EcoLogic will use the Marples Algorithm, [Bundy *et al.* 79], to form the set of equations which constitute the ecological model. The Marples Algorithm starts with a number of prestored ecological formulae. It instantiates these, with the ecological variables specific to the user's problems, to form equations. This requires only one way matching: sub-expressions of formulae are matched to ecological variables. Its final set of equations express the dependent variable(s) in terms of the independent variable(s).

The Marples Algorithm uses a simple form of recursion. The dependent variables are made into a list of 'soughts' and the independent variables into a list of 'givens'. For each sought, the Marples Algorithm first tries to find an equation involving only it, other soughts, some givens, known constants, and functions for which calculation procedures are known. If this fails then it finds an equation which also involves some new 'bridge' variables[6]. These bridge variables are added to the list of soughts, and the process recurses on the new list. In EcoLogic the user will be offered a menu of the available equations for each sought, together with information to help him/her choose. Examples of such guidance information are whether the equation introduces new bridge variables and, if so, a description of these bridge variables.

In the case of our dependent variable, (8), the function *maximum* is calculable. The only unknown part is:

$$\lambda Dy : set(hours).average(\lambda Hr : hours.biomass(shp(F, B), Hr), Dy)$$

where Dy ranges over the elements of yr, *i.e.* Dy is $dy(J)$ for $J \in int_{365}$. Thus, by beta reduction,

$$average(\lambda Hr : hours.biomass(shp(F, B), Hr), dy(J))$$

becomes a sought, which the Marples Algorithm now proceeds to solve for. There is no equation expressing this sought in terms of the givens alone, so it is necessary to introduce bridge variables. There are three applicable formulae, according as *average* is interpreted as mean, median or mode. The formula for mean is:

$$average(Func, Range) \quad = \quad \frac{\Sigma(Func, Range)}{size(Range)}$$

which can be instantiated to form the equation:

$$\begin{aligned} &\forall F \in fields, B \in shp\_brds, J \in int_{365}. \\ &average(\lambda Hr : hours.biomass(shp(F, B), Hr), dy(J)) \\ &\quad = \quad \frac{\Sigma(\lambda Hr : hours.biomass(shp(F, B), Hr), dy(J))}{size(dy(J))} \end{aligned} \tag{9}$$

Since Σ and *size* are calculable, this equation only introduces the new bridge variable:

$$biomass(shp(F, B), hr(I, J))$$

biomass is an ecological state variable, and must be calculated using a differential equation. The precise form of this differential equation will depend on the processes that the user decides to take into account. EcoLogic will provide a menu of possible processes for the user to choose from. In the case of the biomass of sheep this will include grazing, respiration[7], defecation, death, birth, *etc.* These possible processes

[6] In Mecho, bridge variables were called intermediate variables, but that term has a different meaning in ecologically modelling (see appendix B), so we have used an alternative.

[7] Energy loss through metabolic processes.

are stored with the function *biomass*. Each process is accompanied by a note to help in the choice, *e.g.* the time scale it operates on, the strength of the effect, *etc.* For instance, since the user is currently interested in processes acting on sheep on a scale of hours, then (s)he can use this information to decide to ignore birth and death. Suppose (s)he decides to take respiration and grazing into account. The equation formed is:

$$\begin{aligned} &\forall F \in fields, B \in shp\_brds, I \in int_{24}, J \in int_{365}. \\ &rate(\lambda T.biomass(shp(F,B),T))(hr(I,J)) \\ &\quad = \quad \Sigma(\lambda Plt : plants.grazing(shp(F,B), Plt, hr(I,J)), PltRange) - resp(shp(F,B), hr(I,J)) \end{aligned} \tag{10}$$

where $PltRange : set(plants)$ is a new free variable.

Note the following:

- *grazing* depends on both the sheep and the various plants it grazes on. *PltRange* is a set of those plants and we must add up the contributions made by each member of the set. The structure of *PltRange* remains to be specified.
- *resp* depends only on the sheep, so no summation is necessary.
- *grazing* increases the biomass, and so is positive in sign. *resp* decreases the biomass, and so is negative in sign.
- The new bridge variables are $grazing(shp(F,B), Plt, hr(I,J))$ and $resp(shp(F,B), hr(I,J))$.
- The use of compound logical terms to represent ecological variables and the matching of these terms against general-purpose ecological formulae automatically creates terms representing the bridge variables without the need to elicit these from the user. This is a major advantage of our use of logical terms to represent ecological variables.
- To make *rate* well defined a total order must be imposed on the time instants, $hr(I,J)$. EcoLogic can impose the order induced by int_{24} and int_{365} by defining *next* as in §5.

Suppose the user specifies that the sheep only eat grass and only in their own field. This is done by instantiating $PltRange : set(plants)$ to $\{grs(F)\} : set(grass)$, where *grs* has type $locations \mapsto grass$. The bridge variables are now $grazing(shp(F,B), grs(F), hr(I,J))$ and $resp(shp(F,B), hr(I,J))$. Note how the agreement of the two Fs in *shp* and *grs* represent the specification that the sheep only graze on the grass in the field in which they are located. So the summation across different plants is not required in this case and the Σ will eventually be removed from this equation by simplification.

To finish the model it is necessary to form equations for the two outstanding bridge variables. These introduce an bridge unknown for the biomass of the grass, for which we need another differential equation. This in turn introduces the respiration and photosynthesis of the grass, for which further equations are required. With these equations the model is complete. The complete set of equations can be found in appendix A.

Although these equations are asserted for all $F \in fields$ and $B \in shp\_brds$, it is still possible for them to yield different results for the dependent variables for different values of F and B. This is because the parameters and intermediate variables, on which they ultimately depend, may return different results for different values of F and B. For instance, the equation for sheep respiration in appendix A depends on the intermediate variable $bmass\_resp(shp(F,B), temp(hr(I,J)))$. The user must supply values of *temp* for different values of I and J and values of *bmass_resp* for different values of F, B and $temp(I, J)$, *e.g.* by tabulating the values of *temp* and *bmass_resp* or by providing some procedure to calculate them. These differences in the results of *bmass_resp* will be inherited by the calculations of sheep respiration, sheep biomass, *etc.* Note that functions of ecological objects do not have to depend on all dimensions of their substructure, *e.g.* *bmass_resp* may depend on B but not on F. This will be reflected in the procedures for calculating *bmass_resp*.

9 The Running of the Model

The equations in appendix A constitute an ecological model. To run this model EcoLogic must solve them numerically. This involves iterating over successive time increments and using the equations to calculate successive values of the dependent ecological variables in terms of previous values and the independent variables.

Note that time is discrete and ordered by the *next* function. For numerical solution of differential equations on a discrete variable it is best to transform them first into difference equations. This can be done by rewriting all occurrences of *rate* using the formula:

$$rate(UFunc)(Arg) \quad = \quad UFunc(next(Arg)) - UFunc(Arg)$$

as a rewrite rule left to right. For instance, this would transform equation (10) into:

$$\begin{aligned}
&\forall F \in fields, B \in shp_brds, I \in int_{24}, J \in int_{365}. \\
&biomass(shp(F,B), next(hr(I,J))) - biomass(shp(F,B), hr(I,J)) \\
&\quad = \quad \Sigma(\lambda Grs : grass.grazing(shp(F,B), Grs, hr(I,J)), \{grs(F)\}) - resp(shp(F,B), hr(I,J))
\end{aligned}$$

which can be simplified to:

$$\begin{aligned}
&\forall F \in fields, B \in shp_brds, I \in int_{24}, J \in int_{365}. \\
&biomass(shp(F,B), next(hr(I,J))) - biomass(shp(F,B), hr(I,J)) \\
&\quad = \quad grazing(shp(F,B), grs(F), hr(I,J)) - resp(shp(F,B), hr(I,J))
\end{aligned}$$

To initiate the calculation the user must supply values for the various constants, *e.g.* *sp_rte_graze*, and procedures for the parameters, external variables, and intermediate variables *e.g.* *bmass_resp*, *temp*, *etc.* used in the equations in appendix A. When general-purpose procedures are available EcoLogic will prestore them, and invite the user to choose one. Otherwise, the user must provide the procedures in tabular form. The user must also give initial values to the state variables, *e.g.* $biomass(shp(F,B), hr(1,1))$. EcoLogic should prompt for these initial values. It can then use them to calculate the value of $biomass(shp(F,B), hr(2,1))$, and so on, for each of the 6 values of $shp(F,B)$. Equation (9) can then be used to calculate the 6 sets of daily averages, from which the 6 annual maximums can be calculated, as required, and presented to the user for comparison, as required .

10 The State of the EcoLogic Implementation

The initial implementation of EcoLogic was the EL program, [Robertson *et al.* 87]. Various experimental extensions of this program are currently being developed in the Eco project, including the programs ELK (Uschold), SL (Robertson) and NIPPIE (Haggith). Each of these programs implements some of the ideas described above. We aim to use them as the basis for a complete implementation of our ideas.

- ELK consists of a sort hierarchy and function editor and a goal elicitation assistant. The editor helps users to browse through the sort hierarchy and add new sorts, and instances to it, as well as for defining new functions. The hierarchy is presented as a tree and the current editing choices are presented as a frame (see figure 2). Additional functions may be specified by filling slots in a frame. Given the name of an attribute, the most general sort to which it applies, and a value space, the system easily infers the type of the corresponding function (*e.g.* $eye\_colour : animals \mapsto \{blue, green, brown\} : set(colours)$. [8]

  The goal elicitation assistant helps users to construct logical terms to represent dependent and independent variables as described in §6. The current term is displayed as a frame whose slots contain different parts of the term (see figure 3).

- SL contains subsystems for editing the sort hierarchy, creating substructure and eliciting ecological variables. The substructure representation is essentially that given in §5. The method of eliciting ecological variables is based on that described in §6. Logical terms are presented to the user as stylised English text.

- EL and NIPPIE both use the Marples Algorithm (see §8) but they use it for forming a Prolog simulation program out of program chunks rather than putting together equations. It is planned to explore its use in eliciting equations as part of the ELK program. NIPPIE uses a variant of the formalism for representing substructure that we have described above, it elicits this substructure in a manner similar to that described in §7 above and is able to form runnable models containing substructure.

Each of these experimental systems is implemented in Quintus Prolog and runs on Sun3 computers. ELK and NIPPIE also use the Quintus ProWindows package.

As the interfaces of these programs indicate it is possible to protect the user from the mathematical complexities of the logical formulae, while exploiting these complexities to guide the elicitation process. The user can define the ecological variables, the substructure of ecological entities and the differential equations by answering questions which make ecological sense and without being overwhelmed with choices. The resulting formalism can be presented to the user in an ecological intelligible form.

11 Conclusion

In this paper we have described the use of typed lambda calculus to represent ecological models as sets of differential and non-differential equations and discussed a requirements capture system for assisting the user in specifying a model and, hence, constructing these equations.

We make the following claims about this representation.

- In representing the ecological modelling domain we have found a need for second order functions, polymorphism and a rich, first-order, order-sorted, sort structure, with a function *set* from sorts to sorts. We have made a novel use of function application to represent ecological substructure.

- This logical language provides a general formalism for describing ecological models which extends previously proposed formalisms, *e.g.* systems dynamics.

[8] A similar editing facility for processes like *grazing* is forthcoming.

The user creates the sheep instance shp by selecting the edit command for adding instances and filling in the appropriate slots. There are two major classes of commands: DISPLAY *and* EDIT. *The current command and instructions for its use are shown. A portion of the sort hierarchy is shown below from a past use of the display taxonomy command.*

Figure 2: The ELK Sort Hierarchy Editor Interface

The user is defining the dependent variable using the Y entity *window. (S)he starts by selecting shp from the sort hierarchy as the object of interest and then selects biomass, from a generated menu, as the quantity of shp to be modelled. This is then qualified by average and maximum using a separate window. A complete, English translation of the state of the specification is automatically generated.*

Figure 3: The ELK Goal Elicitation Assistant Interface

In particular, unlike systems dynamics, it is possible to represent substructure in a way that permits quantification across instances from different elements of the substructure.

- The special-purpose needs of the model are met by combining general-purpose functions in compound terms. Compatibility is checked by reference to the hierarchy of sorts. Ecological knowledge is expressed via the type structure and the association of dimensions, processes, *etc.* with ecological entities of different types.
- Our formalism facilitates the elicitation of the model from the user. Users are able to build models in a series of refinement steps. Each of these steps can be presented in a form that is ecologically meaningful, *e.g.* as a pre-stored question and a menu of possible answers. Each menu consists of a small number of general-purpose functions, dimensions, processes, *etc.*, limited to those that are compatible with the current state of the model. Thus the user is not overwhelmed by a combinatorial explosion of choices.

Thus our representation has provided a framework for the formal characterisation of the space of choices to be made in ecological modelling. We have also started to fill in this framework by defining a collection of general purpose ecological functions, sorts, *etc.* More empirical work, further testing our system with real ecological models, is required to build this into a comprehensive collection. We have also started to provide automatic guidance to users on how to search the space of ecological models. Much more work is required on this aspect before our EcoLogic system will be usable by ecologists without previous modelling experience.

This use of typed lambda calculus does not call for higher order unification. One way matching is required to instantiate ecological formulae into the equations required for the model, but higher order matching is a much simpler computational problem than higher order unification. Thus, one of the major computational difficulties of using higher order logic is avoided.

The major ideas in this paper have been partially or totally implemented in one of the experimental programs that make up the EcoLogic system. The common underlying logical basis of these programs makes merging them computationally feasible, and we are currently working to this end.

A Equations for Sheep Biomass Model

The following is the complete set of equations which form the model for the major running example in this paper. To be run they require simplification and rewriting into difference equations. The types of all new functions used below is given in appendix C.

$$\begin{aligned}
&\forall F \in fields, B \in shp_brds, J \in int_{365}.\ average(\lambda Hr : hours.biomass(shp(F,B),Hr), dy(J)) \\
&= \frac{\Sigma(\lambda Hr : hours.biomass(shp(F,B),Hr), dy(J))}{size(dy(J))}
\end{aligned}$$

$$\begin{aligned}
&\forall F \in fields, B \in shp_brds, I \in int_{24}, J \in int_{365}.\ rate(\lambda T.biomass(shp(F,B),T))(hr(I,J)) \\
&= \Sigma(\lambda Plt : grass.grazing(shp(F,B), Plts, hr(I,J)), \{grs(F)\}) - resp(shp(F,B), hr(I,J))
\end{aligned}$$

$$\begin{aligned}
&\forall F \in fields, I \in int_{24}, J \in int_{365}.\ rate(\lambda T.biomass(grs(F),T))(hr(I,J)) \\
&= photo(grs(F), hr(I,J)) - resp(grs(F), hr(I,J))
\end{aligned}$$

$$-\Sigma(\lambda Shp : sheep.grazing(Shp, grs(F), hr(I, J)), \{\lambda B.shp(F, B)|shp\_brds\})$$

$$\begin{aligned}
&\forall F \in fields, B \in shp_brds, I \in int_{24}, J \in int_{365}.\ grazing(shp(F, B), grs(F), hr(I, J)) \\
&= \ sp_rte_graze.biomass(grs(F), hr(I, J)).biomass(shp(F, B), hr(I, J))
\end{aligned}$$

$$\begin{aligned}
&\forall F \in fields, B \in shp_brds, I \in int_{24}, J \in int_{365}.\ resp(shp(F, B), hr(I, J)) \\
&= \ sp_rte_resp(shp(F, B)).bmass_resp(shp(F, B), temp(hr(I, J))).biomass(shp(F, B), hr(I, J))
\end{aligned}$$

$$\begin{aligned}
&\forall F \in fields, I \in int_{24}, J \in int_{365}.\ resp(grs(F)), hr(I, J)) \\
&= \ sp_rte_resp(grs(F)).bmass_resp(grs(F), temp(hr(I, J))).biomass(grs(F), hr(I, J))
\end{aligned}$$

$$\begin{aligned}
&\forall F \in fields, I \in int_{24}, J \in int_{365}.\ photo(grs(F), hr(I, J)) \\
&= \ sp_rte_photo(grs(F)).bmass_photo(grs(F), light(hr(I, J))).biomass(grs(F), hr(I, J))
\end{aligned}$$

B Types of Ecological Variables

Ecological modellers call many of the functions that they use *variables*. This terminology is not odd; it is the norm in mathematics. Consider, for instance, the "dependent variable", which is necessarily a function of the "independent variable". In fact, it is the use of the word "variables" by logicians which is out of step. In this paper we use the phrase "ecological variable" and "logical variable" when there might be ambiguity.

Ecologists have identified several type of ecological variable: *state variables, processes, intermediate variables, external variables, parameters, constants.* The main difference seems to be the type of their dominant function. These types can be inferred from the use of the different kinds of ecological variable in system dynamics notation and are given in table 1 below.

Variables	Dominant Function's		Type	Example
State Variable	$objects \times times$	$\mapsto$	$numbers$	$biomass$
Processes	$objects^* \times times$	$\mapsto$	$numbers$	$grazing$
Intermediate Variable	$objects^* \times numbers^*$	$\mapsto$	$numbers$	$bmass\_resp$
External Variable	$times$	$\mapsto$	$numbers$	$temp$
Parameter	$objects$	$\mapsto$	$numbers$	sp_rte_resp
Constants			$numbers$	sp_rte_graze

Table 1: The Types of Ecological Variables

- **State Variables**: represent attributes of ecological objects which vary over time. These attributes are affected by various processes. The variations are defined by differential equations for the attributes in terms of the processes.
- **Processes**: represent the amount of some quantity that is transferred between ecological objects at each time increment. These incremental amounts are defined by non-differential equations in terms of the old values of the attribute and various intermediate variables, external variables, parameters and constants.

- **Intermediate Variables**: represent values that depend on the ecological objects in the model and on other intermediate variables, external variables, parameters or constants.
- **External Variables**: represent values that change over time, but are not affected by anything in the model.
- **Parameters**: represent values that depend on the ecological objects in the model, but do not change over time.
- **Constants**: represent values that do not change at all.

Second order functions, like *average*, are not usually described as ecological variables but as *aggregation operators*, which combine together similar ecological variables across the elements of the substructure.

EcoLogic can already represent a wider range of models than systems dynamics; a potential we are still exploring. As we do this we expect to extend the definitions of the ecological variables given above, which are based on a purely system dynamics view of models. For instance, we do not want to be restricted to variables with numerical values.

C Glossary of Types

In table 2 we record the types of all the logical constants and functions used in this paper, in alphabetical order.

Quantification	Function	Type
$\forall Sort : \varrho(entities), \forall Nums : \varrho(numbers).$	$average :$	$(Sort \mapsto Nums) \times set(Sort) \mapsto Nums$
	$asymptote :$	$set(objects) \mapsto naturals$
	$biomass :$	$lifeforms \times times \mapsto reals$
	$bmass\_photo :$	$lifeforms \times reals \mapsto reals$
	$bmass\_resp :$	$lifeforms \times reals \mapsto reals$
	$dy :$	$naturals \mapsto set(hours)$
	$fields :$	$set(locations)$
	$grazing :$	$animals \times plants \times times \mapsto numbers$
	$grs :$	$locations \mapsto grass$
	$heath :$	$locations$
	$hr :$	$naturals \times naturals \mapsto hours$
	$int_{24} :$	$set(naturals)$
	$int_{365} :$	$set(naturals)$
	$leicester :$	$breeds$
	$light :$	$times \mapsto reals$
$\forall Sort : \varrho(entities), \forall Nums : \varrho(numbers).$	$maximum :$	$(Sort \mapsto Nums) \times set(Sort) \mapsto Nums$
	$meadow :$	$locations$
	$next :$	$times \mapsto times$
	$number :$	$set(objects) \times times \mapsto naturals$
	$pasture :$	$locations$
	$photo :$	$plants \times times \mapsto numbers$
	$pop :$	$set(objects)$
$\forall Sort : \varrho(entities), \forall Nums : \varrho(numbers).$	$rate :$	$(Sort \mapsto Nums) \mapsto (Sort \mapsto Nums)$
	$resp :$	$lifeforms \times times \mapsto numbers$
	$set :$	$\varrho(entities) \mapsto \varrho(entities)$
	$shetland :$	$breeds$
	$shp :$	$locations \times breeds \mapsto sheep$
	$shp\_brds :$	$set(breeds)$
$\forall Sort : \varrho(entities), \forall Nums : \varrho(numbers).$	$\Sigma :$	$(Sort \mapsto Nums) \times set(Sort) \mapsto Nums$
$\forall Sort : \varrho(entities).$	$size :$	$set(Sort) \mapsto naturals$
	$sp\_rte\_graze :$	$reals$
	$sp\_rte\_photo :$	$lifeforms \mapsto reals$
	$sp\_rte\_resp :$	$lifeforms \mapsto reals$
	$t :$	$times$
	$temp :$	$times \mapsto reals$
	$yr :$	$set(set(hours))$

Table 2: The Types of Logical Constants and Functions

References

[Bundy *et al.* 79] A. Bundy, L. Byrd, G. Luger, C. Mellish, R. Milne, and M. Palmer. Solving mechanics problems using meta-level inference. In B. G. Buchanan, editor, *Proceedings of IJCAI-79*, pages 1017–1027. International Joint Conference on Artificial Intelligence, 1979. Reprinted in 'Expert Systems in the microelectronic age' ed. Michie, D., pp. 50-64, Edinburgh University Press, 1979. Also available from Edinburgh as DAI Research Paper No. 112.

[Bundy *et al.* 82] A. Bundy, L. Byrd, and C. Mellish. Special purpose, but domain independent inference mechanisms. In *Proceedings of ECAI-82*, pages 67–74. European Conference on Artificial Intelligence, 1982. Also available from Edinburgh as DAI Research Paper No. 179 and the Electro-Technology Journal Vol. XXXI, No. 1 2.

[Cardelli 88] L. Cardelli. Structural subtyping and the notion of power type. *ACM Proc. POPL*, 1988.

[Church 40] A. Church. A formulation of the simple theory of types. *Symbolic Logic*, 5(1):56–68, 1940.

[Forrester 61] J. W. Forrester. *Industrial Dynamics*. MIT Press, 1961.

[Niven 82] B. S. Niven. Formalisation of the basic concepts of animal ecology. *Erktennis*, 17:307–320, 1982.

[Robertson *et al.* 87] D. Robertson, Alan Bundy, M. Uschold, and R. Muetzelfeldt. Helping inexperienced users to construct simulation programs: An overview of the ECO project. In S. Moralee, editor, *Research and Development in Expert Systems IV*, pages 185–197, Brighton, England, 1987. British Computer Society Specialist Group on Expert Systems, Cambridge University Press. Also available from Edinburgh as DAI Research paper 338.

[Robertson *et al.* 88] D. Robertson, Alan Bundy, M. Uschold, and R. Muetzelfeldt. Using ecological descriptions to guide the construction of simulation programs. In L. Clarke, editor, *Proceedings of UK IT 88*, pages 1531–156, Swansea, Wales, 1988. The Information Engineering Directorate. Also available from Edinburgh as DAI Research Paper No. 381.

[Robertson *et al.* 89] D. Robertson, M. Uschold, Alan Bundy, and R. Muetzelfeldt. The ECO program construction system: Ways of increasing its representational power and their effects on the user interface. *Int. J. Man-Machine Studies*, 31:1–26, 1989. Also available from Edinburgh as DAI Research paper 380.