Idealisation

Abstract

Idealisation is a form of human reasoning which is ubiquitous but is seldom analysed formally. This paper considers a particular form of idealisation in which we produce a more practical description of a problem by replacing functions defined over individuals with analogous functions defined over named subsets. We show how a number of (seemingly) diverse idealisations can be based on a single transformation method. Although general purpose idealisations can never be fully automatic, the transformation gives useful clues to the way in which the non-automatic parts of the idealisation might be built, as well as providing the core of the new definition. A key question once idealisation has been performed is whether the idealised description maintains any correspondence to the original. We define a minimal form of correspondence and show how this can be checked automatically.

1 Idealisation in General

Idealisation is the reasoning we perform when we take a problem for which we are unable directly to answer a question and represent it in a form which, although not directly in correspondence with the original problem, gives us an approximation which is close enough to supply useful answers. This form of reasoning is ubiquitous in modelling. Some examples are:

- Physics or mechanics problems are often addressed by boiling the problem down to a set of equations which can be solved mathematically. In order to do this, it is necessary to represent the problem using components (*e.g.* particles, inelastic connectors) which have just enough resemblance to the problem to give plausible solutions but which are idealised versions of (imperfectly understood) physical reality.

- Simulation models are often built with the express purpose of giving an idealised view of a system which accounts for some of its behaviours. For example, a simulation model of the spread of a virus through a human population would not represent each individual human and viral organism. It would idealise these in a computationally tractable form - probably as groups representing human and viral populations.

- Knowledge acquisition systems usually require domain specialists to supply information in forms which are known to be manageable by the inference engines which the acquisition systems support. Many of the problems in "knowledge engineering" stem from the difficulties people have in describing their problems in just the right (idealised) style to conform to these idealised notations.

Although idealisation is common, it has received little direct attention from the academic community. One reason for this is the mysteriousness of the

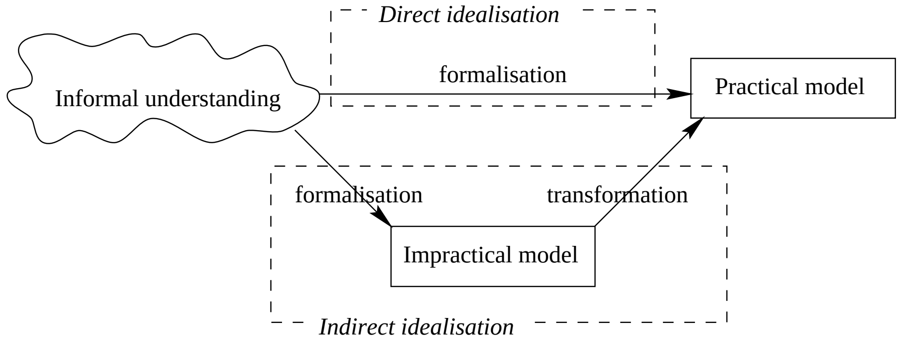

Figure 1: Two routes to practical idealisations

creative intuitions which we use in order to formalise our personal view of a problem - it is impossible to describe formally the relation between two representations if the nature of one of them is unknown. Consequently, support for this first formalisation step is normally provided using domain-specific tools which describe the most common ways of formalising a problem in the target domain. Examples of such tools in the ecological modelling domain are given in [Robertson *et al.* 91] and a more general set of tools is described in [Robertson 96b, Robertson 96a]. Such tools work in one of two ways:

- Some allow domain specialists to represent their problem in a domain-specific formal language which is reasonably familiar to them. This formal representation is then used to control the generation of a model appropriate to that problem - the step between problem description and model being guided by domain-specific heuristics.
- Some allow specification of the model in incremental stages, often working "top down" from a high level description of the main components to finer detail of the model's structure.

Both of these classes of tools are designed to help people supply knowledge which enables the tool to synthesise a practical model. This is summarised as the "direct idealisation" process in Figure 1 where our aim is to assist people in formalising their informal ideas[1].

There is an alternative route which has received much less attention: the indirect idealisation process shown as the lower path in Figure 1. This splits the process into (at least) two stages. The first is a formalisation stage as before but without the requirement that we will obtain a practical model. On the contrary, we expect the formalisation to produce a model which is impractical - for example, it may not allow appropriate questions to be answered in finite time or it may assume the presence of data which isn't readily available. Nevertheless,

[1] It might be argued that problem descriptions are a sort of intermediate model. In practice, such problem descriptions aren't used as models of systems but as cues for the domain-specific model generator.

our impractical model may be executable in limited ways and it is intended to correspond more directly to the features of our problem than will our practical model. We then face a different form of idealisation than we have discussed hitherto: from an impractical model via formal transformation to one which is practical. The advantage of idealising indirectly in this way is that for at least part of the idealisation we can be precise about where we start in idealising (the impractical model) as well as being precise about where we end (the practical model). In doing so, we have not removed the need for support in formalisation. However, we have relaxed the demands on this stage since we no longer require initial formalisation to yield a model which is already idealised appropriately.

Figure 2 depicts both the general scenario for indirect idealisation (as we described above) and a specific example which will feature in Section 5. In this example we expect our specialists to understand about a plant population. Their understanding may be deep but it isn't expressed formally. We assume that their understanding is at the level of individual plants, since these are the most obvious things to see in a plant community, and that they are interested to predict the mass of the plant population. This being so, we assume that it is possible for them to describe formally those interactions between individual plants which influence plant mass. This model might be impractical for various reasons: we may not be able to identify all the individuals in the population; our representation of time may be inappropriate for deriving some kinds of result; we may not be sure of the quantitative relationships determining mass. These impracticalities might prompt us to transform the impractical model to idealised versions which are more practical - for example by grouping the population into age classes; by splitting the times into seasons; or by using qualitative rather than qualitative values for mass. In Section 5 we show precisely how a single transformation can be used to drive this process. Before that, we introduce the formal notation we use to describe models (Section 3); introduce this transformation using a simple example (Section 4) and give the general definition for the transformation (Definition 1).

2 Idealisation Over Sets

Idealisation seems to involve a variety of patterns of reasoning but it frequently involves a shift in focus from functions which involve individuals directly to functions which involve symbolic representations of groups of objects (hence referring to the original objects indirectly). The first step in performing this sort of transformation is easy to describe: we take an expression, $X \in p$, which constrains X to be included in p and rewrite this to the expression $X_s \subseteq p$, which constrains X_s to be a subset of p. If the instances for X_s are the names of sets rather than enumerations of the sets themselves then we lift the "level of discussion" of the model from that of named individuals to that of named groups (for which we may not be able to enumerate all individuals). Although this first step is easy to describe, it has ramifications for the rest of the model which are difficult to control and which are not confined to the predicate in

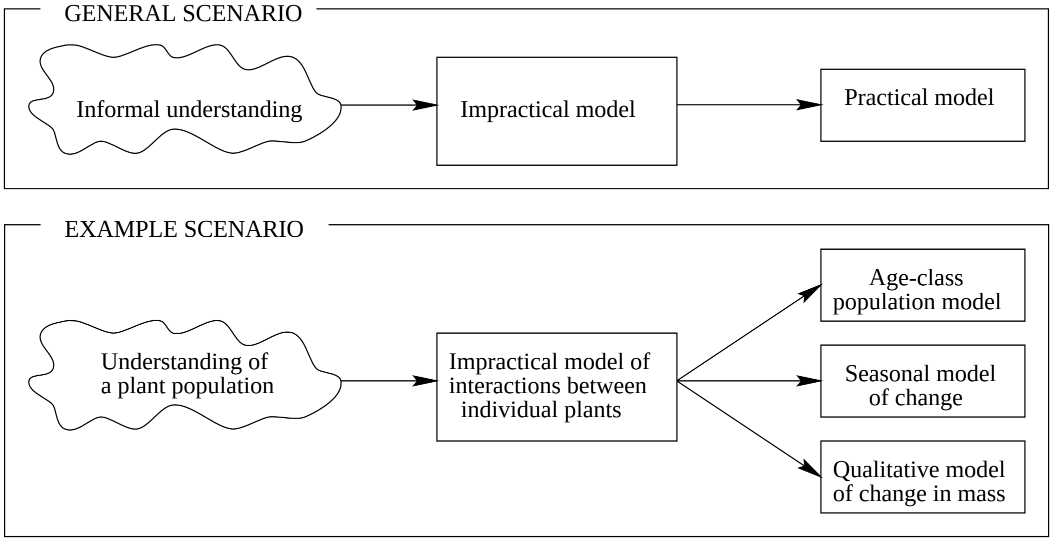

Figure 2: Idealisation via an impractical model

which we make the idealisation, as we show in Section 5.

3 Normal Form for Inclusion Clauses

The idealisation method of the following section requires those Horn clauses which are idealised to be in the following normal form:

- Each clause is either of the form $V \in P(A_1, \ldots, A_n) \leftarrow B$ or is of the form $V \subseteq P(A_1, \ldots, A_n) \leftarrow B$ where V is a variable; P is a unit goal representing a function in the specification; and B is the body of the clause.

- If the head of the clause is, $V \in P(A_1, \ldots, A_n)$, this is informally interpreted as "V is one of the results for function $P(A_1, \ldots, A_n)$". There is a direct mapping from this expression and the head of a Horn clause in more standard notation, where the "result" variable appears as an extra argument to P, giving us $P(A_1, \ldots, A_n, V)$.

- If the head of the clause is, $V \subseteq P(A_1, \ldots, A_n)$, this is informally interpreted as "V names a subset of the results for function $P(A_1, \ldots, A_n)$".

- The body of the clause, B, is a conjunction of unit goals which may be either the normal Prolog goals or may be goals which use the membership ('$\in$') or inclusion ('$\subseteq$') operators described above. Every variable, V', appearing in the head of the clause must appear as at least one conjunct of the body as either a goal of the form $V' \in E$ or $V' \subseteq E$.

We refer to this normal form as "inclusion notation".

4 The Essence of Set Idealisation

This section explains the basic idea of idealisation over sorts, using a simple example, and then defines a transformation which can be used to apply the idealisation to any predicate defined in our inclusion notation. Our example begins with the following Horn clause:

$$p(X,Y) \leftarrow q(X) \wedge r(X,Y) \tag{1}$$

Suppose that we know that the variable Y is the "output" of predicates p and r, and that the variable, X, is the "output" of predicate q. We can then rewrite clause 1 in inclusion notation as:

$$Y \in p(X) \leftarrow X \in q \wedge Y \in r(X) \tag{2}$$

For completeness, we also supply definitions for the objects included in q and r:

$$o1 \in q \tag{3}$$
$$o4 \in q \tag{4}$$

$$o2 \in r(o1) \tag{5}$$
$$o3 \in r(o1) \tag{6}$$

Using clauses 2 to 6 as a standard logic program, we can find two solutions for the goal $Y \in p(X)$: one with $X = o1$ and $Y = o2$; the other with $X = o1$ and $Y = o3$

We can idealise p in two ways: using variable X or using variable Y. In the first case, we are changing the definition of p to use named groups of instances of X, so our transformation must require that each named group (call it X_s) is included in q. Since r also used X and must now use X_s we need to allow for the possibility that it may need to be redefined. To do this, we replace r with the new function symbol, $\mathcal{F}_1$, and record the similarity between $\mathcal{F}_1$ and r using the assertion $similar(r(X), \mathcal{F}_1(X_s))$. We would also like to be able to check, whenever we use the idealised clause to satisfy a goal, that its use is consistent with at least one of the objects in the source clause. This test (which we call a "minimal assumption" in Definition 1) is defined by forming the conjunction of the body of the source clause and the goal $X \in X_s$. Intuitively, this requires that if we have a solution using the idealised clause then there should also exist at least one instance satisfying the conditions of the source clause for an object, X, in the idealised sort, X_s. We record the idealised clause, C, its source check, A, and similarities, S using the tuple (C, A, S) as shown below:

$$\left(\begin{array}{c} Y \in p(X_s) \leftarrow X_s \subseteq q \wedge Y \in \mathcal{F}_1(X_s), \\ \exists X.(X \in q \wedge Y \in r(X) \wedge X \in X_s), \\ \{similar(r(X), \mathcal{F}_1(X_s))\} \end{array}\right) \tag{7}$$

To make tuples like expression 7 easier to read when we move to our larger example in Section 5, we shall use a table to separate the three components. Expression 7 represented in this way would be:

Clause	$Y \in p(X_s) \leftarrow X_s \subseteq q \wedge Y \in \mathcal{F}_1(X_s)$
Test	$\exists X.(X \in q \wedge Y \in r(X) \wedge X \in X_s)$
Similarities	$\{similar(r(X), \mathcal{F}_1(X_s))\}$

However, for this section we stick to the more standard notation of expression 7.

This idealised clause requires definitions for $X_s \subseteq q$ and $\mathcal{F}_1(X_s)$. It is up to us to decide what these should be (guided by the similarity between $\mathcal{F}_1$ and r). we choose the following definitions.

$$sq \subseteq q \tag{8}$$

$$o2 \in \mathcal{F}_1(sq) \tag{9}$$

$$o3 \in \mathcal{F}_1(sq) \tag{10}$$

Using clauses 7 to 10, we can find two solutions for the goal $Y \in p(X_s)$: one with $X_s = sq$ and $Y = o2$; the other with $X_s = sq$ and $Y = o3$. We shall show in Section 6 how the source check of idealisation 7 can be tested as part of the procedure for satisfying goals. For now, it is sufficient to observe that the instantiation of idealisation 7 which allowed us to derive the first solution is:

$$\left(\begin{array}{c} o2 \in p(sq) \leftarrow sq \subseteq q \wedge o2 \in \mathcal{F}_1(sq), \\ \exists X.(X \in q \wedge o2 \in r(X) \wedge X \in sq), \\ \{similar(r(X), \mathcal{F}_1(sq))\} \end{array}\right)$$

Our source check for this instance is $\exists X.(X \in q \wedge o2 \in r(X) \wedge X \in sq)$, which is not satisfiable from the current clauses because we don't know what is a member of sq. However, if we add the information:

$$o1 \in sq \tag{11}$$

Then there is a solution for our source check, namely: $(o1 \in q \wedge o2 \in r(o1) \wedge o1 \in sq)$. In finding this solution we also instantiate the sililarity condition to $similar(r(o1), \mathcal{F}_1(sq))$.

We now turn to our second case of idealisation, using variable Y in clause 2. Here we introduce the inclusion in the head of the clause (since Y is determined by p) and we must again introduce a new function ($\mathcal{F}_2$) in place of r. Our idealisation is:

$$\left(\begin{array}{c} Y_s \subseteq p(X) \leftarrow X \in q \wedge Y_s \in \mathcal{F}_2(X), \\ \exists Y.(X \in q \wedge Y \in r(X) \wedge Y \in Y_s), \\ \{similar(r(X), \mathcal{F}_2(X))\} \end{array}\right) \tag{12}$$

We must then define $\mathcal{F}_2$, which we define simply to include the group sy.

$$sy \in \mathcal{F}_2(o1) \tag{13}$$

Using clauses 12, 3, 4 and 10, we can find only one solution for the goal $Y_s \in p(X)$: $X = o1$ and $Y_s = sy$. The instantiated version of idealisation 12 used to satisfy this goal is:

$$\left(\begin{array}{c} sy \subseteq p(o1) \leftarrow o1 \in q \wedge sy \in \mathcal{F}_2(o1), \\ \exists Y.(o1 \in q \wedge Y \in r(o1) \wedge Y \in sy), \\ \{similar(r(o1), \mathcal{F}_2(o1))\} \end{array}\right)$$

Our source check is : $\exists Y.(o1 \in q \wedge Y \in r(o1) \wedge Y \in sy)$ which is satisfiable if we define (for example) $o3$ to be a member of sy:

$$o3 \in sy \tag{14}$$

In the example above we have described idealisation on sorts informally. Definition 1 defines this transformation precisely and generalises it to predicates which may be recursive.

Definition 1 *Given a goal, G, containing variable, V, and a set of clauses, $\{C_1, \ldots, C_n\}$, defining a predicate capable of solving G, we write:*

$$idealised(\{(C_1, A_1, S_1), \ldots, (I_n, A_n, S_n)\}, V, G, \{(C'_1, A'_1, S'_1), \ldots, (I'_n, A'_n, S'_n)\})$$

if we can idealise G over V. Each C'_i is the idealised clause corresponding to C_i; $A'_i = A_i \cup \{\alpha\}$, where α is the minimal assumption made during the idealisation step; and $S'_i = S_i \cup \sigma$, where σ is the set of similarities between the original and idealised subgoals. If we are idealising from a source model (i.e. one which hasn't been idealised) then $A_i = \{\}$ and $S_i = \{\}$. The idealisation of each clause, C_i, is as follows:

- *Initialise σ to $\{\}$.*
- *Rewrite every subterm in C_i which is of the form $V \in E$ to a subterm of the form $V \subseteq E$. The set of affected variables, Δ, is initialised to $\{V\}$. Form a conjunction, τ, of the rewritten subterms.*
- *If V is the variable restricted by the predicate (i.e. G is of the form $V \in E_g$) then rewrite the head of clause C_i from $V \in P(\ldots)$ to $V \subseteq P(\ldots)$. Also rewrite each recursive subgoal of the form $V' \in P(\ldots)$ to $V' \subseteq P(\ldots)$. Each V' is added to Δ.*

- *If there is a subterm in C_i which is of the form $V_a \in Q(A_1, \ldots, A_m)$ and one of the variables in Δ appears in A_i then rewrite this subterm to $V_a \in \mathcal{F}(A_1, \ldots, A_m)$ where $\mathcal{F}$ is a new variable representing a relation to be supplied by the engineer. Add V_a to Δ. Add to σ the similarity relation $similar(Q(A_1, \ldots, A_m), \mathcal{F}(A_1, \ldots, A_m))$. Exhaustively apply this rewrite until no further subterms can be replaced. The set, σ, is now the full set of similarity relations and the clause has been completely rewritten.*

- *the original clause C_i is of the form $H \leftarrow B$, where H is the head of the clause and B is its body. The minimal assumption, α for the idealisation is $(B \wedge H \wedge \tau)$.*

5 An Example of Set Idealisation

In Section 4 we introduced idealisation using a simple example. We now work through a more complex example, which returns to the scenarios presented in Figure 2 of Section 1 and works through these in detail, using an example model. Section 5.1 describes the model used as the source for our example. We then apply the idealisation transformation given in Definition 1 to part of this model and show (in Sections 5.2 to 5.4) how this can help us to construct three different models, corresponding to the three different idealisations of Figure 2.

5.1 The Source Model

Axioms 15 and 16 define a simple model of mass for a plant.

$$N \in mass(X,T) \leftarrow \begin{array}{l} X \in plant \wedge \\ T \in initial\_time \wedge \\ N \in initial\_mass(X) \end{array} \tag{15}$$

$$N \in mass(X,T) \leftarrow \begin{array}{l} X \in plant \wedge \\ T \in time \backslash initial\_time \wedge \\ T_p \in previous(T) \wedge \\ N_p \in mass(X,T_p) \wedge \\ C = \{X_c \mid X_c \in competitor(X,T)\} \wedge \\ S \in sum\_masses(C,T_p) \wedge \\ N \in mass\_calculation(N_p,S) \end{array} \tag{16}$$

Under some circumstances we may be able to use this model directly to calculate the mass of any plant, given appropriate definitions for the subgoals of the model. For example, we might define axioms 17 to 29. The first three of these (axioms 17 to 19) supply instances of plants (these are a, b and c). The next three (axioms 20 to 22) give the initial values for mass of each individual plant. Axioms 23 and 24 define permitted time points, which are represented as integers of 1 (the initial time point) or greater (later time points). The ordering on successive time points is defined using the predicate $T_p \in previous(T)$

(axiom 25) which restricts the previous time point (T_p) to be the integer below the reference time (T). We allow all plants to compete with each other, so X_c is a competitor of plant X (using axiom 26) if it is any plant other than X. Axioms 27 and 28 calculate the numerical sum, N, of the masses of a set of plants in a standard recursive style. Finally, axiom 29 defines an arithmetic function for calculating the mass, N, of a plant from its previous mass, N_p, and the sum of the masses of competing plants, S.

$$a \in plant \tag{17}$$
$$b \in plant \tag{18}$$
$$c \in plant \tag{19}$$

$$10 \in initial\_mass(a) \tag{20}$$
$$20 \in initial\_mass(b) \tag{21}$$
$$30 \in initial\_mass(c) \tag{22}$$

$$1 \in initial\_time \tag{23}$$
$$T \in time \backslash initial\_time \quad \leftarrow \quad integer(T) \ \wedge \ T > 1 \tag{24}$$

$$T_p \in previous(T) \leftarrow integer(T) \ \wedge \ T > 1 \ \wedge \ T_p = T - 1 \tag{25}$$

$$X_c \in competitor(X, T) \leftarrow X \in plant \ \wedge \ X_c \in plant \ \wedge \ \neg X = X_c \tag{26}$$

$$N \in sum\_masses([X|R], T) \quad \leftarrow \quad \begin{array}{l} N_x \in mass(X, T) \wedge \\ N_r \in sum\_masses(R, T) \wedge \\ N = N_r + N_x \end{array} \tag{27}$$
$$0 \in sum\_masses([], T) \tag{28}$$

$$N \in mass\_calculation(N_p, S) \leftarrow N = N_p + ((N_p/10) * (N_p/(N_p + S))) \tag{29}$$

We can run the model defined by axioms 15 to 29 as a normal logic program. For example, we might want to know the predicted mass of each plant at time 3. For this, we ask for solutions to the goal:

$$N \in mass(X, 3)$$

which can be solved to yield three instances for X and N:

$$\begin{array}{ll} X = a & N = 10.3 \\ X = b & N = 21.4 \\ X = c & N = 33.1 \end{array}$$

This is fine in theory but deficient in practice. There are at least three situations in which the model we have defined would be inadequate:

- We might not be able to identify each individual plant. Real vegetation systems often contain thousands of plants and it isn't possible to identify each one. Even if we could, it may not be practical to define functional relationships like the one in axiom 29 because we don't understand precisely what drives competation between individual plants.
- We may not want an unbounded set of time points. Instead, we may prefer to group time points into a finite collection of larger classes (for example the seasons). This allows us to ask questions which exhaustively explore the space of possible times - for instance, if we want to know whether the mass of any animal can exceed a certain value in any time class.
- We may want to express numerical values, like those for $mass$, qualitatively. This is necessary if we are not confident enough of our quantitative model to commit ourselves to arithmetic equations like the one in axiom 29.

In Sections 5.2 to 5.4 we show how the basic idealisation method described in Section 4 can address each of these deficiencies in turn. In each case, we will be applying the method to one of the three arguments of the term $N \in mass(X,T)$. In Section 5.2 we apply it to X, giving us an idealisation on the population and avoiding the problem of having to identify individual plants. In Section 5.3 we apply it to T, allowing us to idealise the unbounded time sequence into a simple set of seasons. In Section 5.4 we apply it to N, turning our definition of $mass$ into a qualitative one.

5.2 Idealising on the Populations

For this idealisation we want to calculate $mass$ for groups of plants rather than for individuals so we apply the transformation of Definition 1 to the variable, X in clauses 15 and 16. Our interpretation of these groups in biological terms is that these will correspond to classes of plants of a similar age (the young and the old ones for example). This yields the following two idealisations.

Idealisation 1 *Corresponding to Expression 15:*

Clause	$N' \in mass(X_s, T) \leftarrow \; X_s \subseteq plant \wedge \; T \in initial\_time \wedge \; N' \in \mathcal{F}_1(X_s)$
Test	$\exists X, N. \left(\begin{array}{c} X \in plant \wedge \\ T \in initial\_time \wedge \\ N \in initial\_mass(X) \wedge \\ X \in X_s \end{array} \right)$
Similarities	$\{similar(initial\_mass(X), \mathcal{F}_1(X_s))\}$

Idealisation 2 *Corresponding to Expression 16:*

Clause	$N' \in mass(X_s, T) \leftarrow \; X_s \subseteq plant \wedge \; T \in time \backslash initial\_time \wedge \; T_p \in previous(T) \wedge \; N'_p \in mass(X_s, T_p) \wedge \; C' = \{X'_c \mid X'_c \in \mathcal{F}_2(X_s, T)\} \wedge \; S' \in \mathcal{F}_3(C', T_p) \wedge \; N' \in \mathcal{F}_4(N'_p, S')$
Test	$\exists X, N, N_p, C, S. \left(\begin{array}{c} X \in plant \wedge \\ T \in time \backslash initial\_time \wedge \\ T_p \in previous(T) \wedge \\ N_p \in mass(X, T_p) \wedge \\ C = \{Xc \mid Xc \in competitor(X, T)\} \wedge \\ S \in sum\_masses(C, T_p) \wedge \\ N \in mass\_calculation(N_p, S) \\ X \in X_s \end{array} \right)$
Similarities	$\left\{ \begin{array}{c} similar(competitor(X, T), \mathcal{F}_2(X_s, T)), \\ similar(sum\_masses(C, T_p), \mathcal{F}_3(C', T_p)), \\ similar(mass\_calculation(N_p, S), \mathcal{F}_4(N'_p, S')) \end{array} \right\}$

This has introduced several new functions to which we must now give meaningful names. Function $\mathcal{F}_1$ is similar to *initial_mass* but applies over age classes so we name this *age_class_initial_mass*. Similarly, we rename $\mathcal{F}_2$ to *age_class_competitor* ; $\mathcal{F}_3$ to *sum_age_class_masses* and $\mathcal{F}_4$ to *age_class_mass_calculation*.

Instantiated Idealisation 1 *Corresponding to Expression 15:*

Clause	$N' \in mass(X_s, T) \leftarrow \; X_s \subseteq plant \wedge \; T \in initial\_time \wedge \; N' \in age\_class\_initial\_mass(X_s)$
Test	$\exists X, N. \left(\begin{array}{c} X \in plant \wedge \\ T \in initial\_time \wedge \\ N \in initial\_mass(X) \wedge \\ X \in X_s \end{array} \right)$
Similarities	$\{similar(initial\_mass(X), age\_class\_initial\_mass(X_s))\}$

Instantiated Idealisation 2 *Corresponding to Expression 16:*

Clause	$N' \in mass(X_s, T) \leftarrow$	$X_s \subseteq plant \wedge$ $T \in time \backslash initial\_time \wedge$ $T_p \in previous(T) \wedge$ $N'_p \in mass(X_s, T_p) \wedge$ $C' = \{X'_c \mid X'_c \in age\_class\_competitor(X_s, T)\} \wedge$ $S' \in sum\_age\_class\_masses(C', T_p) \wedge$ $N' \in age\_class\_mass\_calculation(N'_p, S')$
Test	$\exists X, N, N_p, C, S.$	$\left(\begin{array}{c} X \in plant \wedge \\ T \in time \backslash initial\_time \wedge \\ T_p \in previous(T) \wedge \\ N_p \in mass(X, T_p) \wedge \\ C = \{Xc \mid Xc \in competitor(X, T)\} \wedge \\ S \in sum\_masses(C, T_p) \wedge \\ N \in mass\_calculation(N_p, S) \\ X \in X_s \end{array}\right)$
Similarities	$\left\{\begin{array}{c} similar(competitor(X, T), age\_class\_competitor(X_s, T)), \\ similar(sum\_masses(C, T_p), sum\_age\_class\_masses(C', T_p)), \\ similar(mass\_calculation(N_p, S), age\_class\_mass\_calculation(N'_p, S')) \end{array}\right\}$	

In order to run this model we need to name the age classes and to supply definitions for the new functions which we introduced. First we give *old* and *young* as named subsets of *plant*.

$$young \subseteq plant \tag{30}$$

$$old \subseteq plant \tag{31}$$

We then define the initial masses for each age class.

$$300 \in age\_class\_initial\_mass(young) \tag{32}$$

$$200 \in age\_class\_initial\_mass(old) \tag{33}$$

We then define competition between age classes in a similar way to that of our source definition (clause 26) but over plant groups rather than individuals.

$$X_c \in age\_class\_competitor(X, T) \leftarrow X \subseteq plant \ \wedge \ X_c \subseteq plant \ \wedge \ \neg X = X_c \tag{34}$$

The remaining definitions simply use the source definitions directly.

$$N \in sum\_age\_class\_masses(C, T) \leftarrow N \in sum\_masses(C, T) \tag{35}$$

$$N \in age\_class\_mass\_calculation(N_p, S) \leftarrow N \in mass\_calculation(N_p, S) \tag{36}$$

We can now use this model to answer questions about the mass of the plant age classes. For example, if we ask for solutions to the goal:

$$N \in mass(X_s, 3)$$

then we obtain three instances for X_s and N.

$$\begin{array}{ll} X_s = young & N = 337.2 \\ X_s = old & N = 216.2 \end{array}$$

5.3 Idealising on Time

For our second idealisation we want to calculate $mass$ over a finite set of seasons (spring, summer, autumn and winter), rather than for the infinite set of all possible time points, so we apply the transformation of Definition 1 to the variable, T in clauses 15 and 16. This yields the following two idealisations.

Idealisation 3 *Corresponding to Expression 15:*

Clause	$N' \in mass(X, T_s) \leftarrow \; X \in plant \wedge T_s \subseteq initial\_time \wedge N' \in initial\_mass(X)$
Test	$\exists T, N. \left(\begin{array}{c} X \in plant \wedge \\ T \in initial\_time \wedge \\ N \in initial\_mass(X) \wedge \\ T \in T_s \end{array} \right)$
Similarities	$\{\}$

Idealisation 4 *Corresponding to Expression 16:*

Clause	$N' \in mass(X, T_s) \leftarrow \; X \in plant \wedge T_s \subseteq time \backslash initial\_time \wedge T_{ps} \in \mathcal{F}_5(T_s) \wedge N'_p \in mass(X, T_{ps}) \wedge C' = \{X'_c, X'_c \in \mathcal{F}_6(X, T_{ps})\} \wedge S' \in \mathcal{F}_7(C', T_{ps}) \wedge N' \in \mathcal{F}_8(N'_p, S')$
Test	$\exists T, T_p, N, N_p, C, S. \left(\begin{array}{c} X \in plant \wedge \\ T \in time \backslash initial\_time \wedge \\ T_p \in previous(T) \wedge \\ N_p \in mass(X, T_p) \wedge \\ C = \{Xc \mid Xc \in competitor(X, T)\} \wedge \\ S \in sum\_masses(C, T_p) \wedge \\ N \in mass\_calculation(N_p, S) \\ T \in T_s \end{array} \right)$
Similarities	$\left\{ \begin{array}{c} similar(previous(T)), \mathcal{F}_5(T_s)), \\ similar(competitor(X, T), \mathcal{F}_6(X, T_{ps})), \\ similar(sum\_masses(C, T_p), \mathcal{F}_7(C', T_{ps})), \\ similar(mass\_calculation(N_p, S), \mathcal{F}_8(N'_p, S')) \end{array} \right\}$

As in the previous idealisation, we must now name our new functions. We use the similarities (as before) to help us think of meaningful names, bearing in mind that we will be looking for versions of the original functions which operate over seasons. We rename $\mathcal{F}_5$ to *previous_season* ; $\mathcal{F}_6$ to *seasonal_competitor* ; $\mathcal{F}_7$ to *sum_seasonal_masses* and $\mathcal{F}_8$ to *seasonal_mass_calculation*.

Instantiated Idealisation 3 *Corresponding to Expression 15:*

Clause	$N' \in mass(X,T_s) \leftarrow X \in plant \wedge T_s \subseteq initial\_time \wedge N' \in initial\_mass(X)$
Test	$\exists T, N. \left(\begin{array}{c} X \in plant \wedge \\ T \in initial\_time \wedge \\ N \in initial\_mass(X) \wedge \\ T \in T_s \end{array} \right)$
Similarities	$\{\}$

Instantiated Idealisation 4 *Corresponding to Expression 16:*

Clause	$N' \in mass(X,T_s) \leftarrow X \in plant \wedge T_s \subseteq time \backslash initial\_time \wedge T_{ps} \in previous\_season(T_s) \wedge N'_p \in mass(X,T_{ps}) \wedge C' = \{X'_c, X'_c \in seasonal\_competitor(X,T_{ps})\} \wedge S' \in sum\_seasonal\_masses(C',T_{ps}) \wedge N' \in seasonal\_mass\_calculation(N'_p,S')$
Test	$\exists T, T_p, N, N_p, C, S. \left(\begin{array}{c} X \in plant \wedge \\ T \in time \backslash initial\_time \wedge \\ T_p \in previous(T) \wedge \\ N_p \in mass(X,T_p) \wedge \\ C = \{Xc \mid Xc \in competitor(X,T)\} \wedge \\ S \in sum\_masses(C,T_p) \wedge \\ N \in mass\_calculation(N_p,S) \\ T \in T_s \end{array} \right)$
Similarities	$\left\{ \begin{array}{c} similar(previous(T), previous\_season(T_s)), \\ similar(competitor(X,T), seasonal\_competitor(X,T_{ps})), \\ similar(sum\_masses(C,T_p), sum\_seasonal\_masses(C',T_{ps})), \\ similar(mass\_calculation(N_p,S), seasonal\_mass\_calculation(N'_p,S')) \end{array} \right\}$

We now must define the temporal groups. Spring is defined to be our only initial time group. The other three seasons are in times other than the initial ones.

$$spring \subseteq initial\_time \tag{37}$$

$$summer \subseteq time \backslash initial\_time \tag{38}$$
$$autumn \subseteq time \backslash initial\_time \tag{39}$$
$$winter \subseteq time \backslash initial\_time \tag{40}$$

We then define on ordering on the time groups according to which precedes each one.

$$spring \in previous\_season(summer) \tag{41}$$
$$summer \in previous\_season(autumn) \tag{42}$$
$$autumn \in previous\_season(winter) \tag{43}$$

The definitions for competition from our source model can then be used directly for the corresponding new functions.

$$X_c \in seasonal\_competitor(X, T_{ps}) \leftarrow X_c \in competitor(X, T_{ps}) \tag{44}$$

$$S \in sum\_seasonal\_masses(C, T_{ps}) \leftarrow S \in sum\_masses(C, T_{ps}) \tag{45}$$

$$N \in seasonal\_mass\_calculation(N_p, S) \leftarrow N \in mass\_calculation(N_p, S) \tag{46}$$

In previous models we had to instantiate the time at which we wished to establish the mass of an object or group. However, in this model we have a finite set of temporal groups so we can leave both the arguments of $mass$ uninstantiated. For example, if we ask for solutions to the goal:

$$N \in mass(X, T_s)$$

then we obtain :

$$\begin{array}{lll} X = a & T_s = spring & N = 10 \\ X = b & T_s = spring & N = 20 \\ X = c & T_s = spring & N = 30 \\ X = a & T_s = summer & N = 10.2 \end{array}$$

and so on, for all the combinations of plants and seasons.

5.4 Idealising on Values for Mass

For our final idealisation we aren't interested in the numerical value for $mass$ but want instead a qualitative estimation of whether it is low, medium or high. Therefore we apply the transformation of Definition 1 to the variable, N in clauses 15 and 16. This yields the following two idealisations.

Idealisation 5 *Corresponding to Expression 15:*

Clause	$N_s \subseteq mass(X,T) \leftarrow X \in plant \wedge T \in initial\_time \wedge N_s \subseteq \mathcal{F}_9(X)$
Test	$\exists N. \left(\begin{array}{c} X \in plant \wedge \\ T \in initial\_time \wedge \\ N \in initial\_mass(X) \wedge \\ N \in Ns \end{array} \right)$
Similarities	$\{similar(initial\_mass(X), \mathcal{F}_9(X))\}$

Idealisation 6 *Corresponding to Expression 16:*

Clause	$N_s \subseteq mass(X,T) \leftarrow X \in plant \wedge T \in time \backslash initial\_time \wedge T_p \in previous(T) \wedge N_{ps} \subseteq mass(X, T_p) \wedge C' = \{X'_c, X'_c \in competitor(X,T)\} \wedge S' \in \mathcal{F}_{10}(C', T_p) \wedge N_s \in \mathcal{F}_{11}(N_{ps}, S')$
Test	$\exists N, N_p, C, S. \left(\begin{array}{c} X \in plant \wedge \\ T \in time \backslash initial\_time \wedge \\ T_p \in previous(T) \wedge \\ N_p \in mass(X, T_p) \wedge \\ C = \{Xc \mid Xc \in competitor(X,T)\} \wedge \\ S \in sum\_masses(C, T_p) \wedge \\ N \in mass\_calculation(N_p, S) \\ N \in N_s \end{array} \right)$
Similarities	$\left\{ \begin{array}{c} similar(sum\_masses(C, T_p), \mathcal{F}_9(C', T_p)), \\ similar(mass\_calculation(N_p, S), \mathcal{F}_{10}(N_{ps}, S')) \end{array} \right\}$

As previously, we rename the new functions to more meaningful names. Function $\mathcal{F}_9$ becomes *approximate_initial_mass* ; $\mathcal{F}_{10}$ becomes *sum_approximate_masses* and $\mathcal{F}_{11}$ becomes *approximate_mass_calculation*.

Instantiated Idealisation 5 *Corresponding to Expression 15:*

Clause	$N_s \subseteq mass(X,T) \leftarrow X \in plant \wedge T \in initial\_time \wedge N_s \subseteq approximate\_initial\_mass(X)$
Test	$\exists N. \left(\begin{array}{c} X \in plant \wedge \\ T \in initial\_time \wedge \\ N \in initial\_mass(X) \wedge \\ N \in Ns \end{array} \right)$
Similarities	$\{similar(initial\_mass(X), approximate\_initial\_mass(X))\}$

Instantiated Idealisation 6 *Corresponding to Expression 16:*

Clause	$N_s \subseteq mass(X,T) \leftarrow$	$X \in plant \wedge$ $T \in time \backslash initial\_time \wedge$ $T_p \in previous(T) \wedge$ $N_{ps} \subseteq mass(X,T_p) \wedge$ $C' = \{X'_c, X'_c \in competitor(X,T)\} \wedge$ $S' \in sum\_approximate\_masses(C',T_p) \wedge$ $N_s \in approximate\_mass\_calculation(N_{ps},S')$
Test	$\exists N, N_p, C, S.$	$\left(\begin{array}{c} X \in plant \wedge \\ T \in time \backslash initial\_time \wedge \\ T_p \in previous(T) \wedge \\ N_p \in mass(X,T_p) \wedge \\ C = \{Xc \mid Xc \in competitor(X,T)\} \wedge \\ S \in sum\_masses(C,T_p) \wedge \\ N \in mass\_calculation(N_p,S) \\ N \in N_s \end{array}\right)$
Similarities	$\left\{\begin{array}{c} similar(sum\_masses(C,T_p), sum\_approximate\_masses(C',T_p)), \\ similar(mass\_calculation(N_p,S), approximate\_mass\_calculation(N_{ps},S')) \end{array}\right\}$	

We must now define the qualitative initial mass for each plant.

$$low \subseteq approximate\_initial\_mass(a) \tag{47}$$
$$medium \subseteq approximate\_initial\_mass(b) \tag{48}$$
$$high \subseteq approximate\_initial\_mass(c) \tag{49}$$

Since we have adapted the definition of *mass* to be qualitative, we must also alter the sum of masses ot perform a qualitative calculation on the groups. This uses a qualitative addition relation which determines (via simple "table lookup") the qualitative result, N, of adding two qualitative values, N_x and N_r.

$$N \in sum\_approximate\_masses([X|R],T) \leftarrow \begin{array}{l} N_x \subseteq mass(X,T) \wedge \\ N_r \in sum\_approximate\_masses(R,T) \wedge \\ qualitative\_addition(N_r,N_x,N) \end{array} \tag{50}$$
$$low \in sum\_approximate\_masses([],T) \tag{51}$$

$$qualitative\_addition(low,low,low) \tag{52}$$
$$qualitative\_addition(low,medium,medium) \tag{53}$$
$$qualitative\_addition(low,high,high) \tag{54}$$
$$qualitative\_addition(medium,low,medium) \tag{55}$$
$$qualitative\_addition(medium,medium,high) \tag{56}$$
$$qualitative\_addition(medium,high,high) \tag{57}$$
$$qualitative\_addition(high,low,high) \tag{58}$$

$$qualitative\_addition(high, medium, high) \tag{59}$$
$$qualitative\_addition(high, high, high) \tag{60}$$

$$medium \in approximate\_mass\_calculation(low, low) \tag{61}$$
$$medium \in approximate\_mass\_calculation(low, medium) \tag{62}$$
$$low \in approximate\_mass\_calculation(low, high) \tag{63}$$
$$high \in approximate\_mass\_calculation(medium, low) \tag{64}$$
$$medium \in approximate\_mass\_calculation(medium, medium) \tag{65}$$
$$medium \in approximate\_mass\_calculation(medium, high) \tag{66}$$
$$high \in approximate\_mass\_calculation(high, low) \tag{67}$$
$$high \in approximate\_mass\_calculation(high, medium) \tag{68}$$
$$high \in approximate\_mass\_calculation(high, high) \tag{69}$$

We can now obtain qualitative answers to questions about the mass of plants. For example, if we ask for solutions to the goal:

$$N_s \in mass(X, 3)$$

then we obtain :

$$\begin{array}{ll} X = a & N_s = low \\ X = b & N_s = medium \\ X = c & N_s = high \end{array}$$

6 Meta Interpretation of Idealised Definitions

In Section 4 we introduced the idea that the transformations which perform sort idealisation could construct a test which accompanies each idealised clause. This is the term which appears in the "Test" row of each of the tables showing idealised clauses in the previous sections. The key idea is that when we instantiate the idealised clause we will also instantiate (through shared variables) the corresponding test. If we remember all the instantiated tests to which we were committed during the solution to a given goal then we can check whether these hold in the source model. If they do, then we have evidence strengthening our belief that (for the chosen goal) our idealisation maintains at least some correspondence to the source model. On the other hand, if the tests fail then either we have an incorrect correspondence between source and idealised model or we have omitted definitions which would enable us to make the appropriate correspondence. The purpose of this section is to define the machinery needed to perform this task.

Assuming that we are finding solutions to goals in the normal Prolog style, the most direct way of accumulating test instances is by writing a meta-interpreter which will attempt to satisfy goals using idealised clauses and, while doing so, accumulate a set of tests. This is defined by expressions 70 to 77 below. The goal $solve(G, I_s, I_f)$ succeeds if the goal, G, is satisfiable from the axioms available. The set I_s contains the test accumulated so far (an empty set when the goal is first called) and I_f is instantiated to the set of tests accumulated once a solution to the goal has been found. Expressions 70 to 72 follow the normal pattern of dealing with conjunction, disjunction and negation. Expression 73 deals with the case where we need the set, S, of results for some goal, G, which we obtain by finding the set of pairs, (X, I), of results, X, for the goal with accompanying tests, I. We then split these pairs into the set of results and their accompanying tests. The collection of each test is performed by expression 74, which first checks that the goal is one which has been idealised. It then finds an idealised clause using the predicate $iclause(G \leftarrow P, C, S)$, which succeeds if there exists an idealised clause, $G \leftarrow P$ with test C and similarities S (which corresponds to the three rows in our idealisation tables of previous sections). If P can be solved then the final set of tests contains C and the tests obtained when satisfying P. Expressions 75 and 76 solve membership and inclusion goals by finding clauses in the model which match the goal with precondition, P, and satisfying P. Finally, expression 77 satisfies system-defined goals (such as arithmetic expressions or unifications) by calling them directly.

$$
\begin{aligned}
solve(A \wedge B, I_s, I_f) &\leftarrow solve(A, I_s, I_n) \wedge solve(B, I_n, I_f) && (70)\\
solve(A \vee B, I_s, I_f) &\leftarrow solve(A, I_s, I_f) \vee solve(B, I_s, I_f) && (71)\\
solve(not(G), I_s, I_s) &\leftarrow not(solve(G, \{\}, I)) && (72)\\
solve(S = \{X|G\}, I_s, I_f) &\leftarrow R = \{(X, I)|solve(G, [], I)\} \wedge && (73)\\
&\qquad I_f = I_s \cup \{I'|(X', I') \in R\} \wedge \\
&\qquad S = \{X'|(X', I') \in R\} \\
solve(G, I_s, I_f) &\leftarrow is\_idealised(G) \wedge && (74)\\
&\qquad iclause(G \leftarrow P, C, S) \wedge \\
&\qquad I_n = I_s \cup \{C\} \wedge \\
&\qquad solve(P, I_n, I_f) \\
solve(X \in S, I_s, I_f) &\leftarrow clause(X \in S, P) \wedge solve(P, I_s, I_f) && (75)\\
solve(X \subseteq S, I_s, I_f) &\leftarrow clause(X \subseteq S, P) \wedge solve(P, I_s, I_f) && (76)\\
solve(G, I_s, I_s) &\leftarrow system\_defined(G) \wedge G && (77)
\end{aligned}
$$

Given this definition of a solution strategy which accumulates tests, we can now define a predicate, $ideal\_solve(G)$ which succeeds if a goal can be solved using idealised clauses in a way which maintains the connection to the source model prescribed by the tests. This is defined as follows:

$$ideal\_solve(G) \leftarrow solve(G, \{\}, I_f) \wedge connected(I_f) \qquad (78)$$

where $connected(I_f)$ succeeds if every test in I_f is satisfiable. This is defined as:

$$connected(I_f) \leftarrow \forall C.(C \in I_f \rightarrow C) \tag{79}$$

7 Conclusion

We have described a particular form of idealisation and have demonstrated that it can be used as a basis for constructing a variety of different forms of idealised model. We have also shown that the transformation which we use for this sort of idealisation can, in addition to the idealised model, construct tests which enable one form of consistency with the original model to be checked automatically. This form of reasoning is unusual. It isn't deductive, since the functions we introduce during idealisation are only loosely similar to the original ones. It isn't inductive, because our idealised description is not intended to "cover" all of the instances of the source model. It isn't abductive, since the idealised clauses aren't a direct logical consequence of the source clauses. It does have some similarities to analogical reasoning, since the purpose of idealisation is to build a model which is similar, but different in key respects, to a source model. However, we do not yet know how close idealisation is to analogy.

The particular form of idealisation described in this paper gives an unusual route to the description of formal models. It is based on the assumption that models directly describing interactions between individual objects can be (comparatively) easily described but may not always be practical for supporting the forms of automated inference necessary to "run" the models. Idealisation is the process used to convert these precise but impractical models into (differently) precise but practical models. In other words, models describing individual objects may be more useful for knowledge acquisition but idealised models may be more practical for inferring answers to modelling problems. Although part of idealisation can be based on automatic transformation, many of the decisions on how to restructure the original model must remain in the hands of the modeller. It would be interesting to explore how the mechanisms for automatic transformation could be used to guide these human decisions.

References

[Robertson 96a] D. Robertson. Distributed specification. In *Proceedings of the 12th Eurpoean Conference on Artificial Intelligence*, Budapest, Hungary, August 1996.

[Robertson 96b] D. Robertson. An empirical study of the lss specification toolkit in use. In *Proceedings of the 8th International Conference on Software Engineering and Knowledge Engineering, Nevada, USA*. Knowledge Systems Institute, Illinois, 1996. ISBN 0-9641699-3-2.

[Robertson *et al.* 91] D. Robertson, A. Bundy, R. Muetzelfeldt, M. Haggith, and M Uschold. *Eco-Logic: Logic-Based Approaches to Ecological Modelling.* MIT Press (Logic Programming Series), 1991. ISBN 0-262-18143-6.