Dealing with expected and unexpected obstacles

Fausto Giunchiglia
Istituto per la Ricerca Scientifica e Tecnologica (IRST)
Loc. Pantè di Povo - 38050 Trento, Italy
Tel.: +39 (461) 314-517 E-mail: fausto@irst.itc.it

Enrico Giunchiglia
Dipartimento di Informatica, Sistemistica e Telematica (DIST)
Università di Genova
Viale Causa, 13 - 16146 Genoa, Italy
Tel.: +39 (10) 353-2811 E-mail: enrico@frege.mrg.dist.unige.it

Tom Costello
Computer Science Department, Stanford University
Palo Alto, CA 94305-2140, USA
Tel.: +1 (415) 723-4202 E-mail: costello@cs.stanford.edu

Paolo Bouquet
Dipartimento di Filosofia, Università di Genova
Via Balbi, 4 - 16100 Genoa, Italy
Tel.: +39 (461) 88-2322 E-mail: bouquet@cs.unitn.it

Abstract

Generality and *locality* have been proposed as two crucial properties for systems formalizing common sense reasoning. The first is the ability of representing knowledge in a way that makes it usable in a wide class of circumstances. The second is the ability of using only a subset of the potentially available knowledge, namely the subset which is held to be relevant in a given circumstance. These two properties seem to be one the opposite of the other, since disregarding part of the available information (locality) may lead to a loss of generality. In this paper, we argue that this is not the case, and propose a general methodology for combining generality and locality. This methodology is essentially based on the notion of *context*. As a case study, we propose a formalization of the Glasgow-London-Moscow example and its mechanization in an interactive theorem prover, `GETFOL`.

1 Introduction. Generality and locality

In [15] McCarthy argues that one of the most serious problems in formalizing common sense is the lack of *generality*, namely the ability to represent knowledge in a way to make it usable in a sufficiently wide class of circumstances. In the same paper, McCarthy clarifies the point with a simple example: "Consider putting an axiom in a *common sense database* asserting that birds fly. Clearly, this axiom must be *qualified* in some way, since penguins, dead birds, and birds whose feet are encased in concrete cannot fly. A careful construction of the axiom might succeed in including the exceptions of penguins and dead birds, but clearly we can think up as many additional examples like birds with their feet encased in concrete as we like". Formalized nonmonotonic reasoning was proposed as an appropriate solution. Circumscription [13, 14], default logics [18], and other nonmonotonic approaches are meant to achieve generality along these lines. The idea is that generality can be increased by expressing the conjecture that the only exceptions that are to be reasoned about are those explicitly entailed by the globally available knowledge.

Nonmonotonic formalisms, however, do not capture an important aspect of common sense reasoning, namely *locality*. According to [6], "[common sense] reasoning is usually performed on a subset of the global knowledge base; we never consider *all we know* but only a very small subset of it". The theories that we use to reason about the world are always *partial* (*i.e.* do not containxs all we know about the world) and *approximate* (*i.e.* describe the world at some level of detail; this notion of approximation is similar to that of [12]). This suggests a different approach to the examples in the above paragraph. The fact that birds fly needs a lot of qualifications, but we do not consider all of them every time we reason about birds (in this paper, we will not discuss this problem – called the qualification problem – in its

extreme consequences. For instance, we do not discuss the fact that most assertions cannot be fully qualified. [1, 2] discuss this aspect in some detail). In a normal conversation, the fact that birds fly can be left almost unqualified; in Antarctic, it is important to consider the fact that birds might be (or usually are) penguins; and in a scientific book about birds all (known) qualifications should be taken into account. In other words, in different contexts different sets of qualifications are usually considered. This idea is at the basis of a lot of work in artificial intelligence and cognitive science, some examples being the multi-context approach [6], mental spaces [4] and partioned representations [3].

Apparently, generality and locality put forward opposite needs. Increasing generality makes locality decrease, and vice versa. Generality requires that we take advantage of all the available information in order to qualify a fact. Locality requires that only a subset of the available information is used, and that different subsets are used in different contexts. The contrast between generality and locality is very clearly identified in the example described below, called the Glasgow–London–Moscow (GLM) example, elaborated in detail in this paper. The GLM example was proposed by McCarthy in the unpublished note *Overcoming an Unexpected Obstacle* (1991):

> "You are planning a trip from Glasgow to Moscow via London. You would like to build the plan maybe without having to think of all the details, *i.e.* by working in a fairly approximated theory. For instance you are willing to consider the fact that you must have a ticket in order to get on a plane but not the fact that the flight could be cancelled. However you want to be able to revise your plan if an expected obstacle arises (*e.g.* you do not have the ticket because you have lost it) and more particularly if an unexpected obstacle arises (*e.g.* the flight is cancelled)".

The requirement of building a plan "without having to think of all the details" goes in the direction of locality. However, the fact that "you want to be able to revise your plan if an expected obstacle arises [...] and more particularly if an unexpected obstacle arises" requires a high degree of generality, since we must be able to detect that an event is an obstacle to the plan.

In this paper we propose to combine generality and locality by exploiting the following idea: generality has to be thought of as a property of a knowledge base (KB) in its globality, whereas locality is a property of the theories in which problems are reasoned about and possibly solved (in the following these theories will be called *working spaces*, WS for short). The KB contains all the knowledge that potentially can be used to solve a problem. For instance, it may contain the information that losing the ticket, or the cancellation of the flight, make the plan fail. A WS contains only a subset of this information, namely the subset that is used to solve a given problem. In particular, in our formalization (*i*) a WS is formulated as a result of a problem formulation by extracting part of the information contained in the KB, and (*ii*) a WS can be reformulated (*e.g.* extracting new information from the KB) if the problem is reformulated (*e.g.* more information becomes available).

The paper is structured as follows. In section 2 we present a solution of the GLM example using circumscription. In section 3 we show that this solution misses the requirement of locality. In section 4 we propose a new solution, using circumscription and a multi-context framework, which solves the problem of locality by performing theory formulation. In section 5 we show that this solution solves also the problem of generality. In section 6 we propose a revised final solution based on the idea of performing problem and theory reformulation. We conclude by describing some parts of the mechanization of the GLM example inside `GETFOL` [7], an interactive theorem prover running on top of a complete reimplementation of the `FOL` system

[19]. The goal of this section is to give the flavour of how contexts are implemented and used in GETFOL.

2 A solution using circumscription

The solution of the GLM example presented in this section uses circumscription. We use the axiomatization of actions originally proposed in [10] and extended to consider unexpected events like losing the ticket (the extension is proposed by McCarthy in the unpublished note mentioned in section 1). Notice that our solution is largely independent of the particular formalization of action we chose; we introduce this formalization only in order to show how the context mechanism integrates with standard tools for mononotonic or nonmonotonic reasoning. As a convention, all free variables must be read as universally quantified. Besides axioms for unique names, we have the axioms given below.

$$succeeds(a, s) \equiv \forall f.\ (precond(f, a) \supset holds(f, s)) \tag{1}$$

This tells us that an action succeeds in a situation s iff all its preconditions hold in that situation. Actually, it's a definition of the predicate *succeeds*.

$$noninertial(a, f, s) \equiv$$

$$succeeds(a, s) \wedge (causes(a, f, true) \vee causes(a, f, false)) \tag{2}$$

This is the definition of noninertial.

$$succeeds(a, s) \wedge causes(a, f, true) \supset holds(f, result(a, s)) \tag{3}$$

$$succeeds(a, s) \wedge causes(a, f, false) \supset \neg holds(f, result(a, s)) \tag{4}$$

If an action succeeds in a situation and it is one that causes a fluent to hold (not to hold), then the fluent holds (does not hold) in the situation that results from the performance of

the action.

$$\neg noninertial(a, f, s) \wedge holds(f, s) \supset holds(f, result(a, s)) \tag{5}$$

This tells us that unless an action affects a fluent, then the fluent holds after the action provided it held before the action.

$$occurs(e, s) \supset outcome(s) = outcome(result(e, s)) \tag{6}$$

$$\forall e.\ \neg occurs(e, s) \supset outcome(s) = s \tag{7}$$

The last two axioms give the effects of events different from actions.

$$rr(e, s) = outcome(result(e, s)) \tag{8}$$

This is an abbreviation for the situation that results from an action after all the events that occur after it have happened.

We are interested in flying from Glasgow to Moscow. The following axioms describe what we know in general about flying.

$$causes(fly(x, y), at(y), true) \tag{9}$$

$$precond(at(x), fly(x, y)) \tag{10}$$

Axiom (9) says that flying from x to y causes being at y. Axiom (10) says that you must be at x to fly from there to y. We also have some commonsense knowledge about tickets.

$$precond(hasticket, fly(x, y)) \tag{11}$$

$$causes(loseticket, hasticket, false) \tag{12}$$

$$causes(buyticket, hasticket, true) \tag{13}$$

Axiom (11) says that you must have a ticket to get on a plane. Axioms (12) and (13) describe the effects of losing and buying a ticket.

Finally we know that flights must exist for us to fly.

$$precond(existsflight(x,y), fly(x,y)) \tag{14}$$

$$causes(cancflight(x,y), existsflight(x,y), false) \tag{15}$$

Axiom (15) describes the effects of cancelling a flight.

What has been described so far is general commonsense knowledge. We also suppose that the following facts hold of the initial situation.

$$holds(at(Glasgow), S_0) \tag{16}$$

$$holds(hasticket, S_0) \tag{17}$$

$$holds(existsflight(Glasgow, London), S_0) \tag{18}$$

$$holds(existsflight(London, Moscow), S_0) \tag{19}$$

That is, the traveller is at Glasgow, he has a ticket and the necessary flights exist.

Then we can apply the following circumscription:

$$circum(Facts; occurs > causes, precond; holds) \tag{20}$$

and show

$$holds(at(Moscow), rr(fly(London, Moscow), rr(fly(Glasgow, London), S_0))). \tag{21}$$

However if we lose the ticket in London

$$occurs(loseticket, result(fly(Glasgow, London), S_0)) \tag{22}$$

then the circumscription in (20) does not allow us to obtain plan (21). However, in this case we can build the following new plan:

$$holds(at\ Moscow,$$

$$rr(fly(London, Moscow),$$

$$rr(buyticket,$$

$$rr(fly(Glasgow, London), S_0)))). \quad (23)$$

3 The problem of locality

With circumscription one starts with a theory of the world and throws away, by minimizing the extension of certain predicates, a set of hypotheses which would prevent the monotonic deduction of the answer. This formalizes a process of "jumping to conclusions", which is extensively used in common sense reasoning [11]. Since what is known is formalized as a unique logical theory, the minimization process is applied all over the KB. This seems counterintuitive and different from what happens in common sense reasoning. People never seem to consider everything they know. Only a small subset of the known facts is explicitly taken into consideration. People jump to some conclusions not only because they explicitly discharge some unknown qualification (as it happens with circumscription) but also because they do not consider all they know about the problem they are trying to solve. The above formalization lacks locality.

Performing local reasoning gives a lot of advantages. For instance, it limits the explosion of the search space. In nonmonotonic reasoning, it avoids the minimization of the reasoner's entire theory of the world. Instead, in the solution to the GLM example given in section 2 we are forced to consider all the possible things which may happen — that we have to be where the plane leaves, that we must have a ticket and that the flight must exist. And this is done any time a new flight is taken.

Local reasoning can be very naturally formalized using contexts as defined in [6]. We take

a context c to be a logical theory presented as an axiomatic formal system, *i.e.* as a triple $\langle L, A, \Delta \rangle$. L is the language of c, A is the set of axioms of c, Δ is the set of inference rules, also called the deductive machinery, of c [6, 5]. (From now on we use the terms "context" and "theory" as synonyms).

A key issue is the extent to which reasoning is local. The language, the set of facts and the inference rules often depend on the problem. People seem able to build theories which contain enough facts and often not many more than needed. The contents of this theory depend on many factors, *e.g.* on the confidence that people have in their ability of solving the problem, on their current attitude (more or less oriented towards details), on the time they have to solve the problem (taking into account more details requires more time) and so on. A general solution to the GLM example must provide a mechanism which, given a particular formulation of the problem, allows for the building of a theory that contains the needed facts. It must provide a function from problems to theories. A theory of this formulation, if stated declaratively in a control theory, will contain statements about (the contents of) contexts. It must have contexts reified as objects, *i.e.* a set of individual constants naming contexts, as part of its language.

4 A solution using contexts and circumscription

We pose the problem in a problem context PC. We take a problem to be a pair "set of assumptions, goal". The set of assumptions may be empty. Assumptions describe contingent knowledge about particular situations. In the chosen formalism, they can have only one of the following three forms: $holds(F, S)$, $\neg holds(F, S)$ and $occurs(E, S)$, where F is a fluent, E is an event and S is some given situation. In the GLM example possible assumptions are

axioms (16), (17), (18), (19) and (22). A goal is a formula of the form $goal(f)$. The statement of the goal to achieve is

$$goal(at(Moscow)). \tag{24}$$

We have also to say what it means to solve a problem. This is done by introducing in PC a predicate on situations, $pbsolves$, which is true iff the goal is true at that situation:

$$pbsolves(s) \equiv holds(f, s) \wedge goal(f). \tag{25}$$

We solve the problem in a work space context WS. The formulation of WS is described in a control context, C. Intuitively, C is a sort of metatheory of all the other contexts and contains statements about what holds inside them. This is achieved by defining a distinguished predicate whose holding in C depends on what holds in the other contexts. In the solution of the GLM example we will use ist as defined in [16], where the intuition is that $ist(c, w)$ is true in C if the formula (denoted by) w is true in the context (denoted by) c. The intended meaning of ist is captured via a set of distinguished inference rules, called *bridge rules* [6], which allow inferring a fact in a context just because some facts are derivable in other contexts. Two examples of bridge rules are the reflection rules [8]:

$$\frac{\langle ist(c1, w), c2\rangle}{\langle w, c1\rangle} \qquad \frac{\langle w, c1\rangle}{\langle ist(c1, w), c2\rangle}$$

The rule on the left is called reflection down, the other reflection up. Notationally, we write $\langle w, c\rangle$ to mean w and that w is a wff of the context c. Thus, the intuitive meaning of reflection down is that we can derive w in $c1$ any time we have derived $ist(c1, w)$ in $c2$. Reflection up has the obvious dual meaning. For the solution of the GLM example we need two schemas of reflection rules

$$\frac{\langle ist(c, w), C\rangle}{\langle w, c\rangle} \qquad \frac{\langle w, c\rangle}{\langle ist(c, w), C\rangle}$$

with the restriction that c is different from C and that the premises of both reflection up and reflection down are theorems.

Using ist we can define in C a lift operation

$$lift(c1, c2) \equiv \forall w.\ (ist(c1, w) \supset ist(c2, w)), \tag{26}$$

where $lift(c1, c2)$ states that anything which holds in $c1$ holds also in $c2$. Whenever $lift(c1, c2)$ can be proved, the reflection rules allow us to derive in $c2$ any fact that can be proved in $c1$. Thus the fact that WS contains the statement of the problem is formalized by the following statement in C:

$$lift(PC, WS). \tag{27}$$

We want to assert the answer in PC

$$ist(WS, pbsolves(s)) \supset ist(PC, pbsolves(s)). \tag{28}$$

(Notice that the names of the contexts involved, that is WS and PC, are constants of the language of C.)

WS also contains knowledge about the problem domain. This knowledge must be extracted from a common sense data base. In our solution, the KB consists of a number of different contexts, called KB contexts. Each KB context formalizes knowledge about a particular topic. Thus, for instance, in the GLM example we have a context *Action* which contains all the axioms that formalize actions (up to axiom (8)). The context *Flying* details what the action fly causes and what its preconditions are (axioms (9) and (10)). The context *Ticket* contains knowledge about tickets (axioms (11), (12) and (13)). The context *Flights* contains the axioms about the existence of flights (axioms (14) and (15)).

A key issue is how to decide from which KB contexts knowledge should extracted. If we use all the KB contexts we lose locality. We should consider all the KB contexts only if we

want to take into account all the details. However, if we want to find a quick solution in a very approximate theory, we will use as few KB contexts as possible. The quest for locality puts an upper bound on the quantity of information considered. However we also have a lower bound, that is, we must have enough information to build a plan. The statements in C of what must be lifted into WS are a set of relevance claims which link the language of the problem in PC to the contexts that make up WS. Depending on whether the problem context has a goal to plan for, *i.e.* if the problem context contains *goal*, we need the axioms in *Action*. If we are planning about being *at* a place we will need the axioms from *Flying*. If the problem formulation mentions *hasticket* or *loseticket*, we need the axioms in *Ticket*. If the fluent *existsflight* is concerned, we need the axioms in *Flights*.

$$In(goal, PC) \supset lift(Action, WS) \tag{29}$$

$$In(at, PC) \supset lift(Flying, WS) \tag{30}$$

$$In(loseticket, PC) \vee In(hasticket, PC) \supset lift(Ticket, WS) \tag{31}$$

$$In(existflight, PC) \supset lift(Flights, WS) \tag{32}$$

In is a decidable predicate which is true if its first argument is a symbol belonging to the language of the context occuring as its second argument.

To recapitulate, the reasoning goes as follows. In the first step, the problem is posed in PC. This causes a second step of theory formulation where WS is constructed out of PC and a subset of the KB contexts. Then in the third step (monotonic or nonmonotonic) reasoning is performed inside WS. In the fourth and last step the solution is asserted in PC.

As an example, let us consider a first formulation of the GLM example where we have, in PC, axiom (16) and goal (24). This means that we formulate a very approximate theory where the only assumption is that the traveller is at Glasgow. By axioms (29), (30) we bring

into WS all the facts in $Action$ and $Flying$. Then, after circumscribing in WS, we can prove (21) and therefore

$$pbsolves(rr(fly(London, Moscow), rr(fly(Glasgow, London), S_0))) \tag{33}$$

The last step is exporting (33) into PC.

Let us now consider a less approximate formulation of the GLM example where we also assume that we have a ticket, *i.e.* we have in PC also axiom (17). This allows us to bring into WS the facts from the contexts $Action$, $Flying$ and $Ticket$ (because of (29), (30) and (31)). Then, after circumscribing in WS, we can prove (21) and therefore (33). However, if we include in PC also axiom (22), we can no longer infer (21) but rather (23) from which we can conclude

$$pbsolves(rr(fly(London, Moscow), rr(buyticket, rr(fly(Glasgow, London), S_0)))) \tag{34}$$

Notice that we have two formulations of the problem posed in the informal description of the GLM example. The first leads to a WS which is more local than the second. Both are more local than the formulation proposed in section 2, as neither of them considers the fact that, in order to fly, there must exist a flight. More in general, starting from a given commonsense database, we can formulate several more or less local theories (WS contexts). Such theories can be partially ordered depending on their contents. That is, for any two WS contexts $ws1, ws2$, we say that $ws1$ is *more local* than $ws2$, in formulae $ws1 \sqsubseteq_l ws2$, if and only if the set of facts lifted in $ws1$ is a subset of the set of facts lifted in $ws2$. For a given problem, we have a most local context which contains the minimal set of KB contexts needed to solve the problem itself. In the GLM problem the most local context contains only the facts from $Action$ and $Flying$. For a given database, we have a least local context which contains the facts of all the KB contexts. In the GLM example this context would contain the facts from

Action, *Flying*, *Ticket* and *Flights*. Figure 1 gives a graphical representation of some of the possible formulations of the GLM example.

5 The problem of generality

In section 4 we have shown how the problem of lack of locality, highlighted in section 3, can be solved by performing *problem formulation* and *theory formulation*. But, what if the traveller is not wearing pants? Neither the theory built in section 2 nor the work spaces built in section 4 are applicable. Something which is relevant to the solution of the problem has been factored away because of some hidden assumptions. For any given WS formulation we can classify the "obstacles" which make our reasoning fail in two classes. In the first class there are the obstacles which violates a precondition that had been previously minimized. One example is the event of losing the ticket, both in the formulation of the GLM example of section 2 and in the second formulation of section 4. In the second class there are those obstacles which cannot be dealt with by the current formulation. One example is not wearing the pants in all the formulations of the GLM example discussed before. Another is losing the ticket in the first formulation of the GLM example of section 4 (notice that the reasoning performed within the second formulation of section 4 is impossible in the first formulation). Obstacles belonging to the first class are called *expected obstacles*, those belonging to the second, *unexpected obstacles*.

Expected obstacles are taken care by circumscription. However, this is not the case with unexpected obstacles. There are two kinds of unexpected obstacles. There are cases in which we really do not know how to deal with the obstacle. For instance, a person who is not an expert of airplanes will not know how to deal with the plane breaking down. This kind of

unexpected obstacle cannot be dealt with by pure reasoning only. The database does not contain the knowledge which allows us to take care of the event. This case is not considered in the rest of the paper. In the case we deal with, an unexpected obstacle arises because, though we know how to deal with it, we have not loaded the relevant information from the common sense database. This case can be solved by pure reasoning. The intuition is to simulate people's commonsense reasoning and to perform a step of theory reformulation which constructs a new theory where the unexpected obstacle is "transformed" into an expected obstacle.

Providing the machinery which deals with unexpected obstacles allows us to solve the problem of generality: given a problem, a theory for solving it can be formulated and reformulated by extracting the relevant information from the commonsense knowledge base. This can be achieved by using the machinery proposed in section 4. In fact one can see problem reformulation as doing problem formulation again. For instance the second example of formulation given in section 4 can be described as a reformulation of the first example. This is not a coincidence. No matter how general it is, (commonsense) reasoning is implicitly local as it relies on a set of unstated assumptions which define its scope. Vice versa, no matter how local it is, (commonsense) reasoning is general enough to deal with all the situations explicitly taken into consideration. *The more local a formulation is the less general it is and vice versa.*

An immediate consequence is that the partial order on locality can be used to define a dual partial order on generality, that is, for any two contexts $c1, c2$ we have $c1 \sqsubseteq_l c2$ if and only if $c2 \sqsubseteq_g c1$. There is therefore a least general context which is also the most local and, vice versa, a most general context which is also the least local. The most general context is obtained by loading all the contexts from the database. Notice that we accept the existence of a most general context. This seems to contradict McCarthy's statement that such a context

cannot exist [15]. However this is not the case. In fact we have defined a most general context for a fixed database. However, when reasoning, in order to deal with unexpected obstacles, people continuously reformulate their WS. When they have loaded all the database they still can go to a more general database by acquiring new knowledge. The fact that this process of knowledge acquisition is not upper bounded is a consequence of the fact that situations are rich objects [17]. Any context or set of contexts will always fail to describe them completely. Figure 2 gives a graphical representation of the levels of generality and locality for the GLM example. The largest dashed circle represents an extension of the work space context, due to an unexpected obstacle, which requires an extension of the database (caused, for instance, by the unexpected obstacle of not wearing the pants).

6 A refined solution

The solution of performing theory reformulation by doing theory formulation again seems quite unsatisfactory. The drawback is that we are always forced to redo everything from scratch. This might be quite expensive computationally. Moreover it contradicts our intuition that people do problem reformulation simply by "adjusting" their work space to the new set of facts. In the following, we show how a simple process of "adjustement" can be formalized and applied to the GLM example. The intuition is to put problem formulation in a wider perspective which allows us to see it as a special case of problem reformulation. This brings in the new issue of having to deal with more than one problem at the same time, *e.g.* the original and the reformulated. A problem can no longer be directly posed in PC and solved in WS as there will be a problem context and a work space context for any problem. Not only must the work space (re)formulation be formalized in C but also the problem (re)formulation.

This means that, instead of posing a problem directly in a problem context, we state in C that we have a problem that we want to solve. Technically this requires that we be able, in C, to talk about problems, that is, that we have problems reified as objects [9].

6.1 Problem formulation

Let us start by considering problem formulation. A problem P^* is formulated in C by stating explicitly its assumptions and goal. This is done by asserting an axiom of the form

$$P^* = pbform(Setofas, Goal) \tag{35}$$

where $P^*, Setofas, Goal$ are constants naming a problem, a set of assumptions and a goal respectively, $pbform$ is a function which, given a set of assumptions and a goal, returns a problem. (Notice that we need to reify also sets of assumptions and goals.)

The second step is to build the problem context and to assert assumptions and goal. To do this, we need an injective function pbc which, given a problem, returns its problem context. The injectivity of pbc guarantees us that two different problems get associated to two different problem contexts. We also need some functions for manipulating problems, *i.e.* a function $pbgoal$ which, given a problem, returns its goal and a function $pbas$ which, given a problem, returns the list of its assumptions. The following general axioms of C allow us to build the contents of the problem context

$$ist(pbc(p), pbgoal(p)) \tag{36}$$

$$w \in pbas(p) \supset ist(pbc(p), w). \tag{37}$$

We also want to know in C the solution to the problem. This can be obtained by adding the axiom

$$solves(s, p) \equiv ist(pbc(p), pbsolves(s)) \tag{38}$$

in C and by having axiom (25) in any problem context.

The third step is to construct the work space context. We introduce a new function constant $wrkc$ which, applied to a problem context, returns the relative work space context. $wrkc$ is injective. Its contents are extracted from the database contexts exactly as in section 4. However we must now take into account the fact that we have multiple problem and work space contexts. Axioms (27), (28) become

$$lift(pbc(p), wrkc(pbc(p))) \tag{39}$$

$$ist(wrkc(pbc(p)), pbsolves(s)) \supset ist(pbc(p), pbsolves(s)) \tag{40}$$

while axioms (29) - (32) become

$$In(goal, pbc(p)) \supset \ lift(Action, wrkc(pbc(p))) \tag{41}$$

$$In(at, pbc(p)) \supset \ lift(Flying, wrkc(pbc(p))) \tag{42}$$

$$In(loseticket, pbc(p)) \vee In(hasticket, pbc(p)) \supset lift(Ticket, wrkc(pbc(p))) \tag{43}$$

$$In(existsflight, pbc(p)) \supset lift(Flights, wrkc(pbc(p))) \tag{44}$$

After these three steps we have contructed a problem plus its problem context and its WS context. Step two corresponds to step one in the solution in section 4 while step three is identical to step two. The fourth and fifth steps are the same as the third and fourth steps described in section 4, respectively. The sixth step is to prove in C that the given problem is solved.

As an example, let us suppose that we want to solve the most approximate formulation of the GLM example, that is the first formulation of section 4. We do this by posing in C a problem $P1$ defined by the following axiom

$$P1 = pbform(\{holds(at\ Glasgow, S_0)\}, goal(at\ Moscow)) \tag{45}$$

that is a particular instance of the general axiom (35). The second step requires that we build a problem context for $P1$ and lift into it the problem description. We can suppose that *pbc* is axiomatized such that we have in C

$$PC1 = pbc(P1) \tag{46}$$

where, therefore, $PC1$ is the name of the problem context of $P1$. Then, by axioms (36), (37) we add goal (24) and assumption (16) to $PC1$. We need to construct the work space for $P1$. Analogously to above, we suppose that $wrkc$ is axiomatized such that we have in C

$$WS1 = wrkc(PC1). \tag{47}$$

where, therefore, $WS1$ is the name of the workspace context of $PC1$. Then, analogously to what described in section 4, $WS1$ contains the facts lifted from *Action* and *Flying* (by axioms (41) and (42)). We can therefore derive (33) in WS (by axiom (39)) which is then lifted into PC (by axiom (40)). This allows us to prove

$$solves(rr(fly(London, Moscow), rr(fly(Glasgow, London), S_0)), P1) \tag{48}$$

in C (by reflection up and axiom (38)).

6.2 Problem reformulation

Let us now consider problem reformulation. We consider the case where the problem reformulation is done by adding a set of assumptions to the problem statement. Let us define a new function $pbreform$ that, given a set of assumptions and a problem, returns a new reformulated problem in which the set of assumptions has been added to the assumptions of the old problem. Then we can state that a problem P^{**} is a reformulation of a problem P^* with an axiom of the form

$$P^{**} = pbreform(Setofas, P^{*}) \tag{49}$$

where P^{*}, P^{**} are constants naming problems and $Setofas$ is a constant naming a set of assumptions. We write $PBreform(p2, p1)$ to mean that $p2$ is a reformulation of $p1$. We have therefore

$$PBreform(pbreform(setofas, p), p) \tag{50}$$

for any problem p and set of assumptions $setofas$. We are now able to state that two contexts are problem contexts of two problems, one reformulation of the other

$$PBreform(p2, p1) \supset PCreform(pbc(p2), pbc(p1))) \tag{51}$$

and the same for the work space contexts

$$PCreform(pc2, pc1) \supset WSreform(wrkc(pc2), wrkc(pc1)) \tag{52}$$

where $PCreform(pc2, pc1)$ ($WSreform(ws2, ws1)$) means that $pc2$ ($ws2$) is a problem (work space) context which is a reformulation of $pc1$ ($ws1$).

We can now state that the problem context of the reformulated problem will contain all the facts of the original problem plus the new set of assumptions

$$PCreform(pc2, pc1) \supset lift(pc1, pc2) \tag{53}$$

$$PBreform(p2, p1) \wedge w \in newas(p2) \supset ist(pbc(p2), w) \tag{54}$$

where $newas(p2)$ returns the new assumptions of the reformulated problem.

The last step is the construction of the new work space. In this context we lift

1. all the facts of its problem context;

2. all the facts of the old workspace context that have been lifted in it by relevance axioms;

3. all the new facts from the KB contexts which must be brought in because of the new assumption.

The first set of facts is lifted by axiom (39). The third set is lifted using the relevance axioms (axioms (41) - (44)). The lifting of the second set is done by the following axiom

$$WSreform(ws2, ws1) \supset \forall kc.\ (lift(kc, ws1) \supset lift(kc, ws2)) \tag{55}$$

where kc ranges over KB contexts.

As an example, let us see how the second formulation of section 4 can be built as a reformulation of the first formulation, as constructed in subsection 6.1. We consider the fact that we have a ticket and that we lose it in London, *i.e.* we add (17), (22) to the assumptions of $P1$. We obtain a problem $P2$ defined by the following axiom

$$P2 = pbreform(\{occurs(loseticket, result(fly(Glasgow, London), S_0)),$$
$$holds(hasticket, S_0)\}, P1). \tag{56}$$

that is a particular instance of axiom (49). As for the case of problem formulation, we can suppose that pbc and $wrkc$ are axiomatized to have in C

$$PC2 = pbc(P2) \tag{57}$$

$$WS2 = wkrc(PC2), \tag{58}$$

that is, $PC2$ and $WS2$ are, respectively, the problem and work space contexts of $P2$. By axiom (53) we lift in $PC2$ all the contents of $PC1$, by axiom (54) we lift in $PC2$ the new assumptions, that is (17) and (22). For what concerns $WS2$, we lift everything lifted in $WS1$ (because of axioms (55), (39)) plus the context $Ticket$ (because of axiom (43)). As a

consequence, we can no longer infer (21), but we can infer (23) and therefore (34) is lifted back into $PC2$. With the last step we prove

$$solves(rr(fly(London, Moscow), rr(buyticket, rr(fly(Glasgow, London), S_0))), P2) \quad (59)$$

in C (by reflection up and axiom (38)).

The problem reformulation described here is pretty obvious and almost as costly as redoing problem formulation from scratch. However the machinery set up is completely general. More interesting forms of problem reformulation can be implemented by suitably modifying axioms (53), (54), (55) or by adding more complicated axioms. For instance it is easy to think of strategies where the theorems of $WS1$ are lifted into $WS2$ (but a lot of care must be taken in making sure that nonmonotonic reasoning inside $WS1$ and work space reformulation interact correctly).

7 Mechanization of the GLM example

We have mechanized the GLM example in `GETFOL`. The goal of this section is to give the flavour of how contexts are implemented and used inside `GETFOL`.

`GETFOL` allows the user to create any number of contexts and to connect them, via bridge rules, into any number of multicontext systems (MC systems) [6]. Intuitively an MC system defines a set of contexts plus the bridge rules that connect them. It is a useful notion which allows us to categorize functionally contexts and the interactions among them. New contexts and MC systems can be created at any moment and added to the existing ones. For the solution of the example we need only one MC system, which we call `GLM`. Thus, as soon `GETFOL` is started, we can type the following set of commands.

```
NAMECONTEXT C;

MAKECONTEXT FLYING FLIGHTS TICKET;

MAKECONTEXT WS1;

MAKEMCONTEXT GLM C FLYING FLIGHTS TICKET WS1;
```

`MAKECONTEXT` creates an empty context with name its argument. `MAKEMCONTEXT` creates an MC system, with name its first argument, which contains any number of contexts, all those whose names come after the name of the MC system. The meaning of `NAMECONTEXT` is slightly more subtle. When we work in `GETFOL`, we are always in a context, the *current context* [15, 6]. The first time `GETFOL` is entered, the current context (is empty and) has no name. `NAMECONTEXT` names the current context and it fails if it is already named. Above we have arbitrarily chosen `C` as the current context.

An MC system comes with a number of bridge rules. `GETFOL` has a repertoire of bridge rules schematic on various parameters, *e.g.* the contexts linked and the predicates used. Any time the user wants to use one, it must declare it and instantiate all its parameters. Bridge rules are implemented as `GETFOL` commands which can be declared and executed only from the context where their conclusion is asserted. For instance, in `C` we need a reflection up rule from the KB contexts.

```
newcommand rupKC IS RUP ist;
```

The result is that `rupKC` is installed as a new command executable from `C`. `rupKC` is an instance of the general bridge rule `RUP` (relection up) which uses `ist` as distinguished predicate.

Any newly created context is empty. We have to define all its components, *i.e.* language, axioms and deductive machinery. The default deductive machinery for reasoning inside a context is the extension of natural deduction defined in [7]. Let us consider some of the

declarations in C. We want to talk about contexts and to distinguish among, for instance, KB contexts, problem contexts and workspace contexts. GETFOL's sorted logic allows us to do this and also to impose a partial order on the various kinds of contexts.

```
DECLARE SORT ncontexts nknwcxts nprobcxts nworkcxts;
MOREGENERAL ncontexts <nknwcxts nprobcxts nworkcxts>;
```

The first command declares a set of sorts. The second states that its first argument is more general than all the sorts listed in the second argument. We also need names and variables.

```
DECLARE INDCONST ACTION FLIGHTS FLYING TICKET [nknwcxts];
DECLARE INDVAR c c1 c2 [ncontexts];
DECLARE INDVAR kc [nknwcxts];
DECLARE INDVAR pc pc1 pc2 [nprobcxts];
```

After the language (*i.e.* constants, variables, function and predicate symbols), the next step is the declaration of axioms. For instance we need axiom (26):

```
AXIOM CLIFT: forall c1 c2.
              (lift(c1,c2) iff forall w. (ist(w,c1) imp ist(w,c2)));
```

We assert (26) as an axiom of C with label CLIFT. As a more complete example, below is the full declaration of the context FLYING.

```
SWITCHCONTEXT FLYING;

DECLARE SORT fluents actions primitivefluents towns;
DECLARE INDVAR t t1 [towns];
```

```
DECLARE INDCONST true false [fluents];

DECLARE FUNCONST at 1;

SETFMAP at(towns) = primitivefluents;

DECLARE FUNCONST fly 2;

SETFMAP fly(towns,towns) = actions;

DECLARE PREDCONST precond 2;

DECLARE PREDCONST causes 3;


AXIOM FP1: forall t t1. precond(at(t),fly(t,t1));

AXIOM FC1: forall t t1. causes(fly(t,t1),at(t1),true);

AXIOM FC2: forall t t1. causes(fly(t,t1),at(t),false);


newcommand rdownFlying IS RDOWN C ist;
```

Various observations are in order. `SWITCHCONTEXT` changes the current context from `C` to `FLYING` [6]. It corresponds to the sequence of the two atomic operations of *leaving* `C` and *entering* `FLYING` (McCarthy discusses a similar idea of leaving and entering a context [15]. His notion of leaving corresponds, in our terminology, to an application of reflection up plus an application `SWITCHCONTEXT`. His notion of entering corresponds to an application of reflection down plus an application of `SWITCHCONTEXT`. Our notions of entering and exiting seem more flexible and more natural to use inside a mechanized reasoning system. For instance, there are cases when one wants to apply a reflection down without changing context and vice versa, there are cases when one wants to change context without applying any reflection rule (*e.g.* to probe a deduction inside that context)). We can be in only one context at the time. Any operation of a given context, *e.g.* declarations and local theorem proving, can only be done

from inside the context itself. Moreover, it can be done only when the context is the current context. This is why SWITCHCONTEXT is the first operation performed. We did not perform a switchcontext before as, because of the NAMECONTEXT command, we were already in C. The first argument of the commands DECLARE FUNCONST and DECLARE PREDCONST is the symbol being declared, the second is its arity. SETFMAP declares the sort mapping of a function symbol. Thus, for instance, fly takes two arguments of sort towns and produces a value of sort actions. The last command declares reflection down from C to FLYING, using ist as a distinguished predicate, as the command rdownFlying.

In order to illustrate how the GETFOL multicontext machinery works, we now give a short example of reasoning inside a context and among contexts. We describe the lifting of axiom FP1 from FLYING to the workspace context WS1. Let us suppose that we have inferred, in C, proof line 28 given below.

```
C@GLM:: SWITCHCONTEXT C;

....

28   lift(FLYING,WS1)
```

C@GLM, before SWITCHCONTEXT, is the GETFOL prompt (which distinguishes between input to and output from the system). The GETFOL prompt is programmable. C@GLM means that we are inside the MC system GLM and inside the context C. 28 is the label of the proof line. It can be used later to refer to it. From 28 and CLIFT we can build the following deduction:

```
C@GLM:: alle CLIFT FLYING WS1;

  116   lift(FLYING,WS1) iff forall w. (ist(FLYING,w) imp ist(WS1,w))


C@GLM:: iffe 116 1;
```

```
  117   lift(FLYING,WS1) imp forall w. (ist(FLYING,w) imp ist(WS1,w))

C@GLM:: impe 117 28;
  118   forall w. ist(FLYING,w) imp ist(WS1,w)

C@GLM:: alle 118 NFP1;
  119   ist(FLYING,NFP1) imp ist(WS1,NFP1)
```

NFP1 is in C the name of axiom FP1 (of FLYING), FLYING and WS1 are the names of the two contexts. iffe, alle and impe are the GETFOL names for iff elimination, forall elimination and implication elimination. Their arguments are the labels of proof lines. The next step is to reflect FP1 up in C.

```
C@GLM:: rupKC FLYING FP1;
 ctext-get: I changed context to: FLYING
 ctext-get: I changed context to: C
  120   ist(FLYING,NFP1)
```

The two lines after the command line are GETFOL output. They tell us that, in order to perform reflection up, the system enters the context FLYING and then it returns back to C. We can now derive ist(NFP1,WS1) by implication elimination.

```
C@GLM:: impe 119 120;
  121   ist(WS1,NFP1)
```

Finally we enter context WS1 and prove what we want by reflection down.

```
C@GLM:: SWITCHCONTEXT WS1;

 You are now using context: WS1

 You are switching to a proof with no name.


WS1@GLM:: rdownWS1 121;

 ctext-get: I changed context to: C

 ctext-get: I changed context to: WS1

  14   forall t t1. precond(at(t),fly(t,t1))
```

rdownWS1 is the reflection down rule between C and WS1. 121 is the label of the theorem of C on which we perform reflection down (see above in the trace). In WS1 the label of the lifted formula is 14 (and not, for instance, 122) as each context has its own local counter for labelling proof lines.

Finally, GETFOL can be asked to show proofs across contexts.

```
C@GLM:: SHOW MCPROOF C:28 - WS1:14;

  C:28   lift(FLYING,WS1)

  ...

  C:116   lift(FLYING,WS1) iff forall w. (ist(FLYING,w) imp ist(WS1,w))

  C:117   lift(FLYING,WS1) imp forall w. (ist(FLYING,w) imp ist(WS1,w))

  C:118   forall w. ist(FLYING,w) imp ist(WS1,w)

  C:119   ist(FLYING,NFP1) imp ist(WS1,NFP1)

  C:120   ist(FLYING,NFP1)

  C:121   ist(WS1,NFP1)

 WS:14    forall t t1. precond(at(t),fly(t,t1))
```

8 Conclusion

In the paper, we have shown that generality and locality, despite their apparent incompatibility, can be combined inside the same system. Our main idea can be summarized as follows. Generality is a property of the KB; locality is a property of the theories that are formulated in order to solve problems. The more information we lift from the KB into a WS, the less the WS is local; and *vice versa*. These ideas are formalized as a process of problem and theory formulation and reformulation in a multi-context framework.

Acknowledgement

John McCarthy has provided many motivations and suggestions. Vladimir Lifschitz has given very useful feedback. The discussions with Luciano Serafini have been stimulating. Alessandro Cimatti and Lorenzo Galvagni have helped in the mechanization of the GLM example in `GETFOL`.

References

[1] P. Bouquet and F. Giunchiglia. Reasoning about theory adequacy: A new solution to the qualification problem. *Fundamenta Informaticae*, 23(2–4):247–262, June,July,August 1995. Also IRST-Technical Report 9406-13, IRST, Trento, Italy.

[2] P. Bouquet and F. Giunchiglia. Reasoning about Theory Formulation and Reformulation. A New Solution to the Qualification Problem. In *Second World Conference on the Fundamentals of Artificial Intelligence (WOCFAI 95)*, 1995.

[3] J. Dinsmore. *Partitioned Representations.* Kluwer Academic Publishers, 1991.

[4] G. Fauconnier. *Mental Spaces: aspects of meaning construction in natural language.* MIT Press, 1985.

[5] F. Giunchiglia. Multilanguage systems. In *Proceedings of AAAI Spring Symposium on Logical Formalizations of Commonsense Reasoning*, 1991. Also IRST-Technical Report 9011-17, IRST, Trento, Italy.

[6] F. Giunchiglia. Contextual reasoning. *Epistemologia, special issue on I Linguaggi e le Macchine*, XVI:345–364, 1993. Short version in Proceedings IJCAI'93 Workshop on Using Knowledge in its Context, Chambery, France, 1993, pp. 39–49. Also IRST-Technical Report 9211-20, IRST, Trento, Italy.

[7] F. Giunchiglia. `GETFOL` Manual - `GETFOL` version 2.0. Technical Report 92-0010, DIST - University of Genoa, Genoa, Italy, March 1994.

[8] F. Giunchiglia and A. Smaill. Reflection in constructive and non-constructive automated reasoning. In H. Abramson and M. H. Rogers, editors, *Proc. of META-88, Workshop on Metaprogramming in Logic*, pages 123–145. MIT Press, 1988. Also IRST-Technical Report 8902-04 and DAI Research Paper 375, University of Edinburgh.

[9] F. Giunchiglia and R.W. Weyhrauch. A multi-context monotonic axiomatization of inessential non-monotonicity. In D. Nardi and P. Maes, editors, *Meta-level architectures and Reflection*, pages 271–285. North Holland, 1988. Also DIST Technical Report 9105-02, DIST, University of Genova, Italy.

[10] V. Lifschitz. Formal theories of action. In F. M. Brown, editor, *The frame problem in Artificial Intelligence*, pages 35–58. Morgan Kaufmann. Los Altos, CA, 1987.

[11] J. McCarthy. Epistemological Problems of Artificial Intelligence. In *Proc. of the 5th International Joint Conference on Artificial Intelligence*, pages 1038–1044, 1977. Also in V. Lifschitz (ed.), *Formalizing common sense: papers by John McCarthy*, Ablex Publ., 1990, pp. 77–92.

[12] J. McCarthy. Ascribing Mental Qualities to Machines. In M. Ringle, editor, *Philosophical Perspectives in Artificial Intelligence*, pages 161–195. Humanities Press, 1979. Also in V. Lifschitz (ed.), *Formalizing common sense: papers by John McCarthy*, Ablex Publ., 1990, pp. 93–118.

[13] J. McCarthy. Circumscription - A Form of Non-monotonic Reasoning. *Artificial Intelligence*, 13(1,2):27–39, 1980. Also in V. Lifschitz (ed.), *Formalizing common sense: papers by John McCarthy*, Ablex Publ., 1990, pp. 142–157.

[14] J. McCarthy. Applications of Circumscription to Formalizing Common-Sense Knowledge. *Artificial Intelligence*, 28(1):71–87, 1986. Also in V. Lifschitz (ed.), *Formalizing common sense: papers by John McCarthy*, Ablex Publ., 1990, pp. 198–225.

[15] J. McCarthy. Generality in Artificial Intelligence. *Communications of ACM*, 30(12):1030–1035, 1987. Also in V. Lifschitz (ed.), *Formalizing common sense: papers by John McCarthy*, Ablex Publ., 1990, pp. 226–236.

[16] J. McCarthy. Notes on Formalizing Context. In *Proc. of the 13th International Joint Conference on Artificial Intelligence*, pages 555–560, Chambery, France, 1993.

[17] J. McCarthy and P. Hayes. Some Philosophical Problems from the Standpoint of Artificial Intelligence. In B. Meltzer and D. Michie, editors, *Machine Intelligence 4*, pages 463–502. Edinburgh University Press, 1969. Also in V. Lifschitz (ed.), *Formalizing common sense: papers by John McCarthy*, Ablex Publ., 1990, pp. 21–63.

[18] R. Reiter. A logic for default reasoning. *Artificial Intelligence*, 13:81–132, 1980.

[19] R.W. Weyhrauch. Prolegomena to a Theory of Mechanized Formal Reasoning. *Artificial Intelligence*, 13(1):133–176, 1980.

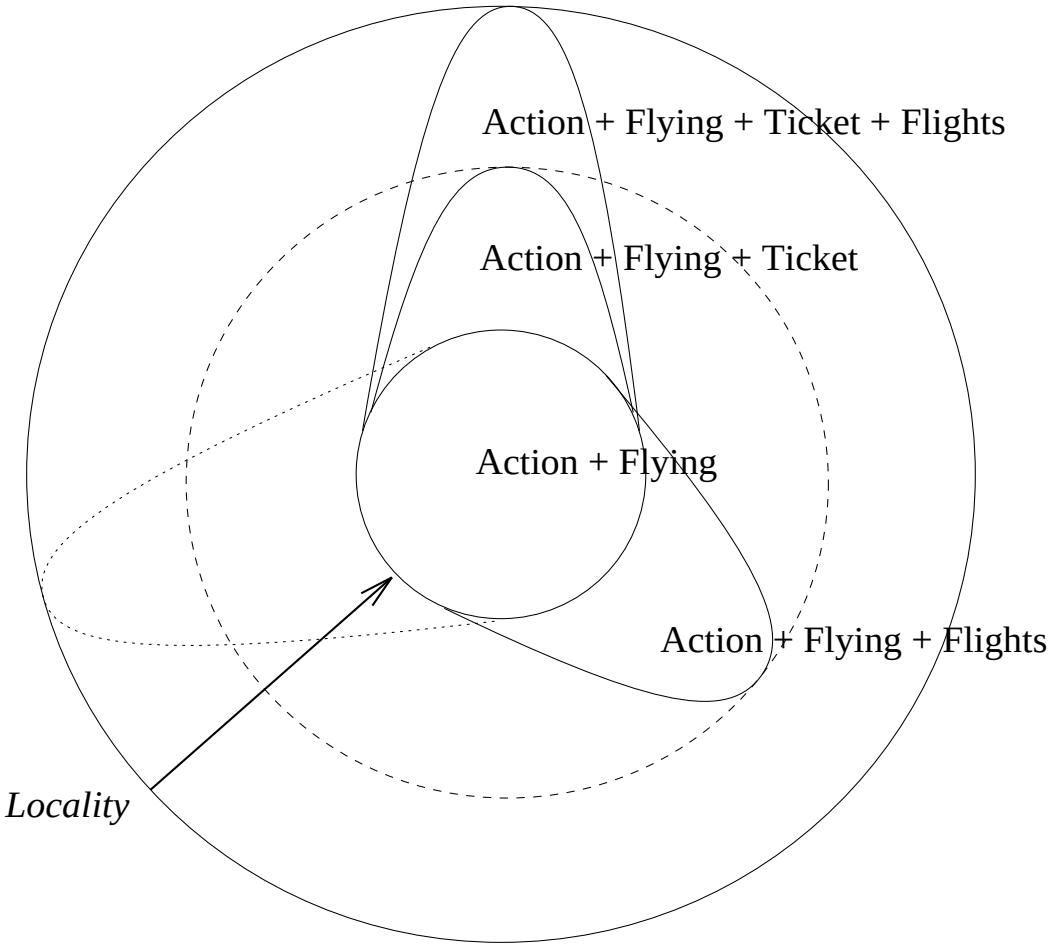

Figure 1: Levels of locality of the work space context

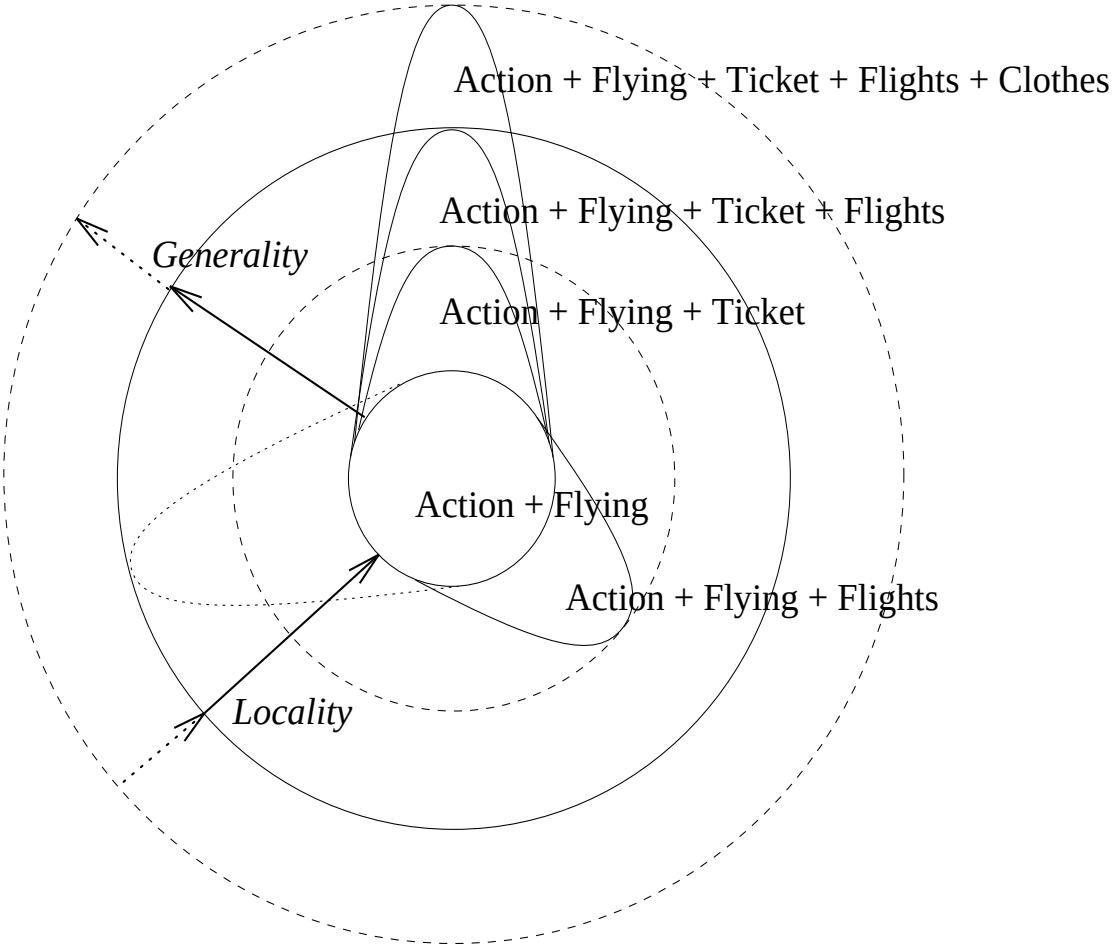

Figure 2: Levels of generality of the work space context