(2)

SECURITY CLASSIFICATION OF THIS PAGE

REPORT DOCUMENTATION PAGE

1a. REPORT SECURITY CLASSIFICATION: Unclassified	1b. RESTRICTIVE MARKINGS
2a. SECURITY CLASSIFICATION AUTHORITY	3. DISTRIBUTION/AVAILABILITY OF REPORT: Approved for public release; distribution unlimited.
2b. DECLASSIFICATION/DOWNGRADING SCHEDULE	
4. PERFORMING ORGANIZATION REPORT NUMBER(S): FINAL REPORT	5. MONITORING ORGANIZATION REPORT NUMBER(S): AFOSR-TR- 89-1868
6a. NAME OF PERFORMING ORGANIZATION: President and Fellows of Harvard College	6b. OFFICE SYMBOL (If applicable)
6c. ADDRESS (City, State and ZIP Code): Holyoke Center, Cambridge MA 02138	7a. NAME OF MONITORING ORGANIZATION: AFOSR
	7b. ADDRESS (City, State and ZIP Code): AFOSR, Bldg 10, Bolling Air Force Base, Washington DC 20332-6448
8a. NAME OF FUNDING/SPONSORING ORGANIZATION: AFOSR	8b. OFFICE SYMBOL (If applicable): NM
8c. ADDRESS (City, State and ZIP Code): same as 7b	9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER: AFOSR-89- 0088
	10. SOURCE OF FUNDING NOS.

PROGRAM ELEMENT NO.	PROJECT NO.	TASK NO.	WORK UNIT NO.
61102F	2304	A2	

11. TITLE (Include Security Classification): Collaborative Planning

12. PERSONAL AUTHOR(S): Barbara J. Grosz, Candace Sidner, Cecile Balkanski

13a. TYPE OF REPORT	13b. TIME COVERED	14. DATE OF REPORT (Yr., Mo., Day)	15. PAGE COUNT
FINAL	FROM 10/17/88 - 10/16/89	12/19/89	24

16. SUPPLEMENTARY NOTATION

17. COSATI CODES			18. SUBJECT TERMS (Continue on reverse if necessary and identify by block number)
FIELD	GROUP	SUB. GR.	

19. ABSTRACT (Continue on reverse if necessary and identify by block number)

Our milestones for this pilot project were to collect additional interaction records of planning by two agents, to analyze the actions and action relations in the data, to refine definitions of action relations to represent adequately the relationships occuring in the data, and to prepare a report summarizing the major findings from the analysis and presenting the new action relationships. We analyzed data from the following three sources: a construction task, a group planning meeting, and a simulated human-computer problem-solving dialogue. A description of this data, the results of the analysis, and the proposed new action relations are described in this report. A paper describing these results (Balkanski, Grosz, and Sidner, "Action Relations in Collaborative Activity") will be submitted to AAAI-90.

20. DISTRIBUTION/AVAILABILITY OF ABSTRACT	21. ABSTRACT SECURITY CLASSIFICATION
UNCLASSIFIED/UNLIMITED ☒ SAME AS RPT. ☐ DTIC USERS ☐	Unclassified/unlimited

22a. NAME OF RESPONSIBLE INDIVIDUAL	22b. TELEPHONE NUMBER (Include Area Code)	22c. OFFICE SYMBOL
Dr A WAKSMAN	(202) 767-5027	NM

DD FORM 1473, 83 APR EDITION OF 1 JAN 73 IS OBSOLETE.

SECURITY CLASSIFICATION OF THIS PAGE

COLLABORATIVE PLANNING

Final Report to the Air Force Office of Scientific Research
Contract number: AFOSR-89-0088

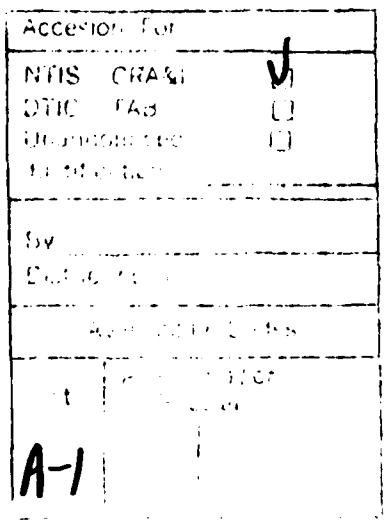
October 17, 1988 through October 16, 1989

Barbara J. Grosz, Principal Investigator

Project Participants:
Candace L. Sidner
Cecile Balkanski

Abstract

Our milestones for this pilot project were to collect additional interaction records of planning by two agents, to analyze the actions and action relations in the data, to refine definitions of action relations to represent adequately the relationships occuring in the data, and to prepare a report summarizing the major findings from the analysis and presenting the new action relationships. We analyzed data from the following three sources: a construction task, a group planning meeting, and a simulated human-computer problem-solving dialogue. A description of this data, the results of the analysis, and the proposed new action relations are described in this report. A paper describing these results (Balkanski, Grosz, and Sidner, "Action Relations in Collaborative Activity") will be submitted to AAAI-90.



Action Relations in Collaborative Activity

Cecile T. Balkanski, Candace L. Sidner, Barbara J. Grosz

1 Data Collection

Our initial description of a framework for modelling collaborative activity included definitions of a small set of action relations [GS89]. These relations provided a way of constructing more complex actions from simpler ones. For instance, the complex action of George and Sue lifting their dining room table could be defined in terms of a simultaneous generation relationship holding between George's lifting one end and Sue's lifting the other. We noticed, however, that to model collaboration required a more complete specification of a wider range of action relations that might participate in any collaborative activity. To expand the set of relations to a set that would span, or more completely span, the space of complex collaborative actions, we investigated data in three domains: a construction task, a group planning session, and a simulated human-computer problem-solving dialogue.

The construction task data were taken from a videotape of the construction of a porch swing glider by two agents; we will refer to this data as the "glidertape data". The group planning session data are from an audiotape of three individuals in a working group meeting who were exchanging information about the month-to-month maintenance of a communication network; we will refer to this data as the "meeting data." These two types of data differ in significant ways: the glidertape data includes many examples of physical actions, action sequences, and rationales for performing actions; it also demonstrates the interleaving of planning and acting (including, but not limited to, situations in which actions fail to achieve their intended effects) that we conjectured was central to collaborative activity. The meeting audiotape discourse centers on sharing information about past actions, and to a lesser extent, choosing actions to be performed at a subsequent time. The meeting data includes extremely useful discourse (communication) data, but much less data about the interrelationships among physical actions. The human-computer problem-solving dialogue data is a set of dialogues between two agents collaborating to manage a network on a daily basis. This data was constructed, but it was based on discussions with network managers as well as examples in the literature. This third source of data included two complex action-types not uncovered in the other two domains: iterations and conditionals.

We have decided to choose network management (at the daily level) as a domain of application. This domain has several characteristics which we found valuable for our research. It offers a natural way to investigate collaborations among agents, one

of which could be a computer. It requires action relationships of some complexity, thereby enabling us to test the action relations we define. The domain also offers a means of exploring the coordination of graphical and linguistic means of communicating between agents, a set of issues we are investigating under sponsorship of U S West Advanced Technologies.

In the following sections most of our examples draw on data from the glidertape, but we also use the network management dialogues. We also have incorporated other problematic examples from the commonsense everyday world when they best illustrate certain problems. The relations we report on here are sufficient for describing the relations in the glidertape, and they cover nearly all of the relations of the network management dialogues. We are still investigating the representation of tests and the actions they comprise. In a related report[GSL89], we present samples of the *recipes* (i.e.act-types and relations among them) for actions in the network management domain.

2 Action Relations

2.1 Introduction

Our aim in this section is to identify and define the relations that hold among actions performed by more than one agent. In previous work [GS89], we have examined generation and enablement relationships between actions performed by a single agent. For example, flipping the light switch generates turning the light on, and buying ingredients enables preparing dinner.

When multiple agents and multiple actions are considered, then new action relations come into play. For example, actions performed by different agents may be simultaneous or sequential. In our earlier work, we identified the following cases:

GEN-Simult:	*two agents lifting up a piano*
GEN-Conjoined:	*two agents setting a table*
GEN-Sequence:	*one agent turning a door knob then pulling on it to open a door*

The glider tape provides numerous examples of actions fitting into the categories given above, as well as examples showing the need to revise them and add new ones. In this section, we discuss the following categories:

Action relations: generation (GEN), enablement (ENABLE),
facilitation (FACIL), prevention (PREVENT), causality (CAUSE)
Complex action formation: sequence, conjunction, simultaneity, iteration

2.2 GENERATION

If action A generates action B then when some agent G performs action A, action B occurs at the same time, without agent G having to do anything more than perform action A [Pol86,Gol70]. For example, with a single agent and action, we have the example given earlier:

GEN(flip(G, light-switch), turn-on(G, light))

Or, to take an example from the glider tape, the action of one agent hammering each arm-assembly against the base leg assembly, to bring the two pieces tightly together, can be represented as follows:

GEN(hit(G, arm-assembly, hammer), secure(G_1,arm-assembly,leg-spreader))

This representation, as well as the following ones, will be formalized in the next chapter. For the time being, it suffices to read GEN(A, B), as A generates B.

With multiple actions and agents, we need to specify how different actions A_1, A_2, ..., A_n, performed by different agents, can combine to produce a composite action B.

2.2.1 GEN-Simultaneous

As the name suggests, simultaneous actions are those which occur over identical intervals of time. Each individual action will have the proper generation relationship with the overall action to be achieved if and only if the other actions are performed at the same time [GS89].

For example, to attach the arm-assemblies to the base of the glider, the agents perform the simultaneous actions of holding the arm assembly in the vertical position and attaching it to two glider straps mounted to the base of the glider. This can be expressed as follows:

GEN-Simult((hold(John, right-arm-assembly,vertical-position) &
attach(Paul,right-arm-assembly, right-strap)).
attach(John&Paul, right-arm-assembly, glider-base))

The number of actions and the number of agents need not be the same. To attach the base spreader to the leg-assemblies, for example, one agent performs two actions simultaneously to achieve the desired result:

GEN-Simult((hold(John, left-hand, left-end-spreader) &
insert(John, right-hand, lag-screw, left-end-spreader)),
attach(John, left-end-spreader, left-leg-assembly))

2.2.2 GEN-Sequence

Actions related by a GEN-Sequence relation are those that, when taken together and performed in a particular order, achieve a desired result. [GS89].

For instance, the recipe for making an angle-food cake can be described (at a very high level of description) as "preheat the oven, make the batter, bake the batter". The first two actions are related by a GEN-Sequence relation: the preheating action must be done before the making the batter action (else the batter will fall while the oven is heating up), but the former does not enable the latter. The enable relationship (described later in this chapter) that does hold in this recipe is between the sequence of the two first actions and the third action, baking the cake. We can express these relationships as follows:

GEN-Sequence((preheat(G, oven) $\wedge$ make(G, batter)), ready-to-bake(batter))
ENABLE(ready-to-bake(batter), bake(G, batter))
GEN(bake(G, batter), make(G, angel-food-cake))

If two or more actions must be done in sequence, then there is a reason for this ordering. This is why GEN-Sequence and Enable relationships are sometimes hard to distinguish. In the previous example, the oven needs to be turned on before making the batter so that it is at the right temperature when the batter is ready. This is a time constraint, but the two actions are not related by enablement. (One could put the batter in a cold oven, and bake it, but the result would not be the expected one.)

However, in the glider tape, there are very few instances of actions related by GEN-Sequence relationships. This may be a consequence of the type of task being performed, namely a construction task, where each building step typically enables the next. Consider for example the following (high level) instructions:

To build the glider:
1. build the base
2. attach the arm assemblies
3. attach the seat frame
4. attach the back rest

Although this set of instructions involves a "sequence" of actions, describing these actions by a GEN-Sequence relationship would be incorrect: here, each individual action involves putting together particular pieces that need to be together in order for the next action to be performable. We therefore have a set of actions related by enablement.

Another reason for which actions may have be to sequential is resource allocation. For example, the last step in building the base of the glider consists in tightening all bolts and screws. So far, the agents have been working simultaneously, on of the right side and the other on the left side of the structure being constructed. But in this step, they work in sequence because they have to share the (single) nutdriver they have available. Here, Paul's tightening the screws on the left side does not enable John to tighten the screws on the right side, therefore, we have a GEN-Sequence, namely:

GEN-Sequence((tighten(Paul, screws, left-side), tighten(John, screws, right-side)),
tighten(Paul&John, screws, glider-base))

2.2.3 GEN-Conjoined

Conjoined actions are those that, when taken together, but performed in any order, achieve a desired result. In such cases, the time interval for each individual action must fall within some time interval during which all actions are performed; these individual actions may overlap, but there is no need for simultaneity, nor for a predetermined order of performance of the individual actions [GS89].

For example, at the beginning of the glider tape, after opening the box and emptying its contents, the agents clear the floor in order to have enough space for building the glider. This clearing action is achieved by pushing aside the box and the pillows which came with it. These two actions, when taken together, achieve the desired result, as described in the following representation:

GEN-Conjoined((push-aside(G1, pillows) & push-aside(G2, box)),
clear(G1&G2, floor))

Another example occurs when the agents are building the base of the glider. One of the first subtasks consists in placing dowels in the front and rear rails. This is accomplished by Paul putting them in the right ends, and John in the left ends; there is no need for them to perform their actions during exactly the same time intervals:

GEN-Conjoined((place(Paul, dowels, right-end-of-rails) &
place(John, dowels, left-end-of-rails)),
prepare(Paul&John, dowels, rails))

2.2.4 GEN-Iteration

Another type of action relation which was clearly illustrated in the glider tape is that of iteration. In this case, one or more agents perform the same action over and over, until, for instance, every object of a given set has been manipulated by that action. This repetition happens, for instance, when the agents are checking all the screws. To accomplish this task, they have to take every screw, one at a time, and match it on the list of parts in the directions sheet. We can express this as follows:

GEN-Iteration((find(screw$_i$) & match(screw$_i$, picture)), check(set-of-screws))

An action to be iterated can itself be a complex action. For example, to tighten all the bolts, the agents have to perform the tightening action on each bolt. This action is a complex action which can be described by a GEN-Simult relationship: to tighten a bolt, John and Paul must, simultaneously and respectively, hold the nut and turn the bolt. This is expressed as follows:

GEN-Simult((hold(John, nut) & turn(Paul, bolt)), tighten(John&Paul, bolt)),
GEN-Iteration(tighten(John&Paul, bolt$_i$), tighten(John&Paul, set-of-bolts))

Although all examples from the glider tape involve iteration over a set of objects, this is not the only type of iterative actions. One other type of iteration involves repeating an action until a given condition is met, as in the following example:

Try the number until you get through!

Here, unlike the other examples, it is not known ahead of time how many times the action will be iterated. The next chapter will discuss these cases.

2.3 ENABLE

If action A enables action B, then the performance of action A brings about conditions that allow the subsequent performance of B. Unlike generation of B from A, if A enables B, B is not done as a result of doing action A [Pol86,Gol70]. For example, with a single agent and action, we have the example given in the introduction:

ENABLE(buy(G1, ingredients), prepare(G2, dinner))

Or to take an example from the glider tape, the agents need to locate the parts in order to use them; i.e.:

ENABLE(locate(G, part), use(G, part))

Mental actions, as well as physical actions, participate in action relationships. Understanding directions, for instance, is a mental action that enables the agents to subsequently follow them; i.e.:

ENABLE(understand(G, directions), follow(G, directions))

As with generation, the actions involved in an enable relationship may themselves be complex actions. For example, at the beginning of the glider tape, the agents discover the warranty as they are checking all the parts. John then comments "If there's any problem, I have the bill". This situation illustrates John's knowledge of disjoined actions in an enable relationship: showing the warranty or showing the bill enables replacing the faulty-part; i.e.:

GEN-Disjoined((show(G, warranty) or show(G, bill)),
show(G, proof-of-purchase))

ENABLE(show(G, proof-of-purchase), replace(G, faulty-part))

Another example occurs when the agents attach the seat-frame. They first place the seat-frame in the right position, acting simultaneously, then, acting conjointly, they each place bolts, washers and nuts in their side of the glider. Since the first action enables the second, we have:

GEN-Simult((position(John, right-side-seat-frame) ∧
position(Paul, left-side-sear-frame)),
position(John&Paul, seat-frame))

ENABLE(position(John&Paul, seat-frame),
attach(John, bolts/washers/nuts, right-side-seat-frame))
ENABLE(position(John&Paul, seat-frame),
attach(Paul, bolts/washers/nuts, left-side-seat-frame))

GEN-Conjoined((attach(John, bolts/washers/nuts, right-side-seat-frame) ∧
attach(Paul, bolts/washers/nuts, left-side-seat-frame)),
attach(John&Paul, seat-frame, glider))

2.4 FACILITATE

The tapes also provides several situations where one action, A, facilitates, or makes easier, another, B. In these cases, doing A brings about a condition which makes the (subsequent) doing of B easier. Contrary to enablement relationships, in facilitate

relationships, A does not have to be done first in order for B to be performable; but if B is done without A having been done before, then some other difficulty may be encountered. We will denote this relationship FACIL.

For example, at the very beginning of the glider tape, the agents realize that reading the directions will help them in their subsequent task of building the glider. Using the new relationship FACIL, we can express this as follows:

FACIL(read(G, directions), build(G, glider))

The following utterance provides another example:

You can tell by the fact that it's a little easier if we turn the whole thing upside down.

FACIL(turn(G1&G2, base, upside-down), attach(G1&G2, rail, screws))

It is interesting to notice that FACILITATE relations are often explicitly marked in the utterance, e.g. by "it's probably good if", and "it's a little easier if". However, the border between FACIL and ENABLE is not always evident from the agents' utterances. In the following utterance, when the agents are discussing the best way to fasten a bolt, an ENABLE, rather than a FACIL, relation seems appropriate, despite the fact that the agent says "it's easier to":

John: *I guess it's easier to hold it and then turn it the other side.*
Paul: *Yeah, once you it get on, then you can turn it.*

ENABLE(hold(G1, nut, left-hand), tighten(G1, screw, right-hand))
GENERATE(tighten(G1, screw), secure(G1, rail, leg-assembly))

2.5 PREVENT relation

Another type of action relationship often encounted is that of one action A preventing another action B. For example, the child's interruptions bring about a situation that prevents the men from building the glider:

PREVENT(interrupt(child, agents), build(agents, glider))

The PREVENT relation may be connected to goals of maintenance. For the construction process to be successful, for example, the agents have to be able to work in an undisturbed environment. The child's interruptions, however, prevent this goal from being maintained, and this in turn prevents the agents from focusing their attention on the construction process. This can be expressed as follows:

PREVENT(interruptions, undisturbed-environment)
ENABLE(undisturbed-environment, build(agents, glider))

A similar example occurs in the following situation, where the agents need to maintain the wood from being stressed:

John: *They say here "do not over tighten bolts, tighten only enough to bring the heads firmly against the wood" otherwise it starts to eat in the wood I assume.*

PREVENT(tighten(bolts, too-much),wood(unstressed))

2.6 CAUSE

One last relation to mention is that of causality. For example, the agents' first attempt at building the base of the glider fails after the structure collapses as a result of hitting it with a hammer. The failure then leads them to decide to coordinate their actions. These action relationships can be described as follows:

CAUSE(hit(G, hammer, leg-assembly), detach(leg-assembly, rail))
CAUSE(*detach(leg-assembly,rail)*, *intend*(G1&G2, coordinate))

Other instances of causal relations occur when the agents realize that they are running out of washers:

CAUSE(question(G, put(G, washer, right-arm-assembly)), count(G, washers))
ENABLE(count(G, washers), realize(G, error))
CAUSE(realize(G, error), disassembly(G, glider))

Causality will be further discussed in the last section of this report.

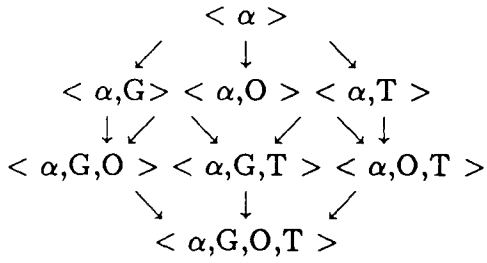
3 Action Relation Definitions

The definitions that follow are presented in the formalization developed during this contract. That formalization suggested several simplifications which we are now investigating. The modifications that result will be (?) part of current effort.

3.1 Action representation

Pollack [Pol86] adopts a 3-way terminological distinction: an *act-type* is a type of action (e.g. calling the hospital), an *action* or *act* is a triple of act-type, agent, and time (e.g. Joan's calling the hospital yesterday afternoon), and an *occurrence* is a realization of some action (e.g. Joan called the hospital yesterday afternoon). [cf p.41]

On the other hand, we have suggested a 2-way distinction: *occurence* versus *act-type*. An act-type is a type of action, namely a singleton, pair, triple or quadruple from the following lattice of specialization, where α is the act-type descriptor, G the agent performing α, T the interval of time over which α occurs, and O the other objects which play a role in the act-type:



(i) $< \alpha >$ read
$< \alpha$, G> John's reading
$< \alpha$, O, T> read a novel today
$< \alpha$, G, O, T> John's reading a novel today

In this representation, an act-type can be simply named, or increasingly specified by other relevant arguments. Pollack, on the other hand, typically represents objects by "attaching" them to the act-type, as in (ii), below, although she does mention an alternative representation, where objects are arguments to the act-type, as in (iii):

(ii) <talk-to-Jane, Sue, now>
(iii) <talk-to(Jane), Sue, now>

The notation in (iii) seems justified given that the notion of agency, unlike that of object, deserves a separate treatment and should therefore be kept separate from the act-type. Furthermore, when there are several objects, the notation in (i) becomes clumsy if they are kept separate from the act type; compare for example <give(painting, friend), Sue, now> with <give, Sue, painting, friend, now>.

Notice that this difference in representation carries over to the OCCURS predicate to which objects must also be added:

(ii) OCCURS(act-type(object), agent, time)
(iii) OCCURS(act-type, agent, objects, time)

We are currently exploring this issue. In the present report, we use the representations (ii) and (iii).

3.2 Generation

3.2.1 GEN

Example 1 *Flipping the light switch to turn the light on.*

For single actions, we adopt Pollack's definitions. She distinguishes CGEN from GEN: CGEN is defined only in terms of act-types, α and β, and the "generation-enabling conditions" C, whereas GEN is defined in terms of actions, i.e. the triple act-type, agent and time, $< \alpha$, G, t>. Her definitions are as follows [cf p.52]:

CGEN(α, β, C) $\Longleftrightarrow$
(i) [$\forall$ G1, t1 [[HOLDS(C, t1) $\wedge$ OCCURS(α, G1, t1)]
$\Longrightarrow$ OCCURS(β, G1, t1)] $\wedge$
(ii) $\exists$ G2, t2 [OCCURS(α, G2, t2) $\wedge$ $\neg$OCCURS(β, G2, t2)] $\wedge$
(iii) $\exists$ G3, t3 [HOLDS(C, t3) $\wedge$ $\neg$OCCURS(β, G3, t3)]]

GEN(α,β, **G, t**) $\Longleftrightarrow$ $\exists$ C (CGEN(α, β, C) $\wedge$ HOLDS(C, t))

The previous example is therefore represented as follows (the condition given is simply illustrative):

CGEN(flip(G, light-switch), turn-on(G, light), on(electricity))

3.2.2 GEN-Simult

Example 2 *Lucie and Chris holding both ends of the piano to lift it up.*

In all following definitions, we make use of the following notation: α is an act-type, α_i runs from $i = 1$ to $i = n$, G is an agent, G_j runs from $j = 1$ to $j = k$, $n \geq k$, G_{α_i} is the agent performing act-type α_i, t is a time interval, t_{α_i} is the interval of time over which act type α_i occurs, and t_e is the interval of time covering all t_{α_i} intervals.

CGEN-Simult($\alpha_1 \& \ldots \& \alpha_n$, β, C) $\Longleftrightarrow$

(i) $[\forall\, G_j, t_e\, [[(\bigwedge_{i=1}^{n} \text{OCCURS}(\alpha_i, G_{\alpha_i}, t_e)) \wedge \text{HOLDS}(C, t_e)] \Longrightarrow \text{OCCURS}(\beta, G_1 \& \ldots \& G_k, t_e)] \wedge$

(ii) $\exists\, G_j, t_e\, [(\bigwedge_{i=1}^{n} \text{OCCURS}(\alpha_i, G_{\alpha_i}, t_e)) \wedge \neg\text{OCCURS}(\beta, G_1 \& \ldots \& G_k, t_e)] \wedge$

(iii) $\exists\, G_j, t_e\, [\text{HOLDS}(C, t_e) \wedge \neg\text{OCCURS}(\beta, G_1 \& \ldots \& G_k, t_e)]]$

GEN-Simult($\alpha_1 \& \ldots \& \alpha_n$, β, $G_1 \& \ldots \& G_k$, t) $\Longleftrightarrow$
$\exists$ C [CGEN-Simult($\alpha_1 \& \ldots \& \alpha_n$, β, C) $\wedge$ HOLDS(C, t)]

Given this definition, and the one presented later for ENABLE, enablement relations between the individual actions involved in a GEN-Simult relationships are excluded. This happens because of time constraints: if α_1 and α_2 are in a GEN-Simult relationship, then they occur at the same time; if, on the other hand, they are in an ENABLE relationship, then α_1 occurs before α_2; therefore, α_1 and α_2 cannot be both in a GEN-Simult and an ENABLE relationship.

Notice that the number of agents participating in the simultaneous action is left unspecified. This will be the case of all definitions. Here, for example, there could be even just one agent: e.g. breaking eggs into a pan with one hand, and turning the sauce in that pan with the other.

Example 2 can then be represented as:

CGEN-Simult((lift(Lucie, left-end) & lift(Chris, right-end)),
lift(Lucie&Chris, piano), not-too-heavy(piano))

3.2.3 GEN-sequence

Example 3 *To make angel-food cake, preheat the oven, make the batter, then bake it.*

CGEN-sequence($\alpha_1 \& \ldots \& \alpha_n$, β, C) $\Longleftrightarrow$

(i) $[\forall\, G_j, t_e\, [[(\bigwedge_{i=1}^{n} \text{OCCURS}(\alpha_i, G_{\alpha_i}, t_{\alpha_i})) \wedge \text{HOLDS}(C, t_e)] \Longrightarrow \text{OCCURS}(\beta, G_1 \& \ldots \& G_k, t_e)] \wedge$

(ii) $\exists\, G_j, t_e\, [(\bigwedge_{i=1}^{n} \text{OCCURS}(\alpha_i, G_{\alpha_i}, t_{\alpha_i})) \wedge \neg\text{OCCURS}(\beta, G_1 \& \ldots \& G_k, t_e)] \wedge$

(iii) $\exists$ G_j,t_e [HOLDS(C, t_e) $\wedge$ $\neg$OCCURS(β, G_1&...&G_k, t_e)]]

where DURING(t_i, t_e) and BEFORE(t_i, t_{i+1}),
and $\neg(\exists\ \alpha_j, \alpha_k$ ENABLE(α_j, α_k))

GEN-Sequence(α_1&...&α_n, β, G_1&...&G_k, t) $\Longleftrightarrow$
$\exists$ C [CGEN-Sequence(α_1&...&α_n, β, C) $\wedge$ HOLDS(C, t)]

Notice that this definition is exactly the same as that of GEN-Conjoined except for the additional constraint that the time interval t_i occur before t_{i+1}, which is not required for GEN-Conjoined.

3.2.4 GEN-Conjoined

Example 4 *Sally and Peter setting the table.*

CGEN-Conjoined(α_1&...&α_n, β, C) $\Longleftrightarrow$

(i) $[\forall\ G_j,t_e\ [[(\ \bigwedge_{i=1}^{n}\ \text{OCCURS}(\alpha_i, G_{\alpha_i}, t_{\alpha_i})) \wedge \text{HOLDS}(C, t_e)]$
$\Longrightarrow$ OCCURS(β, G_1&...&G_k, t_e)] $\wedge$

(ii) $\exists\ G_j,t_e\ [(\ \bigwedge_{i=1}^{n}\ \text{OCCURS}(\alpha_i, G_{\alpha_i}, t_{\alpha_i})) \wedge \neg\text{OCCURS}(\beta, G_1\&\ldots\&G_k, t_e)] \wedge$

(iii) $\exists$ G_j,t_e [HOLDS(C, t_e) $\wedge$ $\neg$OCCURS(β, G_1&...&G_k, t_e)]]

where DURING(t_i, t_e) and $\neg(\exists\ \alpha_j, \alpha_k$ ENABLE(α_j, α_k))

GEN-Conjoined(α_1&...&α_n, β, G_1&...&G_k, t) $\Longleftrightarrow$
$\exists$ C [CGEN-Conjoined(α_1&...&α_n, β, C) $\wedge$ HOLDS(C, t)]

Notice that this definition is exactly the same as that of GEN-Simult except for the fact that the individual actions $< \alpha_i, G_{\alpha_i}, t_{\alpha_i} >$ are not necessarily performed at the same time.

The previous example is then:

CGEN-Conjoined([put(Sally, dishes) $\wedge$ put(Peter, silverware) $\wedge$ put(Sally, glasses)],
set-table(Sally&Peter), [on-table(table-cloth) $\wedge$ ...])

3.2.5 GEN-Iteration

a. Iteration over a set of objects

The first type of iteration we will consider involves iterating over a set of objects, as in the example below. We describe it by the action relationship GEN-Iteration:

Example 5 *Tightening a set of bolts by tightening each one of them.*

$$\textbf{CGEN-Iteration}(\alpha, \textbf{obj}_1\&\ldots\&\textbf{obj}_n, \beta, \textbf{C}) \text{ (version 1)} \iff$$

$$\text{(i) } [\forall\, G_j,t_e\ [[(\ \bigwedge_{i=1}^{n}\ \text{OCCURS}(\alpha(\text{obj}_i), G_{\alpha_i}, t_{\alpha_i})) \wedge \text{HOLDS}(C, t_e)] \Longrightarrow \text{OCCURS}(\beta, G_1\&\ldots\&G_k, t_e)] \wedge$$

$$\text{(ii) } \exists\, G_j,t_e\ [(\ \bigwedge_{i=1}^{n}\ \text{OCCURS}(\alpha(\text{obj}_i), G_{\alpha_i}, t_{\alpha_i}) \wedge \neg\text{OCCURS}(\beta, G_1\&\ldots\&G_k, t_e)] \wedge$$

$$\text{(iii) } \exists\, G_j,t_e\ [\text{HOLDS}(C, t_e) \wedge \neg\text{OCCURS}(\beta, G_1\&\ldots\&G_k, t_e)]]$$

where $\text{DURING}(t_{\alpha_i}, t_e)$ and $\neg(\exists\ \alpha_j, \alpha_k\ \text{ENABLE}(\alpha_j, \alpha_k))$

$$\textbf{GEN-Iteration}(\alpha, \textbf{obj}_1\&\ldots\&\textbf{obj}_n, \beta, \textbf{G}_1\&\ldots\&\textbf{G}_k, \textbf{t}) \iff$$
$$\exists\ C\ \text{CGEN-Iteration}(\alpha, \text{obj}_1\&\ldots\&\text{obj}_n, \beta, C) \wedge \text{HOLDS}(C, t)]$$

This definition is very similar to that of GEN-Conjoined. What differs here is that instead of describing a set of different act-types being performed, this definition describes the same act-type, α, being performed over a set of objects.

Given this definition, the previous example is then:

$$\text{CGEN-Iteration}(\text{tighten}, \text{bolt}_1\&\ldots\&\text{bolt}_n, \text{secure(structure)}, \text{available(tools)})$$

b. Iteration over agents and times

Just as iterations occur over objects and act-types, they also occur over agents and/or times. For example:

Example 6 *Every member of the team did the exercise.*
Take this medicine every four hours.

These cases can be handled by generalizing GEN-Iteration in the following way:

CGEN-Iteration(α, $\text{iter}_1\&\ldots\&\text{iter}_n$, β, C) (version 2) $\Longleftrightarrow$

if iter_i is an object, let $\text{OCCURS}(\alpha_i, G_{\alpha_i}, t_{\alpha_i}))] = \text{OCCURS}(\alpha(\text{obj}_i), G_{\alpha_i}, t_{\alpha_i}))]$

if iter_i is an agent, let $\text{OCCURS}(\alpha_i, G_{\alpha_i}, t_{\alpha_i}))] = \text{OCCURS}(\alpha, G_i, t_{G_i}))]$

if iter_i is a time, let $\text{OCCURS}(\alpha_i, G_{\alpha_i}, t_{\alpha_i}))] = \text{OCCURS}(\alpha, G, t_i))]$

(i) $[\forall\, G_j, t_e\ [[(\ \bigwedge_{i=1}^{n}\ \text{OCCURS}(\alpha_i, G_{\alpha_i}, t_{\alpha_i})) \wedge \text{HOLDS}(C, t_e)] \Longrightarrow \text{OCCURS}(\beta, G_1\&\ldots\&G_k, t_e)] \wedge$

(ii) $\exists\, G_j, t_e\ [(\ \bigwedge_{i=1}^{n}\ \text{OCCURS}(\alpha_i, G_{\alpha_i}, t_{\alpha_i}) \wedge \neg\text{OCCURS}(\beta, G_1\&\ldots\&G_k, t_e)] \wedge$

(iii) $\exists\, G_j, t_e\ [\text{HOLDS}(C, t_e) \wedge \neg\text{OCCURS}(\beta, G_1\&\ldots\&G_k, t_e)]]$

where $\text{DURING}(t_{\alpha_i}, t_e)$ and $\neg(\exists\, \alpha_j, \alpha_k\ \text{ENABLE}(\alpha_j, \alpha_k))$

GEN-Iteration(α, $\text{iter}_1\&\ldots\&\text{iter}_n$, β, $G_1\&\ldots\&G_k$, t) $\Longleftrightarrow$

$\exists$ C CGEN-Iteration(α, $\text{iter}_1\&\ldots\&\text{iter}_n$, β, C) $\wedge$ HOLDS(C, t)]

c. Iteration over multiple sets of objects

The definition given above for GEN-Iteration is in fact a special case of a more general definition which allows more than one set of objects to be iterated over. Consider for instance the following example:

Example 7 *Alex put a nail in every board.*

In this situation, both a set of nails and a set of boards are iterated over. Given its current definition, GEN-Iteration does not handle this case. We therefore introduce the term *object collection* to refer to the sets of objects that are to be iterated over. These sets typically correspond to different thematic roles of the verb expressing the given act-type. In the above example, we have an object collection consisting of all nails (i.e. filling the direct object position) and all boards (i.e. filling the indirect object position). Interation can also involve objects filling other thematic roles, such as instrument or destination for example.

The following revised definition will handle these more complex cases of iteration. It is exactly the same as the previous ones, except that obj_i, the ith object being iterated over, is replaced by $obj\text{-}coll_i$, the ith collection of objects being iterated over. When only one set of objects will be iterated over, as is the case most of the time, then $obj\text{-}coll_i$ will consist of a single object obj_i.

CGEN-Iteration(α, $\text{iter}_1\&\ldots\&\text{iter}_n$, β, C) (version 3) $\Longleftrightarrow$

if $iter_i$ is a collection of objects,
  let $OCCURS(\alpha_i, G_{\alpha_i}, t_{\alpha_i}))] = OCCURS(\alpha(\text{obj-coll}_i), G_{\alpha_i}, t_{\alpha_i}))]$
if $iter_i$ is an agent, let $OCCURS(\alpha_i, G_{\alpha_i}, t_{\alpha_i}))] = OCCURS(\alpha, G_i, t_i))]$
if $iter_i$ is a time, let $OCCURS(\alpha_i, G_{\alpha_i}, t_{\alpha_i}))] = OCCURS(\alpha, G, t_i))]$

(i) $[\forall\, G_j, t_e\ [[(\ \bigwedge_{i=1}^{n}\ OCCURS(\alpha_i, G_{\alpha_i}, t_{\alpha_i})) \wedge HOLDS(C, t_e)] \Longrightarrow OCCURS(\beta, G_1 \& \ldots \& G_k, t_e)] \wedge$

(ii) $\exists\, G_j, t_e\ [(\ \bigwedge_{i=1}^{n}\ OCCURS(\alpha_i, G_{\alpha_i}, t_{\alpha_i}) \wedge \neg OCCURS(\beta, G_1 \& \ldots \& G_k, t_e)] \wedge$

(iii) $\exists\, G_j, t_e\ [HOLDS(C, t_e) \wedge \neg OCCURS(\beta, G_1 \& \ldots \& G_k, t_e)]]$

where $DURING(t_{\alpha_i}, t_e)$ and $\neg(\exists\, \alpha_j, \alpha_k\ ENABLE(\alpha_j, \alpha_k))$

GEN-Iteration$(\alpha, \text{iter}_1 \& \ldots \& \text{iter}_n, \beta, G_1 \& \ldots \& G_k, t) \Longleftrightarrow$
  $\exists\, C$ CGEN-Iteration$(\alpha, \text{iter}_1 \& \ldots \& \text{iter}_n, \beta, C) \wedge HOLDS(C, t)]$

where $OCCURS(\alpha(\text{obj-coll}_i), G_{\alpha_i}, t_{\alpha_i}) = OCCURS(\alpha(\text{obj}_{i_1}, \ldots, \text{obj}_{i_k}), G_{\alpha_i}, t_{\alpha_i})$
  with each obj_i being an argument of the act-type α,
and $DURING(t_{\alpha_i}, t_e)$,
and $\neg(\exists\, \alpha_j, \alpha_k\ ENABLE(\alpha_j, \alpha_k))$

d. Scope of iteration

The definition we have been refining is still unsatisfactory. Consider for example the following task:

Example 8 *Fill all boxes and bring them to the truck.*

There are two ways of performing this task: one gives wider scope to the iteration over objects and the other to the sequence of actions to be performed; in other words, we have the two following possibilities:

(i) Fill all boxes, then bring them to the truck.
(ii) Fill and bring down one box after the next.

We would like to express both these possibilities in the formalism developped so far. Since the two act-types involved are in a GEN-Sequence relationship (although it is more appropriate to fill the boxes before bringing them to the truck, the former

action does not enable the latter action in any way), we can do so with the following relationships:

(i) CGEN-Iteration(fill(G, box), box_1&...&box_n, fill(G, set-of-boxes), available(set-of-boxes))
CGEN-Iteration(bring-down(G, box), box_1&...&box_n, bring-down(G, set-of-boxes), not-too-heavy(set-of-boxes))
CGEN-Sequence((fill(set-of-boxes) & bring-down(set-of-boxes)), ready-to-load(truck), available(truck))

(ii) CGEN-Sequence((fill(G, box) & bring-down(G, box)), prepare-for-loading(G, box), available(box))
CGEN-Iteration(prepare-for-loading(G, box), box_1&...&box_n, ready-to-load(truck), available(truck))

But suppose now that the actions involved are the following:

Example 9 *Bring down all boxes and load them into the truck.*

This task can again be accomplished in two different ways; but here, the formalism does not allow us to express the second possibility, namely when the set of two actions is iterated over the objects.

The problem is due to the fact that these actions are related by an ENABLE relationship instead of an GEN-Sequence relationship. As can be seen in the following representation, the relations as they are now defined do not provide a "handle" onto the complex action consisting of one action enabling another. The inability of the relation to adequately represent this case is a source of future work and we expect ot explore an alternative formulation for a future report.

(i) CGEN-Iteration(bring-down(G, box), box_1&...&box_n, bring-down(G, set-of-boxes), ready-to-go(set-of-boxes))
CGEN-Iteration(load-into-truck(G, box), box_1&...&box_n, load-into-truck(G, set-of-boxes), available(truck))
CGEN-Sequence((bring-down(set-of-boxes) & load-into-truck(set-of-boxes)), read-to-go(truck), available(truck))

(ii) ENABLE(bring-down(G, box), load-into-truck(G, box))
CGEN-Iteration(???, box_1&...&box_n, ready-to-load(truck), available(truck))

e. Iteration until/while

So far, iteration has been restricted to cases of the form "for $i = 1 \ldots n$, do X_i". Other cases, however, are of the form "do X while/until Y", as in the examples given below:

Example 10
(i) Keep trying the number, until the line is no longer busy.
(ii) While you're in France, go to good restaurants!

These examples involve actions that are to be repeated until, or while, a given condition is met. Typically, an "until" construct involves an action to be repeated while the condition is false and until it is true, whereas a "while" construct involves an action to be repeated while the given condition is true and until it is false. We define a new iteration relation, GEN-Iteration-test, to represent this case. It is defined as follows:

CGEN-Iteration-test(α, β, C, test) $\Longleftrightarrow$
(i) [$\forall$ G_j,T, with T max such that HOLDS(test, T)

$$[[\text{HOLDS(C, T)} \wedge (\bigwedge_{i=1}^{n} \text{OCCURS}(\alpha, G_\alpha, t_i))] \Longrightarrow \text{OCCURS}(\beta, G_1 \& \ldots \& G_k, T)] \wedge$$

(ii) $\exists$ G_j,T [$(\bigwedge_{i=1}^{n} \text{OCCURS}(\alpha, G_\alpha, t_i)) \wedge \neg\text{OCCURS}(\beta, G_1 \& \ldots \& G_k, T)] \wedge$
(iii) $\exists$ G_j,T [HOLDS(C, T) $\wedge$ $\neg$OCCURS(β, $G_1 \& \ldots \& G_k$, T)]]
(iv) $T_{max} = 0 \Longrightarrow \neg$(HOLDS(test,T))

where DURING(t_i, T) and $\neg(\exists\ \alpha_j, \alpha_k$ ENABLE(α_j, α_k))

GEN-Iteration-test($\alpha_1 \& \ldots \& \alpha_n$, β, $G_1 \& \ldots \& G_k$, t, test) $\Longleftrightarrow$
$\exists$ C [CGEN-Iteration-test($\alpha_1 \& \ldots \& \alpha_n$, β, C, test) $\wedge$ HOLDS(C, t)]

The "test" argument in this definition will be adapted to a true or false proposition depending on whether a "while" or "until" construct is involved. For the examples given above, the arguments to CGEN-Iteration-test would be as follows:

(i) CGEN-Iteration-test(dial(number), call(Sally), busy(line))
(ii) CGEN-Iteration-test(go-to(restaurants), enjoy(yourself), in(you, France))

There are still cases, however, that do not fit into this definition. These involve actions to be continued, rather than repeated. For example:

Example 11
(i) We ought to think about these instructions until we understand them.
(ii) Keep working while the system is up.

These examples do not involve a repetition of actions, but an action which occurs over the maximal interval of time during which a given condition holds, namely ¬understand(we, instructions) for (i) and up(system) for (ii). We call GEN-continue the relation which describes this type of situation. It is defined as follows:

CGEN-continue(α, β, C, test) $\Longleftrightarrow$
(i) [$\forall$ G_j,T, with T max such that HOLDS(test, T)
[[HOLDS(C, T) $\wedge$ OCCURS(α, G_α, t))]
$\Longrightarrow$ OCCURS(β, G_1&...&G_k, T)] $\wedge$
(ii) $\exists$ G_j,T [OCCURS(α, G_α, T)) $\wedge$ ¬OCCURS(β, G_1&...&G_k, T)] $\wedge$
(iii) $\exists$ G_j,T [HOLDS(C, T) $\wedge$ ¬OCCURS(β, G_1&...&G_k, T)]]
(iv) $T_{max} = 0 \Longrightarrow$ ¬(HOLDS(test,T))

where ¬($\exists$ α_j, α_k ENABLE(α_j, α_k))

GEN-continue(α, β, G_1&...&G_k, t, test) $\Longleftrightarrow$
$\exists$ C [CGEN-continue(α, β, C, test) $\wedge$ HOLDS(C, t)]

3.3 ENABLE

Example 12 *Annie recently installed cable TV to be able to see more movies.*

CENABLE(α, β, C) $\Longleftrightarrow$
[$\forall$ G1,t1,t2, $\exists\gamma$ [OCCURS(α, G1, t1) $\wedge$
GEN(α, (ACHIEVE (HOLDS (C, t1))), G1, t1) $\wedge$
CGEN (γ, β, C)]

ENABLE(α, β, G, t) $\Longleftrightarrow$ $\exists$ C, CENABLE(α, β, C)

NB: the time of α's occurring is not necessarily the time of the CGEN of γ and β.

This definition says that the occurence of α brings about condition C and that this condition C is necessary for achieving β (via γ). The example given above can then be represented as follows:

ENABLE(install(Annie, cable-TV), see(Annie, more movies), available(channells))

Unlike generation, there is no need for defining new ENABLE relations for multiple actions, because multiple actions in an ENABLE relation can be expressed with the existing GENERATION and ENABLE relations. The two following examples, and their representation (repeated from the previous chapter), illustrate this point.

Example 13
(i) You need to show the warranty or the bill in order to replace a faulty part.
(ii) John and Paul positioned the seat-frame, then attached it with bolts, washers and nuts, each working on one side of the glider.

(i) GEN-Disjoined((show(G, warranty) $\vee$ show(G, bill)),
show(G, proof-of-purchase))

ENABLE(show(G, proof-of-purchase), replace(G, faulty-part))

(ii) GEN-Simult((position(John, right-side-seat-frame) $\wedge$
position(Paul, left-side-sear-frame)),
place(John&Paul, seat-frame))

GEN-Conjoined((place(John, bolts/washers/nuts, right-side-seat-frame) $\wedge$
place(Paul, bolts/washers/nuts, left-side-seat-frame)),
attach(John&Paul, seat-frame, glider))

ENABLE(place(John&Paul, seat-frame),
attach(John&Paul, seat-frame, glider))

3.4 PREVENT

Example 14 *Mark's playing the piano prevented Charlie from sleeping.*

CPREVENT(α, β, C) $\Longleftrightarrow$
[$\forall$ G1,t1,t2, $\exists\gamma$ [OCCURS(α, G1, t1) $\wedge$
GEN(α, (ACHIEVE ($\neg$HOLDS(C, t1))), G1, t1) $\wedge$
CGEN (γ, β, C)]

PREVENT(α, β, G, t) $\Longleftrightarrow$ $\exists$ C, CPREVENTα, β, C)

NB: the time of α's occurring is not necessarily the time of the CGEN of γ and β.

The PREVENT relationship is very similar to the ENABLE relationship. They both involve two act-types, α and β, related by a generation condition C; the difference is that in the ENABLE case, the occurence of α brings about that condition, wherease in the PREVENT case, it does not. This explains the similarity in the definitions, the only difference being in HOLDS(C, t) versus $\neg$HOLDS(C, t).

3.5 FACILITATE

Example 15 *Learning French will make it easier for you to communicate and get around in Montreal.*

CFACIL(α, β, C) $\Longleftrightarrow$
$[\exists$ C, CENABLE(α, β, C) $\vee$
$\exists$ G1,t1, ($\neg$OCCURS(α, G1, t1)] $\wedge$
$\exists$ C', t2, HOLDS(C', t2) $\Longrightarrow$ (CGEN(γ, β, C') $\wedge$
$\exists$ t3, t3$\neq$t2, HOLDS(C', t3) $\Longrightarrow$ $\neg$CGEN(γ, β, C')]

FACIL(α, β, G, t) $\Longleftrightarrow$ $\exists$ C, $\hat{\text{C}}$FACIL(α, β, C)

This definition is a disjunction. The first disjunct says that there is a condition C such that the occurence of α will bring about that condition, thereby enabling the occurence of β. The second disjunct handles the cases when α does not occur. In these cases, there is some other condition C' under which γ will *sometimes* conditionally generate β. The "sometimes" is represented by the last implication: it states that there is a time interval t3 during which γ will not conditionally generate β.

The *sometimes* is necessary to convey the intuition that doing α is "easier" than doing some other action (namely γ) that will bring about the C' condition allowing the occurence of β. If this clause were not included, then FACIL(α, β, G, t) would be equivalent to

CENABLE(α, β, C) $\vee$ CENABLE(γ, β, C')

3.6 Some open questions

3.6.1 HINDER

Distinguishing FACIL from ENABLE opens the way for introducing HINDER as opposed to PREVENT. Also, in the same way as PREVENT is related to $\neg$ENABLE, so is HINDER related to $\neg$FACIL. These four relationships are constrasted in the following diagram:

$$
\begin{array}{rcl}
\text{ENABLE}(\alpha,\beta) & \Longleftrightarrow & \text{FACIL}(\alpha,\beta) \\
\text{HOLDS(C,t)} \wedge \text{CGEN}(\gamma,\beta,\text{C}) & & \text{HOLDS(C,t)} \wedge \text{helps(C,}\beta\text{)} \\
\Uparrow & & \Uparrow \\
\Downarrow & & \Downarrow \\
\text{PREVENT}(\alpha,\beta) & \Longleftrightarrow & \text{HINDER}(\alpha,\beta) \\
\neg\text{HOLDS(C,t)} \wedge \text{CGEN}(\gamma,\beta,\text{C}) & & \neg\text{HOLDS(C,t)} \wedge \text{helps(C,}\beta\text{)}
\end{array}
$$

where *helps* is a short hand for the second disjunct in the definition of CFACIL.

The definition of HINDER then follows as a dual of FACIL, in the same way that ENABLE forms a dual with PREVENT.

3.6.2 CAUSE

Example 16 *The glider structure collapsed when Paul hammered the leg-assembly too strongly.*

Causality is intuitively similar to generation because both relations involve two actions such that the occurence of the first results in the occurence of second. They are not the same however, because if GEN(α,β, G, t), then α and β occur at the same time, which is not necessarely the case for causality. In the following example, the breaking of the window occurs after the throwing the ball:

Example 17 *Mary threw the ball up high into the air; as it came back down, it went right over the balcony railing and through the living room window, which broke with a loud noise.*

Causality is also different from generation in that GEN(α,β, G, t) implies that the same agent(s) performed α and β, which is not normally the case for CAUSE(α,β), as the preceding examples show.

Causality, in fact, often involves agentless actions, such as the collapsing and breaking actions in the previous examples, or the rising action in "putting yeast in the dough caused the cake to rise". The glider structure, the ball and the cake, *are* the agents of these actions, but not in the sense used in the rest of this study (e.g. agents have intentions and beliefs). By dealing with agentless actions, causality relates events rather than actions.

Finally, one last difference worth mentioning concerns intention. Given the relation, GEN(α,β, G, t), we expect the agent G to *intend* to do β by doing α, which is not the case for actions related by causality.

This relation is therefore quite different from the other ones presented in this report and requires separate treatment.

3.6.3 Negation

Agents may discuss actions or states to be avoided. For example, the dialog during the glider construction includes the two following (non consecutive) statements:

They say here "do not over tighten bolts, ...".
We're not liable to disassemble it at this stage.

Negation, however, is a function rather than a relation between actions since it takes act-types into act-types, and is therefore not a primary concern here.

3.6.4 States versus actions

It often occurs that states enter into action relationships. The question is whether this is acceptable or not, and if they are, how to express such relationships. The example involving the child's interruption for example made reference to the state "undistrubed-environment".

There are two possible answers to this issue. One is to allow states to enter into action relations just like actions do. The predicate ACHIEVE was used by Pollack to turn states into actions, e.g. ACHIEVE(undisturbed-environment). The other answer is to introduce new relationship names when states are involved. For our work, we have chosen to use the ACHIEVE predicate.

3.6.5 Actions participating in several relationships

Actions need not participate in a single action relationship. Sometimes, an action can be analyzed from several perspectives, and each one will be represented by a different action relationship. For example, we can treat the agent's reference to the warranty either as part of the ongoing plan of sorting out parts or as an independent plan of discussing the use of a warranty; i.e.:

GEN-disjoined((show(G, warranty) or show(G, bill)),
show(G, proof-of-purchase))

GEN-Conjoined((check(G, warranty) & check(G, bolts) & check(G, tools))
check(G, parts))

3.6.6 Basic relations and relations of explanation

Generation and *enablement* can be viewed as "basic" relations in that they describe facts about the world, namely how actions are related to each other. Other relations, such as *facilitate* and *prevent* for example, seem less "basic" in that they provide

advice (facilitate), or explanations (prevent) about action relations rather than describing them. This difference might have consequences in the way "recipes" are defined and represented. We leave this topic to future research.

References

[Pol86] Martha E. Pollack. A model of plan inference that distinguishes between the beliefs of actors and observers. In *Proceedings of the 24th Annual Meeting of the Association for Computational Linguistics*, pages 207-214, Association for Computational Linguistisc, New York, June 1986.

[Gol70] Alvin I. Goldman. *A Theory of Human Action.* Princeton University Press, Princeton, NJ, 1970.

[GS89] Barbara J. Grosz and Candace L. Sidner. Plans for Discourse. To appear in *Intentions in Communication*, P. Cohen, J. Morgan and M. Pollack (eds), M.I.T. Press, 1989

[GSL89] Barbara J. Grosz, Candace L. Sidner, Karen E. Lochbaum, Models of Plans to Support Communication. Forthcoming, Harvard Technote, 1989.