MASSACHUSETTS INSTITUTE OF TECHNOLOGY
ARTIFICIAL INTELLIGENCE LABORATORY

Memo No. 423

August 3, 1977

COMEX:
A Support System for a Commodities Expert.

by
James L. Stansfield

Abstract

The intelligent support system project is developing a program (COMEX) to assist a commodities expert in tasks such as interpreting data, predicting trends and intelligent noticing. Large amounts of qualitative and quantitative information about factors such as weather, trade and crop condition need to be managed. This memo presents COMEX-0, a prototype system written in FRL, a frame-based language (Goldstein & Roberts, 1977). COMEX-0 has a complaint handling system, frame-structure matching and simple reasoning. By conversing with a user, it builds groupings of frame structures to represent events. These are called CLUSTERS and are proposed as a new representation method. New CLUSTERS are built from previously defined ones using INSTANTIATION and AGGREGATION, two methods which combine with frame inheritance and constraints to make up a general event representation mechanism. CLUSTERS capture the idea of generic patterns of relationships between frames and raise an issue named the GENERIC CONSTRAINT PROBLEM concerning constraints between the parts of a cluster. The final section presents plans for future work on qualitative reasoning within COMEX and includes a hypothetical scenario.

Descriptive terms: qualitative reasoning, frames, intelligent support systems, clusters, generic constraint problem, event representation.

This report describes research done at the Artificial Intelligence Laboratory of the Massachusetts Institute of Technology. It was supported by the Advanced Research Projects Agency of the Department of Defense under Office of Naval Research contract N00014-75-C-0643.

CONTENTS

Acknowledgement

I would like to thank Steve Rosenberg for many helpful discussions, and Ira Goldstein, Candy Bullwinkle, Bruce Roberts and Adolfo Guzman for valuable comments on the draft.

Section 1: The Commodity Analyst Project.

Many problems today involve manipulating large amounts of data. The data is often presented in English and inter-related by inferences though incomplete and sometimes inconsistent. This presents a management problem which traditional quantitative methods alone cannot solve. A qualitative advance is needed in which programs understand the information they handle. To this end we are developing an intelligent support system for database management with the following capabilities.

1) manipulating many data sources to find facts relevant to the user's goals.
2) inferencing from new data
3) cross-checking data and theories for reliability
4) monitoring events
5) explaining
6) weighing evidence
7) alerting the user
8) preparing reports
9) answering queries

Our experimental domain is the production and trade of key commodities and we concentrate on wheat. This choice has several advantages. Wheat information is given daily in newspapers, and in journals from grain centres. The U. S. Department of Agriculture produces statistics on which most analyses are based. Brokerage houses publish reports giving reasoned arguments about the state of the wheat world. There is also an accessible base of expertise about how reports affect prices. Expert commodities analysts demonstrate their forecasting skill in newsletters from which their analytical methods can be infered. Methods of analysis range from straightforward numerical techniques to qualitative reasoning using a wide range of knowledge.

Economic forecasting is crucially important. Simulation techniques apply to industry, the economy and the global situation. There are over a hundred models of sectors of the commodity market. However, all are limited by special purpose representations making them unsuitable as support systems to experts such as a commodities broker. One typical system is a simulation of the world coffee economy (Epps 1975). It is based on a system of equations and is not amenable to qualitative reasoning. So, if there were a sudden frost that killed the crop, the user must know to interpret this as a change in some number. Since we envisage systems which will report on information from news wires, articles and other sources we require general methods for representing events and the relationships between them.

COMEX (commodity expert) is based on four areas of research; natural language, representation, common sense reasoning and user modelling. We plan to use existing language processing techniques for syntax, surface case structures, and the reference problem (Marcus 1976, Bullwinkle 1977). To understand discourse one must build expectations, "read between the lines" and follow arguments. For this we are studying the discourse structure of news articles and reports (Rosenberg, 1977) and are developing deep semantic representations based on frames. COMEX-0 and the user currently interact in LISP and a simple pseudo-English.

The representation scheme is based on FRAMES (Minsky 1975). We use a frame representation

language, FRL, (Goldstein & Roberts, 1977) to represent knowledge of wheat supply and demand, weather, competition, transport, uses of wheat, storage, export patterns, and so on. As a development of this, COMEX-0 uses CLUSTERS of frames to represent events. It can also represent reports of various types and sources. Government reports, issued at regular times, have diverse formats and topics. Common subjects include supply, planted acreage, crop state, weekly export inspections, and predicted yields. Sources are not equally reliable and the proportion of facts to opinions and statistics to commentary varies. The system will need to know what supports its beliefs to approach new information with accurate expectations and to correct inconsistencies.

"Common sense" reasoning is shallow but broad and much of it seems to take place immediately. In our domain, reasoning frequently involves causality, actions and opinions. The reasoner must analyze its own arguments, compare them and suggest evidence that might decide between them. COMEX-0 does not have extensive reasoning capabilities yet. Only the kinds of immediate or implicit inferences that are available are described here. The event representation provides a foundation for more complex reasoning and this is one major direction of the work.

A model of each user is a prerequisite for presenting only relevant facts and arguments to him. It includes his particular expertise and goals. Goldstein (Carr & Goldstein, 1977) has developed an overlay method for user modelling in the context of a tutoring program for a decision theory game (Stansfield, Carr and Goldstein, 1976). It applies to any system with a rule-based expert such as COMEX.

Section 2 presents a scenario with COMEX-0, a prototype of part of COMEX developed to study representation issues, and discusses user interaction with the system. This is really background for the main focus of the paper which is a discussion of the representation of concepts as CLUSTERS of frames. Section 3 describes CLUSTERS and gives simple examples from the event description facilities of COMEX-0. A major new problem is raised and named the generic comstraint problem. Section 4 describes some initial facilities for inference in COMEX-0. The last section gives an example of qualitative reasoning in the commodities domain and proposes some ideas about representing plans and actions that seem necessary for the next version of the system.

Section 2: COMEX-Ø: A Prototype Analyst.

Frames representation.

COMEX-0 can represent about twenty types of events together with commodities, quantities and prices. Since the representation scheme is frame based it is necessary to give a brief description of frames as they are used in FRL. In its simplest form, a frame is a collection of slots. Each slot denotes a property of the frame and can contain values for that property. For example, a frame for a person may have slots for age, height and name. Frames can be instances of other frames. The frame for Fred is an instance of the frame for people which is an instance of the frame for animal. This means that frames form a classification hierarchy. The instance relationship between a frame and its class is represented by the AKO (a kind of) slot. Frames have two particularly useful features, inheritance and procedural attachment. When one frame is an instance of another, as when Fred is a person, the instance inherits all the generic

properties located in the type. Inherited information is accessed by following the AKO link. The slots of a frame may contain procedures whose purpose may be user defined. Standard purposes are finding values for a slot, checking new values fit requirements and asking the user for values. Procedural knowledge belonging to a class of items is stored in the frame for that class and will be inherited by the instances. The frame structure acts as a hierarchical "filing cabinet" for procedures. This is a powerful organising principle, which COMEX-0 uses to advantage.

Because of the large choice of places for procedural knowledge, COMEX-0 is easily extended. Procedures can be located naturally in the contexts where they will be needed. Since new knowledge can be added by attaching more procedures, this is a form of modularity. It has a cost since procedural attachment of demons leads to deep recursion manifested in persistent side-tracking. This behaviour is like that of a scatter-brain who never completes anything because something else always crops up. Although several approaches are available for controlling this, COMEX-0 is uncommitted. Here are three possibilities. KRL (Bobrow & Winograd 1977) keeps several queues with different priorities on which to schedule events. When a new task comes up it can be scheduled for later so that the current job can be completed. AMORD (Doyle, Steele & Sussman 1977) separates the control structure from the logical dependancies among assertions to provide non-chronological backtracking. The dependancies provide control. A combination of FRL and the Actor control principles of PLASMA (Hewitt 1976) would allow the programmer to tailor-make control structures and escape from them when they are inconvenient.

A dialogue with COMEX-0.

COMEX-0 builds frame structures by means of a dialogue with a user. It asks questions to find values for slots of new frames. Each frame type has procedures for requesting information and for phrasing its queries in pseudo-english. Requirements attached to slots of frames are procedures which inspect new values. Unsatisfied requirements complain to the user and give a reason for unacceptability. The user may try again. The program has simple expectations which keep it from asking unnecessary questions. Examples of these capabilities are presented in the dialogue below.

We begin by telling COMEX-0 that there is a new "exporting" event. Its goal is to instantiate this frame and execute procedures that will be triggered during the instantiation. The dialogue is driven by the instantiation process and results in a collection of instantiated frames called a "frame gestalt". It will be clear from this example dialogue that a simple frame instantiation mechanism places restrictions on both the order and phrasing of the input. This raises interesting points about the form of a natural language interface for the system. The later subsections of this section discuss these.

COMEX-0 first asks us about the new frame. The user can create new instances of other frames can be created as replies and has the option of giving a simple description of the frame or waiting to be asked. The program will recursively enter a new dialogue for these and return to its old dialogue later. The user is prompted with ">>>".

```
>>> (MAKE AN 'EXPORTING)
```

```
WHEN DID EXPORTING1 BEGIN ?        >>> (MAKE A 'DAY)
```

Days have a month and a number. Time is represented in a rudimentary way and COMEX will eventually use FRL's time expert (Roberts & Goldstein, 1977).

```
WHAT MONTH IS DAY2 IN ?            >>> MAY
WHAT DAY OF THE MONTH IS DAY2 ?    >>> 3
WHEN DID EXPORTING1 END ?          >>> (MAKE A 'DAY)
WHAT MONTH IS DAY3 IN ?            >>> APRIL
WHAT DAY OF THE MONTH IS DAY3 ?    >>> 5
DAY3 VIOLATES THE CONSTRAINTS OF THE END-TIME OF EXPORTING1
THE END-TIME MUST BE AFTER TIME2. TRY AGAIN
```

Slots of frames have requirements which check that new values are acceptable. When a requirement is violated. the initial dialogue procedures give the user another try. A complaint handler explains the problem using instructions from several places. Requirements can have specific procedures for such explanations. Some use more general procedures which complain whenever a requirement of a certain type is violated. For instance, a common generic requirement is that a new value be of a particular type. There is a standard complaint procedure for violations of this. If no procedure can be found then a default complaint is output. Questions are formed in the same way as complaints. Some are produced by specially located procedures, others by more general procedures dealing with a class of cases.

```
WHEN DID EXPORTING1 END ?              >>> (MAKE A 'DAY)
WHAT MONTH IS DAY4 IN ?                >>> MAY
WHAT DAY OF THE MONTH IS DAY4 ?        >>> 5
WHERE DID EXPORTING1 TAKE PLACE ?      >>> (A COUNTRY WHOSE (NAME IS USA))
WHAT IS THE ACTOR1 OF EXPORTING1 ?                    >>>  USA
WHAT IS THE ACTOR2 OF EXPORTING1 ?                    >>> USSR
YOU HAVEN'T TOLD ME ABOUT USSR. IS USSR A COUNTRY ?   >>> YES
WHAT COMMODITY IS BEING EXPORTED IN EXPORTING1 ?      >>> (SOME 'WHEAT)
WHAT IS THE QUANTITY OF WHEAT14 ?                     >>> (A 'QUANTITY)
WHAT IS THE NUMBER OF QUANTITY15 ?                    >>> 3
WHAT IS THE UNIT OF QUANTITY15 ?                      >>> FOOT
QUANTITY15 VIOLATES THE CONSTRAINTS OF THE QUANTITY OF WHEAT14
FOOT IS NOT A WEIGHT UNIT
```

The violated constraint is between WHEAT14 and the quantity which is to be a part of it. Their units must match. The violation occurred when the entire quantity was being added to the WHEAT14 frame so the user is asked to give the entire quantity again.

```
WHAT IS THE QUANTITY OF WHEAT14 ?                  >>> (A 'QUANTITY)
WHAT IS THE NUMBER OF QUANTITY16 ?                 >>> 3
WHAT IS THE UNIT OF QUANTITY16 ?                   >>> MILLION-BUSHEL
WHAT IS THE PRICE OF THE WHEAT IN EXPORTING1 ?     >>> (SOME 'MONEY)
HOW MUCH MONEY ?
>>> (A  'QUANTITY WHOSE (NUMBER IS 100000) AND WHOSE (UNIT IS DOLLAR))
```

The system has the information it needs and has finished constructing the instance of "exporting". Certain other frames have also been constructed. The frame structure it has produced is described later.

Complaint handling.

At several points in the dialogue, proposed values violated slot requirements. In each case, a message was output to convey the complaint. Requirements may be simple restrictions or may relate a slot's value to other slots in its frame or even in other frames. Complaints are handled in the same way for each. DECREASING provides an example. The DECREASING frame has the structure shown in figure 1. It is a kind of CHANGING and so inherits slots that specify what is being changed, the OLD-VALUE and the NEW-VALUE. A constraint is added to make certain that when something is decreasing its old value is bigger than its new one. There is also some code attached to that constraint which describes how the system should complain when the constraint is violated.

```
DECREASING
 AKO          $VALUE     CHANGING
 ACTION       $VALUE     DECREASE
 OLD-VALUE    $REQUIRE   ((THE NEW VALUE IS LESS THAN THE OLD)
                          (COMPLAIN: (SAY The old-value must
                                          be greater than the new)))
 NEW-VALUE    $REQUIRE   ((THE NEW VALUE IS LESS THAN THE OLD)
                          (COMPLAIN: (SAY The new-value must
                                          be less than the old)))
```

Figure 1. Complaint handler on the DECREASING frame.

Currently, COMEX-0 rejects the proposed bad value and tries again. In the future, a complain function will try to put right the least significant part of the knowledge involved in the violation.

Output.

The output of COMEX-0 is generated by procedures that are attached to slots of frames. They know how to phrase requests for the values of the slot or to complain about unsatisfied requirements. This approach could be extended so that frames describe themselves. Each frame type would have a procedure to describe its instances. When that procedure describes a part of its frame it can call upon the part to describe itself. Knowledge about producing output can be spread through the system, is modular and can be used flexibly. It is an ideal approach for a developing system. Later, a more general output program will be used (McDonald, 1976) and the local procedures altered to use its formal internal language.

Consider how the frame for a rate describes itself. "Feet per second" asks its numerator to describe itself, then outputs "per" and then asks its denominator to describe itself. An acceleration such as "feet per second per second" works automatically by simple recursion. Simple grammatical transformations making the output more natural are easy to apply. For

example, a list of nouns can be transformed to use commas and one "and".

Use of context would improve parts of the example dialogue. In particular, QUANTITY15 is an eyesore. It would be easy to give COMEX-0 a general procedure for describing quantities which examines the frame structure around QUANTITY15 and says instead "the amount of wheat which is being exported". If a description of the dialogue so far is made available it may be possible to incorporate some simple types of ellipsis.

Mixed initiative dialogue.

In the dialogue presented, COMEX-0 had almost all the initiative. The user could only change the pattern of discourse by presenting new frames for discussion. It would be better for the user to say what he knows about each frame and for the system to ask only for what it still needs. This presupposes some model of what one usually needs to know about a frame and some way to represent when something else might be useful to a particular goal. It is already possible for the user to give information about a frame when he first presents that frame. The last sentence in the dialogue is an example and it made it unnecessary for COMEX-0 to ask for data. The general problem is much harder. Consider what the mix of initiatives could be like during a discourse whose purpose is to resolve a contradiction.

The example in good english.

A hypothetical version of the example dialogue will demonstrate what is to be aimed for. It includes the kind of self-description mentioned above and also several examples of ellipsis. Mixed-initiative is illustrated in the third question and answer. The system did not know when the exporting took place and by default used the past tense. This assumption was wrong but the user corrected it by refering to the event in the future in his reply. Some annotation of the example is given in parentheses.

```
>>> Construct an EXPORTING.

--- When did it happen?                          (uses "it" reference)
>>> The trade was made on January 17 1977.       (refers to the trade part)

--- Who was the seller?                          (uses a role name)
>>> The Bunge corporation.                       (sentence fragment & ellipsis)

--- What was exported?                           (assumes past tense)
>>> Some wheat will be.                          (corrects the assumption)

--- How much wheat?                              (sentence fragment)
>>> 200,000 bushels.                            (sentence fragment)

--- When will it be shipped?                     (refers to part of "exporting")
>>> During July 1977.                           (incomplete specification)

--- Where will it be exported to?
```

```
>>> USSR.

--- Where will it be shipped from?          (varies the verb used)
>>> A Gulf Coast port.
```

Section 3. Representation.

This section begins with an account of a description mechanism available in COMEX-0. Descriptions of concepts are central to representation. The section then deals with representation of objects and follows with a progression from representation of simple events to representation of the more complex event structure produced in the example dialogue of the preceding section.

Descriptions.

COMEX-0 has a convenient way to specify simple frame structures in a pseudo-english form. This is used to define frames, specify requirements and construct simple rules. Here is an example of a frame definition containing a requirement.

```
(A PILOT IS
   (A PERSON WHOSE (ACTIVITY MUST-BE
      (A FLYING WHOSE (VEHICLE MUST-BE
         (A PLANE))))))
```

The vocabulary consists of frame names and the function words "a, is, whose, must-be, and". Verbs and relations cannot be used and all descriptions must be indefinite. The language is clearly still very simple.

Each description is a pure LISP s-expression so although parentheses must be used no parser is needed. Each function word is a LISP fexpr and represents the semantics of that word. If "the" were included it would be a procedure for finding referents. Each time a description is evaluated it returns a corresponding frame structure. "A" produces a new frame and evaluates the slot-descriptions for that frame. Slot-descriptions are expressions using "is" or "must-be". "Is" adds a value to the frame and "must-be" adds a requirement. Value-descriptions may describe frame structures and the expressions may be nested as much as desired. Requirements are handled by "match" statements. A frame structure (pattern) for the requirement is constructed by evaluating its description and is enclosed in a "must-match" expression which is added to the requirements of the relevant slot. The matcher takes care of the requirement when it is invoked. The frame structure corresponding to the description given above is as follows.

```
PILOT
AKO         $VALUE      PERSON
ACTIVITY    $REQUIRE    (MUST-MATCH FLYING1)
```

```
FLYING1
AKO         $VALUE      FLYING
VEHICLE     $REQUIRE    (MUST-MATCH PLANE2)

PLANE2
AKO         $VALUE      PLANE
```

It is important that descriptions be used interchangably with frame structures. COMEX-0 translates syntactic descriptions into frame structures and then treats these as the description so this criterion is met immediately. Requirements are descriptions of the values allowed in a slot. In COMEX-0 a typical requirement is that a value match a given frame structure. The structure is used generically since one copy applies to all instances by inheritance. The requirement can be used to test candidate values of the slot for acceptability as with any procedural requirement. However, requirements give information about values. Even when an actual value is unknown it may be constrained and the constraint may be used in reasoning. A requirement that is expressed as a match against a description is especially useful since it can be used almost directly in place of the unknown value. COMEX-0 uses this idea by treating requirement descriptions as templates for building the value when needed. Suppose that a PERSON has a PLACE-TO-LIVE which must be a HOUSE which must have a LIVING-ROOM. If we are now told that John's living-room is blue and so far have no frame for his place-to-live, we can use the requirement to construct one and can assert its color.

A class of complex networks of frames may have a description. Any instance of such a class will be a particular network whose frames are combined according to the description. A description representing an arch would refer to frames for each part of the arch and for important relationships between these parts. Links in the network arise from frames containing others as slot values. Because one frame may be the value of several slots, the links form a graph rather than a tree. There are also constraints between parts of the network. In the case of "arch", the support and abut relationships constrain the parts. Winston (1970) used networks to describe concepts such as arch. I have combined this with the ideas of inheritance and procedural attachment from FRL and I call the result CLUSTERS. If we think of clusters as words and definitions from a dictionary, their interrelationships become clear. Clusters are made from clusters which are made from clusters. And so on. This is not exactly a hierarchy since circularities are possible.

A description is a generic cluster and an instance is a specialization of it. It should be possible to define generic clusters so that instances of them will inherit all their properties including procedural constraints. This raises a problem illustrated in the dialogue of section 2 at the point where DAY3 violated a constraint. The only part of DAY3 that needed changing was the month. Notice that COMEX-0 needed all the information about DAY3 before it noticed the violation. That is because the constraint is a generic one stored in the EVENT frame. It makes sure that the value of the BEGIN slot is before the value of the END slot. Since constraints are triggered when values are added to the slots that refer to them, this constraint will be triggered as soon as COMEX-0 adds uninstantiated TIME frames to the BEGIN and END slots of EXPORTING1. This is premature. Execution of the constraint should be delayed until the time frames are instantiated. In particular cases, I have arranged for triggers to the generic constraint to be passed down into the parts of the instance as soon as the parts appear. A trigger, like an if-

added, watches a slot and invokes a procedure when a value is added to that slot. The term is used when several if-addeds in different slots invoke the same procedure. If EXPORTING1 were given an uninstantiated DAY, it would push triggers down onto the MONTH and NUMBER slots of the DAY. To generalize, it is difficult to express generic constraints between slots within parts of a generic frame so that they will be triggered at exactly the right time. I call this the "generic constraint problem". The problem is not peculiar to FRL but must be faced in any system that fits new information into generic descriptions which contain constraints.

I think clusters begin to approach the complexity implied in Minsky's original notion of a frame. FRL makes it easy to describe single frames, to constrain single slots and to allow frame inheritance. COMEX-0 uses this base to address the problem of defining clusters of frames and allowing inheritance of all properties of a cluster. An FRL frame represents the view from a point in a semantic net. Its properties are like the relations that emanate from that point so it resembles a spider with an arbitrary number of arms. A cluster is like a set of spiders holding hands and perhaps even holding other spiders. The constraints are between particular spider's hands and govern the things they can hold. A generic cluster is a set of rules that describe a class of clusters. Just as FRL obtains its power from inheritance and procedural attachment, the power of clusters comes from inheritance of constraints and procedural expression of these constraints. As a consequence, the generic constraint problem is crucial.

KRL is a language for complex descriptions. Reports about KRL suggest that a top-down approach was taken, beginning with specifications of desirable language characteristics before deciding on a mechanism that makes these work. In contrast, my approach with COMEX-0 was to work upwards from a basic frame representation language and see which more complex constructs could be easily defined. This is why COMEX-0 uses frame structures as descriptions.

Quantities and Commodities.

In the commodities world we must represent substances and objects. These basic concepts are likely to have implications for the rest of our conceptual world, so representations should be developed with care. Some of the frames in COMEX-0 are shown in figure 2 which displays the classification hierarchy.

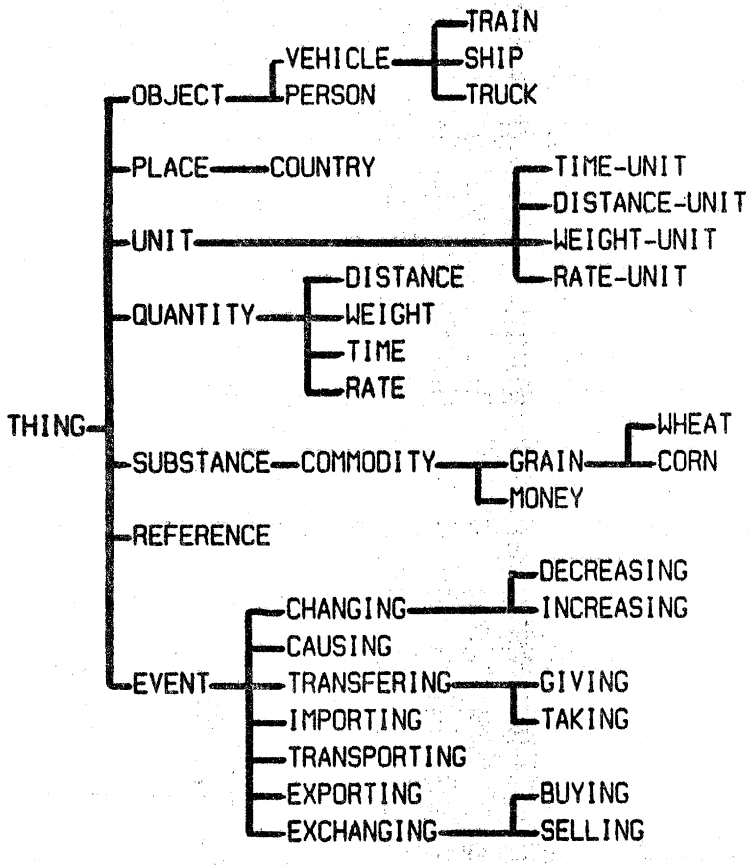

Figure 2. Generic Frames of COMEX-0

Consider representations for SUBSTANCES, OBJECTS, QUANTITIES and UNITS. SUBSTANCE is a generic frame with UNIT and QUANTITY slots. The UNIT slot gives the allowable units for the substance and the QUANTITY slot is used to specify the quantity of each particular instance of that substance. A QUANTITY has UNIT and NUMBER slots. SUBSTANCE has requirements that the values of the UNIT and QUANTITY slots must be UNIT and QUANTITY frames respectively. Also, one of the UNITS of any instance of a SUBSTANCE must be the same as the UNIT of that substances QUANTITY. Two classes of frame, generic and token, can be instances of a generic frame. Types refer to concepts in general semantic memory and tokens represent specific instances of these concepts. They use the same AKO link and a single inheritance mechanism. They differ in form only by a mark and are stored together.

COMEX-0 could have used different frames for a substance and an amount of it. A QUANTITY would then have an extra slot for the substance it was a quantity of. Instead, any instance of a substance which has a value for its quantity represents a particular physical instance of that substance. The former method is appropriate for objects since they are quantified as collections rather than by amount. Objects have form and are made up of related parts each composed of a substance with its characteristic physical properties. Objects made of a single substance can be refered to either as a substance or an object as, for example, "a bottle" versus "some glass". Commodities markets treat most commodities as substances.

To COMEX-0, COMMODITIES are a type of SUBSTANCE and inherit their properties. More

precisely, a commodity is any type of substance or object that is generally traded. It should be defined in terms of its ROLE and not by its structure. Roles, such as "business-man, trader and hedger," form a hierarchy in which properties should be inherited just as in the AKO hierarchy. For example, a hedger has all the properties of a trader. The problem of representing roles and allowing inheritance is called the role problem. If we place the roles of a person in a slot called role, the inheritance will not be automatic AKO inheritance. However, if we make someone a kind of person and a kind of X for each of his roles X, he will always inherit from many places. But we must restrict role inheritance to the times we consider the person as playing that role.

Multiple inheritance crops up in other situations. Since, in general, a commodity could be either a substance or an object it may seem that COMMODITY should have AKO links to both of these. This would make every commodity both a substance and an object so it is not a correct representation. The following two correct representations seem at first to need only one AKO link but on closer investigation need two. First consider defining a new type, of which substance and object are both instances. COMMODITY would be an instance of this type and so needs only one AKO link. However, a particular substance which is also a commodity now requires two AKO links since it must inherit from COMMODITY and SUBSTANCE. Alternatively we could have two generic instances of COMMODITY, those that are substances and those that are objects. In this case, those that are substances must inherit both from COMMODITY and SUBSTANCE. The problem is that COMMODITY/NON-COMMODITY and SUBSTANCE/OBJECT are orthogonal and each of the four possibilities has specific information associated with it. Such orthogonality occurs often.

GRAIN and MONEY are types of COMMODITY. The entry in the unit slot for GRAIN is a requirement that the value must be a WEIGHT-UNIT. Examples of WEIGHT-UNITS are BUSHEL and METRIC-TON. It is straight-forward to add procedures to convert from any unit to any related one, to perform calculations involving quantities such as determining the price of X pounds at Y dollars per pound, and to take into account information about actual or usual prices and the units commonly used. Such a procedure can be a requirement or if-added on a slot, that converts new values into standard form.

In future versions of COMEX, objects will be related to events. Just as an event has a beginning and an end, so does an abject. Associated with the generic representation of the object will be information regarding the events that cause its production or disappearance. In the commodities domain, objects have their form changed by various transformation processes which an expert knows about. Seed turns into wheat and then into flour and bread. The amounts produced by each process and their timing are important to supply-demand analysis.

Representing events.

In the next sub-sections, COMEX-0's event representation is described. As an introduction here is a summary of the topics.

The events COMEX-0 represents are instances of TRANSFER, GIVE, BUY, SELL, TRANSPORT, CHANGE, TAKE, INCREASE, DECREASE, CAUSE, EXCHANGE, MOVE, EXPORT and IMPORT. Objects and actors participate in an event according to a pattern specific to that event type and described by a set of relationships. A buying event, for example, relates a buyer, a seller, the

money and the purchased item around various actions of transfering location and ownership. Events have time and location relationships between participants to specify their changing positions and important times such as the beginning and ending of the event. The relationships of participants to an event are classified into cases such as OBJECT and RECIPIENT.

An event will often be a collection of subevents. A buying event has at least two parts which are transferings of ownership. Transfering of money can again be divided into giving and receiving money. There is a difference between the dictionary meaning of an event-word such as "buy" and the complete description of a particular buy event. An actual instance of a buying event would be completely described only if details were given such as the particular hand used to give the money. Microscopic analysis like this does not belong to the semantics of "buy" but may appear in an instance of "buy". This fits into the scheme of things as follows. The meaning of "buy" is a description of the essential sub-events and their relationships. In a particular case of "buying", instances of the sub-events and participants will be arranged according to the description. The sub-events also have descriptions of the structures they may have and the instantiation process may be carried on to indefinite detail. The details, however, are described elsewhere than in the main event. This does not imply that an events description only states relationships between its immediate sub-events. A description can equally relate parts of parts of a sub-event to parts of parts of another. The important point is that the further down into details one goes the fewer constraints apply from the very top-level.

Events can be thought of as structures in time. Event types are descriptions of classes of structures with common features. An event structure is made up of many related atomic changes. In COMEX-0, CHANGING and TRANSFERING are the PRIMITIVE ATOMIC EVENTS from which all others are ultimately built.. Two combinators, INSTANTIATION and AGGREGATION, are used to derive further events. Instantiation specializes an event. The instance inherits all the meaning of the generic event and may also have extra rules or values. Events are combined by aggregation, resulting in a frame that holds frames for the subevents. Such a frame is a simple cluster. In a generic aggregate, certain relationships must hold between parts of the subevents. For example, EXCHANGING is a generic cluster formed from two TRANSFERINGS which must be in opposing directions. Rules for these constraints are kept in the generic cluster to be inherited by any instance. Instantiation can be applied to the results of aggregation. In other words, clusters can be instantiated to produce more specialised versions.

I have informally investigated more complex scenarios of events such as rain falling and being absorbed or running off the ground. A storm includes winds and heavy rainfall as subevents. The frame representation allows aggregates like these, and each generic event frame contains the information which relates its parts. Complex aggregations may contain sub-events arranged in time. The time relationships are expressed as generic constraints on the event. All events have a beginning and an end. Some events are instantaneous in which case the beginning and end are regarded as the same. Several sub-events may begin at the beginning of the aggregate.

Events are not reduced to primitives in the way proposed by Schank (Schank, 1975). COMEX'S events form a hierarchy and specific semantic information can be attached at any level. This may be in the form of new values, requirements or if-added procedures.

The primitive atomic events.

CHANGING and TRANSFERING are used to describe primitive changes in the state of the frame data-base.

CHANGING		
AKO	$VALUE	EVENT
CHANGED	$REQUIRE	(must be a REFERENCE)
OLD-VALUE		
NEW-VALUE		

A CHANGING specifies that something has changed from OLD-VALUE to NEW-VALUE. Being an event it inherits slots describing time. Whatever has changed must be the value of some slot S in a frame F. An instance of CHANGING refers to this slot by name for it is not enough to specify the value. If Fred's location changes from Boston to New York, Boston does not change. The "location of Fred" changes. A generic frame type, REFERENCE, is used to refer to slots as opposed to their values. A REFERENCE has a FRAME slot and a SLOT slot.

REFERENCE		
AKO	$VALUE	THING
FRAME	$REQUIRE	(must be a FRAME)
SLOT	$REQUIRE	(must be a slot of that frame)

CHANGING1			REFERENCE2		
AKO	$VALUE	CHANGING	AKO	$VALUE	REFERENCE
CHANGED	$VALUE	REFERENCE2	FRAME	$VALUE	FRED
OLD-VALUE	$VALUE	BOSTON	SLOT	$VALUE	LOCATION
NEW-VALUE	$VALUE	NEW YORK			

TRANSFERING is the dual of CHANGING. In the FRED example we could add an inverse location slot, CONTAINS, to each PLACE to specify the objects there. FRED is then transfered from one place to another. The difference between a CHANGING and a TRANSFERING is simply one of perspective. From Fred's point of view, his location changes. From a spatial point of view, he is transfered. It turns out that we conceptualise some events as CHANGES (e.g. MOVING) and some as TRANSFERS (e.g. GIVING). The choice may depend on who does the action, the thing being altered or some other thing. The frame for TRANSFERING has slots for DONOR, RECIPIENT, RELATION and OBJECT. The RELATION slot holds the predicate changed by the action and will be a slot name common to the DONOR and RECIPIENT.

TRANSFERING1		
AKO	$VALUE	TRANSFERING
DONOR	$VALUE	BOSTON
RECIPIENT	$VALUE	NEW YORK
OBJECT	$VALUE	FRED
RELATION	$VALUE	CONTAINS

Building events by instantiation.

MOVING is an instance of CHANGING where the LOCATION slot of a frame is being changed.

```
MOVING
AKO         $VALUE     CHANGING
CHANGED     $REQUIRE   (must be a REFERENCE whose SLOT is LOCATION)
SUBJECT     $REQUIRE   (must be a PHYSICAL-OBJECT)
                       (must be the FRAME of the CHANGED slot)
OLD-VALUE   $REQUIRE   (must be a PLACE)
NEW-VALUE   $REQUIRE   (must be a PLACE)
```

MOVING inherits from CHANGING. It has an extra slot for the object moved and has requirements that specify the slot values in more detail. MOVING also constrains its SUBJECT to be the same as the FRAME of the REFERENCE in the CHANGED slot. GIVING is another instance of construction by instantiation.

```
GIVING
AKO         $VALUE      TRANSFERING
DONOR       $IF-ADDED   (map-into FOCUS)
FOCUS
RELATION    $VALUE      OWNS
```

A difference between GIVING and TAKING is the party who initiated the transfer. COMEX-0 represents this with a FOCUS slot. This use of FOCUS must not be confused with the linguistic use. The FOCUS of a GIVING is the DONOR. A value asserted into the DONOR slot will be mapped into the focus slot. Map-into is an if-added procedure which is triggered when the value of the donor is asserted.

Aggregation.

Aggregation is exemplified by EXCHANGING which has two parts, both of them TRANSFERINGS occuring at the same time as the EXCHANGING.

```
EXCHANGING
AKO          $VALUE      EVENT
TRANSFER1    $IF-ADDED   (must be a TRANSFERING)
TRANSFER2    $IF-ADDED   (must be a TRANSFERING)
ACTOR1       $IF-ADDED   (map-into TRANSFER1 DONOR)
             $IF-ADDED   (map-into TRANSFER2 RECIPIENT)
ACTOR2       $IF-ADDED   (map-into TRANSFER1 RECIPIENT)
             $IF-ADDED   (map-into TRANSFER2 DONOR)
OBJECT1      $IF-ADDED   (map-into TRANSFER1 OBJECT)
OBJECT2      $IF-ADDED   (map-into TRANSFER2 OBJECT)
BEGIN        $IF-ADDED   (map-into TRANSFER1 BEGIN)
             $IF-ADDED   (map-into TRANSFER2 BEGIN)
END          $IF-ADDED   (map-into TRANSFER1 END)
             $IF-ADDED   (map-into TRANSFER2 END)
```

When we instantiate EXCHANGING, the map-into commands construct the two subcomponents TRANSFER1 and TRANSFER2 in such a way as to represent that ACTOR1 transfered OBJECT1 to ACTOR2 and ACTOR2 transfered OBJECT2 to ACTOR1. This mapping is inference by definition expansion, a common part of mathematical proofs. Assertions about transfering are infered from the definition of exchanging. A map-into adds a value to a specified slot of a frame which itself is part of the exchanging. If the frame is not already present, map-into constructs it. It uses the requirements on the TRANSFER1 and TRANSFER2 slots as descriptions of their contents and constructs frames fitting these descriptions.

Mapping is used here in the sense of a correspondence. Values in one frame are related to values in another. If John bought a car from Joe, John transfered money to Joe and there is a mapping from the buying to the transfering which represents an inference. MERLIN (Moore & Newell, 1973) mappings are also correspondences between concepts. They are used to view one concept as though it were another. There are clearly some similarities between the two types of mapping. MERLIN maps go between types and subtypes and COMEX-0 mappings go between a concept and its parts. Although an exchanging is composed of two transferings we could say it can be viewed as two transferings in the MERLIN sense. In general, mappings go from a set SA of parts and relations of a structure A to a set SB of parts and relations of a structure B. Relations and parts are put into correspondence. A mapping is useful if further relations SB' in B, implied by SB, can be mapped back into SA' of A. The mapping enables B to be used as a model for A. SA' can be said to be deduced from B by analogy. Figure 3 represents this. In the EXCHANGING example, an inference in SB' is that before the action John had a car.

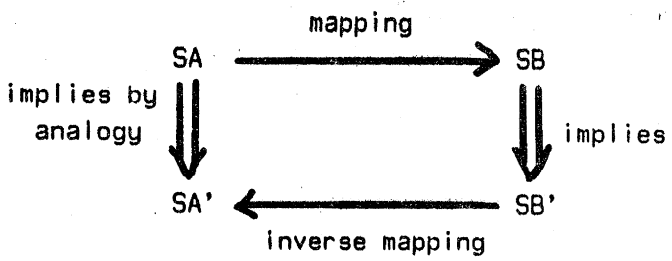

Figure 3. Transitivity diagram.

Frame structure built by the example dialogue.

During the example dialogue with COMEX-0, a frame structure for an exporting event was constructed. Enough of the representation mechanism has now been described for a presentation of this structure. Figure 4 shows a schematic version of it. Notice the distinction between "parts-of" signifying aggregation and "a-kind-of" signifying instantiation. EXPORTING1 has two components, a TRADING and a TRANSPORTING. TRANSPORTING7 has two parts, a MOVING and a CAUSING. Someone caused something to move. The TRADING is SELLING10 which consists of two TRANSFERINGS, wheat from the USA to the USSR and money the other way.

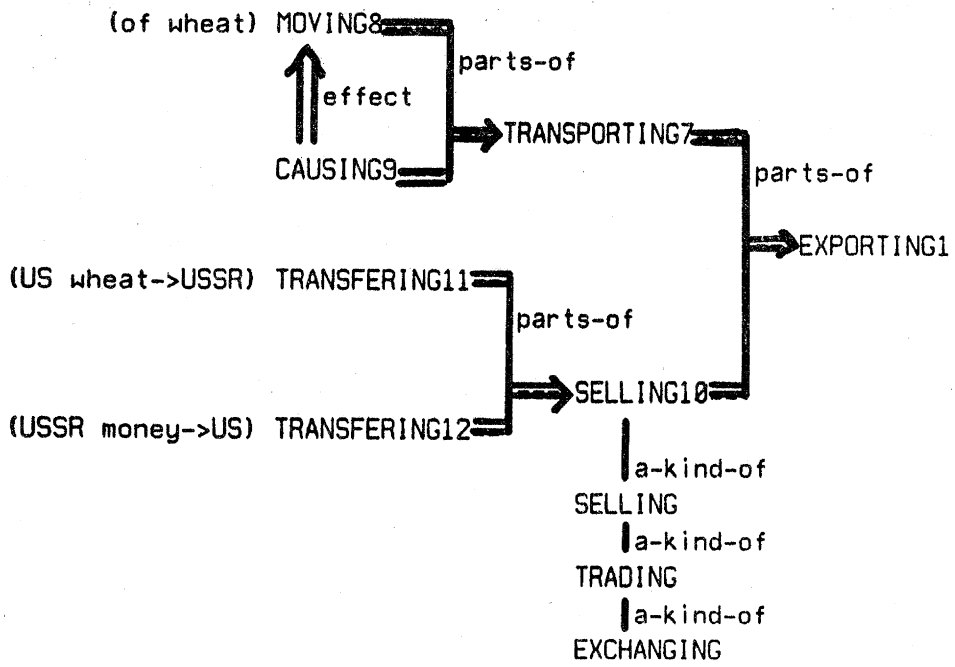

Figure 4. Event structure of EXPORTING1.

Figure 5, shown next, gives some of the frame structure for EXPORTING1 in frame-like syntax. The AKO slots are ommitted from figure 5 as they are self-evident from the frame names.

EXPORTING1

BEGIN-TIME	$VALUE	DAY2	END-TIME	$VALUE	DAY4
TIME-PERIOD	$VALUE	TIME5	PLACE	$VALUE	USA
ACTOR1	$VALUE	USA	ACTOR2	$VALUE	USSR
TRANSPORT	$VALUE	TRANSPORTING7	TRADE	$VALUE	SELLING10
COMMODITY	$VALUE	WHEAT14	PRICE	$VALUE	MONEY17

TRANSPORTING7

PART-OF	$VALUE	(EXPORTING1 TRANSPORT)			
SOURCE	$VALUE	USA			
MOVE	$VALUE	MOVING8	DESTINATION	$VALUE	COUNTRY13
CARGO	$VALUE	WHEAT14	CAUSE	$VALUE	CAUSING9

MOVING8

PART-OF	$VALUE	(TRANSPORTING7 MOVE)
FROM	$VALUE	USA
TO	$VALUE	USSR

CAUSING9

PART-OF	$VALUE	(TRANSPORTING7 CAUSE)
EFFECT	$VALUE	MOVING8

SELLING10

PART-OF	$VALUE	(EXPORTING1 TRADE)			
ACTOR1	$VALUE	USA	ACTOR2	$VALUE	USSR
TRANSFER1	$VALUE	TRANSFERING11	TRANSFER2	$VALUE	TRANSFERING12
OBJECT1	$VALUE	WHEAT14	OBJECT2	$VALUE	MONEY17

TRANSFERING11

PART-OF	$VALUE	(SELLING10 TRANSFER1)
DONOR	$VALUE	USA
RECIPIENT	$VALUE	USSR
OBJECT	$VALUE	WHEAT14

TRANSFERING12

PART-OF	$VALUE	(SELLING10 TRANSFER2)
DONOR	$VALUE	USSR
RECIPIENT	$VALUE	USA
OBJECT	$VALUE	MONEY17

WHEAT14

QUANTITY	$VALUE	QUANTITY16

```
QUANTITY16
NUMBER      $VALUE  3
UNIT        $VALUE  MILLION-BUSHEL
```

figure 5. Frame structure for EXPORTING1.

Figure 6 shows the events represented in COMEX-0. There are two kinds of arrows shown, some have one stem and others have two. One-stemmed arrows are AKO links and appear when events are derived by instantiation. Two-stemmed arrows represent PART-OF and appear in aggregations. MOVE is a change of location. TRANSPORT is to cause to move and has two parts. IMPORTING & EXPORTING each contain a TRADE and a TRANSPORT and have rules relating them appropriately. A TRADE is an EXCHANGE of ownership and BUY & SELL involve money. GIVE & TAKE are kinds of transfer.

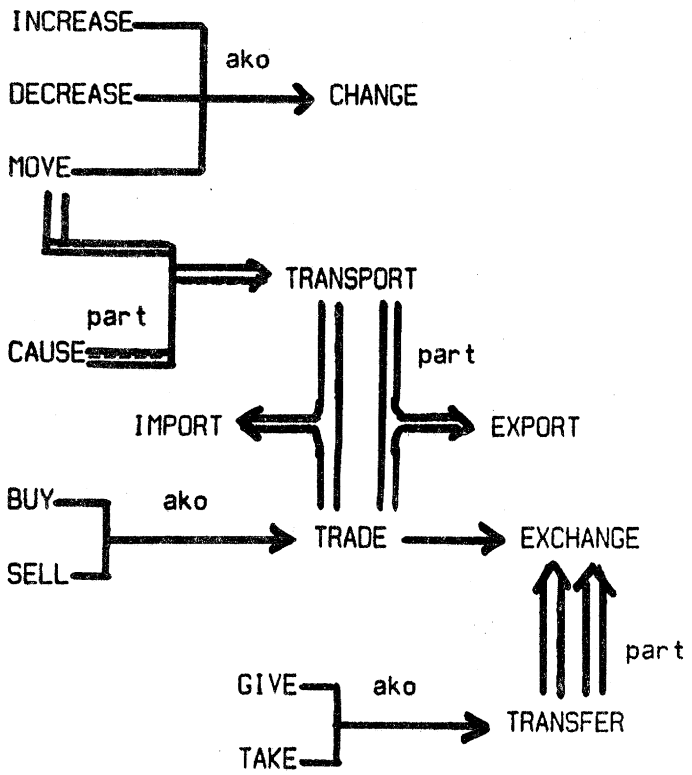

Figure 6. Relationships between event types.

Section 4. Inferencing.

Although COMEX-0 has no general reasoning mechanism, several modes of inferencing became clear while developing its representation scheme. They followed from using FRL as a representation language. A list is given here and the following subsections describe them. "Definition expansion", has already been discussed. It occurs when the system maps a concept into a lower level structure that defines it. Another kind of inference uses procedures that constrain the values of sets of slots. It is exemplified by the arithmetic constraints within the SUPPLY-DEMAND frame. Inferencing also happens when two matching frame structures are

merged since the information from each becomes accessible together. The last method dealt with is "Autopositioning". Since a frame inherits knowledge from the frame hierarchy it is important to position that frame at the lowest point possible. For example, when there is enough information to know that an animal is a cat, it should be moved down the frame hierarchy where it will inherit information about cats.

Constraints.

One of the inferencing modes incorporated in COMEX-0 is constraint handling. Some inferences about a "concept" can be handled by constraints among the slots of the corresponding frame and supply-demand reports provide an example. They are issued by the USDA and specify the amount of wheat in various sectors of the market during one year. A supply-demand frame has slots for amounts of imported wheat, production and so on. The slots are shown in figure 7. Because of conservation of wheat there are arithmetic constraints between the values of these slots. In COMEX-0, the constraints are represented as if-added procedures stored in the generic supply-demand frame and triggered when enough information becomes present in any particular supply-demand instance. Figure 7 shows the three constraints within SUPPLY-DEMAND. Each constraint requires an if-added theorem to be placed in all relevant slots. Since the if-added will be triggered when a value is added to any relevant slot it first checks to see if enough slots are filled for the constraint to apply.

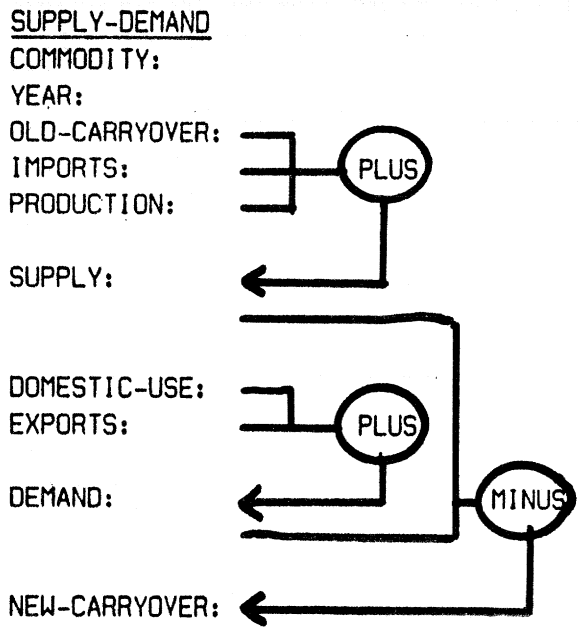

Figure 7. Supply-demand constraints.

The constraints are shown in action in the following dialogue with COMEX-0. This example has been rephrased in English. COMEX-0 supports the interaction but in a more stilted form.

```
>>> Make a supply-demand frame for wheat in 1976.
>>> The old-carryover is 600 million bushels.
>>> Imports are 20 m.bushels.
```

```
>>> Production is 1,300 m.bushels.

--- So the total supply must be 1.920 m.bushels.

>>> Domestic use is 800 m. bushels.
>>> Demand is 1,600 m.bushels.

--- So the exports must be 800 m.bushels.
--- The new-carryover must be 320 m.bushels.
```

If we now alter one of the values in the frame the if-addeds are triggered again to signal inconsistency. The three theorems in this frame are organised in a simple hierarchy. Old-carryover, imports and production give the supply; exports and use give demand; demand and supply give new-carryover. Constraints which deal with simultaneous equations or which operate in the reverse direction are best dealt with in a more global way. To handle more complex constraint sets and to resolve inconsistencies it seems that assertions must be annotated with statements justifying them (i.e. giving their derivations). AMORD is a recent system that handles justifications well (De Kleer, Doyle, Sussman 1977) and its method may be appropriate for COMEX.

Matching.

Matching is related to inferencing in several ways. First, matching formalisms allow restrictions to be expressed easily and concisely. This was their use in MICRO-PLANNER (Sussman, Winograd & Charniak, 1970) where rules only apply if the goal for the situation matches the goal of the rule. Production systems also use matching as their basic control primitive (Newell and Simon, 1972). COMEX-0 makes two uses of the restriction property of matching. First, requirements on slots are specified by means of match statements so that only values that match are allowed into the slot. The second use is to describe the trigger of an if-added procedure which is invoked when a value added to a slot matches some frame structure. Besides these two uses, matching allows the following type of inference. When two matching frame structures represent the same object and each has information that the other hasn't, merging them results in all the information being accessible together.

COMEX-0's matcher has special properties because it matches frame structures. First consider what a preliminary test for matchability of two frames would be. One might expect a necessary condition to be that one frame is AKO the other. A human may match a mortal since HUMAN is a kind of MORTAL. However, HUMAN1 may match HUMAN2 and in this case neither frame is an instance of the other. To see what the condition should be, consider two frames, QUANTITY2 and WHEAT1, which do not match. The reason is that they have different slots. A weight has a number and a unit but some wheat has a quantity and a type. This suggests that two frames are matchable if the slots of one form a subset of the slots of the other. For example, QUANTITY only has slots for NUMBER and UNIT so any frame at all with these two slots will be considered a QUANTITY and may therefore match. This is the criterion COMEX-0 uses.

Having decided that two frames are matchable we check that corresponding slots match. If the slots have values these must match each other. The requirements on a slot are also important

for a value from the slot of a frame must fit the requirements of the corresponding slot in the matching frame. Since requirements are often specified in terms of match statements, the matcher is often called recursively. But what if neither slot has a value but both have requirements? We should check that these requirements are compatible. This leads to reasoning for several inference steps may be needed to show that anything satisfying one description also satisfies the other. In COMEX-0, requirements that are represented as match statements can easily be used in this process. The two frame structures being used in the requirements must match.

To summarize, the matching criterion for two frames is that the definition of each frame's type should fit the other and the descriptions of corresponding slots should fit. COMEX-0 currently uses the set of slot names of a frame as a simple definition of its type.

Frame structures are often circular. The matcher temporarily marks its path in a frame structure and if it returns, it must return in the corresponding structure. This ensures that matching frame structures have the same topology and that the matcher never gets caught in an infinite loop. Matching can get out of control in other ways not dealt with by COMEX-0. For instance, there are some slots in some frames which are not directly relevant to a match. Two frames may have unequal but matching values for such a slot and it can happen that the matching process spends a lot of effort "away from the point". To terminate such a match it seems that some idea of the boundary of a concept would be useful. This would prevent the matcher straying into territory that is not pertinent. Clusters may provide one natural boundary. However, the boundary may also depend on the particular goal of the match. In this case we have partial matching of the form "frame1 can be viewed as frame2" as in MERLIN.

FRL encourages judicious location of procedures within frame structures. The advantages of inherited procedural knowledge follow from a good location for a requirement or an if-added procedure. COMEX-0 also locates certain procedural knowledge about matching at appropriate points in the frame structure. Consider a match between a "point in time" frame and an "interval in time" frame. These match if the point is within the interval. For example, "John skied on Sunday" should match "John skied on Sunday afternoon at 3.00". This special matching condition is really knowledge about time. There are many special matching conditions and it is cumbersome for the matcher to know them all. In COMEX-0, "time-period" and "time-point" know how to test for matches against themselves. The matcher simply orders them to do the match. With this technique, specific knowledge about matching can be distributed through the frame structure. Specific matching procedures are attached to frames using a slot called MATCH. In this way, general matching expertise can be inherited when needed.

Autopositioning.

Autopositioning is another way in which a frame representation leads to inferencing. Since much knowledge is inherited within a frame system, the position of a frame in the hierarchy is important. If a frame is placed too far up the hierarchy only general knowledge about it will be available. As new information about a frame arrives or is deduced it gives clues to the frame's more precise position in the tree. I call this autopositioning and plan to include it in future COMEX systems.

Each frame type should take responsibility for positioning its instances in the frame tree. As an example, consider a frame for ANIMAL which has two generic instances CAT and DOG. When ANIMAL is given a new token instance, it should try to determine whether it is a cat or a dog. One way is to query the information source but another is to use information about dogs. If the new animal pants to keep cool and has a tail for example, COMEX might reasonably move it down the tree to DOG. To conclude the animal is a dog COMEX-0 needs a set of sufficient conditions for a frame to be a dog. Such a set is a definition of DOG and could be stored in that frame. There may be many condition sets based on features of dogs that are sufficient. They will form an implicational structure composed of constraints on the slots of the DOG frame.

Autopositioning might correspond to that part of perception that fills out a hypothesised model and specialises it.

An example passage.

Consider the following passage.

> *"The USDA reduced its estimate of the exports of wheat this year. Estimates of exports to Russia had decreased."*

The first sentence gives rise to two expectations. First, it is likely that some country has reduced its purchases of wheat from the USA. Also, since no exporting country is mentioned, USA is taken as the default since the report is by the USDA. The second sentence validates the first of these expectations by matching it. Merging the matching structures reveals that Russia caused the reduction and it was Russia's wheat exports that changed. The example shows that connections exist in discourse; that they are revealed by matching expectations with later input; and that merging the connecting parts makes new knowledge available. Connections like this have been studied by Rosenberg (1977) as part of the intelligent support system project. Such discourse structure is needed to know that the second sentence of the passage may be a reason for the first.

In this section I show how COMEX-0 creates the expectation in this case and how we can use its matcher to merge this with the second sentence. COMEX-0 has no knowledge of discourse structure however. Nor does it know about expectations. The example is used only to demonstrate the representation facilities in an area where they will ultimately be useful.

An if-added theorem on the changed slot of a decreasing frame represents the inference rule that total exports fall because the exports to some country drop. This theorem can be written in the description language.

```
(IF THE CHANGED IS
   (A REFERENCE
     WHOSE (FRAME IS (A 'SUPPLY-DEMAND)) AND
     WHOSE  (SLOT IS ('EXPORTS)))
THEN THE REASON IS
   (A DECREASING
     WHOSE  (CHANGED IS
```

```
(A REFERENCE
  WHOSE  (SLOT IS ('AMOUNT)) AND
  WHOSE  (FRAME MUST BE
    (AN EXPORTS-TO-ONE-COUNTRY)))))))
```

The frame structure for the first sentence can be constructed in a dialogue with the program. We do this by making a DECREASING frame and instantiating it appropriately. The rule will be triggered and the REASON slot of the decreasing will be filled with a frame structure representing the expectation.

```
>>> (MAKE A 'DECREASING)
WHAT FRAME IS BEING ALTERED IN DECREASING22?          >>> SUPPLY-DEMAND19
WHAT SLOT IS BEING DECREASED IN SUPPL-DEMAND19?       >>> EXPORTS
WHAT IS THE NEW VALUE OF THE EXPORTS OF SUP-DEM19?    >>> 300
```

DECREASING22

AKO	$VALUE	DECREASING
CHANGED	$VALUE	REFERENCE23
REASON	$VALUE	DECREASING25
OLD-VALUE	$VALUE	400
NEW-VALUE	$VALUE	300

The entry DECREASING25 uses a reference to say that some frame representing the exports to some country has had its amount reduced. This frame is represented by a requirement on the slot it is to fit when it is found. This is specified using the matcher.

REFERENCE23

AKO	$VALUE	REFERENCE
FRAME	$VALUE	SUPPLY-DEMAND19
SLOT	$VALUE	EXPORTS

DECREASING25

AKO	$VALUE	DECREASING
CHANGED	$VALUE	REFERENCE26

REFERENCE26

AKO	$VALUE	REFERENCE
SLOT	$VALUE	AMOUNT
FRAME	$REQUIRE	(MUST MATCH EXPORTS-TO-ONE-COUNTRY27)

We can use a description to construct the frame structure for the second sentence in the passage.

```
(MAKE A DECREASING
   WHOSE  (CHANGED IS
      (A REFERENCE
```

```
        WHOSE FRAME IS
           (AN EXPORTS-TO-ONE-COUNTRY
               WHOSE (COUNTRY IS
                  (A COUNTRY
                      WHOSE (NAME IS ('USSR))))))
AND WHOSE (SLOT IS ('AMOUNT))))
```

We use the matcher to see if this frame structure matches the reason for the first passage. It does, and we merge the two combining their information. The second sentence says nothing about wheat but the reduction in Russian exports caused the change in total wheat exports so the commodity is the same in both instances. This kind of inference is easily done by merging the two frame structures.

Section 5: Plans for Qualitative Reasoning.

Reasoning can be divided into two classes, filling out the gaps in a frame structure and deducing new frame structures from old ones. The first is achieved in COMEX-0 by means of constraints between slots of frames. Its goal is to produce a frame gestalt. This section sketches some plans for deduction in the next COMEX system, COMEX-1. In the commodities world, deduction deals with states of the world, actions which change these states, actors who perform these actions and their motivations. Several actors may be involved and may reason about the plans of the others. From a frame structure describing one state, COMEX-1 should be able to deduce other frame structures describing future or previous states. A scenario will illustrate these concepts.

> *During 1976 France had a 400,000 ton surplus of dried milk which the government wanted used as feed. French farmers were using less expensive soybean meal from the U.S.*

In this situation, COMEX-1 would alert the user that the French government might impose an import duty on soybeans. The static representation has a COMPETITION frame containing two competing trades, one an INTERNATIONAL-TRADE and one a DOMESTIC-TRADE. The system expands on its input by using frame constraints to produce this frame gestalt. INTERNATIONAL-TRADE knows that import duty increases the price. TRADE knows the effect of a government subsidy.

Dynamic knowledge concerns actions that can be taken by the French government or the French farmers. In the initial state W1, soy is less costly than milk so the COMPETITION frame infers that farmers will buy soybeans. The government has the goal G1 that farmers buy milk, so COMEX-1 examines the reasoning that might lead from this goal. The government must alter the world so that soy is more expensive than milk. This is G2 which generates two alternate subgoals, G3, to reduce the price of milk and G4, to increase the price of soy. The government may subsidize milk leading to W2 where its price is less than soy. An import tax would raise the price of soy giving a world W3 with the same result.

To realize that changing the price relationship will alter the farmers behavior requires analysis of the farmers plans. Farmers had a need for animal feed that generated the goal G5 that they

possess the feed. This resulted in a plan whose action A1 was to buy feed. A1 has instantiations A2 and A3 corresponding to buying soy and buying milk. In the usual AI planning situation, either successfully achieves the goal. Actually the farmer places different values on the actions and decides between them accordingly. Feldman & Sproull (1977) discuss a planning program based on STRIPS (Fikes & Nilsson, 1971) that includes a decision theory component which determines the best of several alternative plans. COMEX-1 will need the extra capability of including such decisions in its domain of reasoning. For example, the French government must analyze the farmers' plan to discover why they are buying soy and from this to determine how to alter the farmers' behaviour. COMEX-1 should be able to answer the question "Why did the farmers buy soy?" with the decision theory answer "It was cheaper than milk" and not just with the goal oriented answer "They needed feed." There are parallels with Abelson's (1973) script model of interaction between actors.

Representation for Reasoning

In the usual paradigm, causal reasoning and planning take place on a domain of states and actions. Actions change worlds which are sets of assertions about state. Each world is transformed into a well-defined successor by a single fully specified action. This gives rise to a planning tree and to a search paradigm for reasoning using either forward chaining, backward chaining or means-end analysis. The frame problem (Raphael, 1970) classically results and some means is provided to take account of parts of the world which an action does not change as well as those it does. We propose several changes to this scheme.

The commodities world deals with continuous actions. Two projects have considered this. Hendrix (1973) extended STRIPS to deal with processes such as water pouring into buckets. Skuce (1976) wrote a program which qualitatively models biological processes in a medical situation. The latter is closely related to the commodities domain for it relates a qualitative description of a process with its simulation. Continuous actions have several implications for representation. First, a world must contain, besides its state assertions, a set of actions which are in progress. Second, an action may relate two worlds which are not consecutive. Third, many actions may relate two worlds.

Consider W1, a world in which John is at Logan airport, W2 in which he is on the plane drinking a cocktail and W3 where he lands in Heathrow. Each has a set of state assertions such as (ON JOHN AIRPLANE) which are organised into frames. W2, however, has the further property that A1, the action in which John is flying from Boston to London, is in progress. Second, A1 connects W1 and W2 which are not consecutive. Third, W2 is the result of many actions including A2, John boarded the plane and A3, the pilot boarded the plane. When dealing with real worlds rather than toy worlds these three extensions must be taken into account.

STRIPS assumes an initial state is fully specified and determines all succeeding states toward a goal. In the commodities world information is usually incomplete. Incompleteness arises in three ways. First, any world is incompletely known. Second, the sequence of worlds may be incompletely known. Yesterday, John was in Boston, now he is in my office but I don't know where he was in between. Third, actions can be incompletely specified. The french farmers bought some feed but it is unknown what kind it was. From these three forms of incompleteness, it follows that the usual context mechanism will not work. In a fully specified

sequence of worlds we can arrange for properties of a world which are unchanged through the sequence to be inherited from the earliest world in which they appeared. The other extreme from a context mechanism is to deduce every property from changes that have been made. Some kind of trade-off between these two seems necessary.

The changes in representation combined with incompleteness give rise to changes in reasoning patterns that are crucial to an intelligent support system. These include, determining the outcome of a known action, sifting evidence to determine the state of the world, determining a cause for a situation and determining the details of a known action. In the first case, the system may know that it rained in the Midwest and must predict the effect on the crop. The outcome is unknown but probabilities can be assigned given the time of year and the state of the crop and evidence can be sought to verify the possibilities. In the second case, we may wish to know the crop forecast for Russia but have access to indirect and incomplete information. In the third case, we may require an explanation for the sudden increase in export orders. In the fourth case, we may wish to examine the details of the action "The U.S. exported 6 billion bushels of wheat last year."

Conclusion.

COMEX-0 already represents simple events and the actions which connect worlds. Generic event clusters promise qualitative descriptions of more complex processes if the generic constraint problem can be managed. It is still an experimental question how deductive knowledge is best organized in frames. It is clear that the commodities domain has proved a source of many ideas that are both intuitively appealing and interesting.

References.

Abelson, R.P. 1973, The structure of belief systems. in Computer models of thought and language. Eds. Schank & Colby, Freeman.

Bobrow, D. G. & Winograd, T. An overview of KRL, a knowledge representation language. Cognitive Science vol 1, 1. 1977.

Bullwinkle, C. 1977, Levels of complexity in discourse for reference disambiguation and speech act interpretation. MIT AI Lab memo 413.

Doyle, J., Steele, G. L. Jr. & Sussman, G. J. 1977, AMORD: A dependency-based problem solving language. Proc. SIGART-SIGPLAN Symposium on AI & Programming Languages. 1977.

Epps, M. L. 1975, A simulation of the world coffee economy. In Labys, Quantitative models of commodity markets. Ballinger.

Feldman, J.A. & Sproull, R.F. 1977, Decision theory & AI. II: The hungry monkey. To appear in Cognitive Science.

Fikes, R.E. & Nilsson, N.J. 1971, STRIPS: A new approach to the application of theorem proving to problem solving. Artificial Intelligence vol 2, 189-208.

Goldstein, I.P. & Carr, B.P. 1977, Overlays: A theory of modelling for computer aided instruction. MIT AI memo 406.

Goldstein, I.P. & Roberts, R.B. 1977, NUDGE: a knowledge based scheduling program. MIT AI Lab memo 405.

Hendrix, G.G. 1973, Modelling simultaneous actions & continuous processes. Artificial Intelligence, Vol 4, 145-180.

Hewitt, C. 1975, How to use what you know. MIT AI Lab working paper 93.

Marcus, M. 1976, A design for a parser for english. Proc. ACM.

McDonald, D.D. 1976, Linguistic reasoning during language generation. MIT Technical Report, AI-TR-404.

Minsky, M. 1975, A framework for representing knowledge. in The psychology of computer vision. Ed. Winston, McGraw-Hill.

Moore, J, & Newell, A. 1973, How can MERLIN understand? in Knowledge & Cognition Ed. L Gregg. Erlbaum.

Newell, A. & Simon, H. 1972, Human problem solving. Englewood Cliffs.

Raphael, B. 1970, The frame problem in problem solving systems. Proc. Adv. Study Inst. on Artificial Intelligence & Heuristic Programming, Menaggio, Italy.

Roberts, R.B. $ Goldstein, I.P. 1977 TIMEX: time expert. forthcoming AI memo MIT.

Rosenberg, S.T. 1977, Frames based text processing. forthcoming AI memo MIT.

Skuce, D. 1976, Towards a semantics for a scientific knowledge base. Proc. 1st CSCSI/SCEIO National Conference, Vancouver.

Stansfield, J.L. 1975 Programming a dialogue teaching situation. Ph.D. Thesis Dept. of AI, Edinburgh.

Stansfield, J.L., Carr, B.P. & Goldstein, I.P. 1976, WUMPUS advisor 1: a first implementation of a program that tutors logical and probabilistic reasoning skills. MIT AI Lab memo 381.

Sussman, G.J., Winograd, T. & Charniak, E. 1970, Micro-planner reference manual. MIT AI Lab memo 203.

Winston, P.H., 1970, Learning structural descriptions from examples. MIT AI Lab AI TR-231