26 Computer–based techniques for the representation of automatic problem–solving in archaeology

Juan A. Barceló

26.1 INTRODUCTION

Archaeological theories, like all theories, are symbolic constructs: a representation of artefacts and ecofacts in terms of symbolic or semiologic systems of some sort. The operations involved in the formation and formulation of theories are carried out on symbols, and should have no effect at the representation domain–level. There are many ways of representing or expressing such theories, but in all of them a specific link should exist between the nature of the knowledge we want to represent, and the final representation: the behaviour of the represented system must be analogous to the behaviour of the external entity it represents.

To represent an archaeological theory we have to build the semiologic system according to the architectural specifications of our theory. Those specifications can be defined according to a cognitive metaphor: archaeologists solve problems when they are explaining data, and an archaeological meaning is one of the answers to that problem. Problem–solving is a cognitive mechanism, but some scientists have discovered that it can be analysed using computational techniques (Newell & Simon 1972; Sacerdoti 1977; Pearl 1985; Laurière 1986; Torasso & Console 1988; Brown & Chandrasekaran 1989; Gilhooly 1989). Consequently, archaeological theories may be represented using the techniques that computer scientists use in automatic problem–solving (Figure 26.1).

26.2 DATA *VERSUS* KNOWLEDGE

There are many different meanings for the word *Knowledge*. For instance:

- «Knowledge is a justified true belief» (Waern 1989)
- «Knowledge is the symbolic representation of aspects of some named universe of discourse» (Frost 1986)
- «Knowledge is an organised set of statements of fact or ideas, presenting a reasoned judgement or an experimental result, which is transmitted to others through some communication medium in some systematic form» (Bell 1979).

A piece of knowledge of some universe of discourse may have varying degrees of complexity. To represent an archaeological theory, the simplest knowledge unit is an *empirical datum*, that is to say, a single observation result (Bunge 1983): artefacts and ecofacts we have measured and described during an archaeological excavation are empirical data. The most complex knowledge units in the theory are *interpretations* or *meanings*.

This degree of complexity is determined by the *information content* of a piece of knowledge, and this is a measure of the extent to which that piece of knowledge tells us something that we did not previously know. The amount of information obtained by the receiver of a message is related to the amount by which that message reduces the receiver's uncertainty about some aspect of the universe of discourse (Shannon 1948). Therefore, the information content of an interpretation is greater than the information content of empirical data. This affirmation is easily understandable given the Computer Science definition of the word "problem":

«A goal we want to achieve but we do not know how»

Empirical data do not reduce uncertainty because they represent the initial state in a problem. On

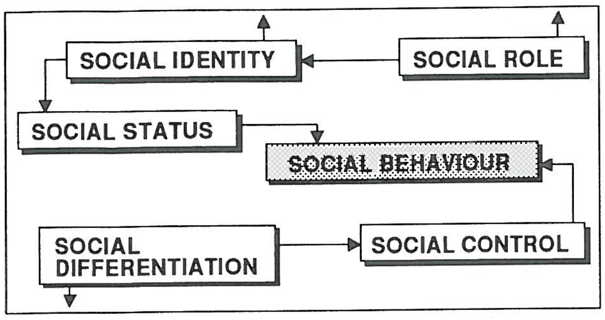

Figure 26.4: A goal–subgoal network.

quently, we represent goals specifying the object and the attribute which slot we want to fill in with a value. The program implementing the theory interprets goal units as requests for information and will attempt to find a value for that attribute.

There is no easy way to find the values of attributes defining theoretical entities: we need hundreds of minor logical calculations using previously instantiated goals. Therefore there is not a single goal in a computational theory, but a very complex tree or network of subgoals. A subgoal is a question on the value of an attribute needed to calculate the unknown value of another attribute. This subgoal network is the result of some analytic *problem decomposition* process; therefore, it has to be considered as an explicit piece of knowledge, and we must implement it computationally before any inference be produced. Figure 26.4 shows a subset of the problem–decomposition analysis needed to implement a standard social theory (Berger *et al.* 1989; Fararo 1989).

Instantiated subgoals represent the problem's *intermediate states*. The ordering of these units has an extraordinary importance, because goal and sub–goal units are linked in taxonomic structures. We say that a unit is a *child* of its parent and is a *descendant* of all units from which its parent *inherits* the value and the name of its attributes. Goals and subgoals are organised into a multiple hierarchy or network, with each element at a lower level *inheriting* the properties of the element at a higher level. These father/son links represent sub–set relationships, where *inheritance* allows for knowledge shared among a set of elements with uniform structure.

Inheritance is based on the concept that goal units tend to form groups and that members within a group tend to share common properties. By using inheritance we can organise our knowledge in such way that allows the discovering of unknown attribute values, because goal attributes are defined only once in some unit, and shared by all the descendants of that unit. This means that each goal will inherit all the attributes of its ancestors, since its parents will inherit the attributes of their parents, and so on. Figure 26.5 shows the code necessary to implement a subgoal network in a commercial Expert Systems Shell.

```
SOCIAL BEHAVIOUR
  inherits from:
    SOCIAL STATUS
    SOCIAL CONTROL
  Actor {inherited from SOCIAL STATUS}
    Characteristics
      [info]-
  Context {inherited from SOCIAL CONTROL}
    Definition
      [info]-
SOCIAL CONTROL
  inherits from:
    SOCIAL DIFFERENTIATION
  is inherited by:
    SOCIAL BEHAVIOUR
  Context
      [inquiry]
SOCIAL DIFFERENTIATION
  is inherited by:
    SOCIAL CONTROL
SOCIAL IDENTITY
  is inherited by:
    SOCIAL STATUS
SOCIAL STATUS
  inherits from:
    SOCIAL IDENTITY
  is inherited by:
    SOCIAL BRHAVIOUR
  Actor
      [info]-
```

Figure 26.5: An example of an inheritance system

26.4 REPRESENTING "EMPIRICAL DATA"

Empirical data are not *knowledge units*; in the same sense, a computational theory is not useful for an archaeologist if it does not include empirical data. Without empirical data a theory is a virtual entity unable to generate an action. If there is nothing to explain, we can explain nothing! We need theories to explain our artefacts, and we need artefacts to use our theories. Data and Theories may arrive at the "Knowledge Level" if we represent them as "potentials to generate an action". That action is to solve a problem by validating a goal, which consists in finding out the unknown values of the attributes that define a computational entity representing the final goal. Then, to transform archaeological artefacts into archaeological knowledge we *have to be able to validate some goal using a representation of them.* In

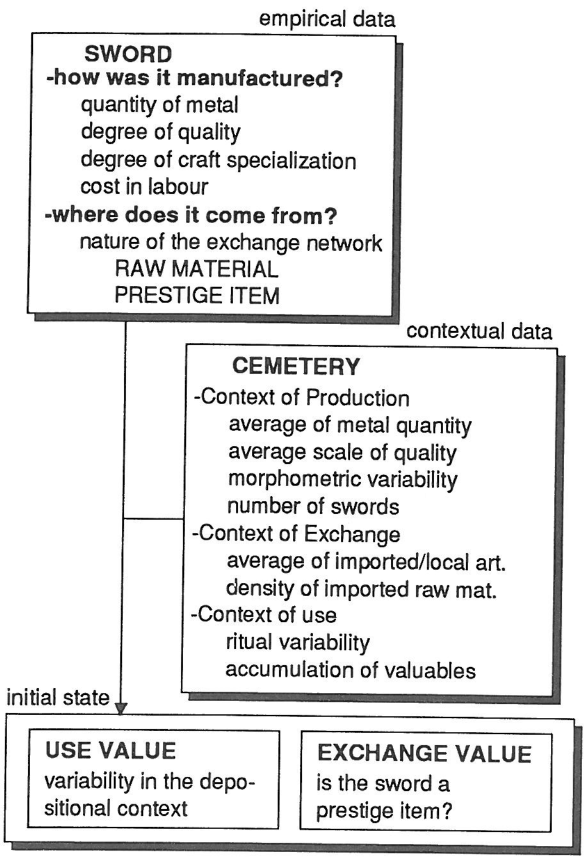

Figure 26.6: Creating a problem initial state from empirical data.

other words, both empirical data and theories have to be integrated into the problem–solving mechanism.

The obvious way to integrate empirical data in a computational theory is by using them in the definition of the problem initial state. A problem state may be defined as the knowledge about a problem available to an archaeologist at a given moment during problem–solving. Then, the initial state is represented by the amount of knowledge available to an archaeologist when he begins solving a problem. It is not reduced to an estimation of the knowledge needed to solve the problem, but a representation of the situation in which has arisen the necessity of solving a specific problem.

In the Problem Architecture presented in previous pages, an Initial State coincides with bottom–level subgoals in the hierarchical goal– network; that is to say, those frames whose attributes cannot be calculated from other frames. Using the previous example about social behaviour, and if we assume that the knowledge available concerns only the technology used in the manufacture of the sword (standardised, non standardised, labour intensive,...), and the related exchange network, then there are two possible attributes to represent the initial state:

- USE VALUE
- EXCHANGE VALUE

Nevertheless, to calculate these values we need to combine knowledge about the sword (a possible prestige symbol), and about the cemetery (the context of use). Figure 26.6 shows a possible integration of both knowledge sources.

We have two different ways to "acquire" the attribute values:

- as explicit declarative units, defined directly by the user
- they have to be read and transformed from external computational databases.

In the first case we are working with a *simulation mechanism*, in the second one with an *automatic problem–solving mechanism*. In this paper, I will only deal with the second one.

A database is a set of observation results described in terms of some generally accepted distinctions. These distinctions represent the criteria we use to organise observations; therefore they have to be structured in some way. There are many different ways of data modelling. The most common, and useful for computational purposes is the *entity–relationship model* (Chen 1976): an entity is an object or thing that exists and can be distinguished from other entities. Entities are described in terms of attributes or properties. A relationship is defined as an ordered list of entity sets. From a structural point of view Empirical Data are represented like goals: frames and database records follow the entity–relationship model. They are n–tuple relations, where n is the number of attributes. Consequently, we may use attributes and relationships in the database as building blocks in the task of calculating the attribute unknown values in bottom–level subgoals.

However, the attributes in the Database have not to be the same terms used as attribute identifiers in subgoal units. We need some inference mechanism as a *bridge* between both computational structures. In the architecture presented in this paper, such a mechanism is represented by a kind of computer program called a *parser*. These programs are sophisticated grammar checkers which prove if an attribute, set of attributes or relationship in the database may be used as a value for any of the attributes in the subgoal unit. There is an important difference between the concepts of "recognition" and "parsing" (Winograd 1983; Frost 1986):

- RECOGNITION is the process of determining whether a given sequence of terminals is a sentence of a given grammar
- PARSING is recognition together with the implicit or explicit construction of the syntax tree.

In other words, our data–parser is much more than a program able to read and transfer data. It is a program that searches for knowledge units and builds new ones. However, it is important to realise that it is a piece of *declarative programming*; the user or programmer must explicitly write the equivalencies between descriptive terms or attributes in the database and subgoal values.

There are some different techniques for writing a parser. In an Expert Systems Shell, parser expressions are usually embedded in the frame structure:

```
subgoal.1
  Attributes:
    number.1
    [inquiry]-TextRead("Object|attribute",ID,
    <filename>)
    number.2
    [inquiry]-TextRead(AttributeName
    (<record|set>),ID,<filename>)
```

where the [inquiry] slots contain the parser expressions. Most Expert System Shells provide specific functions to implement data parsers. In this paper, I use those available in the Mahogany package (Emerald Intelligence, Inc.). In the previous example you can see a call to one of these special functions: TextRead. It performs the following activities:

- TextRead() finds a named database file
- TextRead() selects the records specified by the named record set. If the named record set attribute is empty or the parameter is the keyword "NONE", no records are selected. If the record set parameter is the keyword "ALL", all records in the database are selected.
- TextRead() reads a specified field from each of these records
- TextRead() assigns the values it reads to a receiver attribute or to another named attribute.

The syntax of this function is the following:

```
TextRead(SetName,FieldId,FileName,
[ReceiverName])
```

The first parameter (*SetName*) sets the *name* of a knowledge base attribute whose values are the record numbers that the function will read. These record numbers can be set by an inference process (using rules, inquiry functions, etc.), they may be predefined in the computational theory, or they may be user configured. This parameter can hold a single record number, a range of record numbers, or a set of non–contiguous record numbers. The second parameter (*FieldId*) selects which field in each record is to be read, and fills that value in the slot specified in the fourth parameter (*[ReceiverName]*). The parameter (*FileName*) names the file that holds the empirical data records.

The most interesting cases are those in which the record set and the field identifier are calculated by the parser, and not predefined as an explicit value. In these cases, we have to guide the parser so that it can find the correct value. We use a function called AttributeName, which returns the name of a record attribute in the database. For example, the parser expression:

```
TextRead(AttributeName(<select|site>)
(<db|start>-<db|end>),"Database.fil"
```

returns the value of the attribute "site" in the "select" frame as the record number it has to fill in an [inquiry] slot. The field ID parameter also contains an embedded attribute reference, which computes the column start and end position of the field. In this case, we have two non–theoretical entities: "Select" and "db"; the first one provides knowledge to select the best site to validate a theory. The "db" frame contains two attributes "start" and "end", which contain knowledge about the structure of the database file.

We may also use frames from a different Goal Network. In this case, the values of the record set and the field identifier are the unknown slots in a goal unit. There is a theory whose function is to calculate such values. In other words: we can use a different theory to calculate the specific integration between empirical data and the goal network. "Experimentation" in Natural Sciences or Archaeological Middle–Range Knowledge may be seen as the necessary sub theories to use a higher–level theory.

Sometimes we need more sophisticated parsers, especially when we want to use the theory as a control structure for the search mechanism in the database. For example, we can create a new record set from the records in the database that match some selection criteria. To do that we use the function FindRec. It performs the following activities:

- finding a named database file
- selecting the records specified by the named source record set. If the source record set parameter is the keyword "ALL", then all records in the database are selected. If there are no

valid values in the source record set attribute, no records are selected.

- testing each selected record set against all criteria strings.
- assigning the record number of each record that meets all criteria to the attribute, which holds the results record set. This is the receiver attribute, or the one named in a call to FindRec.
- returning the name of the record set attribute containing the new record set.

The syntax of this function is:

```
FindRec(SetName,CriteriaName,FileName,
[ResultsName])
```

The only difference with the TextRead function is in the *CriteriaName* parameter: the name of the attribute that holds the selection criteria values. Each value of the criteria's attribute includes a database field identifier and a search string. The first part of a criteria's value is a field identifier and the last part, the search string. For example:

```
"<db|site> >4.5 <9.2"
```

selects in a database those archaeological sites with inter–site's distances between 4.5 and 9.2 km.

In sophisticated systems, subgoal units are created at run–time. They do not exist before the system queries the database, but their production mechanism exists as a declarative program. The Instance function creates new frames using information stored in a database. Each of these frames is created as a member of a class, that is to say, as a *child* in the inheritance network. The class must exist before the function is used. In our case, a *class* makes reference to high–level goals, and *instance* to the lowest subgoals in a network: inference starts with the automatic generation of the initial state from some empirical data description.

Instance syntax is the following:

```
Instance(ValueSet,Class,ID Attribute)
```

The first parameter is the *name* of the attribute used as the value set. Each value of this attribute is used to make a separate instance frame. These values, along with the class name ("GOAL"), are used to form the name of the instance, and are used to assign a value to the new instance's ID attribute. The second parameter is the name of a class. This class is used as the inheritance link for each created instance frame. Every subgoal will inherit its characteristic from the goal class we name here. The last parameter names the ID attribute. This must be an attribute that already exists in the class. This is the name of the attribute that is used to assign the unique ID value for each instance. The ID value is taken from the value set, and is used as part of the instance's object name.

The Instance function has to be used together with the other two. Instance creates only an empty frame; its attributes are inherited from the class frame that it belongs to. These attribute values can be filled in using the TextRead and the FindRec functions. An example of a complex parser expression would be:

```
IF
     (1)the SPECIFICATION is "Chronology 9th. cen-
        tury B.C.
THEN
     (1)CRITERIA is "2, Red burnished Pottery"
     (2) SET NAME is
        FindRec(ALL,AttributeName(<criteria>),
        <filename>,AttributeName(<Subset|value>))
     (3)INSTANCE clear is DelMbrs(INITIAL STATE)
     (4)INSTANCE is Instance(<Set|Name>,INITIAL
        STATE,ID)
```

Data parsers are not the best way to integrate empirical data in a computational theory, but they are very useful in our present conditions. The ideal way would be to describe empirical data using the attributes in the Theory Initial State; that is to say, to *impose* a particular way to *see* data. Description is not a self–sufficient mechanism, nor it has universal value. We have to describe in order to be able to do something; descriptions must allow the generation of archaeological meanings, and this would be possible if there is some correspondence between descriptive terms and attributes defining the Initial State. Nevertheless, methodologists have to work with existing databases. It is very difficult to convince an archaeologist that he must rebuild his/her database; then, we are limited to *extract* knowledge, and not to *create* it.

26.5 REPRESENTING "MEANING"

Databases and sets of facts may be seen as theories, if they contain some procedural knowledge able to yield solutions. Nevertheless, empirical data are not useful to obtain answers to "abstract" questions (goals defined by non–observational attributes). Consequently, we need an other kind of knowledge to instantiate the highest–level units in the Goal Network. Theoretical entities and scientific laws are for high–level goals as empirical data are for bottom–level ones. There is a considerable discussion in the philosophy of science about the meaning and usefulness of theoretical entities and scientific laws, and I do not pretend to solve the question. I use these terms as two kinds of computational data structures needed to calculate the unknown values of attributes in high level goal units.

The obvious way to represent scientific law is by means of *production rules* (or Condition/Action pairs), that is to say, computational structures like this one:

```
RULE #1 priority 50 - SOCIAL STATUS DEFINITION
  IF ————————————
     (1) the INITIAL STATE Use Value is "SO-
       CIAL SYMBOL" [threshold 0.20]
  THEN ————————————
     (1) SOCIAL STATUS Actor is "ELITE MEMBER"
       [certainty 0.75]
```

Rules like this one combines known attribute values to calculate unknown values in successive goal units; therefore they link the successive states in a problem–solving mechanism.

Production Rules are well known because of their use in expert systems; some authors have studied also their relevance in scientific modelling (Holland *et al.* 1986; Langley *et al.* 1987; Thagard 1989; Havránek 1990; Gardin 1990), but there is also much criticism about their relevance in scientific knowledge representation (Hofstadter 1992). Nevertheless, all this criticism can be avoided if we do not consider productions as stand–alone particles of meaning. Like attributes, rules are minimal declarative units whose knowledge content depends on the particular structure they have contributed to build. Consequently it is better to see them as "programming instructions", than as a representation of some scientific law. As programming instructions, rules implement an *associative memory*, and it is there where we have to look for knowledge content.

If a "memory" is any device that can store knowledge and recall it on demand, an "associative memory" will be a memory that stores knowledge by associating, or correlating, it with other information in a memory. Therefore, scientific laws and theoretical entities are represented by the pattern of associations, and not only by the associated declarative units, which are meaningless without associations. In this memory, attributes have not any single location, the responsibility for storing and retrieving each declarative unit is shared instead by a number of elements in the memory: the goal units.

To implement that associative memory, production rules are organised in implicit or explicit *clusters of rules* with similar conditions (Holland *et al.* 1986); these are the *relational knowledge units*. The structure given to goal units produces the clustering of rules concerning a particular state or possible interpretation; then, theoretical entities can be defined by identifying multiple rules with similar conditions but varying actions. Consider, for example, a theory in which R_1 is a part of a cluster of rules representing the concept SWORD and R_2 represents the concept PRESTIGE SYMBOL. Then, in the problem–solving mechanism SWORD will invoke PRESTIGE SYMBOL only if R_1 sends a message that activates R_2 ; problem–solving is *directed* from R_1 to R_2 . In more general terms, R_1 influences the activation of R_2 only when the message posted by the action part of R_1 satisfies the condition part of R_2. We will say, then, that rule R_1 is *coupled* to rule R_2 . In other words, we describe one rule as being coupled to another one if the action part of the first rule generates a message that satisfies one or more conditions in the condition part of the second one. Then, when the first rule is active of a given time–step, the second one tends to become active at the next time–step.

Each goal unit acts as the framework for the location of "meanings" in the associative memory; they are *embedded* in that memory because they have to be represented in the *same terms*. Goal units are represented *by* the rule clusters; the frame units are only computational devices to see the output of the system, and not active units in themselves; they act as the external location of goals in the memory. The action part in the rules can be seen as "messages" sent by one goal unit to other goal units; consequently, slots in a given goal unit are filled in by the actions triggered by a previous (lower–level) subgoal in the Goal Network. This behaviour is also a consequence of the associative nature of knowledge units stored in the rule–base, and the associative nature of the retrieval mechanism.

Consequently, archaeological theories are not represented only by rule bases, because rules cannot provide answers alone. We need an explicit representation of the theory *architectural principles*, that is to say, some meta–knowledge about the way to solve the problem. We represent this *architecture* using hierarchy links and dependency paths between goal units. Inheritance links and taxonomic relationships between goal units represent this architecture.

26.6 REPRESENTING "INFERENCES"

Inference is the process by which unknown attribute values are derived from known values. The program that finds these values and makes decisions is called the *inference engine*, and uses the associative knowledge stored in the rule–base, and the theory architectural principles stored as inheritance links between rule clusters.

The associative memory acts as a conditions–to–actions knowledge base where the different goal and subgoal units calculate the values in their slots. The global system looks like a distributed blackboard system, where each frame is no more than a shared data structure used by the members of a specific cluster of rules (Engelmore & Morgan 1988; Jagannathan *et al.* 1989; Murdoch & Johnson 1990). Clusters of rules do not compete among them; the inference engine uses the taxonomic structure built in the Goal Network to guide the activation of the "conditions–to–actions" rule–base instead. This method of inference is called *backchaining* or top–down processing: the program starts by identifying the highest goal unit by reading all dependency paths that lead to that unit. As a result, the inference engine is able to build the Initial State using as building blocks the set of units where dependency paths begin. Once identified, the program runs the data parser to fill in the slots identified. The data parser has to be capable of calculating the highest quantity of low–level attribute values possible. The more comprehensive the Initial State is, the better. If the database we are using is not related to the problem, the resulting Initial State will be incomplete and the program will not be able to run the inference mechanism.

The inference engine uses the Initial State to select the most appropriate higher–level units in the context defined by this Initial State. In other terms, each cluster of rules tries to match their conditions to the attribute values in the Initial State. The action part of the fired rules modifies this set of conditions to allow the selection of the highest–goals in the network. Given the definition of rule clusters as those rules with the same condition part, the selection conditions of each problem–state have to be explicitly asserted. As a result, inference is represented as an accumulation of single decision–making steps. The problem–solving mechanism moves step by step, from Initial States to the Final one, through some Intermediate States, accumulating the relevant knowledge and making further decisions. We call this process *spreading activation.*

Spreading Activation theory was proposed by Collins & Loftus (1975), and has been developed by other authors (Anderson 1983, Holland *et al.* 1986; Newell 1990). This theory asserts that the "activation" of a rule measure how closely associated is the knowledge unit (an attribute value, a goal, a rule) we want to select, to the amount of knowledge available to select it. Most of the times, this "measure" is a numerical index or threshold, and the activation looks like a mathematical continuous function, where the output appears as an increase or decrease of the function values, and the input the sum of activation values in the previously activated units. The resulting level of activation determines the firing of rules and goals, if the adequate threshold values have been introduced in their condition part. The values of the activation function increase or decrease according to the quantity and nature of the rules fired.

Some times it is preferable to use discrete and qualitative activation functions, where the input is not a sum of mathematical values, but an accumulation of declarative units (attributes). In this case, the *spreading activation* is not a mathematical function, but it corresponds to the propagation and "accumulation" of support for goals and rules. Then, instead of threshold values as criteria for firing rules, the associative memory uses a pattern–matching mechanism to decide which productions will apply: a rule is selected for firing when elements accumulated in working memory at a given moment match the elements of the rule's condition.

Bottom–level goals (the problem initial state) provide the "springs" from which spreading activation flows throughout the problem–decomposition network. Activation is spread from various *source nodes*, each of which supports a particular pattern of activation. Consequently, the total activation pattern is the accumulation of the patterns supported by the individual source nodes. Goal units can become source nodes in a number of ways: the lowest–level units receive their input from an external database through the data parser; intermediate units receive input from already validated goal units.

Activation spreads in a computational theory by the following cycle:

1) Bottom–level subgoals make their clustered rules available for matching.
2) The *conditions* of these rules are matched against the input produced by the data parser. A number of rules are fired, and their actions are executed. These actions have to be considered as "messages" sent to the other clusters of goals in the associative memory, representing the next goal in the network,
3) The *conditions* of target clusters are matched against those "messages". A successful match means that the goal represented by the activated cluster of rules is appropriate in the context defined by the Initial State.
4) The actions of the rules in the new cluster are carried out. Pattern–matching starts again, using as target another cluster in the associative memory.
5) Inference spreads until the final state is matched, or the set of attributes in the working

memory cannot fire any other rule. In this case, the goal attached to the last activated rule becomes the final state.

The particular inference engine we are describing here does not support parallel or concurrent spreading activation (see Gupta 1986; Holland *et al.* 1986) because the slots in the current goal have to be calculated before the inference engine process a different (higher–level) one. This is not a limitation due to the programming environment, but a consequence of the representation criteria we are using: inference proceeds by accumulation of activation conditions, therefore, the activation of the problem final state *depends* on the "prior" activation of units above it. Inference steps follow an explicit path, because they are constrained by the architectural principles we have programmed. However, it would be possible to build a computational theory with non–constrained inference paths, in which inheritance links between goal and subgoals were created at run–time by an analogical pattern–matching mechanism (Falkenhainer 1990).

26.7 CONCLUSIONS

In this paper I have proposed to represent computational theories as associative memories with some explicit architectural principles built on them. Free or non–constrained associative memories (Neural networks) are not very useful as scientific representation techniques, although we can integrate them in the pattern matching mechanism.

Many authors have critisised the use of computer programs (specially expert systems technologies) to represent scientific knowledge (Winograd & Flores 1985; Dreyfus & Dreyfus 1986; Penrose 1989). In this paper I have tried to answer to those criticisms proposing an "associative" way of representation. The set of facts, hypotheses, concepts and laws, resides in the pattern of associations between attributes, or representation atoms in this framework. Therefore, archaeological meanings are not represented as finite and discrete units, like a dictionary entry, but in some associative and contextual way, and in terms of the set of relations holding with other units. P. Churchland calls this the *translational content;* it is a result of the inferential/computational relations a goal and the related cluster of rules bears to all the rest of goals and clusters in the problem space (Churchland 1989:43).

Of course, computational theories cannot be considered as the definitive answer to the Philosophy of Science problems: we are constrained by the classical limits of formal systems, and the use of attributes as non–deduced atoms in the system. Nevertheless the Computational Philosophy of Science (Thagard 1988; Churchland 1989) is a very young discipline, and it will take advantage from recent advances in computer science and cognitive psychology.

References

Abelson, R.P. & M., Lalljee
1988 Knowledge structures and causal explanation. In D., Hilton, (ed.) *Contemporary Science and Natural Explanation. Commonsense conceptions of causality.* The Harvester Press, Brighton.

Anderson, J.R.
1983 *The Architecture of Cognition.* Harvard University Press, Cambidge (MA).

Bell, D.
1979 The social framework of the information society. In M.L. Dertouzos and J. Moses (eds.), *The Computer Age: A Twenty–Year View.* The MIT Press, Cambridge (MA).

Bench–Capon, T.J.M.
1990 *Knowledge Representation. An approach to Artificial Intelligence.* Academic Press, London.

Berger, J., M. Elditch, & B. Anderson (eds.)
1989 *Sociological Theories in Progress. New Formulations.* Sage, Newbury Park (CA).

Brown, D.C. & B. Chandrasekaran
1989 *Design Problem Solving Knowledge Structures and Control Strategies.* Pitman, London.

Bunge, M.
1983 *La investigación científica.* Ariel, Barcelona.

Cercone, N. & G. McCalla (eds.)
1987 *The Knowledge Frontier. Essays on the Representation of Knowledge.* Springer–Verlag, New York.

Chen, P.P.
1976 The entity–relationship model. Toward a unified view of data. *ACM Transactions on Database Systems,* 1 (1).

Churchland, P.M.
1989 *A Neurocomputational Perspective. The Nature of Mind and the Structure of Science.* The MIT Press, Cambridge (MA).

Collins, A.M. & E.F. Loftus
1975 A spreading–activation theory of semantic processing. *Psychological Review,* 82:407–428.

Dreyfus, H. & S. Dreyfus
1986 *Mind over Machine*: The Free Press, New York.

Engelmore, R. & T. Morgan
1988 *Blackboard Systems.* Addison–Wesley, Reading (MA).

Falkenhainer, B.
1990 A Unified approach to explanation and theory formation. In J. Shrager & P. Langley (eds.) *Computational models of Scientific discovery and theory formation.* Morgan Kaufman, San Mateo (CA).

Fararo, T.J.
1989 *The meaning of general theoretical sociology. Tradition & Formalization.* Cambridge University Press, Cambridge.

Frost, R.A.
1986 *Introduction to Knowledge Base Systems.* Collins, London

Gardin, J.C.
1990 The structure of archaeological theories. In A. Voorrips (ed.) *Mathematics and Information Science: a flexible framework.* Studies in Modern Archaeology, vol. 3. Holos–Verlag, Bonn.

Gilhooly, K.J. (ed.)
1989 *Human and Machine Problem Solving.* Plenum Press, New York.

Gupta, A.
1986 *Parallelism in Production Systems.* Pitman, London.

Havránek, T.
1990 Rule–based systems ruled out?. In J.E. Tiles, G.T. McKee & G.C. Dean, (eds.) *Evolving Knowledge in Natural Science and Artificial Intelligence.* Pitman, London.

Hofstadter, D.
1992 *Fluid concepts and creative analogies.* Harvester Press, Brighton.

Holland, J.H., K.J. Holyoak, R.E. Nisbett & P.R. Thagard
1986 *Induction. Processes of Inference, Learning and Discovery.* The MIT Press, Cambridge (MA).

Jackson, P., H. Reichgelt, & F. van Harmelen (eds.)
1989 *Logic–based knowledge representation.* The MIT Press, Cambridge (MA).

Jagannathan, V., R. Dodhiawala & L.S. Baum (eds.)
1989 *Blackboard Architectures and Applications.* Academic Press, New York.

Jones, A.
1991 *Logic and Knowledge Representation.* Pitman, London.

Langley, P. H.A. Simon, G.L. Bradshaw & Zytkov, J.M.
1987 *Scientific Discovery. Computational Explorations of the Creative Process.* The MIT Press, Cambridge (MA).

Laurière, J.L.
1986 *Intelligence Artificielle. Resolution de problèmes par l'Homme et la Machine.* Eyrolles, Paris.

Levesque, H.
1984 Foundations of a Functional approach to Knowledge Representation *Artificial Intelligence* 23 (2):155–212.

Minsky, M.
1975 A framework for representing knowledge. In *The Psychology of Computer Vision.* New York McGraw–Hill.

Murdoch, S., & L. Johnson
1990 *Intelligent Data Handling.* Chapman and Hall London.

Newell, A.
1982 The knowledge Level. *Artificial Intelligence* 18: 87–127.
1990 *Unified Theories of Cognition.* Harvard University Press, Harvard.

Newell, A. & H.A. Simon
1972 *Human Problem Solving.* Prentice Hall, Englewood Cliffs (NJ).

Pearl, J.
1985 *Heuristics. Intelligent Search Strategies for computer problem solving.* Addison–Wesley, Reading (MA).

Pernrose, R.
1989 *The Emperor's new mind. Concerning computers, minds and the laws of Physics.* Oxford University Press, Oxford.

Pitrat, J.
1990 *Métaconnaissance. Futur de l' Intelligence Artificielle.* Hermès, Paris.

Reichgelt, H.
1990 *Knowledge Representation. An A.I. Perspective.* Ablex Publ, Hillsdale (NJ).

Ringland, G.A. & D.A. Duce (eds.)
1988 *Approaches to Knowledge Representation.* Research Studies Press, New York (John Wiley Publ.).

Sacerdoti, E.D.
1977 *A structure for Plans and Behavior.* Elsevier, New York.

Shannon, C.E.
1948 A mathematical theory of communication. *Bell Systems Technology Journal*, 27:374–423.

Thagard, P.
1988 *Computational Philosophy of Science.* The MIT Press, Cambridge (MA).

Torasso, P. & L. Console
1988 *Diagnostic Problem Solving. Combining heuristic, approximate and causal reasoning.* Chapman and Hall, London.

Waern, Y.,
1989 *Cognitive aspects of computer supported tasks.* John Wiley Publ, New York.

Winograd, T.
1983 *Language as a cognitive process.* Academic Press, New York.

Winograd, T. & F. Flores
1986 *Understanding Computers and Cognition. A new foundation for design.* Ablex Publ,Norwood (NJ).

Author's address
Juan A. Barceló
Dpt. Història de les Societats Precapitalistes i Antropologia Social
Facultat de Lletres. Edifici B.
Universitat Autonoma de Barcelona.
E–08193 Bellaterra