A PROBLEM REDUCTION MODEL FOR NON INDEPENDENT SUBPROBLEMS

G Levi and F. Sirovich
Istituto di Elaborazione della Informazione
Consiglio Naxionalc delle Ricerche, Pisa, Italy

Abstract

A hypergraph model is introduced, which besides including the AND/OR graph and state space graph models as particulars, is adequate for problem solving tasks involving non independent subproblems. The hypergraph model is shown to be grounded on a nonstandard notion of conjunction such that the truth of a conjunction does not necessarily imply the truth of the conjuncts. A hypergraph search algorithm is given and shown to be equivalent to a resolution-based theorem prover in a first order logic augmented with the special conjunction. A characterization is given of the class of problems requiring the full descriptive power of our model. The class includes problems involving resources, plan formation, simplification of predicate logic programs.

I Introduction

AND/OR graphs arc widely used as problem solving model through problem reduction [/) The assumption underlying both the model and the existing search algorithms is that subproblems can be solved independently, i.e. the solutions to the original problem can be obtained by linear composition of the solutions to the reduced subproblems. The independence assumption yields a nice and clean yet not sufficiently genera) model. Examples have been proposed in the literature *[2-6]* of problems for which the AND/OR graph model is inadequate because the independence assumption is not valid The examples fall into three classes we will try to briefly outline.

The first class consists of problems in which the solution to a subproblem may modify other subproblems. Problems in this class, which will be called *interaction* problems, arise when variables are needed for problem descriptions and subproblem descriptions exhibit common variables

The second class consists of problems whose solution involves the expenditure of scarce resources *(resource* problems) A typical example is the one given by Simon \2\, where John (owning S5000) needs a car(costing $5000| and a yacht (costing $5000) in order to seduce an actress. The independence assumption would lead to a positive (yet erroneous) solution to the problem "Can John seduce the actress[9] ".

The third and more general class consists of problems whose formulation requires the use of concepts defined as conjunctions of non independent terms. Examples of problems in this class, which we call *conjunction* problems, are also given in *[2]* We will give in Section S a detailed description of conjunction problems and show their relevance to artificial intelligence.

Difficulties arising from the AND/OR graph model inadequacy are generally duped by means of problem dependent tricks. The authors have introduced a model [5], based on a generalization of AND/OR graphs, which is adequate for problem reduction in the most general case. This model, which is defined in the next Section, provides a unifying framework for representing the above classes of problems.

2. Problem reduction hypergraphs

A problem solving task is defined as a set *of problems* D and a finite set of *reduction operators* F Set D must contain two distinct elements 1 (bottom) and I (lop) denoting the "undefined solution problem' and the "terminal (trivially solved) problem'', respectively Each reduction operator f_t maps an m_1-tuple of p r o b l $d^1_1, \ldots, d^1_{m_1}$ t o an n tuple of problems $d^2_1, \ldots, d^2_{n_1}$. and M is represented hy the *production*

$$f_t \cdot d^1_1, \ldots, d^1_{m_t} \rightarrow d^2_1, \ldots, d^2_{n_t} .$$

The problems d^1_j (j=1,...,m_t) and d^2_k(k=1,...,n_t) are called f_t-*input* problems and *$f_t$output* problems, respectively.

A problem solving task can be represented by a *problem reduction hypergraph* (prh) H which is defined as a finite directed hypergraph* whose nodes are in one-to-one correspondence with (and labelled by) the elements of D and whose edges are in one-to-one correspondence with (and labelled by) the elements of F. Note that prh's include both AND/OR graphs and state space graphs as particulars. AND/OR graphs are prh's whose edges have a single initial node, while state space graphs are prh's whose edges have both a single initial node and a single final node (hence are edges in the usual sense).

An edge labelled f can be graphically represented by a special node whose input edges come from the nodes labelled by f -initial problems and whose output edges lead to f_i-final problems. Figure I represents the prh corresponding to the problem solving task in Table I When a prh reduces to an AND/OR graph, prh edges and nodes obviously correspond to OR nodes and AND nodes, as defined by Nilsson |/|.

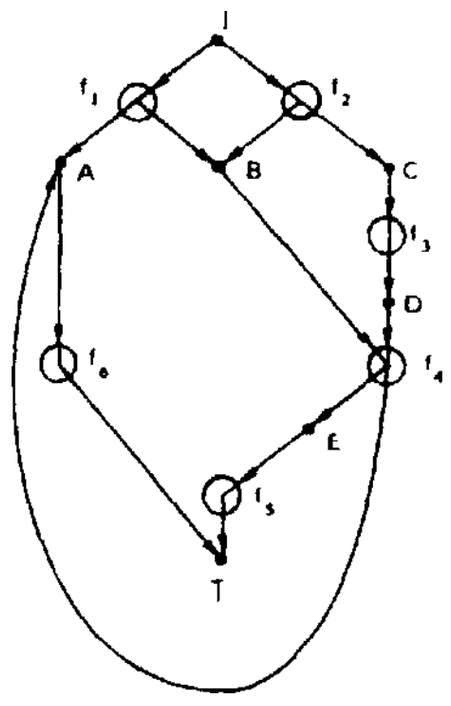

Figure 1 - The hypergraph corresponding to the problem solving task shown in Table 1.

Table 1

$D = \{\perp, \top, A, B, C, D, E\}$
$F = \{f_1, f_2, f_3, f_4, f_5, f_6\}$.
$f_1 : \perp \rightarrow A, B$
$f_2 : \perp \rightarrow B, C$
$f_3 : C \rightarrow D$
$f_4 : B, D \rightarrow E, A$
$f_5 : E \rightarrow \top$
$f_6 : A \rightarrow \top$

Prh's are searched in order to find *solutions,* to show how the untiefined solution problem 1 can be solved by reducing it to the trivially solved problem T Solutions arc defined over the (generally infinite) hypertree 11' *associated* to the prh II and which is obtained by unfolding II. i.c. by re-

* A *finite hypergraph* [7] is a pair H - (N,E), where N is a finite set of *nodes,* and E is a set of subsets (edges) of N. A *finite directed hypergraph* is a pair H=(N,E), where N is a finite set of *nodes,* and E is a set, such that each element (*directed edge*) e_i is an ordered pair (I_i, O_i) of subsets of N. The elements of I_i and of O_i are called respectively *initial* and *final* nodes of e_i.

cursivcly duplicating the output edges of those nodes having more than one input edge, starting from the node labelled by i.

In order to exactly define solutions, let us firstly give the definition of context. A *context* of a prh H is any subgraph c of the hypertree H' associated to H. such that:

i) The node labelled 1 belongs to c,

ii) (or any node n belonging to c, either n has no output edges in c (i.e. n is a *leaf),* or e contains exactly one edge *v* such that n is an initial ndo of e in ll' and all the initial nodes of e in l1' belong, to c,

iii) lor any edge e belonging to c, all the final nodes of c in H' belong to c.

A *solution* is a context such that all its leaves arc labelled by T.

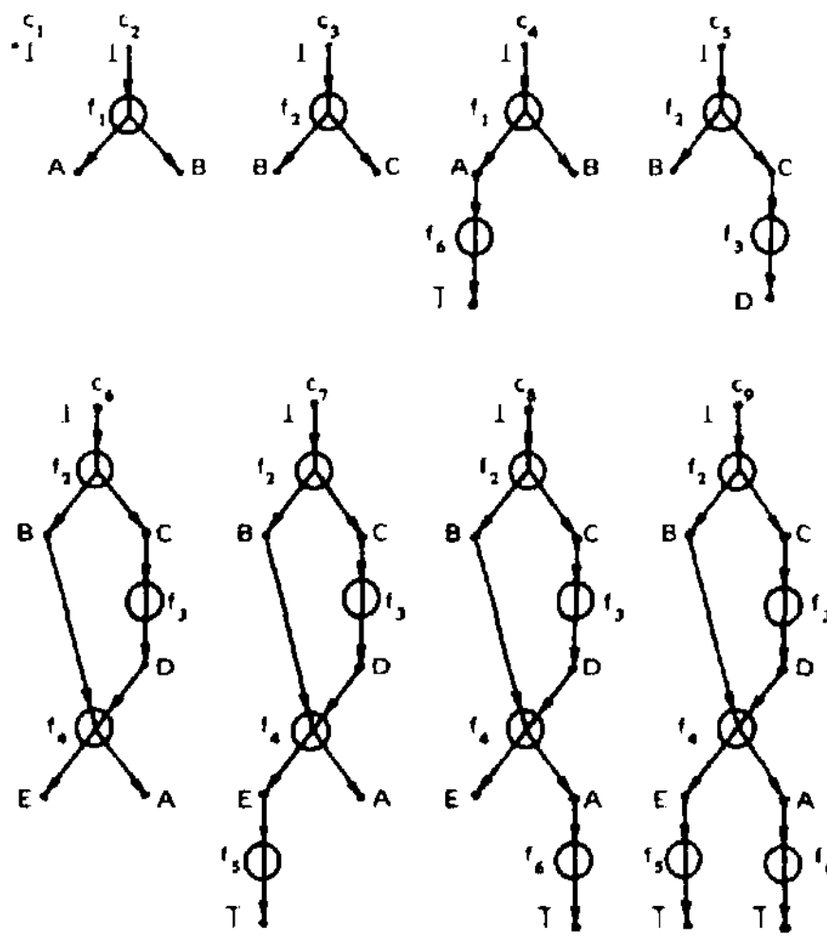

Figure 2 - The contexts of the problem reduction hypergraph shown in Figure 1.

Figure 2 shows all the contexts of the prh in Figure 1. Context c_9 is the (only) solution.

Given a problem solving task, implicitly defining a prh H, a *top-down algorithm* searches the space of contexts of H in order to find a solution. The basic operation of such an algorithm is the *expansion* of a context c_i by applying all the applicable productions. A production f_k is *applicable* to a context c_i iff for each f_k-input problem d_j there exists a separate *selected* leaf of c_i labelled by d_j. Clearly f_k may identify different sets of selected leaves on c_i. For each set S_v of selected leaves a *successor* context is obtained from c_i by connecting the set of nodes in S_v to a set of nodes labelled by the f_k-output problems, by means of an edge labelled by f_k. (For example, context c_6 in Figure 2 is obtained by applying production f_4 to context c_5). The expansion of a context yields the set of all its successor contexts. A context with no successors is either a solution or a *failure*.

An admissible and optimal heuristic top-down algorithm was given in [5]. Here, we will instead describe a *backtrack algorithm*.

Step. 1. Let c_0 be the context consisting of a node labelled by 1, and S be an empty stack.

Step. 2. Expand c_0 to obtain its successor contexts $c_1, ..., c_k$. If no successor contexts are found (i.e. c_0 is a failure) backtrack to Step 3. If one of the successor contexts, say c_j, is a solution, exit with c_j; otherwise set c_0 to c_1, put the successor contexts* $c_2, ..., c_k$ (if any) on stack S and iterate Step 2

Step 3. If S is empty, exit with failure; otherwise set c_o to the head of S, pop S and go to Step 2

When the backtrack algorithm is applied, for example, to the problem solving task of Table 1. the contexts represented in Figure 2 are expanded in the following sequence c_1, c_2, c_4 (failure), c_3, c_5, c_6, c_7. The successor c_9 of c_7 is the solution

* Note that it is sufficient to store on stack S simply the list of the leaves of each context.

.V Interaction problems and nunileU'inunistii algorithms

Problem reduction hypcrgraphs go beyond the capacity of AND/OK graphs with respect to two characteristics

i) The prh reduction operators arc context-sensitive, i.e. the may have more than one input problem,

ii) The prh search space is the space of prh's contexts, while the AND/OR graph search space is the AND/OR graph itself (/. 8. *9]*.

The first characteristic will be discussed later at lenght. In order to show the relevance of the second one, we will consider the interaction problems, i.e. problems whose reduction operators contain variables and therefore must be represented by production schemes A classical example, often used to explain the behavior of PLANNER programs, is the following.

f_1: $1 \rightarrow Fallible(x), Greek(x)$
f_2: $Fallible(x) \rightarrow Human(x)$
f_3: $Human(Turing) \rightarrow T$
f_4: $Human(Socrates) \rightarrow T$
f_5: $Greek(Socrates) \rightarrow T$

Since variables occur in problem descriptions, the application of a production requires a problem description matching operation and results in a unifying substitution associated to the edge corresponding to the production application.

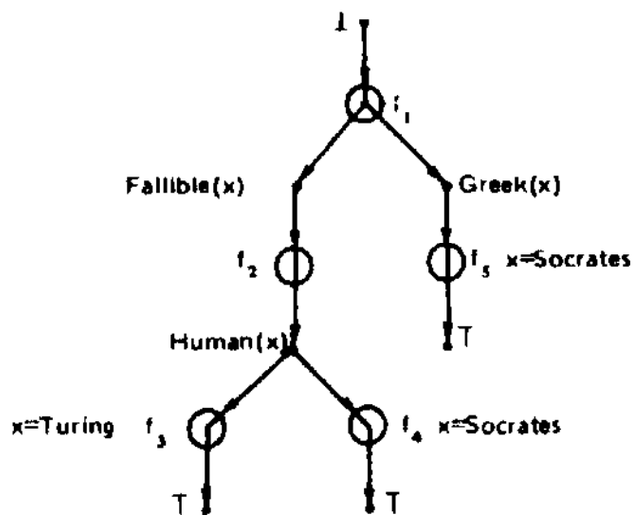

Figure 3 - The problem reduction hyper tree corresponding to the Fallible-Greek problem. Variable bindings are associated to the edges.

The prh describing the example is shown in Figure 3. In this case, the prh is an AND/OR tree and represents also the search space of the AND/OR algorithm. It is clear that the problems *Fallible/x)* and *Greek/x)* cannot be solved independently. Once a solution is found to *Fallible/x)* in which a substitution for variable *x* occurs, the substitution must be carried over to *Greek/x).* A depth-first AND/OR search algorithm will first solve *Fallible/x)* binding x to *Turing* The modified problem *Greek/Turing}* is then unsolvable, thus making unsolvablc the 1 problem. The only way out is to backtrack to the already solved problem *Human/x)* to find a different solution. The need then arises of bringing the search tree back to the state in which *Human/x)* was firstly expanded This may be obtained by *undoing* all the operations performed from that expansion on. In conclusion, the machinery that is needed for correctly handling interaction problems is a substantial running over the AND/OR graph model.

* The algorithm in [10] is the only known AND/OR graph search algorithm whose search space is tomewhat akin to prh's.

Alternatively, the state of the AND/OR tree could be saved every time a nondeterministic expansion occurs. Such a *state saving* leads to a modification of the search space,i.e. to the prh's context space. In the above example, when context c_1 is expanded (see Figure 4) context c_2 is considered for further expansion, while the other successor contexts, among which is c_3, are saved onto the stack. Note that problem interaction is accounted for by applying the unifying substitution to all the leaves of the newly generated context When context c_2 is recognized to be a failure, the search algorithm backtracks to context c_3 whose successor is the solution.

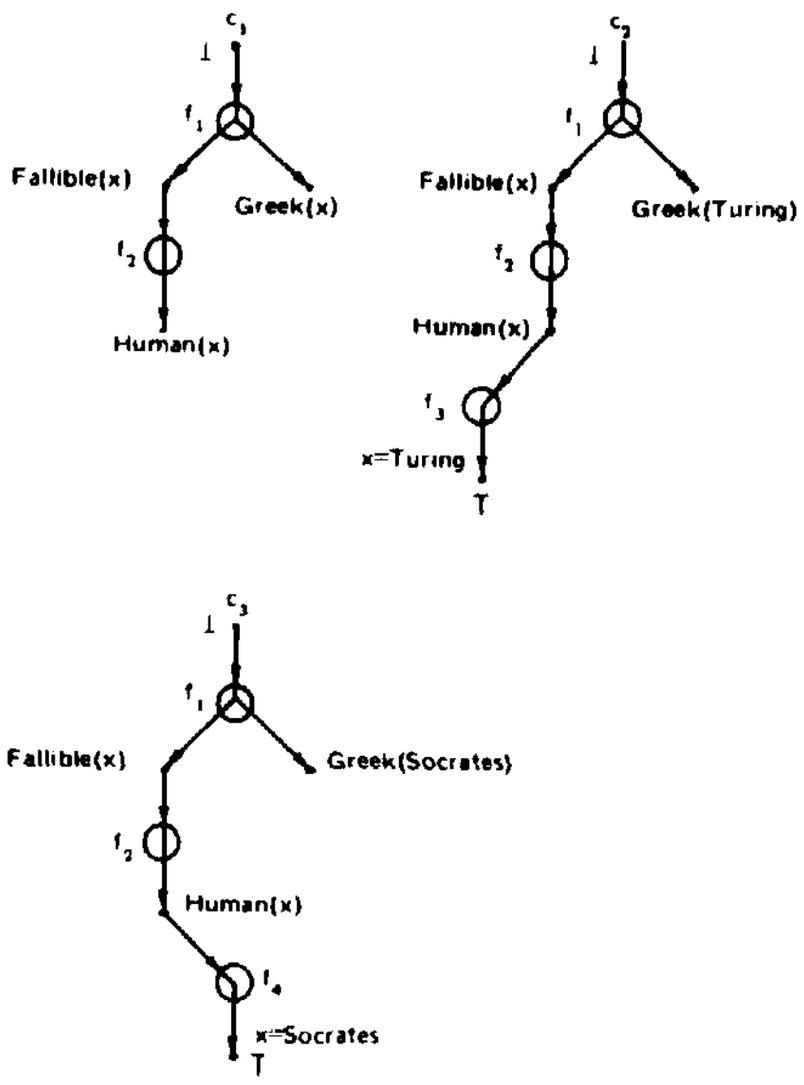

Figure 4 - Some contexts of the prh in Figure 3, which show the state saving mechanism.

The nature of interaction problems, which need either undoing or state saving, shows a close relationship to *nondeterministicalgohthmsThe* search for a solution to a problem defined by a set of productions can be seen as the execution of a nondeterministic algorithm The state of a possible problem solution is described by the leaves of a context, and corresponds to the state of a nondeterministic algorithm computation. Actually, the name "context" has been borrowed from the *context mechanism* used by some languages for artificial intelligence (for example, QA4 [*11]* and MAGMA LISP [12] to allow nondeterministic programming.

4. Resource problems and context dependency

We will now discuss the first prh's characteristic we mentioned above, i.e. the context-dependency of prh reduction operators and show how this feature is crucial to handle resource problems. A resource problem is formulated as a set of "goals" to be achieved through the expenditure of (some of the) limited available resources. The task can be tackled by problem reduction, where reduction operator application may depend upon the availability of some resource and result in the consumption of the resource. Resources are then represented as problems themselves, which act as context for problem reduction. Since the solution of a resource problem does not require the expenditure of all available resources, suitable productions reduce each resource to the terminal problem T. As an example, the actress problem described in Section I can be represented in the following way.

f_1: $\bot \rightarrow$ *Having-$5000, Seduce-the-actress*
f_2: *Seduce-the-actress* $\rightarrow$ *Get-a-car, Get-a-yacht*
f_3: *Having-$5000, Get-a-car* $\rightarrow$ T
f_4: *Having-$5000, Get-a-yacht* $\rightarrow$ T
f_5: *Having-$5000* $\rightarrow$ T

Productions f_3 and f_4 are context-sensitive because they have more than one input problem. Actually, they represent that interaction between problems and the available resource *(Having-$5000)* which makes this problem solving task unsolvable

Let us note that interaction problems can be seen as resource problems. In fact, the state of a variable acts as a "context" for the application of reduction operators and could accordingly be represented as a resource.

An important class of problems that can be handled as resource problems is the class of problems dealing with changing world states, typically robot planning problems, in which world states are modified by reduction operators Each state component [13] describes a slate property Each component may act as a "context" for, and be modified by, the application of reduction operators, hence is represented by a problem. The planning task is reduced to the description of the desired final state while the description of the initial state is reduced to the terminal problem T. This representation is similar to the state-space representation proposed in *[13]* and similarly it is not subject to the frame problem. On the other side, our representation docs not constrain an operator to have a single input problem.

As an example, we will consider the following simple planning problem.

stop: $\bot \rightarrow UP(B1,r),UP(B2,s),AT(R1,t),AT(R2,u)$
go(w,x): $AT(R1,x) \rightarrow AT(R1,w)$
push(b,z,x): $AT(b,x),AT(R1,x) \rightarrow AT(B,z),AT(R1,z)$
lift(b): $UP(b,x),AT(R2,x) \rightarrow AT(b,x),AT(R2,x)$
start: $AT(B1,A),AT(B2,B),AT(R1,C),AT(R2,D) \rightarrow$ T

The initial state of this task is represented by production *start* and consists of two boxes and two robots, at locations *A.B.C* and *D* respectively. Robot *RI* can move around and push bo\es, while robot *R2* can only lift boxes up into a shelf The reduction operator *go(w,x)* describes *RI's* action of moving from location *w* to location *x.* Note that since the task is solved working backward the reduction operators are inverted descriptions of the actual actions. Reduction operator *push/b.z.x)* described K/'s action of pushing box *b* from location *z* to location *x,* and reduction operator *liftfb)* describes *R2's* action of storing box *b* in a shelf. Finally, production *stop* describes the desired state in which both boxes are stored on a shelf Note that action describing productions are similar to STRIPS' operators [*4]*

Figure 5 shows one prh solution to the above planning task. The plan is obtained by "reading back" the prh solution from node T. Note that the plan is not bound to be an action sequence but it is in generat represented by a partial ordering showing that subptans can be done in parallel. The plan obtained by the solution in Figure 5 is shown in Figure 6

We have thus considered the classes of interaction problems and resource problems. The next Section is concerned with the relationship between problem reduction hypergraphs and first order logic theorem proving. We will then be able to introduce the most'general class of problems that can be modelled by problem reduction hypergraphs

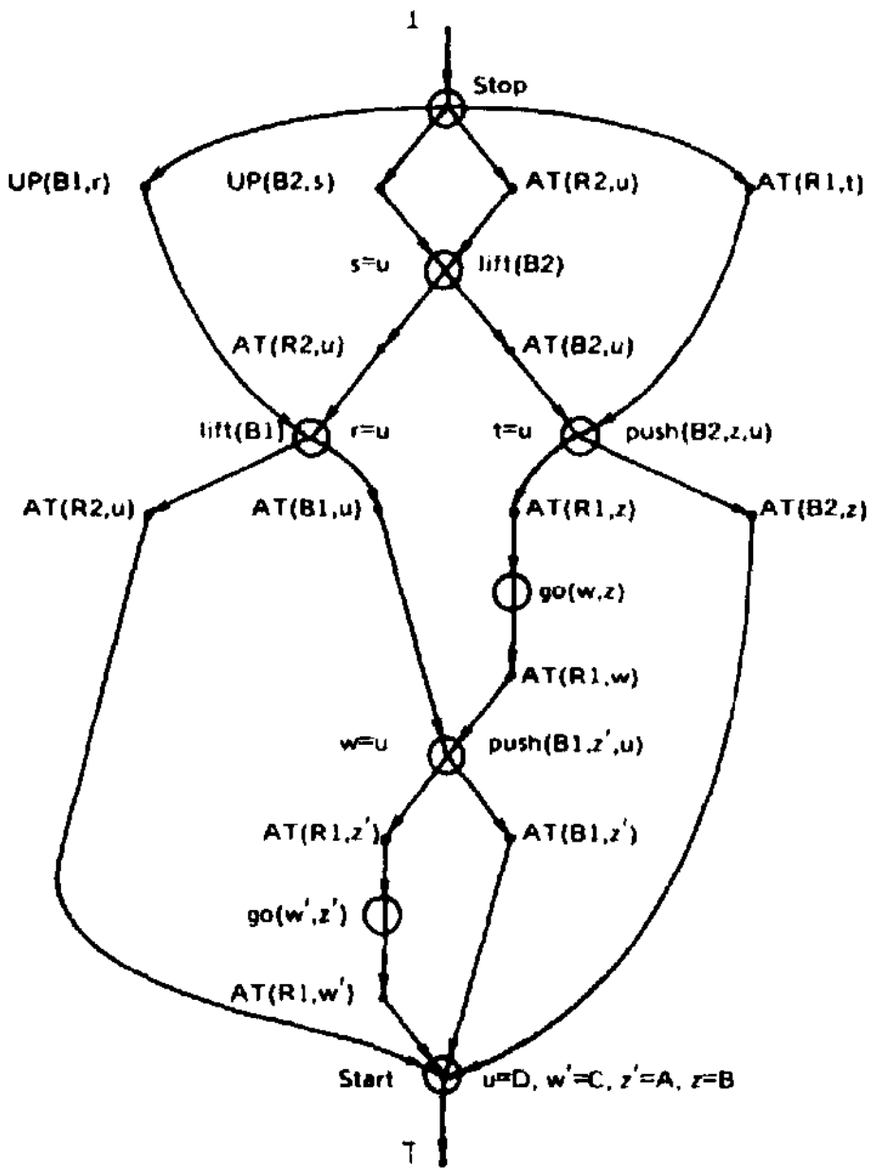

Figure 5 - A solution to the planning problem. Edges are labelled by variable bindings and instantiated actions.

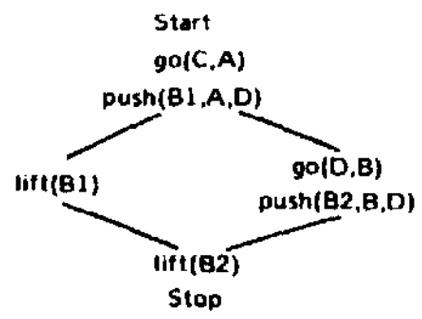

Figure 6 - The plan obtained from the solution in Figure 5.

5. First order logic and conjunction problems

Let us consider the following problem solving task.

k_1: $1 \rightarrow Times(s(0),s(s(0)),u_1), Times(s(0),s(s(0)),u_2), Plus(u_1,u_2,r)$
k_2: $Plus(x,s(y),z) \rightarrow Plus(s(x),y,z)$
k_3: $Times(x,s(y),z) \rightarrow Times(x,y,w), Plus(x,w,z)$
k_4: $Times(x,z,v_1), Times(y,z,v_2), Plus(v_1,v_2,w) \rightarrow Plus(x,y,v_3), Times(v_3,z,w)$
k_5: $Plus(x,0,x) \rightarrow T$
k_6: $Times(x,s(0),x) \rightarrow T$

Problems arc first order logic atomic formulas whose terms are built from universally quantified variable symbols, the only constant symbol *0* (zero), the monadic function symbol s(successor). The symbols 1 and T can be interpreted as the truth values 'false' and 'true' respectively. Any context-free production is therefore a Horn clause |6], whose equivalent clause is the disjunction of the left hand formulas and the negated right-hand for mulas The clauses c_1, c_2, c_3, c_5, c_6 equivalent to k_1, k_2, k_3, k_5, k_6 arc the following.

c_1: $\sim Times(s(0),s(s(0)),u_1) \vee \sim Times(s(0),s(s(0)),u_2) \vee \sim Plus(u_1,u_2,r)$
c_2: $Plus(x,s(y),z) \vee \sim Plus(s(x),y,z)$
c_3: $Times(x,s(y),z) \vee \sim Times(x,y,w) \vee \sim Plus(x,w,z)$
c_5: $Plus(x,0,x)$
c_6: $Times(x,s(0),x)$

Any context-sensitive production can be converted into a *hyper-clause* which is a disjunction of the single *conjunction* of the left-hand formulas, and the negated right-hand formulas. In our example, k_4 is equivalent to

c_4: $(Times(x,z,v_1) \wedge Times(y,z,v_2) \wedge Plus(v_1,v_2,w)) \vee \sim Plus(x,y,v_3) \vee \vee \sim Times(v_3,z,w)$.

The production arrow symbol is interpreted as "is implied by", so that a production of the general form

$$A_1, A_2, \ldots, A_m \rightarrow B_1, B_2, \ldots, B_n$$

is interpreted as $A_1 \wedge A_2 \wedge \ldots \wedge A_m$ *if* $B_1 \wedge B_2 \wedge \ldots \wedge B_n$*.

The productions with the 1 symbol correspond to the clauses obtained from the negation of the well-formed formula to be proved, while the remaining productions correspond to a set of axioms. The application of a reduction operator can easily be seen to correspond to an application of the *resolution principle*. In particular, $1 \rightarrow A,B,C$ and $A,B \rightarrow D$ can be reduced to $1 \rightarrow C,D$, while equivalently the corresponding clauses $\sim A \vee \sim B \vee \sim C$ and $(A \wedge B) \vee \sim D$ resolve, with respect to the pair A and B, to the clause $\sim C \vee \sim D$. A problem reduction hypergraph search algorithm can then be seen as a resolution based, yet non complete, theorem proving strategy.

In a pure first order logic framework, a hyper-clause $(A_1 \wedge \ldots \wedge A_m) \vee \sim B_1 \vee \ldots \vee \sim B_n$ is equivalent to (and would be accordingly transformed into) the set of m clauses

$$A_1 \vee \sim B_1 \vee \ldots \vee \sim B_n$$
$$\vdots$$
$$A_m \vee \sim B_1 \vee \ldots \vee \sim B_n .$$

The equivalence does not apply to problem reduction hypergraphs. This peculiarity, which corresponds to the context-sensitivity feature of prh's, is grounded by a different logic, in which a conjunction does not necessarily imply its conjuncts. In such a logic, "I want cigarettes *and* matches" does not imply "I want matches", thus settling the ambiguity reported by Simon [*2*]. As correctly Simon pointed out, the ambiguity stems from the non independence of the terms of this kind of conjunction.

Such an interdependence is accounted for in problem reduction hypergraphs by left-hand side special conjunctions. The hyper-clause $(A_1 \wedge A_2 \wedge \ldots \wedge A_m) \vee \sim B_1 \vee \ldots \vee \sim B_n$ allows only to assign a denotation to the single *compound* relation $(A_1 \wedge A_2 \wedge \ldots \wedge A_m)$, while the individual conjunct denotations are undefined. On the other hand, the compound relation cannot be represented by a single compound predicate, since the A_i's may occur in other clauses.

The semantics of a problem reduction hypergraph H is strongly dependent upon the above defined semantics for special conjunctions. In fact, if H is transformed according to the standard conjunction semantics, a different nonequivalent problem solving task is obtained. Consider for example the actress task of Section 4, which would be transformed in the following obviously nonequivalent and misleading formulation.

f_1: $1 \rightarrow Having\text{-}\$5000.\ Seduce\text{-}the\text{-}actress$
f_2: $Seduce\text{-}the\text{-}actress \rightarrow Get\text{-}a\text{-}car.\ Get\text{-}a\text{-}yacht$
f_3: $Get\text{-}a\text{-}car \rightarrow T$
f_4: $Get\text{-}a\text{-}yacht \rightarrow T$
f_5: $Having\text{-}\$5000 \rightarrow T$

The *conjunction problems* mentioned in Section 1 are exactly those problems whose formulation requires the above defined special conjunction,

* Note that hyper-clauaei *arc* difti'icnl *from* non-Horn clauses |6| hn-aynr thru left-hind sidcs arc not disjunctions.

in addition to the usual one, i.e problems in which the interdependence among some of the subproblems cannot be ignored. This class includes the class of problems described in Sections3and 4. In fact, in interaction problems, variables introduce a simple kind of interdependence among conjunct subproblems. The context-sensitive productions for resource problems define compound relations among interdependent state components. Therefore, conjunction problems are the most general class of problems with subproblem interdependence The problem reduction hypergraph model is the only known model which is adequate for this class of problems.

Wc will conclude by showing that problem reduction hypergraphs come forth as candidate model for an extension of artificial intelligence goal oriented languages Let us consider Kowalski's predicate logic [15] which is a model of present goal oriented languages The context-free productions shown at the beginning of this Section are actually statements of Kowalski's language. In the example, statements k_2 and k_5, are procedures defining (by means of relations) the function *Plus* over the domain of the integers, defined in terms of the successor function *s* and the constant *0.* Statements A_3 and K_6 are procedures defining the function *Times* and *, is a goal statement requiring to compute *r* as $(1*2) + (1*2)$. Statement k_4 cannot be interpreted tn Kowalski's language and means that $w = (x*z) + (y*z)$ if $w = (x+y)*z$, i.e. represents one possible way of applying the distributivity property This context-sensitive production, which can be seen as a multi-name procedure, is intended to reduce the task of computing an expression of the form $(x*z) + (y*z)$ into the task of computing the simpler expression $(x+y)*z$. This production conveys therefore a pragmatic meaning which can only be expressed in terms of a complex relation involving three non independent predicates. The above example has been introduced in order to show the relevance of context-sensitive productions to the issue of computation optimization

As a final remark, let us note that, similarly to goal oriented languages, a bidirectional search of problem reduction hypergraphs is possible, since a bottom-up search algorithm can easily be defined.

Conclusions

Problem reduction hypergraphs have been shown to be adequate models for problem solving tasks involving interdependent subproblems, and provide a unifying view of problems apparently of different nature, as problems involving variables, resource problems, plan formation, and program simplification problems. The two features which are essential to a correct treatment of conjunction problems are related to the search space and the context-sensitivity of the productions. The search algorithm is a correct and completely general implementation of nondeterminism and minors the state-saving approach to nondeterministic algorithm interpretation. The context-sensitive productions, which intend to augment first order logic with a special kind of conjunction, are a generalization of Horn clauses and provide a basis for an extension of goal oriented laguages.

References

1 Nilsson, NJ*Problem Solving Methods in Artificial Intelligence.* McGraw Hill, New York, 1971.

2. Simon. HA On reasoning about actions, in *Representation and Meaning,* HA. Simon and L. Siklossy, Eds ,414-430. Prentice Hall, Englewood Cliffs, 1972.

3. Ernst, G.W The utility of independent subgoals in theorem proving. *Information and Control 15.* 237-252 (1971)

4 Loveland, D.W. and Shekel, ME. A hole in goal trees Some guidance from resolution theory *Proc. Third Int'l Joint Conference on Artificial Intelligence.* 153161 (1973).

5. Levi, G. and Sirovich, I Gencrali/cd AND/OR graphs and their relation to formal grammars. Nota Interna B73-15 Istituto di Elabora/ionc deirinforma/ione, Pisa (November 1973)

6. Kowalski. R.A Logic for problem solving Memo No 75. Dept of Computational Logic, School of Artificial Intelligence, University of Edinburgh (March 1974).

7. Berge, C. *Graphes et hypergraphs.* Dunod, Paris, 1970

8. Nilsson, N.J Searching problem solving and game playing trees for minimal cost solutions *Information Processing* 68, 11125-130 (1968)

9 Martelli, A and Montanari, U Additive AN[)/OR graphs. P*roe Third Int'l Joint Conference on Artificial Intelligence*, I-11 (1973)

10. Chang, C.I and Slaglc, J.R An admissible and optimal algorithm for searching AND/OR graphs. *Artificial Intelligence 2,* 117-128 (1971)

11 Rulifson, J.F., Derksen. J.A and Waldinger, R.J QA4, a procedural calculus for intuitive reasoning SRI Al Center Technical Note 73 (November 1972)

12. Montangero, C, Pacini, G. and Turini. F MAGMA-LISP A "machine language" for Artificial Intelligence (in these proceedings)

13 VanderBrug.G J. and Mmker, J State-space, problem-reduction, and theorem proving - Some relationships. *Comm ACM 18,* 107 115 (1975).

14 Fikes, R.E and Nilsson, NJ. STRIPS A new approach to the application of theorem proving to problem solving. *Proc Second 1 nt'l Joint Conference on Artificial Intelligence,* 608-620 (1971)

15 Kowalski, R.A. Predicate logic as programming language. *Information Processing 74.* 569-574(1974).