Platypus: a Constraint-Based Reasoning System

William S. Havens* & Paul Stephen Rehfuss

Computer Research Laboratory
Tektronix, Inc
P.O.Box 500 M.S. 50-662
Beaverton, OR 97077

Abstract

Platypus is a constraint-based reasoning engine for synthesis, diagnosis and other recognition tasks. While its target applications are similar to those of many rule-based expert system shells, the architecture of the underlying reasoning engine is not. Platypus is part of our goal at Tektronix of defining an architecture for *Smart Instruments*. In this architecture, a model-based knowledge representation provides rules for constructing object-centered descriptions from the input data. The descriptions produced make explicit the entities recognized in the task domain, their identifying parameters and the semantic constraints that exist among the entities. Platypus is described using a synthesis task, the configuration of Tektronix 4300-series workstations.

1 Introduction

Platypus is a constraint-based reasoning system based on the thesis that many synthesis and diagnosis problems are best viewed as constraint-based recognition tasks. Implemented as an extension to Scheme, and compiling into Scheme code, Platypus provides a portable, efficient, coherent architecture (shell) for a large class of model-based expert systems.

Platypus is motivated by our interest at Tektronix in developing *Smart Instruments*: instrument systems that perform analysis of signals, not just measurement. Smart instruments are active gatherers and analyzers of input, not just passive transducers. As such analysis is inherently domain and application-specific, we expect the software architecture of these instruments to be highly knowledge-based. While the analysis task is fundamentally diagnostic, engineering applications of expert systems include many synthesis (configuration and design) tasks. There are obvious benefits to having a single architecture for both.

* New affiliation: Expert Systems Laboratory, Simon Fraser University, Burnaby, B.C., Canada V5A 1S6.

1.1 Recognition

We view recognition tasks as having the following structure: The knowledge base contains models for task entities, each model specifying possible components, states and relationships for its corresponding entity. The reasoning engine uses these models to interpret input data. An interpretation is a structural description composed of instances of models in the knowledge base representing the recognized entities and their actual states and relationships. So viewed, recognition subsumes both synthesis and diagnosis including many expert system tasks of current interest both in research and industrial applications.

In synthesis, the input is specified as requirements, desires, choices, or measurements. The knowledge base contains models of solution steps, design components and their configurations. The output is a structural description of the synthesized entity. In diagnosis, the input takes the form of signals, measurements or symbolic tokens. The knowledge base holds models of components, faults, or conditions, and the output is a structural description of the faulty situation.

Platypus is designed for "constraint-based" recognition tasks. Constraints are part of the natural ontology for context-sensitive recognition. In the context-free case, recognition degenerates into the weak method "MATCH" used in R1 [McDermott, 1982a]. The relationship between diagnosis and design was noted in [Pople, 1982]. For a description of recognition in scene understanding and other disciplines, see [Havens, 1984]. The whole notion of recognition and representation by partially described schemata goes back at least to Minsky [1975].

1.2 Platypus as an Expert System Shell

Platypus can be viewed as a new form of expert system shell [Havens, 1988]. Rule-based methodology is mature; has had many successes, but its limitations are now apparent. Unstructured knowledge bases and shallow knowledge representations imply *ad hoc* control, lack of meaningful explanations, and knowledge acquisition and maintenance bottlenecks. These problems are due both to the lack of structure in the knowledge base and to a lack of methodology for its construction. Current expert system

shells offer many new features, but at best integrate them at the programming level. For recognition tasks, Platypus implements a more coherent, "knowledge level" [Newell, 1982] architecture. It supports interactive choice and retraction of input; gives a structural description as solution; allows incremental refinement of the solution description; supports default reasoning, and can provide explanations in terms of the semantics of the models in the knowledge base.

2 Platypus Applied to Configuration

Configuration is a well-known application of expert system technology (R1/XSEL [McDermott, 1982a, 1982b], COSSACK [Frayman, 1987]). As systems become more complex, their manufacture involves increasing numbers of mutually constraining options and features. Any mistakes made in configuration by sales or manufacturing engineers are costly both for the manufacturer and for the customer. Configuring the system correctly at the time the order is made is the preferred solution. The generic configuration task has been formally defined as follows:

> *Given a fixed, pre-defined set of components, an architecture that defines certain ways of connecting these components, and requirements imposed by a user for specific cases, either select a set of components that satisfies all relevant requirements or detect inconsistencies in the requirements. [Frayman, 1987]*

There are several things to note in this definition. First, configuration is distinguished from more general design and synthesis tasks by the requirement that components come from a fixed predefined set. Second, configuration is seen as a *satisficing* task. There are multiple competing criteria for optimization (*e.g.* cost and performance), and hence no definition of an optimal solution. Third, a useful configuration system must not only detect inconsistent requirements, but must be able to report the inconsistencies in a helpful way, perhaps suggesting alternatives.

We are building a configurator for Tektronix 4300-series workstations as an initial demonstration of the Platypus architecture. Configuring 4300 workstations is a typical configuration task. There are optional boards, peripherals and expansion cabinets, constraints on placement of boards within cabinets, memory requirements to be met for optional software packages, and installation information needed by the customer. The configurator is designed to be used at the point of sale both to capture the customer's requirements and to complete the resulting order..

In Platypus, there is no separation of these input and verification/completion phases (as in XSEL/R1 or COSSACK). Rather, specification of requirements can be completely incremental, with the opportunity to view the implications of the current set of requirements before adding another. The Platypus architecture guarantees that only the necessary incremental amount of work is done at each step.

Being able to specify requirements incrementally allows us to provide a conceptually simple interface to the user. The configurator always presents a current configuration that is complete and is consistent with current user choices and preferences among defaults. The presentation includes parallel iconic and textual views of the current configuration. The user interacts with the interface by directly modifying the current configuration. After each modification, the configuration is made consistent with the new change or the modification is rejected as inconsistent with previous user choices. As much as possible, user choices are restricted to those consistent with previous choices (by constraint propagation) but this is not guaranteed, and some modifications may be rejected. On demand, rejected or disallowed choices can explained as being inconsistent with specific previous choices.[1] A hand-coded prototype of this interface has been tested and the interface is now being integrated with Platypus.

3 Smart Instrument Architecture

The Platypus architecture will be described using examples taken from the 4300 configurator. The architecture underlying Platypus is a combination of four technologies currently of research interest in Artificial Intelligence. They are 1) a model-based knowledge representation; 2) a rule processor engine; 3) a constraint propagator engine; and 4) a truth maintenance system.

Figure 1 shows a schematic diagram of the overall design of Platypus. The knowledge base contains models for the objects in the task domain represented using a schema (frame, unit) representation. Each schema consists of a set of parameters, a ruleset, a set of component models, and a set of constraints on the model's parameters. Instances of the schemas become the nodes in the composed network description and inherit their properties. Schemas are implemented using the AMOS portable object system for Scheme developed by Adams and Rees [1988].

In Platypus, the rule processor drives the reasoning process. Its task is to interpret the input data in terms of the rules defined in the models. The rule language is chosen to be a conventional horn clause logic similar to Prolog (with some concessions to Scheme). The rule processor uses a unification pattern matcher and a backchaining control strategy. Dependency-directed backtracking (via a straightforward justification-based truth maintenance system (TMS) [Doyle, 1979]) is used for exploring alternative search paths on failure or network inconsistency.

A major aspect of the recognition engine in Platypus is propagation of constraints on model parameters as their values become specified. Our view of reasoning is that search creates hypotheses which provide constraints on the search process itself. Search and constraint propagation form a positive feedback system as illustrated in Figure 2. The choice of an hypothesis provides constraints on the search space which helps choose another hypothesis and so on.

---

1. Dependency information is available that would allow us to indicate in detail what properties of models in the knowledge base imply an inconsistency, but we have no current plans for this or for any sophisticated generation of explanation text.

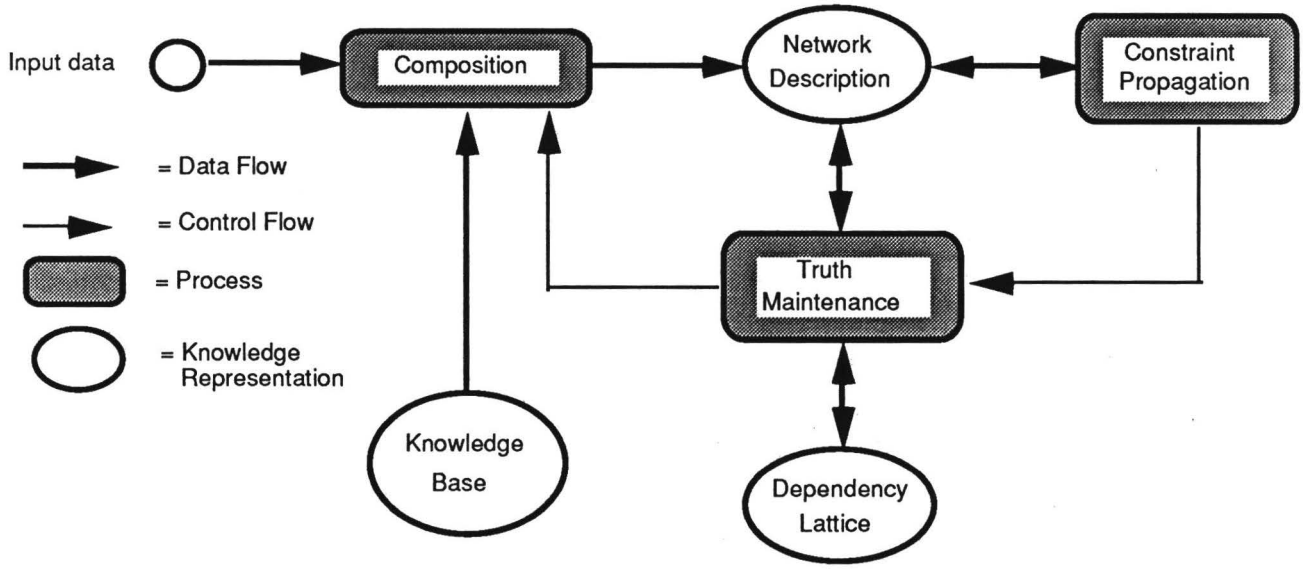

Figure 1: Platypus Overview

The result of rule invocation is the construction of the output description which is a network of schema instances. The network makes explicit the objects recognized in the input data, their identifying parameters and their mutual relationships and provides a persistent intensional representation for the search space. The network consists of nodes which represent parameters or state variables and a set of k-ary constraints over the possible values for the parameters. This representation is natural. To both the rule processor and the user, the network represents instances of domain objects recognized in the input data. The arcs between instances are the semantic relationships among objects known in the knowledge base. The parameters of the instances represent the possible identities of the instance. To the constraint propagator, the network represents a set of state variables, their variable domains and a set of k-ary relations over those domains. The constraint propagator can be viewed as an independent engine which is applied to the network description as it is composed incrementally by the rule processor. The constraint propagator refines the domains of the state variables under the constraints thereby refining the descriptions of the instances in the network.

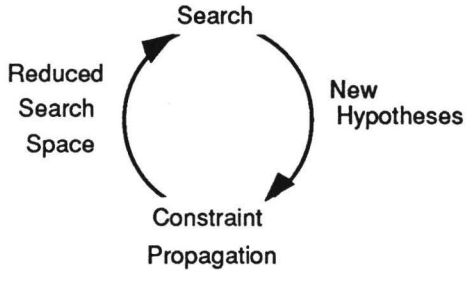

Figure 2: Constraint-Based Recognition Cycle

The Platypus contains an embedded TMS to maintain the consistency of the network description composed by the rule processor. A dependency lattice is built as the historical record of the user choices, assumptions, hypotheses, constraint propagations and their side effects made to the network. The TMS interacts with the rule processor by analyzing inconsistencies and failures and then proposing a dependency backtrack point to the rule processor. The dependency lattice allows the backtrack of both the constraint network and the rule processor as necessary.

4 Recognition and Platypus Knowledge

The architecture outlined above is justified by the belief that recognition tasks have a common underlying organization [Havens, 1984]. These organizing principles include the following observations

4.1 Models

Recognition knowledge is naturally organized as models. This organization is dictated by the ontology of perception. Both the natural world and the world of human artifact are composed of conceptual entities. These entities are grouped into semantically related classes by type. Individual members are identified by their specific attributes and their relationships with other entities.

Platypus attempts to formalize the ontology of recognition by employing a model-based knowledge representation. The knowledge base contains models for the classes of interest in the task domain. Each model represents a class of a particular type whose members all share common identifying characteristics. Members are distinguished only if they differ in the value of some parameter or participate in different component relationships. Each model contains a set of possible components, a small number of parameters and a set of constraints over the

parameters of the model and its components.

As an example, from the 4300 Configurator system, we will examine the model for the physical slots which hold the various electronic circuit boards of the workstation.

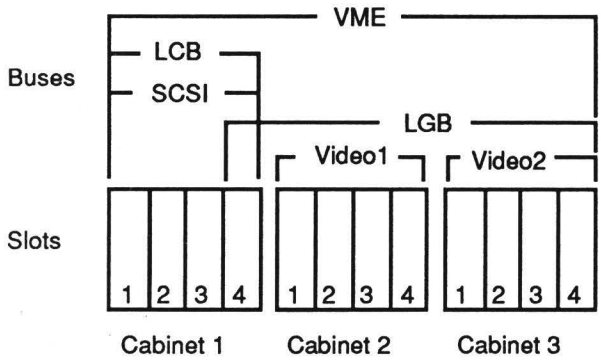

Figure 3: Electrical Busses

The semantics of the slot model is the following: A slot is in position 1, 2,3, or 4 of its containing cabinet. Each slot may contain a single circuit board or be empty. There are various electrical busses each connecting certain slots in certain cabinets (Figure 3). A slot may contain only a board which is supported by the busses resident on that slot. The Platypus representation is called **Slot-Model** and is shown pictorially in Figure 4. The parameters defined in the model are **Pos#** which indicates the position of the slot instance in its cabinet; **Bus** which records which electrical busses reside on this slot; and **BoardSet** which indicates which circuit boards may be placed in the slot. With each parameter is its domain of possible values. The constraint propagator manipulates the parameter domains to ensure constraints hold.

The components of the slot model are **Board, Cabinet** and **Neighbors**. When the model is instantiated, **Board** will contain a circuit board of a type allowed by the **BoardSet** parameter. **Cabinet** will point to the cabinet instance which contains this **Slot-Model** instance. **Neighbors** will contain the **Slot-Model** instances immediately to the left and right of this slot instance or be empty.

The constraints defined in the slot model relate its parameters values to specified parameters of the model's components. In **Slot-Model**, there are five such constraints. We will examine one of these constraints later (§4.4).

Finally, the rules associated with the model are **NewSlot** which creates a new **Slot-Model** instance and installs it in the constraint network and **addBoard** which attempts to place a specified board instance in the slot. An example of a Platypus rule will also be considered later (§4.3).

4.2 Output descriptions and the generative paradigm

The generative paradigm of Chomsky endures as a knowledge representation mechanism. The knowledge base must be finite but account for arbitrary input configurations by producing a correct description for each valid input or reporting failure Recognition is seen as the mapping:

Recognition:
Input × KnowledgeBase ⇒ Description

In Platypus, the output description is structural, a constraint network which makes explicit the entities recognized (represented as instances of the models in the knowledge base), the actual parameters of the instance (thereby characterizing the entity), the component relationship between entities (representing structure) and the constraints existing among entities (specifying the semantic relations present in the description). The constraint network inherits the structure of the models in the knowledge base, in particular each instance in the network inherits the type, parameters and constraints of its parent model. Figure 5 shows part of the constraint network representation for a Tek4336 workstation, illustrating model instances, their parameters and associated value domains, their components and the constraints among parameters.

Type: Slot-Model

Parameters:
Pos# = {1 2 3 4}
Bus = { VME LCB LGB SCSI Video1 Video2 }
BoardSet = { CE BA CP PP ZB Mem8 Mem16 FB4 FB8 FB8S FB12 CM12 }

Components:
Boards: nil
Cabinet: nil
Neighbours: nil

Constraints:
LCB-DaisyChain
LGB-DaisyChain
VideoBusConstraint
SlotCab2Bus
Bus2Board

Rules:
NewSlot
addBoard

Figure 4: Platypus Slot Model

4.3 Rules & Non-Determinism

Search is a necessary part of recognition. Our knowledge of the world is always incomplete and often erroneous. The input data sensed may be ambiguous in its local interpretation and is frequently insufficient to uniquely determine recognition.

In Platypus, these limitations necessitate a reliance on rule-driven search for recognition. Consequently the process of constructing the network description is non-deterministic. Rules are associated with individual models to drive the recognition process for that model.

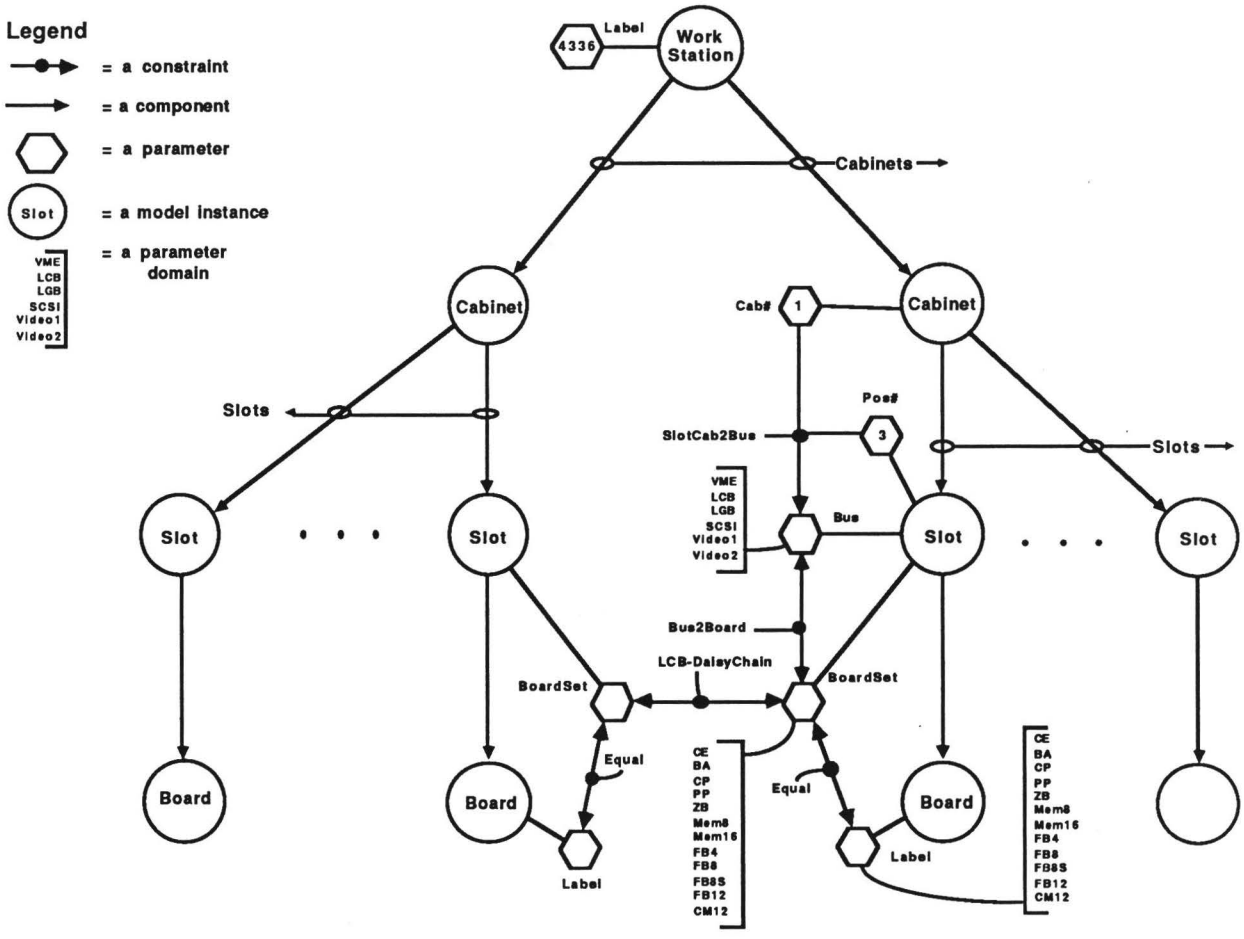

Figure 5: a Tek4336 Network Description

These rules test parameter values and component existence, and change the network description by creating, deleting, specializing or composing model instances, or by constraining parameter values. Non-determinism is implemented via the TMS, which does dependency-directed backtracking of rule side effects.

For example, the **WorkStation** model has a rule called **Option2ARule** for upgrading a 4 bitplane workstation to 8 bitplanes:

> **Option2ARule:**
> Check that the workstation has 4 bitplanes.
> If an FB4 board is present then remove it.
> Create a new FB8 board and attempt to place
> the FB8 in the highest number slot
> possible, starting with the highest
> numbered cabinet.
> Otherwise, fail.

If this rule is applied by the rule processor for some **WorkStation** instance, it will augment the composed constraint network as indicated above. If the augmented network remains consistent, then the rule returns success. Otherwise, an inconsistency is noted by the TMS and it attempts a dependency directed backtrack, choosing either another slot for the FB8 or moving the existing board for some other slot such that the FB8 can be placed consistently.

4.4 Relations, constraints and constraint propagation

Relational knowledge expresses how entities can be associated with one another in any particular network composition. For instance: "The CE board must be in slot 1 of the CEM cabinet" or "The FB8, FB12, CM12 board sequence must be in the highest numbered slots, be consecutively placed and be in the same cabinet".

In Platypus, relational knowledge is expressed as constraints over the parameters of a model and its components. For example, the following constraint called **LCB-DaisyChain** relates the allowed adjacency of circuit boards in the slots of the first cabinet:

> **Semantics:** The local compute bus (LCB) connects the slots of the CEM. The processor (CE), display memory (Mem8 & Mem16) and bus adaptor (BA) boards use this bus to communicate. There is a prescribed ordering to these boards. In particular, following the last BA board must be a memory board, an empty slot or the end of the cabinet. This constraint is expressed as the

following relation:

Constraint:
LCB-DaisyChain $\subseteq$ **Slot.BoardSet** $\times$ **Slot.BoardSet**
LCB-DaisyChain = **{ (CE, BA)(BA, BA)(BA, Mem8)**
(BA, Mem16)(BA, nil)(~BA, ~BA) }

The role of the constraint propagator in Platypus is to incrementally enforce consistency on the output description as it is composed by the rule processor. The constraint propagation algorithm used in Platypus is a variation of hierarchical arc consistency (HAC) [Mackworth, Mulder & Havens, 1985]. The algorithm provides pairwise consistency for k-ary relations in hierarchically organized variable domains. Other stronger forms of consistency could be applied. HAC provides an implementation of "partial matching" as defined by Mittal [1987].

4.5 Constructing interpretations

A consistent constraint network is an intensional description of structure. Human recognition may stop with an incompletely specified representation, but in most applications of a system like Platypus, one wishes eventually to construct extensions or *interpretations* of the description.

In Platypus, a globally consistent interpretation of the current output description need not exist, since the constraint propagation algorithm only guarantees local consistency. If one exists, it can be found by iteratively choosing possible values for not yet fully constrained parameters, and propagating constraints, backtracking in the event of inconsistency.

In choosing these possible values, we use the fact that the problem domain normally has a notion of *preferences* among interpretations and a desire to produce the most preferred one first. In diagnosis tasks, the preference dimensions involve typicality and criticality. Preferences have thresholds and the desire is to produce all interpretations having one or more preference value over its threshold. In synthesis tasks, preference dimensions involve cost and performance. The desire is a single optimal interpretation, but optimality not usually well defined. One thus either satisfices or optimizes a single dimension subject to satisficing the others.

Platypus currently allows expression of a preference as an ordering among the possible values of a parameter. This then defines the order in which the remaining possible values of an not-completely-specified parameter will be chosen by the system in constructing an interpretation. As the description is further specified, such system choices, or *default values*, can be (silently) overridden by user choice or the rule processor. There is currently no provision for describing preferences about the order in which parameters are given default values.

Generating interpretations based on such locally defined preferences is of course a heuristic, and not guaranteed to produce optimal interpretations. Satisficing, however, can be accomplished by incorporating preference thresholds as constraints.

5 Conclusion

The contribution of this work is a description of an expert system architecture based on a coherent theory of how recognition tasks are organized. Platypus was motivated originally as part of our goal of developing smart instruments but is generally applicable to many synthesis and diagnosis expert system tasks. The system has been implemented as a portable extension to the Scheme language. A prototype application for configuring Tektronix 4300-series graphics workstations is being developed as a demonstration of the architecture. Further development of the current system has been postponed until an agreement can be reached with the first author's new organization. Future research will focus on the problems of accommodating noisy data and representing uncertainty within the constraint propagation paradigm.

References

Adams, N. and Rees, J. (1988) *Object-oriented Programming in Scheme*, Proc. ACM Conf. on Lisp and Functional Programming, Snowbird, 1988.

Doyle, J. (1979) A Truth Maintenance System, **Artificial Intelligence 12**, pp.231-272.

Frayman, F. and Mittal, S. (1987) *COSSACK: A Constraints-Based Expert System for Configuration Tasks*, in **Knowledge Based Expert Systems in Engineering: Planning & Design**, ed. D. Sriram & R. A. Adey

Havens, W. S. (1988) *Platypus Constraint Reasoning System: Programming Description*, Technical Report 88-12, Computer Research Laboratory, Tektronix, Beaverton, Oregon 97077, October 1988.

Havens, W. S. (1984) *A Theory of Schema Labelling*, **Computational Intelligence 1**, no.4, National Research Council of Canada, Ottawa.

Mackworth, A. K., Muldar, J. A., and Havens, W. S. (1985) *Hierarchical arc consistency: Exploiting structured domains in constraint stisfaction problems*, **Computational Intelligence 1**, no. 3-4, 1985

McDermott, J. (1982a) *R1: A Rule-Based Configurer of Computer Systems*, **Artificial Intelligence 19**, no. 1, September 1982.

McDermott, J. (1982b) *XSEL: A Computer Sales Person's Assistant*, **Machine Intelligence 10**, 1982.

Minsky, M.. (1975) *A Framework for Representing Knowledge*, in: J. Haugeland (ed.) **Mind Design**, 1981.

Mittal, S. and Frayman, F. (1987) *Making Partial Choices in Constraint Reasoning Problems*, Proc. 6th Nat. Conf. on Artificial Intelligence, Seattle, 1987.

Newell, A. (1982) *The Knowledge Level*, **Artificial Intelligence 18**, 1982

Pople, H. (1982) *Heuristic Methods for Imposing Structure on Ill-Structured Problems: The Structuring of Medical Diagnostics*, in: P. Szolovits (ed.) **Artificial Intelligence in Medicine**