A Theory of Schema Labelling

William Havens

Laboratory for Computational Vision
Department of Computer Science
University of British Columbia
Vancouver, British Columbia
Canada V6T 1W5

Technical Report 84-16
Revised June, 1985

Abstract

Schema labelling is a representation theory which focuses on composition and specialization as two major aspects of machine perception. Previous research in computer vision and knowledge representation have identified computational mechanisms for these tasks. We show that the representational adequacy of schema knowledge structures can be combined advantageously with the constraint propagation capabilities of network consistency techniques. In particular, composition and specialization can be realized as mutually interdependent cooperative processes which operate on the same underlying knowledge representation. In this theory, a schema is a generative representation for a class of semantically related objects. Composition builds a structural description of the scene from rules defined in each schema. The scene description is represented as a network consistency graph which makes explicit the objects found in the scene and their semantic relationships. The graph is hierarchical and describes the input scene at varying levels of detail. Specialization applies network consistency techniques to refine the graph towards a global scene description. Schema labelling is being used for interpreting hand-printed Chinese characters [10], and for recognizing VLSI circuit designs from their mask layouts [2].

1. Introduction

This report describes a synthesis of schema knowledge representations and network consistency recognition techniques. The combined formalism, called *schema labelling*, is a natural union which makes explicit how the objects in the representation correspond to classes of individuals in the world and how these classes can be manipulated for machine perception. We believe this correspondence promotes a natural organization for knowledge which is indigenous to the nature of perception. Consequently, a major goal of this work is epistemological: to characterize how perception is organized and to transfer that architecture to machine perception. This is, of course, a common goal for much knowledge representation research within Artificial Intelligence. Marr [38] argued that the constraints inherent to the perceptual process should be characterized independently of any particular implementation of that process. Newell [41] advocates studying knowledge at the "knowledge level" before considering the form of its representation or how it is to be used. Yet a major aspect of any computational theory of perception is structural. How should knowledge be organized such that it can be effectively represented (in some formalism) and efficiently applied to recognition (by some processor)? By studying the particular knowledge necessary for a perception task, we can hope to infer its form; to understand its architecture.

Schema labelling is described here from the scene analysis perspective, but the technology is applicable to other Artificial Intelligence domains which can be viewed as recognition tasks. Previous network consistency methodology has been applied successfully to scene analysis research [51,32] and its formal properties are well understood [33,34]. Related probabilistic relaxation methods have also been carefully studied [42,22]. However, the technology is mature and has reached its inherent limitations [24]. On the other hand, the capabilities of *schemas* [5] as a computer knowledge representation are still emerging.[1] The term is generic applying to a variety of similar representations including *frame systems* [39,52,12], *scripts* [45], *plans* [1], and schemas [44,23], as well as related *semantic network* representations [53,27,48]. In order for any of these

[1] We will use the terms "schema" and "schemas" for the singular and plural forms respectively.

formalisms to constitute an adequate representational language, their semantics must be formally specified. Hayes [26] has criticized frame systems from this perspective. Woods [53] analysed the semantics of semantic networks and others have made it more precise [46,38,11]. We begin by formally specifying a representation for schema labelling.

1.1. Schema Representations

Schema knowledge representations characterize the perceptual world as composed of objects, events, situations, and abstract configurations of these objects in space or time. Knowledge of this world is object-centered, organized into modular units that act as perceptual models for recognition. Since the "real" world contains an arbitrarily large number of identifiable objects, both concrete and abstract, it is advantageous to group similar individuals into classes. In this theory, a schema is a generative representation for a class of semantically related individuals. The representation is finite but may describe a class containing an arbitrary number of members. Class membership may be known or unknown and is not necessarily unique. The representation is also explicit so that recognition processes may examine and modify the contents of the knowledge base. The knowledge representation contains two types of knowledge: *general knowledge* about classes known to the system and *particular knowledge* about the specific objects appearing in a given scene. This distinction is pervasive in representational formalisms. Object-centered programming languages, beginning historically with Simula [15] and including Smalltalk[21], KRL[6], PSN[31], KLONE[7], Athena[16], and others, define a plethora of datatypes for representing general and specific knowledge.

Likewise, phrase structure grammars utilize a finite set of production rules to represent a potentially unlimited set of sentences. Each instantiation of a rule represents a particular sentential form found in some sentence. We shall define a schema datatype for representing both general and particular knowledge.

A schema *Knowledge Base* (KB) is a static collection of schemas. The KB provides generic models for the various classes of objects known to the system. Each *generic schema* in the KB represents a general class of individuals by specifying its membership in terms of their semantic relationships with other classes. As well, the theory provides a generative mechanism for creating new *derived schemas* from existing generic prototypes. A derived schema represents a particular subclass of the class of its prototype. It is the task of schema labelling to construct a set of derived schemas to represent the objects present in the input data. The actual class membership of each derived schema is constrained by semantic relationships inherited from the prototypes. These schemas with their mutual constraints form a *network consistency graph* (NCG) which provides the output description of the input scene. The NCG is modular, structural and describes the recognized classes and their semantic relationships at multiple levels of detail.

1.2. Composition and Specialization

Schema labelling is characterized as the intimate interaction of two complementary processes: *composition* and *specialization*. Tsotsos [50] has identified these as two intrinsic representational axes for recognition. Complex objects in the world have internal structure and can be represented efficiently as specific legal compositions of their constituent parts. Each member of such a class of objects is assumed to obey the same set of composition rules and, consequently, to have a similar structure. This knowledge, expressed as relationships among a small number of other classes in the knowledge base, is localized and modular. The resulting hierarchy is often called the *composition hierarchy*[35] or *part-of hierarchy*[31][8]. Network descriptions of scene objects produced from this hierarchy retain this hierarchical structure. Recognition processes can exploit this organization by using a combination of top-down and bottom-up search strategies in either goal or data-driven modes[23].

The composition hierarchy developed for the Mapsee2[25] sketch understanding system is shown in Figure 1. In this system, a geographic World is composed of Geo-Systems which represent the landmasses and waterbodies which may appear in a sketch map. Each Geo-System is composed of some combination of Road-Systems, River-Systems, Towns, Shores, Mountain-Ranges, and other interior Geo-Systems. These schemas are, in turn, composed of simpler schemas, finally terminating in the lowest level primitive objects which appear directly in the input image. In the downward direction, the hierarchy represents, for example, that River-Systems are composed of Rivers, Shores, and Bridges. Conversely, in the upward direction, Bridges are part-of both River-Systems and Road-Systems.

Machine perception can also be characterized as a process of specialization from general descriptions of a large class towards refined descriptions of particular individuals. The purpose of specialization is to reduce the overall ambiguity of the scene description derived from incomplete and erroneous data using imperfect knowledge. This process cannot proceed by sequential selection of individual object models to account for the input data until the right model is found. Attempts to ameliorate this problem have used various exception handling mechanisms, notably *similarity links* to recommend a replacement model on failure[39][49][30]. However, such mechanisms, although useful, depend entirely on failure-driven processing and violate the desirable modularity property. Each model should only contain knowledge about its own set of possible members and their allowed compositions.

In schema labelling, classes are merged along the specialization dimension. Classes can be naturally organized using subclass and superclass relationships. We associate a type or *label* with each subclass from a finite set of possible labels for the class called its *labelset*. The labelset segments the class into subclasses by type.[2] Each label names a particular subclass which itself may be a schema present in the KB.

Specialization defines a natural partial ordering of the knowledge base called the *specialization hierarchy* or sometimes the *IS-A hierarchy*. The descendants of each schema in the hierarchy, called *subschemas*, represent subclasses of their parent. The arcs from schema to subschema allow an inheritance of properties and constraints thereby achieving an economy of representation. If the arcs are interpreted as labels signifying distinguishing attributes for each subclass, then the hierarchy becomes a discrimination network. The hierarchy facilitates the associative retrieval of the correct schema to represent a given scene object by a top-down search of its nodes. It is an active process resulting in the construction of the network description of the scene, but nevertheless an associative retrieval from the KB. Although IS-A is a powerful organizational principle for knowledge representation, its imprecise use has been criticised by

[2] The subclasses of a schema are not necessarily disjoint.

Woods[53] and its multiple meanings recently analysed by Brachman[8].

1.3. Network Consistency

In schema labelling, composition and specialization are inextricably intertwined processes. Both processes operate on the same knowledge representation to produce a structural description of the scene as an NCG from rules defined in each schema. The NCG explicitly depicts the objects found in the scene and their semantic relationships. The NCG is hierarchical and describes the input scene at varying levels of detail. Each node in the NCG represents objects either known or hypothesized to exist in the scene. The k-ary arcs among nodes represent constraints on object labelsets which must be synthesized for each possible network composition.

Network consistency techniques are used to refine the NCG towards an unambiguous labelling for each schema. Network consistency is a direct method for representing and manipulating ambiguous network descriptions. By applying the constraints (conceptually in parallel) to the nodes, the scene description is refined towards a globally consistent interpretation. Network consistency techniques have been used to great advantage in computer vision. Waltz [51] used a network consistency algorithm to interpret perfect line drawings of children's toy blocks. The NCG was constructed directly from the input drawing. Picture junctions in the drawing corresponded to corner models over the labelsets of their connecting edges. Following his success, Mackworth [32] showed that network consistency techniques were applicable to task domains more general than understanding the "blocks world". His original *Mapsee* program used network consistency to automatically interpret hand-drawn sketch maps of geographic scenes. Network consistency methods have been extended to allow probability measures to be associated with the labelsets [42] and generalised to work with algebraic constraints [9]. Indeed, constraint-based approaches have been prevalent in computer vision systems [28,49,9,43,4,29]. See [3] for a good review of this technology.

Network consistency has provided a uniform representational framework for scene analysis [33]. Unfortunately, the paradigm is not a panacea for either scene analysis or machine perception in general. Havens and Mackworth [24] identify a number of limitations of the methodology and show that combining it with schema representations can enhance their joint descriptive and procedural adequacy. Schema labelling uses a straightforward combination of schema and network consistency methodology.

1.4. Schema Labelling

An overview of schema labelling is shown in Figure 2. The diagram indicates that the interaction of composition and specialization produces a network description of the input scene. Input to the system is derived from a sensor which samples data from the scene. Segmentation operates on this data and is assumed to produce symbolic image features such as advocated by Marr[38] and to be controlled by procedures local to schema models[52]. It is not necessary that the segmentation be either complete or error free.[3] The subsequent input to the composition process consists of the output from segmentation plus access to the KB. The task of composition is to match these generic schemas in the KB to the image features and produce an NCG of derived schemas as output. The notion of matching used here is analogous to that advocated by Bobrow

[3] For other perceptual tasks, segmentation may be as straightforward, for example, as lexical analysis in natural language understanding or as difficult as phoneme identification in speech recognition.

and Winograd[6]. The schemas in the knowledge base are used to account for the input data while assigning structure to the output description. The parallels between schema recognition and formal parsing algorithms have also been described[23].

The partially constructed NCG provides the input data structure for specialization. The schemas in the NCG represent ambiguously labelled scene objects. Each schema may have one of a number of possible roles in the global scene description. The relations among schemas asserted during composition provide the constraints necessary to remove this ambiguity. Specialization can apply network consistency techniques to the network to produce a global scene description. As well, there is an interchange of control information between composition and specialization. As new schemas are added to the NCG, their consistency with the current global network consistency must be checked. By so doing,.the entire network description may be refined. Constraints on the interpretations of individual schemas may be propagated arbitrarily far in the NCG. Conversely, when the constraints on a schema completely preclude its possibility of contributing to the current network description, then a different composition for the network must be attempted. Standard automatic backtracking algorithms or more sophisticated control structures can be used [17,28]. In the remainder of this paper, we define the schema KB and present a method for constructing the NCG to represent the input scene.

2. Composition

Chomsky's [13] generative paradigm endures as an important organizational principle for knowledge representation. The formalism in which we encode our knowledge about a particular domain must be finite yet capable of characterizing an arbitrary number of inputs. A phrase structure grammar is a well understood mechanism for this purpose. Grammars are generative structural representations for arbitrarily large classes of strings. Parsing is a recognition task which exploits the representational adequacy of grammars. In particular, the grammar is the knowledge base used by the parsing algorithm to construct a hierarchical network description of the input sentence as a parse tree. The parse tree captures the meaning of the sentence by explicitly representing the linguistic objects and their relationships that appear in a given input sentence. The set of strings for which parse trees can be constructed constitute the language of the grammar.

In the theory presented here, schema representations have properties similar to grammars. A schema is a generative and structural representation for a class of semantically related individuals. A schema represents this class as the union of a set of subclasses which satisfy a specified set of constraints. The constraints limit the membership of the class (and recursively all of its subclasses) to those individuals which share specific semantic relationships with other classes, called its *components*. The theory provides a mechanism for generating new schemas from existing ones. Schema labelling is the process of composing and specializing schemas to represent the various objects perceived in the input data. The output is the hierarchical NCG description of these objects and their mutual relationships.

The KB is finite and static. The schemas in the KB describe general classes of objects, their membership, and their allowed relationships with other classes. It will be convenient to refer to schemas by name. For this purpose we associate a unique identifier with each schema. Let the set of names of all the schemas in the KB be an index set, A. A schema, S_i, is in the KB if and only if $i \in A$. The contents of A can be integer subscripts or any other unique set of elements. Normally, we will want to associate character string names as the identifiers for schemas.

Likewise, the NCG consists of a set of schemas derived from the KB and interconnected by semantic constraints. We will refer to these schemas by a second index set, Q. S_i is part of the NCG, if, and only if, $i \in Q$. The elements in Q are also unique identifiers but their names normally will be generated automatically by the system. Note that the KB is distinct from the NCG which it produces, that is, $A \cap Q = \emptyset$.

Every schema in either A or Q has the same form and is represented as:

$$S_i = (m_i, P_i, \alpha_i, \Gamma_i, \Delta_i)$$

where

> $m_i \in A$ is the *prototype* schema for S_i.
>
> P_i is a set of *composition rules* for constructing α_i and Γ_i.
>
> $\alpha_i \subset Q$ is the set of *components* contained in S_i.
>
> Γ_i is a set of *constraints* on Δ_i.
>
> Δ_i is the *labelset* for S_i.

Every schema S_i, $i \in Q$ is derived from a prototype, S_{m_i}, $m_i \in A$. S_i inherits the composition rules, P_i, and labelset, Δ_i, from its prototype. We will use the notation, $\omega(S)$, to denote the class represented by a schema, S. The class represented by S_i is a particular subclass of S_{m_i}:

$$\omega(S_i) \subseteq \omega(S_{m_i}) \tag{1}$$

Conventionally, since every schema has a prototype, if S_i is generic, $i \in A$, then $m_i = i$. Every generic schema is its own prototype.

The components, α_i, of S_i are also schemas in Q. Only derived schemas have components. If $i \in A$, then $\alpha_i = \emptyset$. The notion of component is more than a simple "part of" relationship. The components of S_i are those other derived schemas which have been asserted by composition to be semantically related to S_i in the constructed NCG for the input scene. For a given Q, α_i will contain schemas which are part of S_i, schemas of which S_i is a part, and all of the constructed subschemas of S_i.

The class membership of a schema is structured along the specialization dimension. Δ_i is a discrete and finite set of labels for S_i. Each label, $a_i \in \Delta_i$, is the name of a particular subclass, $\omega(S_{a_i})$, of S_i. S_{a_i} is called a *subschema* of S_i and may or may not be explicitly represented in A or Q. $\omega(S_i)$ is represented as the union of all of its subclasses:

$$\omega(S_i) = \bigcup_{a_i \in \Delta_i} \omega(S_{a_i}) \tag{2}$$

Specialization edits Δ_i under the constraints, Γ_i, applied to its components, α_i, to represent $\omega(S_i)$. Editing involves both deletions and substitutions of the labels in Δ_i.

The schemas in A and their labelsets define the specialization hierarchy for the system. The specialization hierarchy is a static organization of generic classes into their subclasses recursively. It is represented by a hierarchy of schemas and subschemas. Each interior node, i, in the hierarchy corresponds to a schema, S_i, $i \in A$. The descendants of node, i, are the labels, $a_i \in \Delta_i$. If any node, $a_i \in A$, then a_i is also an interior node in the hierarchy and its descendants are recursively the labels contained in Δ_{a_i}.

The specialization hierarchy for a class recursively divides the class into smaller and smaller subclasses. For a finite KB, this process must eventually terminate with singleton classes or with classes that have no schema representation in A . We shall call both such indivisible classes *atomic classes* and represent them only by name as labels in the labelsets of existing schemas. Atomic classes are the leaf nodes of the specialization hierarchy which have no composition and their membership is not constrained by relationships with other classes. In general, if $i \notin A \cup Q$, then $\omega(S_i)$ is unknown. Its membership is homogeneous and must be considered as a whole.[4] However, if S_i is known *a priori* to represent a singleton class, then

$$\omega(S_i) = \{i\} \tag{3}$$

By convention, individuals are represented only by name as singleton classes.

The constraints, Γ_i, are derived from the generic constraints, Γ_{m_i}, defined in the prototype, S_{m_i}. Each constraint, $R_i^u[J] \in \Gamma_i$, is a relation over the labelsets, Δ_j, of a set of schemas, $j \in J$, for $J \subseteq A \cup Q$, $i \in J$. Figure 6 below illustrates a typical set of constraints for the example which follows. If S_i is generic, $i \in A$, then $\forall R_i^u[J] \in \Gamma_i$, $J \subseteq A$ and

$$R_i^u[J] \subseteq \prod_{j \in J} \Delta_j \tag{4}$$

[4] Usually it is unimportant whether the extension of a class is represented in the KB. Instead, the goal of the system is to classify each object present in the input by assigning to it the most specialized subclass possible.

The composition hierarchy for the KB indicates the possible compositions for every generic schema in A. It is a hierarchical digraph where each schema S_i, $i \in A$ is a node in the hierarchy. The arcs between nodes correspond to the possible components for each schema. For every constraint, $R_i^u[J] \in \Gamma_i$, there exists a directed arc from node, i, to every other node, $j \in J$. The actual composition, α_i, for any schema, S_i, $i \in Q$, will contain components derived from the neighbour nodes of its prototype, S_{m_i}, in the composition hierarchy. Not all the possible components of S_{m_i} will appear as actual components in any particular composition for S_i.

Analogous to their counterparts in grammars, the composition rules, distributed among the schemas in A, drive the construction of Q to represent the input scene. We do not specify a language for the rules explicitly. In Mapsee2, the composition rules were embedded in Lisp procedures in each schema. For S_i, $i \in Q$, the responsibilities of P_i are twofold: 1) to specify which combinations of other schemas are valid compositions for S_i given the features present in the input data; and 2) given an α_i, to *compile*[5] new constraints, Γ_i, between its labelset, Δ_i, and the labelsets of other schemas in $A \cup Q$. Each addition to α_i is a new hypothesis about the role of S_i in Q and permits new constraints to be asserted on S_i, given that hypothesis. In other words, the composition rules for a schema allow us to constrain the membership of its class as a function of the membership of other classes to which it is related in the network description of the scene.

[5] "Compile" is used here in the general sense of "to tabulate".

2.1. A Model Railroad

Consider the following hypothetical system to recognize railroad trains from drawings. Figure 3 depicts four types of trains. Each train is composed of an engine followed by a non-empty sequence of cars (called a *carset*) and ending with an optional caboose. Trains are classified as either ***long-haul*** or ***short-haul*** trains. Long-haul trains can be further specialized to the ***express*** and ***freight*** subclasses while short-hauls include *freight, local* and *commuter* subclasses. The type of each train is determined from the mix of cars in its carset, the type of its engine, and whether it has a caboose. Cars may be typed as either *freight-cars* or ***passenger-cars*** which specialize further to ***box, flat,*** *tank* and *hopper* or *coach, mail,* ***observation***, and ***saloon*** respectively. There are two types of engines: powerful ***locomotives (locos)*** and little ***switchers.*** The semantic constraints for each type of train are as follows: An express passenger train has a locomotive engine pulling a mixture of passenger cars. There is no caboose. The commuter train also consists of a sequence of passenger cars but is pulled by a smaller switcher engine. The freight train has a locomotive in front of any sequence of freight cars and ends with a caboose. Finally, the local train is partly passenger and partly freight. It is a small train containing a mixed carset, pulled by switcher type engine, and may or may not have a caboose.

The KB for this example contains schemas for the classes described above:[6]

```
A = { Train,
      LongHaul,
      ShortHaul,
      Engine,
      Car,
      FreightCar,
```

PassCar,
CarSet,
Caboose }

The definitions of these schemas are elaborated in Figure 4. For example, consider the definition of the Train schema. Since every schema in A is generic, Train has no components ($\alpha = \emptyset$) and it is its own prototype ($m = Train$). The schema has two compositon rules ($P = \{p_1, p_2\}$) which are discussed later in Section 2.3. The labelset for Train lists its two subclasses ($\Delta = \{LongHaul, ShortHaul\}$) which are also explicitly represented as schemas in A. The class, $\omega(Train)$, is represented as the union of all long-haul and all short-haul trains:

$$\omega(Train) = \omega(LongHaul) \cup \omega(ShortHaul) \tag{5}$$

The schemas in A and their labelsets define the specialization hierarchy for the railroad. Figure 5 shows this hierarchy for the classes: Train, Car, Engine, and CarSet. Conceptually, we imagine the specialization hierarchy to be a lattice. The top node is the general *Object* schema of which every other schema is a specialization. The single bottom node is the *Null* schema containing no members and being a specialization of all others. The partial ordering between nodes in the lattice is the subclass relation.

Γ for the Train schema specifies the semantic constraints known between Train and the other generic schemas in A. Any schema derived from Train must share these semantic relationships with these other schemas. The arguments to the constraints in Γ are the other schemas in the KB which can be its possible components. For the Train schema, its possible components are CarSet, Engine, and Caboose. Every legitimate

[6] Schemas for the various parts of each car or engine (such as wheel trucks, chassis, couplers, bodies and fittings) are omitted from this example.

Train instance will be composed of some combination of these other schemas. The entire set of constraints for the railroad KB is given in Figure 6.

The composition hierarchy for the railroad is illustrated in Figure 7. Any Train must be composed of an Engine instance followed by a single instance of class CarSet followed optionally by a Caboose. CarSets are composed of a single Car instance or a Car followed by another CarSet instance, thereby allowing an arbitrary length train.

Significantly, knowledge of composition and specialization are isolated. The Train schema is a perceptual model for all trains regardless of their actual type. The same economy of representation also occurs for Engine, Car and CarSet. In this example, the search for a valid scene composition is made efficient by generalizing individual schemas (for example, schemas for Local-Train, Freight-Train, Express-Train and Commuter-Train) into a single class (Train). There is no need to back up and try a different schema if constraints eventually preclude the compatibility of some of its components. For example, attempting to recognize an instance of a Commuter-Train alone will succeed only until the first non-passenger car is discovered. The (perhaps considerable) effort expended so far will be wasted unless the mode of the failure can be used to follow a similarity link to the correct Local-Train train schema. The complexity of such a failure-driven reasoning subsystem is not clear and a simpler approach is advocated here. Whenever individual classes can be merged into larger classes and the resulting ambiguity of interpretation represented by network consistency techniques, then the complexity of recognition is greatly reduced. Mulder [40] is exploring further this phenomenon in a program called *Mapsee3*.

The schema definitions for the ShortHaul and LongHaul classes are also given in Figure 4. The constraints defined in the Train schema have corresponding constraints in its specializations, ShortHaul and LongHaul. They represent a refinement of constraint along the specialization dimension. For example, $R^1_{Train}[\{Train, CarSet\}]$ restricts the type of the train (to either ShortHaul or LongHaul) as a function of the type of the CarSet which it contains (to either PassCarSet, MixedCarSet, or FreightCarSet). That same constraint is manifest in a more refined form in both subschemas. In the ShortHaul schema, $R^1_{ShortHaul}[\{ShortHaul, CarSet\}]$ constrains the type of the train to be either Freight, Local, or Commuter. Likewise for the LongHaul schema, $R^1_{LongHaul}[\{LongHaul, CarSet\}]$ specializes this class to Express or Freight.

2.2. Network Descriptions

Each node in the NCG, Q, is a schema representing a particular collection of objects found in the input scene. The arcs in Q are the semantic constraints which have been asserted to hold between each node and its components. The task of schema labelling is to construct Q such that it adequately accounts for all the input data and every schema in Q is complete. A schema, S_h, $h \in Q$, is *complete* if its composition, α_h, satisfies some rule in P_h and its labelset, $\Delta_h \neq \emptyset$, after applying all the constraints in Γ_h to α_h.

We define a function,

$$New(m, P, \alpha, \Gamma, \Delta)$$

which constructs a new derived schema from its arguments. The new schema has a unique identifier which is automatically added to Q. Although this function can be used to construct the nodes in Q, specifying its arguments for a given target class of objects in the world can be a difficult task. We must collect a finite set of subschemas to cover the membership of the target class. The subschemas may also have to be constructed. Next, we have to compile sufficient semantic constraints on these subschemas to restrict the membership of the new schema to the intended target. To simplify this task, we assume that each new class can be represented as a derived copy of a generic schema from A in conjunction with a set of known generic constraints on other schemas. To this end, the schemas in A all function as prototypes for the schemas in Q. Constructing Q is reduced to deriving new schemas from the these prototypes, searching for valid compositions, and specializing their memberships to correspond to the objects actually present in the input data.

Let $S_h = New(i, P_i, \emptyset, \emptyset, \Delta_i)$, $i \in A$, $h \in Q$. The new S_h inherits the composition rules, P_i, and labelset, Δ_i, of its prototype, S_i. α_h and Γ_h are initially empty and $\omega(S_h)$ is an unidentified subclass of $\omega(S_i)$. The constraints, Γ_i, in the prototype are used to construct the new constraints, Γ_h, incrementally as new components are added to α_h. Each generic constraint, $R_i^u[J] \in \Gamma_i$, $J \subseteq A$, can derive a new constraint, $R_h^u[H] \in \Gamma_h$, $H \subseteq Q$, by letting $R_h^u = R_i^u$ and substituting for every $j \in J$ a corresponding $k \in H$ such that $m_k = j$.

To decide $\omega(S_h)$, we are interested in knowing which other schemas to look for as the possible components for S_h. This knowledge is expressed initially in the composition hierarchy. The set of schemas connected to S_i in the composition hierarchy are the possible components for any particular instantiation of S_h. Search for S_h could proceed "brute force" by looking for instances of these schemas which are themselves complete. If no complete composition can be found for S_h, then it is a bad hypothesis and can be deleted from Q. However, more efficient search methods are possible utilizing both top-down and bottom-up search of the composition hierarchy. In particular, the current components, α_h, limit the set of possible components which need be considered. These issues are closely related to predictive parsing methods [18] but are not considered further here. For more details see Havens [23].

2.3. Building the Railroad

For convenience, the composition rules for the Train, CarSet, and Engine schemas are listed together in Figure 8. In this example, we assume a simple top-down, goal-driven control paradigm with automatic backtrack on failure. The top-down paradigm is popular in logic programming languages such as Prolog [14] and is well understood. The rules for each schema can be invoked to find an instance of that schema in the input data. They are called as subgoals by other schemas in order to complete their own compositions. Each rule returns either a new complete instance on success or failure otherwise.

Consider the rules, p_1 and p_2, for the Train schema which, for convenience, have been expressed in a Prolog-like syntax. Each rule has a single consequent on its *left-hand-side* (LHS) which holds if the conjunction of the predicates on its *right-hand-side* (RHS) can be established. The rules are evaluated in left-to-right order with backtrack on failure.[7] We assume the existence of two functions, *end1* and *end2*, which return the

[7] We assume that the implementation language has the capability to reverse side effects made to the network when backtracking.

respective ends of a car, engine or caboose. As well, there is a predicate, *Coupled*, which is true if the two specified ends are connected in the scene. Rule, p_1, produces a Train composition having an Engine, a CarSet, and a Caboose. Subgoal calls to these schemas appear as predicates in the RHS of the rule. When the rule is invoked, it initially creates a new derived schema from its prototype, *Train*, as described above. The new schema is bound to variable, v_1, in the rule. The labelset for v_1 is $\Delta = \{ShortHaul, LongHaul\}$ indicating that the unrecognized train can be a member of the general class of all trains. α and Γ are both empty and will be incrementally augmented as the rule is further evaluated. p_1 then attempts to find schemas for v_2, v_3 and v_4 such that all the predicates in the rest of the RHS remain satisfied. For example, the subgoal call to the generic Engine schema invokes rule p_5 in that schema passing v_1 as its argument. If p_5 can construct a new Engine (with the Train as a component), then it is returned to p_1 and bound to variable, v_2, in that rule. v_2 is now a new component for the Train which allows p_1 to assert a new constraint, $R^2_{Train}[\{Train, Engine\}]$, between Δ for the Train and Δ for the new Engine. The constraint, given in Figure 6, restricts the class of the Train to be LongHaul if the Engine is a Loco, else a ShortHaul if Engine is a Switcher. Next the predicate, *Specialize*, is called with the Train, Engine, and new constraint as its arguments. *Specialize* adds the Engine to α and the constraint to Γ for the Train. *Specialize* then refines the Train labelset, Δ, under the new constraint and constructs its subschemas for each consistent label in Δ, as necessary.[8] If Δ has been refined, then network consistency is propagated throughout Q. If the Train remains consistent, *Specialize* returns success to

[8] Specialize is considered in more detail in Section 3.

p_1, else it returns failure.

The composition process proceeds, guided by the rule, to look for components, establish new constraints, and call *Specialize* to refine the incrementally growing network. If the RHS of the rule becomes exhausted, then the Train is complete and it is returned as the value of the subgoal. Otherwise failure is propagated back to its caller.[9] The second rule, p_2, represents the alternative composition for the Train schema. No Caboose is part of this composition. These two rules cover all the legitimate compositions for the various example train types. The remaining rules, p_3 through p_5, specify compositions for the CarSet and Engine schemas.

The network, Q, produced for this example is shown in Figure 9. The names of the schemas were generated by appending a unique integer suffix to the name of each prototype. The n-ary arcs among nodes are the constraints compiled by the composition rules. Connected to each node is its final labelset, Δ.

3. Specialization

Network consistency techniques can provide a powerful engine for specialization. In schema labelling, each S_h, $h \in Q$, is a node in the NCG. Its labelset, Δ_h, is the corresponding domain of possible values for the node. The domain is both discrete and finite. Finally, the constraints, Γ_h, provide relations among the nodes. Once the NCG is constructed, an algorithm (such as the *arc consistency* (AC-3) or *path consistency* (PC-2) algorithms of Mackworth [33]) can be applied to refine the network towards a correct scene description. Unfortunately, because the constraints are local, neither arc

[9] In this example, Train is the top-level goal.

nor path consistency can guarantee a globally consistent network. These algorithms operate by discarding from each labelset those labels which do not satisfy a local constraint for some node and therefore cannot possibly be part of any global interpretation. The methodology neither constructs global interpretations nor ensures their existence. Search is then required to verify a global solution.[10]

Freuder [19] has proposed a technique, called *k-consistency*, for synthesizing solutions as a global relation over a set of n-variables given their discrete domains and a set of k-ary constraints over subsets of those variables, $0<k\leq n$. However, the computational complexity of k-consistency grows exponentially in both space and time with the number of variables considered. Basically, the algorithm constructs the power set of all possible relations over the variables (including the desired unique n-ary relation). Seidel [47] has recently shown its complexity to be $O[n^n]$. Clearly, k-consistency cannot be used practically to interpret scenes containing possibly hundreds of objects (and hence variables).

Fortunately for schema labelling, the structure of both the composition and specialization hierarchies can greatly facilitate extending local consistency to global scene interpretations. They cannot eliminate the necessity of search but can ameliorate considerably its complexity.

(1) If the NCG is a tree and, hence, has no cycles, it has been shown recently [34] that arc consistency alone can establish the existence of a global solution in time $O(a^3n)$ and construct all such solutions in time $O(an)$ where a is the uniform

[10] See Freuder [19] for a simple map colouring problem which is both arc and path consistent yet glo-

size of each variable domain. A composition hierarchy is a knowledge structure which attempts to organize the world into a tree of decomposable objects. Each object is limited in its interaction to those other objects which are its component parts and to those more complex objects of which it is part. Although a practical composition hierarchy may not be a strict tree,[11] if we can construct Q from this hierarchy such that it contains as few cycles as possible, then straightforward arc consistency can efficiently refine the instances in Q towards a global labelling.

(2) Alternatively, for an arbitrary NCG, if every node has a single label in its domain, then arc consistency guarantees a unique global solution (which is simply the product of all the singleton domains). By representing the labelset of each schema as a specialization hierarchy of other schemas, the number of labels in each labelset can often be kept very small. In general, the behaviour of AC-3 is $O(a^3 n^2)$ for any graph. By reducing the domain size, a, as much as possible, the behaviour of specialization is again enhanced. A *hierarchical arc consistency* (HAC) algorithm has recently been formalised that exploits the structure of hierarchical labelsets [37]. HAC can manipulate entire subsets of a labelset as a single interior label in the hierarchy thereby frequently obtaining an improvement in efficiency over AC-3.

3.1. Network Consistency

We develop a related form of hierarchical arc consistency for schema labelling by considering which labels in a derived schema's labelset are consistent under the constraints applied to its components. For schema, S_h, $h \in Q$, a label, $a_h \in \Delta_h$, is

bally unsatisfiable.

[11] For example, the Mapsee2 composition hierarchy of Figure 1 has cycles.

consistent under $R_h^u[J]$, $J \subseteq \alpha_h \cup \{h\}$, $h \in J$, if, and only if, it is possible to construct an element of R_h^u from the prototype, m_{a_h}, of a_h and the prototypes, m_{a_j}, of some label, $a_j \in \Delta_j$, for every other component, $j \in J$. For atomic classes, $j \notin A \cup Q$, we assume that $m_{a_j} = a_j$. Atomic classes are their own prototypes. The boolean function, *Consistent* $(a_h, R_h^u[J])$, given as Algorithm 1, implements the above definition:

Consistent provides the basic filtering mechanism necessary for maintaining consistent labelsets. It need be applied to a labelset whenever it is possible that some label may no longer be a consistent member. This situation can arise in schema labelling in two ways: 1) For a schema, S_h, a new constraint, $R_h^u[J]$, has been added to Γ_h by composition, such that $J \subseteq \alpha_h \cup \{h\}$, or; 2) A labelset, Δ_g, for a component, $g \in \alpha_h$, has been refined, removing one or more of its labels and $\exists R_h^u[J] \in \Gamma_h$, $g \in J$. In either case, some label, $a_h \in \Delta_h$, may no longer be consistent. If so, a_h must be deleted from Δ_h and the consistency of every neighbour, S_j, $j \in Q$, such that $h \in \alpha_j$, must also be checked. For this purpose, we define a recursive procedure, *Propagate* (j, h), which is

```
1  Consistent $(a_h, R_h^u[J])$ boolean
2       Let $n = |J|$
3       Return $(\forall j \in J, h \neq j, \exists a_j \in \Delta_j,$
               $(m_{a_1}, \ldots, m_{a_h}, \ldots, m_{a_j}, \ldots, m_{a_n}) \in R_h^u)$
4       End.
```

Algorithm 1: Consistent

listed as Algorithm 2. Given a deletion from Δ_h, *Propagate* checks the consistency of Δ_j for every constraint, $R_j^u[J] \in \Gamma_j$, $h \in J$. If any labels are thus deleted from Δ_j, the procedure is repeated for each neighbour, S_k, $k \in Q$, $j \in \alpha_k$. When *Propagate* terminates, the labelset of every schema in Q will be arc consistent under its constraints or pathologically, every labelset in Q will be empty, indicating an invalid network composition.

3.2. Subclass Constraints

The task of specialization is to refine $\omega(S_h)$ as much as possible for every S_h, $h \in Q$. This process can also add new schemas to Q. From equation 2, $\omega(S_h)$ is represented as the union of all its subclasses, $\omega(S_{a_h})$, for $a_h \in \Delta_h$, such that every a_h is consistent. If S_{a_h} is also a schema, $a_h \in Q$, then $\omega(S_{a_h})$ can also be expressed in

```
Propagate ($j$, $h$)
        Let $\Delta_j' = \emptyset$
        Let $change \leftarrow false$
        For every $a_j \in \Delta_j$,
            If for every $R_j^u[J] \in \Gamma_j$, $h \in J$, $J \subseteq \alpha_j$,
                    Consistent ($a_j$, $R_j^u[J]$)
                then $\Delta_j' \leftarrow \Delta_j' + \{a_j\}$
                else $change \leftarrow true$
        $\Delta_j \leftarrow \Delta_j'$
        If $change$ then
            for every $k \in Q$, $j \in \alpha_k$,
                Propagate ($k$, $j$)
        End.
```

Algorithm 2: Propagate

terms of its own labelset, Δ_{a_h}. In this case, we note that a_h can only be a legitimate label for S_h if it is consistent and $\Delta_{a_h} \neq \emptyset$. In other words, if $\omega(S_{a_h})$ is to make any non-empty contribution to $\omega(S_h)$, then its labelset cannot be empty.

Consequently, we can define within the specialization hierarchy a distinguished *subclass constraint*, $R_i^S[\{i, k\}]$, between S_i, $i \in A$, and all of its non-atomic subschemas, S_k, $k \in \Delta_i$, $k \in A$:

$$R_i^S[\{i, k\}] = \{(a_i, a_k) \mid a_i \in \Delta_i, a_k \in \Delta_k, k = a_i\} \tag{6}$$

The subclass constraints for the model railroad are included in Figure 6. For example, $R_{Train}^S[\{Train, LongHaul\}]$ allows the label, *LongHaul*, for *Train* only if its subclass, *LongHaul*, has at least one label, either *Express* or *Freight*. Likewise for $R_{Train}^S[\{Train, ShortHaul\}]$ where *ShortHaul* is a legitimate label for *Train* only if the *ShortHaul* subclass also has a non-empty labelset.

3.3. Network Synthesis

Finally, the connection between composition and specialization can be elaborated. Whenever composition attempts to add a new schema to the network, every schema of which it is a new component must be specialized. If the augmented network remains consistent, then composition can proceed further. As a side effect, specialization will have refined the membership of those schemas constrained (directly or indirectly) by the new component. On the other hand, if consistency fails from the inclusion of the new component, then the composition is bad. An alternative network composition must be attempted. Here we rely on backtracking on failure to search for alternative compositions although the theory does not require it.

Defined in Algorithm 3 is the boolean procedure, *Specialize* $(h, k, R_h^u[J])$, which is called from a composition rule in P_h whenever a new component, S_k, is added to a schema, S_h, $h \in Q$. *Specialize* first adds S_k to α_h (in step 2) and its associated new

```
1   Specialize $(h, k, R_h^u[J])$ boolean
2       $\alpha_h \leftarrow \alpha_h + \{k\}$
3       $\Gamma_h \leftarrow \Gamma_h + \{R_h^u[J]\}$
4       Let $change = false$
5       Let $\Delta_h' = \emptyset$
6       For every $a_h \in \Delta_h$,
7          If $Consistent(a_h, R_h^u[J])$
8             then
9                If $a_h \in A$ then
10                  Let $S_g = New(a_h, P_{a_h}, \emptyset, \emptyset, \Delta_{a_h})$
11                  $\alpha_h \leftarrow \alpha_h + \{g\}$
12                  $\Gamma_h \leftarrow \Gamma_h + \{R_h^S[\{h, g\}]\}$
13                  Let $H = J - \{h\} + \{g\}$
14                  If $Specialize(g, k, R_{a_h}^u[H])$
15                     then $\Delta_h' \leftarrow \Delta_h' + \{g\}$
16                     else $change \leftarrow true$
17               else-if $a_h \in Q$ then
18                  Let $g = m_{a_h}$
19                  Let $H = J - \{h\} + \{a_h\}$
20                  If $Specialize(a_h, k, R_g^u[H])$
21                     then $\Delta_h' \leftarrow \Delta_h' + \{a_h\}$
22                     else $change \leftarrow true$
23               else   $\Delta_h' \leftarrow \Delta_h' + \{a_h\}$
24      $\Delta_h \leftarrow \Delta_h'$
25      If $change$ then
26         for every $j \in Q$, $h \in \alpha_j$,
27            $Propagate(j, h)$
28      Return $(\Delta_h \neq \emptyset)$
29      End.
```

Algorithm 3: Specialize

constraint, $R_h^u[J]$, for $h, k \in J$, to Γ_h (in step 3). As a consequence, the network consistency of Q may be disturbed. A new empty labelset, $\Delta_h{}'$, is established for S_h (in step 5) and every label, $a_h \in \Delta_h$, is tested for consistency under the new constraint (in steps 6 and 7). $\Delta_h{}'$ is constructed as follows. In order to represent $\omega(S_h)$, *Specialize* must construct all of its subclasses, as subschemas in the specialization hierarchy for S_h as necessary. Every new component of S_h is also a new component of each subschema. Likewise, each new constraint in S_h has a parallel but specialized constraint in the subschema. Also *Specialize* must establish the subclass relation between S_h and each of the derived subschemas.

In this implementation, *Specialize* constructs all subschemas in depth-first order. Recall that a new derived schema inherits its labelset from its prototype. For each consistent label in Δ_h, there are three possibilites (in steps 9, 17 and 23 respectively):

(1) $a_h \in A$ implying that $\omega(S_{a_h})$ has not yet been constructed. *Specialize* creates a new copy, S_g, of S_{a_h} (in step 10) as described in Section 2.2. S_g becomes a new component of S_h and the subclass relation, R_h^S, is asserted between the two schemas (in steps 11 and 12). Next, *Specialize* is called recursively on S_g with S_k as its new component and with a new derived constraint, $R_{a_h}^u[H]$, on its labelset, Δ_g (in steps 13 and 14). If the specialization is successful, then $\Delta_h{}'$ gets the new label, g, for its consistent subschema, S_g (by step 15), else the label is omitted and consistency must be later propagated throughout Q.

(2) In the second case, $a_h \in Q$ indicating that S_{a_h} has already been constructed in a previous call to *Specialize*. A new constraint is derived for S_{a_h} from its prototype,

m_{a_h}, and *Specialize* recursively called on S_{a_h} with its new component, S_k, and new constraint, $R_g^u[H]$ (steps 18 through 20). If successful, then a_h is copied into Δ_h' (in step 21).

(3) In the last case, a_h is a consistent label for S_h but a_h is atomic. Since it does not appear in A, it can not be specialized and $\omega(S_{a_h})$ remains a consistent but unknown subclass of $\omega(S_h)$. a_h is simply copied to Δ_h' (in step 23).

After checking every label in Δ_h, the new labelset, Δ_h', is copied back to Δ_h (in step 24). If the labelset has been changed, then network consistency must be propagated to every neighbour schema, S_j, such that $h \in \alpha_j$ (in steps 25 through 27). Finally, *Specialize* returns true if the new labelset is non-empty and false otherwise (in step 28).

In the example network of Figure 9, specialization constructed subschemas, *ShortHaul*-10, *PassCar*-11, *PassCar*-12 and *FreightCar*-13 and established the final labelsets indicated for each schema. The global scene interpretation for the network can be read by examining each node, its labelset, and the relations among nodes. At top-level, there is a *Train*-1 schema labelled as a ShortHaul class which contains *Engine*-2, *Carset*-4, *Caboose*-3, and *ShortHaul*-10 as its components. The *Engine*-2 schema is labelled as a Switcher. *Carset*-4 is labelled as a MixedCarSet class and contains *Car*-5 (labelled as a PassCar) and *CarSet*-6, which contains *Car*-7 and *CarSet*-8, and so forth. At a finer level of detail, the description of *Train*-1 is specialized to *ShortHaul*-10, which is labelled as a Local train. Likewise, *Car*-5, *Car*-7 and *Car*-9 are specialized respectively to *PassCar*-11, *PassCar*-12 (both labelled as a Coach) and

4. Conclusion

Although the machinery developed here is extensive, the task it solves is a difficult one. How can knowledge about objects in the world be effectively organized and efficiently applied to machine perception? Indeed, much of the research in knowledge representation addresses these two issues and neither can be considered in isolation from the other. We have argued that schema knowledge representations and network consistency constraint propagation techniques can be integrated into a coherent design for machine perception. We presented a formalism for schema representations. Schemas represent both general and specific classes of objects. Each class is represented as the union of a set of subclasses contained in the schema's labelset. Each subclass may also explicitly be represented. Membership in the class is limited by a set of constraints on the labelset of the schema and the labelsets of its components. A schema KB is a static collection of generic schemas. We defined a method of deriving new schemas from their generic prototypes. The goal of schema labelling is to construct a hierarchical structural description of the input scene as an NCG of derived schemas. The NCG makes explicit the objects recognized in the scene and their relationships at multiple levels of detail. Schema labelling identifies composition and specialization as two major aspects of recognition. Composition constructs the NCG by using a set of composition rules distributed among the schemas in the KB. Specialization applies network consistency techniques to refine the membership of each schema to correspond to the objects actually present in the data.

Schema labelling is currently being evaluated in two experimental domains. The first experiment applies schema labelling to the interpretation of hand-printed Chinese characters [10]. Character recognition is an important but difficult problem. The tremendous variability in character size, placement, and writing style makes the recognition of hand-printed characters difficult for traditional pattern recognition techniques. In this system, each schema represents a class of structurally similar characters in terms of its component parts and the spatial constraints that must exist between components. The composition hierarchy for the system defines schemas for characters, their components, called radicals, and the basic strokes from which the radicals and characters are constructed. Segmentation of the input data provides features from which composition can construct schemas for the character, its radicals and their strokes. The spatial relationships present among features allow composition to assert the constraints among the schemas in the NCG. A network consistency algorithm is then used to refine the labelset for each schema towards a unique description for the character.

The second application of schema labelling is the recognition of VLSI circuit designs from their mask layout specifications [2]. The design of integrated circuits remains an art despite recent advances in computer aided design techniques. Very expensive errors proliferate into fabrication despite sophisticated design rule checkers and circuit simulators. Schema labelling provides an alternate approach by recognizing an abstract functional description of the circuit from the topology of the mask layout for the device. The electrical behaviour of the device is not simulated. Instead, a low-level description of the transistors and their interconnections is extracted and used as the input data. For this system, the KB contains schemas for high-level logical functions (such as registers and adders). These objects are composed of flip-flops and boolean gates which are, in turn, composed of an interconnected network of transistors (and possibly other passive devices). Each schema in the KB specifies how the devices in its class can be implemented from its possible components. Given a particular interconnection of components, the constraints in the schema limit the class of the device to the circuit actually implemented in the mask layout.

Acknowledgements

I am grateful to many people in the Laboratory for Computational Vision for their help in preparing this manuscript, particularly Alan Carter, Rachel Gelbart, Lisa Higham, Alan Mackworth, Ted Moens, Jan Mulder, and Ray Reiter. This work was supported by NSERC under grants A5502 and SMI-51, and by the University of British Columbia.

References

1. R. P. Abelson (1975) Concepts for Representing Mundane Reality in Plans, in *Representation and Understanding,* D.G. Bobrow & A. Collins (eds.), Academic Press, N.Y., pp.273-309.
2. A. Alon & W. Havens (1984) Recognizing VLSI Circuits from Mask Artwork by Schema Labelling, Tech. Report 85-1, Dept. of Computer Science, Univ. of British Columbia, Vancouver, Canada (in preparation).
3. D. H. Ballard & C. M. Brown (1982) *Computer Vision,* Prentice-Hall, Englewood Cliffs, N.J.
4. H. G. Barrow & J. M. Tenenbaum (1976) MSYS: A System for Reasoning about Scenes, Tech. Note 121, AI Center, SRI International, March 1976.
5. F. C. Bartlett (1932) *Remembering,* Cambridge Univ. Press, Cambridge, England.
6. D. G. Bobrow & T. Winograd (1977) An Overview of KRL: A Knowledge Representation Language, *Cognitive Science 1, no. 1.*
7. R. J. Brachman (1979) On the Epistemological Status of Semantic Networks, *Associative Networks,* N. Findler (ed.) Academic Press, N.Y., p.3.
8. R. J. Brachman (1982) What ISA is and isn't, *Proc. Canadian Soc. for Computational Studies of Intelligence,* Saskatoon, Canada, May 1982, p.212.
9. R. Brooks, R. Cereiner & T. Binford (1979) The ACRONYM Model-Based Vision System, *Proc. IJCAI-79,* Tokyo, Japan, Aug. 1979, p.105.
10. T. Bult (1985) Schema Labelling Applied to Hand-Printed Chinese Character

Recognition, M.Sc. Thesis, Dept. of Comp. Science, Univ. of British Columbia, Vancouver, Canada (in preparation).

11. N. Cercone & L. Schubert (1975) Towards a State-Based Conceptual Representation, *Proc. 4-IJCAI,* Tbilisi, USSR, pp.83-90, Sept. 1975.
12. E. Charniak (1975) Organization and Inference in a Frame-Like System of Knowledge, *Proc. Theoretical Issues in Natural Language Processing,* Cambridge, Mass., June 1975.
13. N. Chomsky (1957) *Syntactic Structures,* Mouton and Co.
14. W. Clocksin & C. Mellish (1981) *Programming in PROLOG,* Springer-Verlag, New York.
15. O. Dahl & K. Nygaard (1976) SIMULA - An Algol-Based Simulation Language, *CACM 9,* Sept. 1976.
16. C. A. Dent & R. G. Smith (1983) A Guide to ATHENA: A Knowledge Representation Language, DREA Tech. Memo 83/G, Dept. of National Defence, Dartmouth, N.S., Canada.
17. J. Doyle (1979) A Truth Maintenance System, *Artificial Intelligence 12,* pp.231-272.
18. J. Earley (1970) An Efficient Context-Free Parsing Algorithm, *CACM 13,* no.2, pp.94-102.
19. E. C. Freuder (1978) Synthesizing Constraint Expressions, *CACM 21, no. 11,* November, 1978, pp.958-966.
20. E. C. Freuder (1976) A Computer System for Visual Recognition using Active Knowledge, AI-TR345, MIT AI Lab, Cambridge, Mass.
21. A. Goldberg & D. Robson (1983) *Smalltalk-80: The Language and its Implementation,* Addison-Wesley, Reading, Mass.
22. R. M. Haralick & L. G. Shapiro (1979) The Consistent Labelling Problem: Part I, *IEEE Trans. PAMI 1, no. 2,* April 1979, pp.173-184.
23. W. Havens (1983) Recognition Mechanisms for Schema-Based Knowledge Representations, *Int. Journal Computers and Mathematics 9, no. 1,* Pergamon Press, pp.185-199.
24. W. Havens & A. Mackworth (1983) Representing Knowledge of the Visual World, *IEEE Computer 16, no. 10,* October, 1983, pp.90-96.
25. W. Havens, A. Mackworth, & J. Mulder (1985) The Mapsee2 System: Representational Adequacy for Computational Vision, (in preparation).
26. P. Hayes (1981) The Logic of Frames, in B. Webber & N. Nilsson (eds.) *Readings in Artificial Intelligence,* Tioga Publishing, Palo Alto, CA., pp.451-458.
27. G. Hendrix (1975) Expanding the Utility of Semantic Networks through Partitioning, *Proc. 4-IJCAI,* Tbilisi, USSR, pp.115-121, Sept. 1975.
28. M. Herman & T. Kanade (1984) The 3D MOSAIC Scene Understanding System: Incremental Reconstruction of 3D Scenes from Complex Images. CMU-CS-84-102, Dept. of Computer Science, Carnegie-Mellon Univ., Pittsburgh, PA., Feb. 1984.
29. G. Hinton (1979) Relaxation and its Role in Vision, Ph.D. thesis, Univ. of Edinburgh, Edinburgh, Scotland, Dec. 1979.
30. Y. Lesperance (1980) Handling Exceptional Conditions in PSN, *Proc. Canadian*

Society for Computational Studies of Intelligence, Victoria, Canada, May 1980, p.63.

31. H. Levesque & J. Mylopoulos (1979) A Procedural Semantics for Semantic Networks, in N. Findler (ed.) *Associative Networks,* Academic Press, N.Y., p.93.
32. A. K. Mackworth (1977) On Reading Sketch Maps, *Proc. 5-IJCAI,* MIT, Cambridge, Mass.,pp.598-606, August 1977.
33. A. K. Mackworth (1977) Consistency in Networks of Relations, *Artificial Intelligence 8, no. 1,* February, 1977.
34. A. K. Mackworth & E. C. Freuder (1985) The Complexity of Some Polynomial Network Consistency Algorithms for Constraint Satisfaction Problems, *Artificial Intelligence 25,* pp.65-74.
35. A. K. Mackworth & W. S. Havens (1981) Structuring Domain Knowledge for Visual Perception, *Proc. 7-IJCAI,* Univ. of British Columbia, Vancouver, Canada, August 1981, p.625.
36. A. K. Mackworth (1983) Recovering the Meaning of Diagrams and Sketches, *Proc. Graphics Interface '83,* Edmonton, Canada, May, 1983, pp.313-317.
37. A. Mackworth, J. Mulder & W. Havens (1985) Hierarchical Arc Consistency: Exploiting Structured Domains in Constraint Satisfaction Problems, Tech. Report 85-7, Dept. of Comp. Science, Univ. of B.C., Vancouver, Canada.
38. D. Marr (1982) *Vision,* W.H. Freeman, San Francisco, CA.
39. M. Minsky (1975) A Framework for Representing Knowledge, in *The Psychology of Computer Vision,* P. Winston (ed.) McGraw-Hill, N.Y.
40. J. Mulder (1985) Representing Ambiguity and Hypotheticality in Visual Knowledge, Ph.D. thesis, Dept. of Computer Science, U. of British Columbia, Vancouver, Canada (in preparation).
41. A. Newell (1982) The Knowledge Level, *Artificial Intelligence 18, no. 87.*
42. A. Rosenfeld, R. Hummel, & S. W. Zucker (1976) Scene Labelling by Relaxation Processes, *IEEE Trans. Systems, Man, & Cybernetics, SMC-6, no. 6,* June 1976, pp.420-433.
43. S. Rubin (1980) Natural Scene Recognition using Locus Search, *Computer Graphics & Image Processing 13, no. ??????*
44. D. E. Rumelhart & A. Ortony (1976) The Representation of Knowledge in Memory, TR-55, Cent. for Human Info. Processing, Dept. of Psych., Univ. of Calif. at San Diego, LaJolla, CA.
45. R. C. Schank (1975) The Structure of Episodes in Memory, in *Representation and Understanding,* D.G. Bobrow & A. Collins (eds.), Academic Press, N.Y., pp.237-272.
46. L. Schubert (1975) Extending the Expressive Power of Semantic Networks, *Proc. 4-IJCAI,* Tbilisi, USSR, p.158, Sept. 1975.
47. R. Seidel (1984) On the Complexity to Achieve K-Consistency, unpublished tech. report, Dept. of Computer Science, Cornell Univ., Ithaca, N.Y.
48. J. F. Sowa (1984) *Conceptual Structures: Information Processing in Mind and Machine,* Addison-Wesley, Reading, Mass.
49. J. Tsotsos (1981) Temporal Event Recognition: An Application to Left Ventricular Performance Assessment, *Proc. IJCAI-81,* Vancouver, Canada.

50. J. K. Tsotsos (1984) Representational Axes and Temporal Cooperative Processes, RCBV-TR-84-2, Dept. of Computer Science, Univ. of Toronto, Toronto, Canada, April 1984.
51. D. L. Waltz (1972) Generating Semantic Descriptions from Drawings of Scenes with Shadows, AI-TR-271, M.I.T. A.I. Lab, Cambridge, Mass.
52. T. Winograd (1975) Frame Representations and the Procedural-Declarative Contraversy, in *Representation and Understanding,* D.G. Bobrow & A. Collins (eds.), Academic Press, N.Y., pp.185-210.
53. W. A. Woods (1975) What's in a Link: Foundataions for Semantic Networks, in *Representation and Understanding,* D.G. Bobrow & A. Collins (eds.), Academic Press, N.Y., pp.35-82.

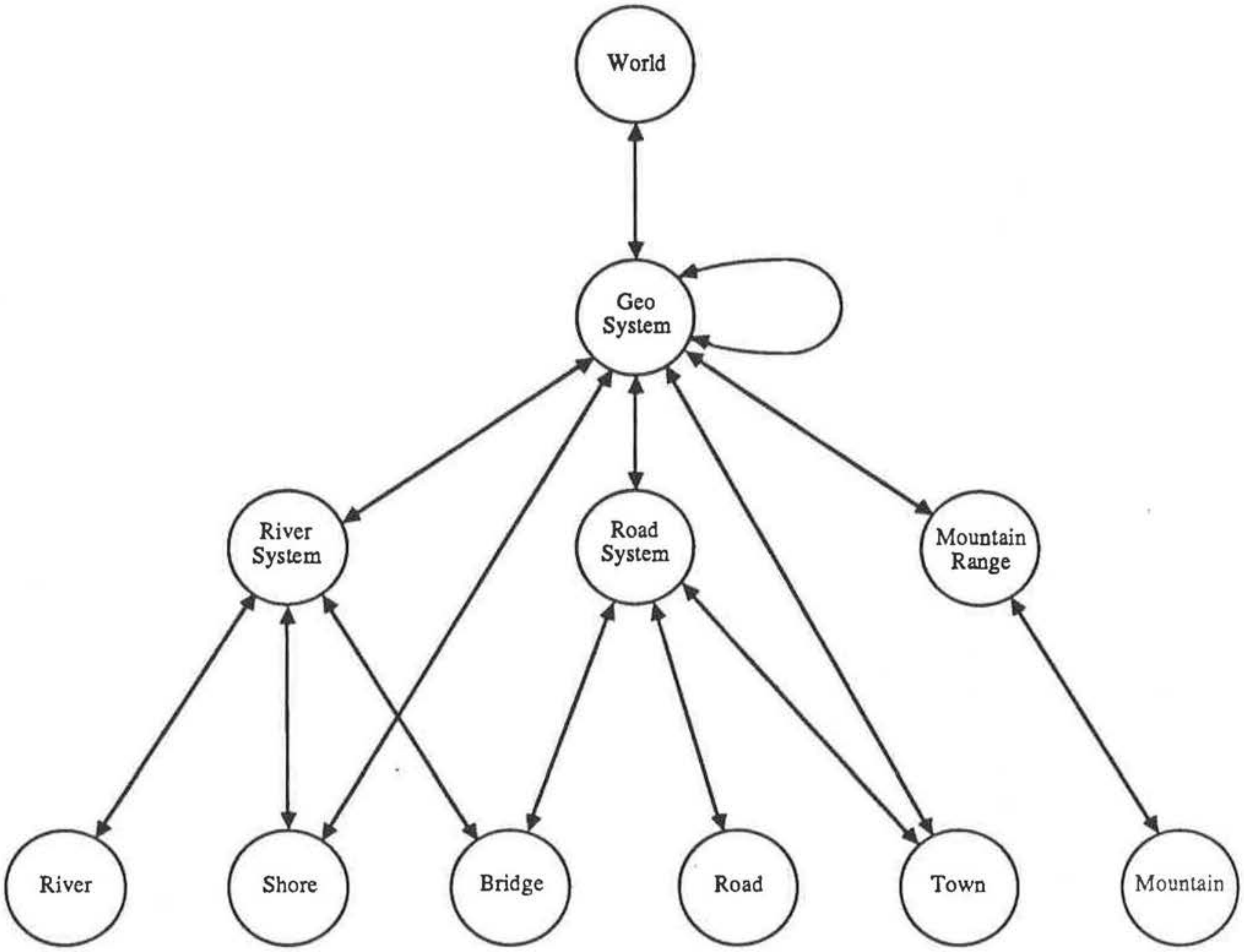


Figure 1: Mapsee2 Composition Hierarchy

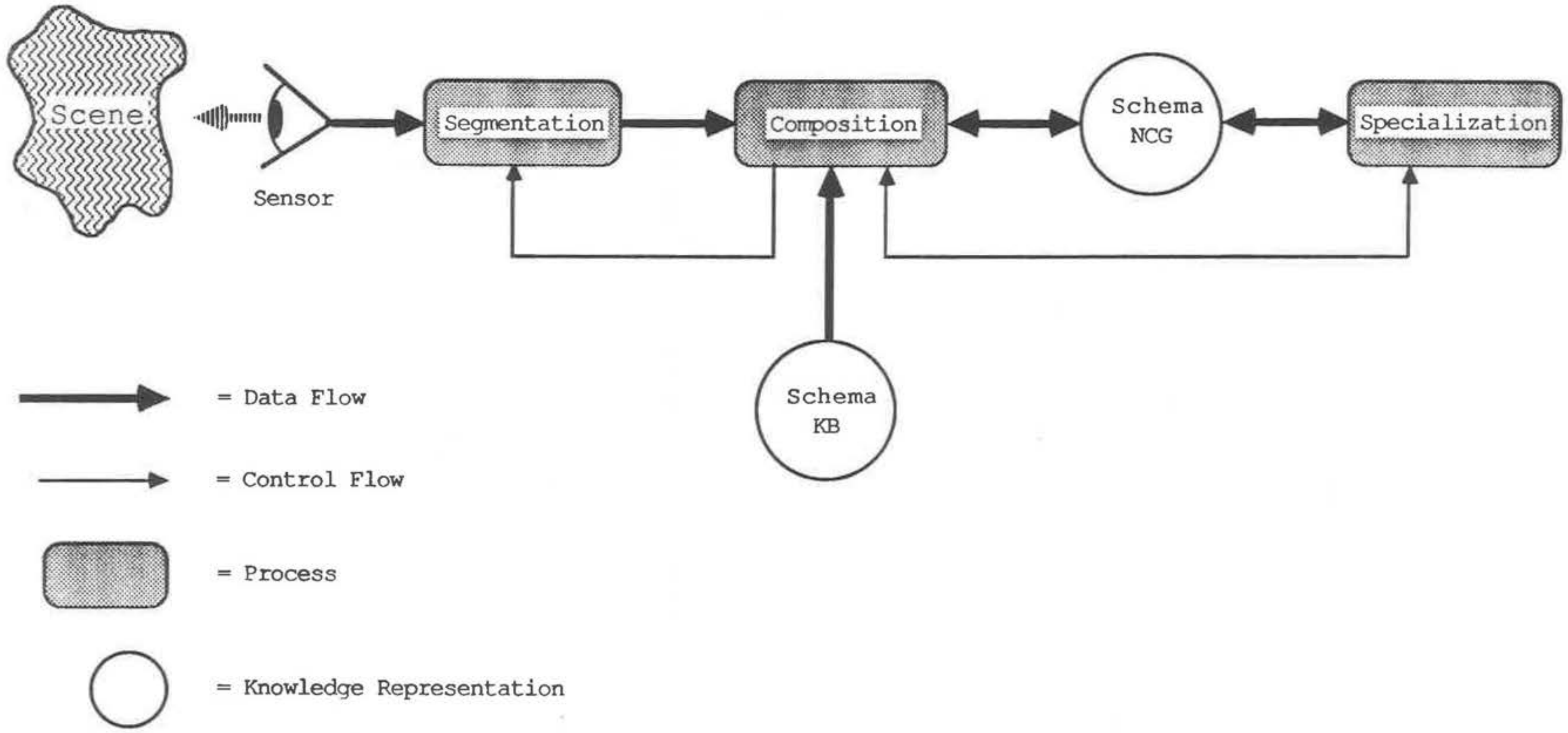
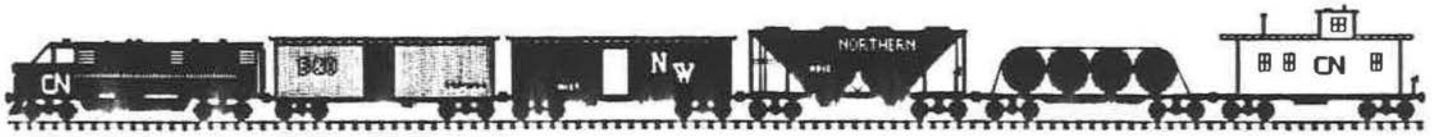
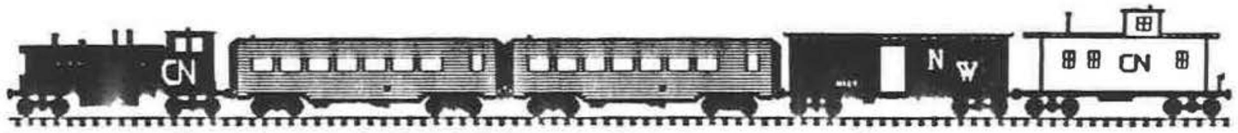


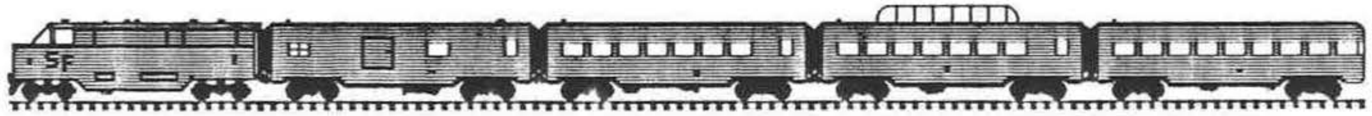
Figure 2: Overview of Schema Labelling



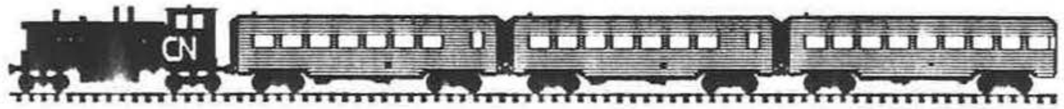
(a) Long-Haul Freight



(b) Local Train



(c) Express Train



(d) Commuter Train

Figure 3: Example Railroad Trains

Train:

$$\begin{aligned}
&M = Train\\
&P = \{p_1, p_2\}\\
&\alpha = \emptyset\\
&\Gamma = \{R^1_{Train}[\{Train, Carset\}],\\
&\qquad R^2_{Train}[\{Train, Engine\}],\\
&\qquad R^3_{Train}[\{Train, Caboose\}],\\
&\qquad R^S_{Train}[\{Train, LongHaul\}],\\
&\qquad R^S_{Train}[\{Train, ShortHaul\}]\}\\
&\Delta = \{LongHaul, ShortHaul\}
\end{aligned}$$

LongHaul:

$$\begin{aligned}
&M = LongHaul\\
&P = \emptyset\\
&\alpha = \emptyset\\
&\Gamma = \{R^1_{LongHaul}[\{LongHaul, CarSet\}],\\
&\qquad R^2_{LongHaul}[\{LongHaul, Engine\}],\\
&\qquad R^3_{LongHaul}[\{LongHaul, Caboose\}]\}\\
&\Delta = \{Express, Freight\}
\end{aligned}$$

ShortHaul:

$$\begin{aligned}
&M = ShortHaul\\
&P = \emptyset\\
&\alpha = \emptyset\\
&\Gamma = \{R^1_{ShortHaul}[\{ShortHaul, CarSet\},\\
&\qquad R^2_{ShortHaul}[\{ShortHaul, Engine\}],\\
&\qquad R^3_{ShortHaul}[\{ShortHaul, Caboose\}]\}\\
&\Delta = \{Local, Freight, Commuter\}
\end{aligned}$$

Engine:

$$\begin{aligned}
&M = Engine\\
&P = \emptyset\\
&\alpha = \emptyset\\
&\Gamma = \{R^1_{Engine}[\{Engine, Train\}]\}\\
&\Delta = \{Loco, Switcher\}
\end{aligned}$$

CarSet:

$$\begin{aligned}
&M = CarSet\\
&P = \{p_3, p_4\}\\
&\alpha = \emptyset\\
&\Gamma = \{R^1_{CarSet}[\{CarSet, Car\}],\\
&\qquad R^2_{CarSet}[\{CarSet, Car, CarSet\}],\\
&\qquad R^3_{CarSet}[\{CarSet, Car\}]\}\\
&\Delta = \{FreightCarSet, PassCarSet, MixedCarSet\}
\end{aligned}$$

Car:

$$\begin{aligned}
&M = Car\\
&P = \emptyset\\
&\alpha = \emptyset\\
&\Gamma = \{R^S_{Car}[\{Car, FreightCar\}],
\end{aligned}$$

$R^{S}_{Car}[\{Car\ ,\ PassCar\ \}]\}$

$\Delta = \{FreightCar\ ,\ PassCar\ \}$

PassCar: $M = PassCar$

$P = \emptyset$

$\alpha = \emptyset$

$\Gamma = \emptyset$

$\Delta = \{Coach\ ,\ Mail\ ,\ Observe\ ,\ Saloon\ \}$

FreightCar: $M = FreightCar$

$P = \emptyset$

$\alpha = \emptyset$

$\Gamma = \emptyset$

$\Delta = \{Box\ ,\ Flat\ ,\ Tank\ ,\ Hopper\ \}$

Figure 4: Schema Knowledge Base for Model Railroad

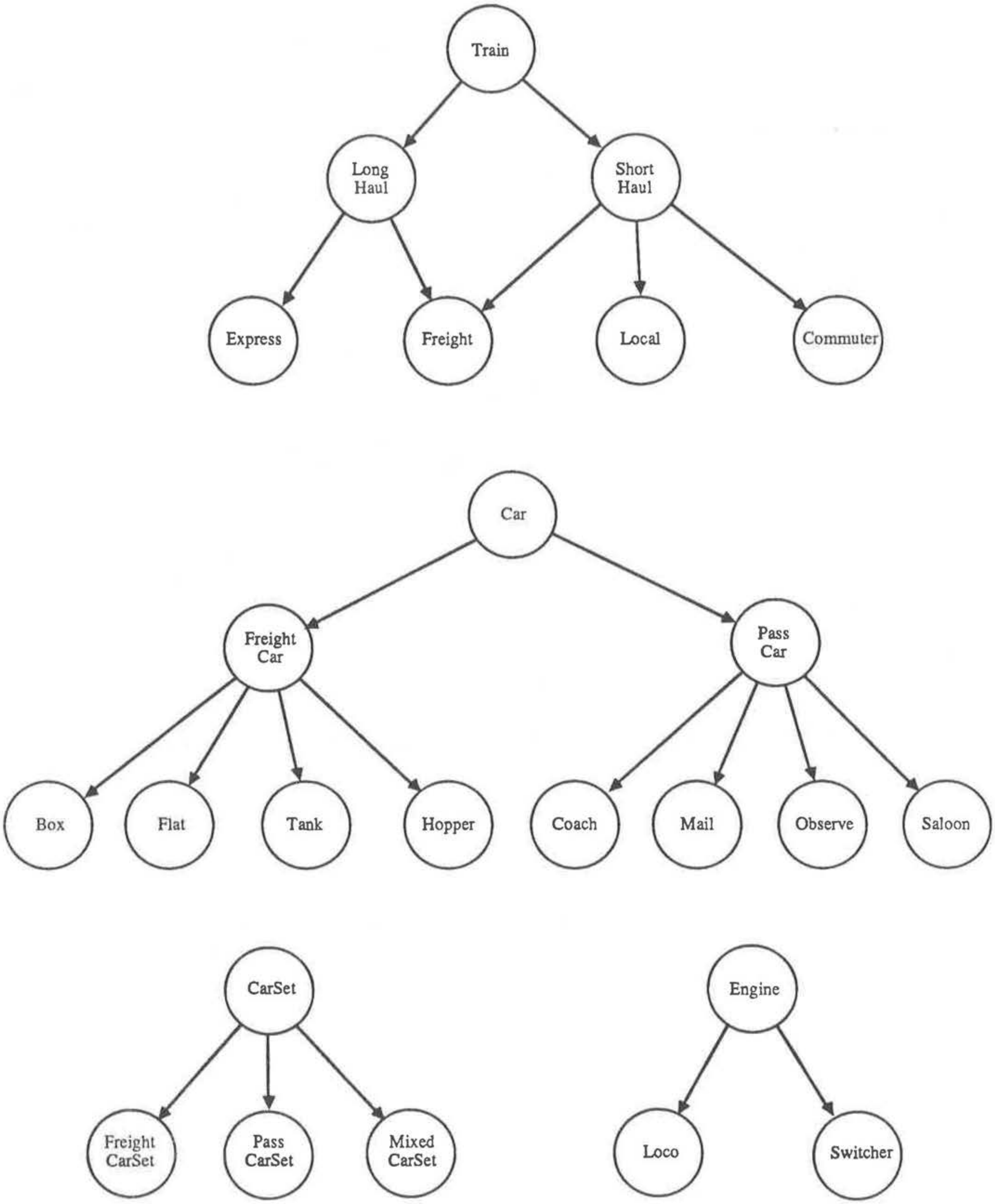


Figure 5: Specialization Hierarchy for Model Railroad

R^{1}_{Train} = {(LongHaul,FreightCarSet),(LongHaul,PassCarSet),
(ShortHaul,FreightCarSet),(ShortHaul,PassCarSet),(ShortHaul,MixedCarSet)}

R^{2}_{Train} = {(LongHaul,Loco),(ShortHaul,Switcher)}

R^{3}_{Train} = {(LongHaul,Generic),(ShortHaul,Generic)}

R^{S}_{Train} = {(LongHaul,Express),(LongHaul,Freight),(ShortHaul,Freight),
(ShortHaul,Local),(ShortHaul,Commuter)}

$R^{1}_{LongHaul}$ = {(Freight,FreightCarSet),(Express,PassCarSet)}

$R^{2}_{LongHaul}$ = {(Express,Loco),(Freight,Loco)}

$R^{3}_{LongHaul}$ = {(Freight,Generic)}

$R^{1}_{ShortHaul}$ = {(Freight,FreightCarSet),(Local,MixedCarSet),(Commuter,PassCarSet)}

$R^{2}_{ShortHaul}$ = {(Freight,Switcher),(Local,Switcher),(Commuter,Switcher)}

$R^{3}_{ShortHaul}$ = {(Freight,Generic),(Local,Generic)}

R^{1}_{Engine} = {(Loco,LongHaul),(Switcher,ShortHaul)}

R^{1}_{CarSet} = {(PassCarSet,PassCar),(FreightCarSet,FreightCar),
(MixedCarSet,FreightCar),(MixedCarSet,PassCar)}

R^{2}_{CarSet} = {(PassCarSet,PassCar,PassCarSet),(FreightCarSet,FreightCar,FreightCarSet),
(MixedCarSet,PassCar,MixedCarSet),(MixedCarSet,FreightCar,MixedCarSet),
(MixedCarSet,PassCar,FreightCarSet),(MixedCarSet,FreightCar,PassCarSet)}

R^{S}_{Car} = {(FreightCar,Box),(FreightCar,Flat),(FreightCar,Tank),
(FreightCar,Hopper),(PassCar,Coach),(PassCar,Mail),(PassCar,Observe),
(PassCar,Saloon)}

Figure 6: Constraints for the Railroad

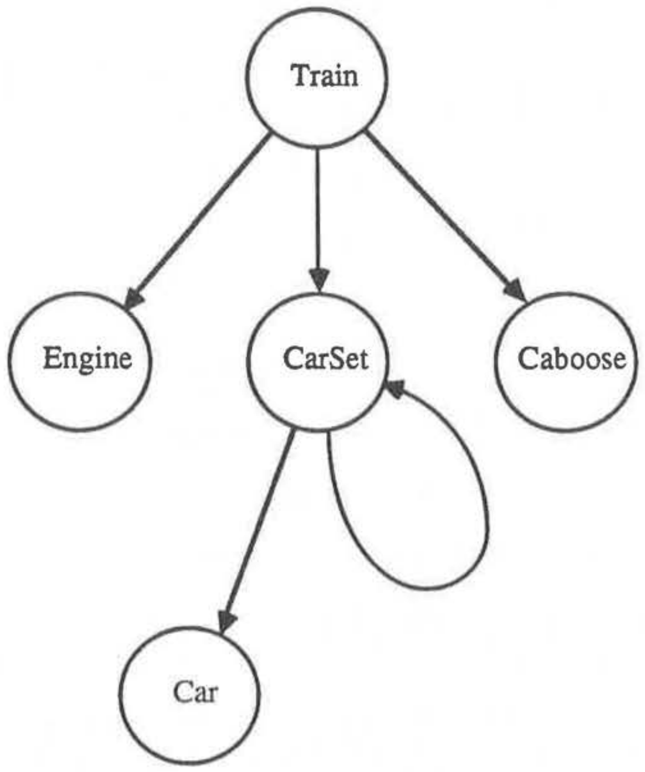


Figure 7: Railroad Composition Hierarchy

$$
\begin{aligned}
p_1\text{:}\quad & \mathit{Train}\,[v_1] \leftarrow \text{Let } v_1 = \mathit{New}\,[\mathit{Train}\,,\, P_{Train}\,,\, \emptyset,\, \emptyset,\, \Delta_{Train}\,] \wedge \\
& \quad \mathit{Engine}\,[v_2,\, v_1] \wedge \\
& \quad \mathit{Specialize}\,(v_1,\, v_2,\, R^2_{Train}\,[\{v_1,\, v_2\}]] \wedge \\
& \quad \mathit{CarSet}\,[v_3] \wedge \\
& \quad \mathit{Coupled}\,[\mathit{end}\,2[v_2],\, \mathit{end}\,1[v_3]] \wedge \\
& \quad \mathit{Specialize}\,[v_1,\, v_3,\, R^1_{Train}\,[\{v_1,\, v_3\}]] \wedge \\
& \quad \mathit{Caboose}\,[v_4] \wedge \\
& \quad \mathit{Coupled}\,[\mathit{end}\,2[v_3],\, \mathit{end}\,1[v_4]] \wedge \\
& \quad \mathit{Specialize}\,[v_1,\, v_4,\, R^3_{Train}\,[\{v_1,\, v_4\}]].
\end{aligned}
$$

$$
\begin{aligned}
p_2\text{:}\quad & \mathit{Train}\,[v_1] \leftarrow \text{Let } v_1 = \mathit{New}\,[\mathit{Train}\,,\, P_{Train}\,,\, \emptyset,\, \emptyset,\, \Delta_{Train}\,] \wedge \\
& \quad \mathit{Engine}\,[v_2,\, v_1] \wedge \\
& \quad \mathit{Specialize}\,[v_1,\, v_2,\, R^2_{Train}\,[\{v_1,\, v_2\}]] \wedge \\
& \quad \mathit{CarSet}\,[v_3] \wedge \\
& \quad \mathit{Coupled}\,[\mathit{end}\,2[v_2],\, \mathit{end}\,1[v_3]] \wedge \\
& \quad \mathit{Specialize}\,[v_1,\, v_3,\, R^1_{Train}\,[\{v_1,\, v_3\}]].
\end{aligned}
$$

$$
\begin{aligned}
p_3\text{:}\quad & \mathit{CarSet}\,[v_1] \leftarrow \text{Let } v_1 = \mathit{New}\,[\mathit{CarSet}\,,\, P_{CarSet}\,,\, \emptyset,\, \emptyset,\, \Delta_{CarSet}\,] \wedge \\
& \quad \mathit{Car}\,[v_2] \wedge \\
& \quad \mathit{Specialize}\,[v_1,\, v_2,\, R^1_{CarSet}\,[\{v_1,\, v_2\}]] \wedge \\
& \quad \mathit{CarSet}\,[v_3] \wedge \\
& \quad \mathit{Coupled}\,[\mathit{end}\,2[v_2],\, \mathit{end}\,1[v_3]] \wedge \\
& \quad \mathit{Specialize}\,[v_1,\, v_3,\, R^2_{CarSet}\,[\{v_1,\, v_2,\, v_3\}]].
\end{aligned}
$$

$$
\begin{aligned}
p_4\text{:}\quad & \mathit{CarSet}\,[v_1] \leftarrow \text{Let } v_1 = \mathit{New}\,[\mathit{CarSet}\,,\, P_{CarSet}\,,\, \emptyset,\, \emptyset,\, \Delta_{CarSet}\,] \wedge \\
& \quad \mathit{Car}\,[v_2] \wedge \\
& \quad \mathit{Specialize}\,[v_1,\, v_2,\, R^1_{CarSet}\,[\{v_1,\, v_2\}]].
\end{aligned}
$$

$$
\begin{aligned}
p_5\text{:}\quad & \mathit{Engine}\,[v_1,\, v_2] \leftarrow \text{Let } v_1 = \mathit{New}\,(\mathit{Engine}\,,\, P_{Engine}\,,\, \emptyset,\, \emptyset,\, \Delta_{Engine}\,) \wedge \\
& \quad \mathit{Specialize}\,[v_1,\, v_2,\, R^1_{Engine}\,[\{v_1,\, v_2\}]].
\end{aligned}
$$

Figure 8: Composition Rules for Railroad

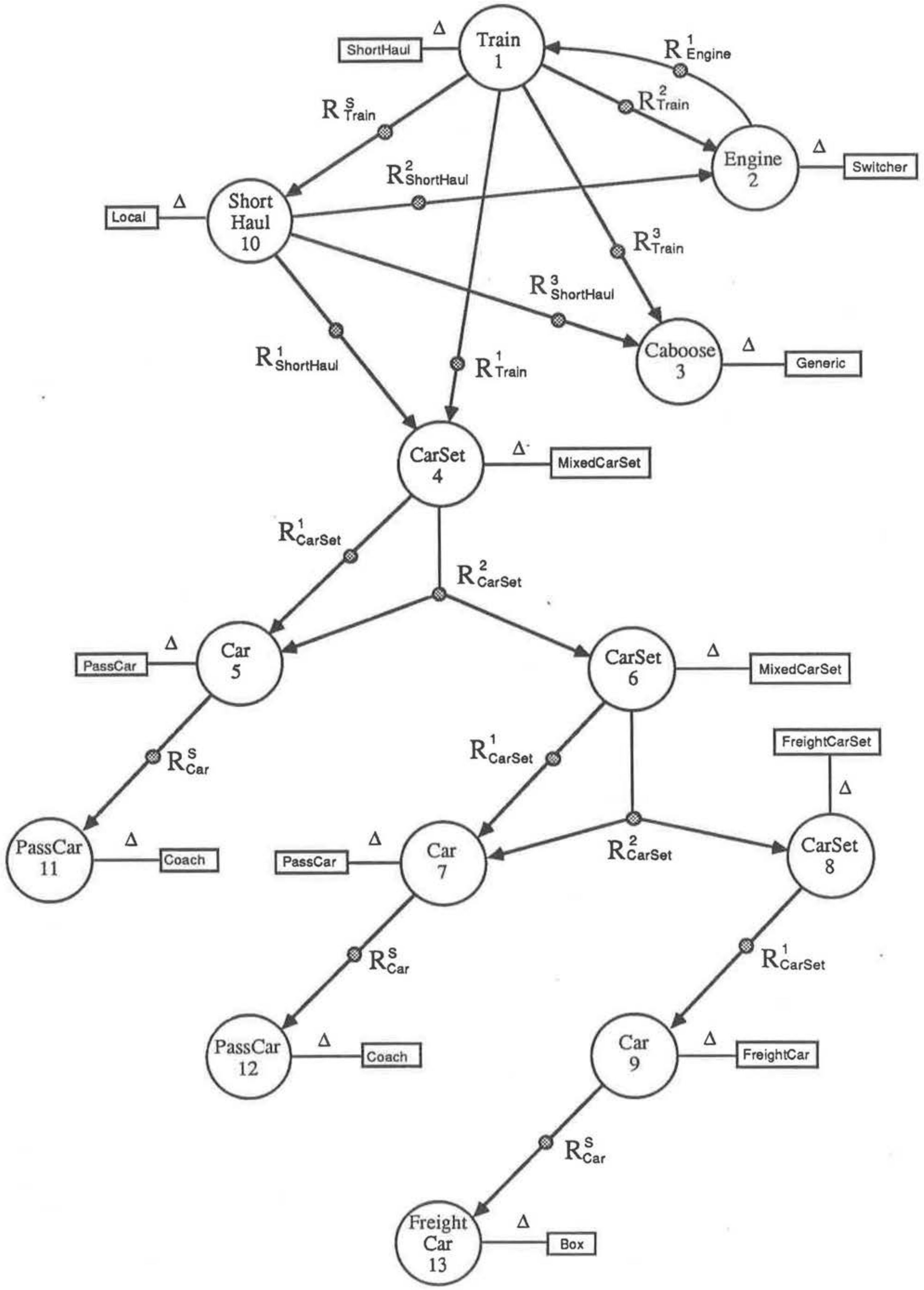


Figure 9: Network Consistency Graph for Local Train