Process Transition Networks: What Are They For?

Jacek Malec
Department of Computer and Information Science
Linköping University
S–581 83 Linköping, Sweden
jam@ida.liu.se

Abstract

Process Transition Networks (PTNs), recently introduced by Sandewall [6], are a graphical language to describe complex qualitative processes; processes, which are difficult to express quantitatively even with the help of causal dependencies. Road or air traffic monitoring, process control, or computer systems maintenance are some examples of possible application areas for PTNs. The language primitives are introduced informally, followed by an example of more complex nature. The relation between PTNs and Petri nets is considered, and it is shown that although PTNs have no more expressivity than Petri nets, their flexibility as a representation language for complex processes justifies their existence.

1 Introduction

For the purpose of describing complex real world domains, such as road or air traffic, manufacturing processes, systems maintenance, etc., one needs to develop systematic techniques able to:

1. describe static and dynamic aspects of environment in a sound, systematic, and user-oriented way (taking not only formal precision, but also legibility for domain experts into account);
2. formally analyze the descriptions developed using techniques mentioned in the previous point, in order to verify their consistency, completeness, and computational complexity (the last one being especially important in real-time domains of application);
3. help implement tools such as simulators, planners, and controllers, for such environments. These tools should be tied to the description languages from the point 1, and conform to the requirements mentioned in point 2.

In this paper I present a formalism called Process Transition Networks (PTNs), recently introduced by Sandewall [6]. They are intended to serve two purposes:

- make the knowledge acquisition process easier (much less effort is needed to construct PTN diagrams describing particular traffic situations than with other representation tools, e.g. logic formulae, or Petri nets);
- serve as a tool for knowledge explanation to non-experts.

PTNs can be seen as a language to describe complex qualitative processes; processes, which are difficult to express quantitatively even with the help of causal dependencies. Currently our application area consists of typical traffic scenarios, but it seems that possible application areas of PTNs are much more numerous. The PTN language is based on graph theory, and its expressive power is covered by Petri nets, what will be shown in one of the sections. The development of the language follows requirements (1) — (3) listed in the previous paragraph, although only point (1) is covered in this paper.

The number of existing knowledge representation tools, including several graphical languages, raises the question as to whether yet another graphical knowledge representation tool is necessary. This paper can be seen as an attempt to (indirectly, but positively) answer this question. First, the language is defined formally (both its syntax and

[0]Manuscript received July 29, 1991. This research was supported by the IT4 research program.

semantics) [4], which makes it possible to compare it with other similar tools, and (in the best case) to prove its useful properties. Second, PTNs appear to be well suited for expressing common sense conceptualizations of complex environments, especially in the case where causal dependencies between described entities are involved.

Although we claim that PTNs might be useful in several different application areas, we currently use them only for expressing knowledge about typical traffic situations. They serve as a tool for transferring knowledge from "domain experts" (which, in case of traffic scenarios, all the drivers feel to be ones) to a driver support system, currently being developed in our lab. As we use several knowledge representation tools within the system, the possibility of moving freely between representations becomes an additional advantage. In this context, clear definition of syntax allows us to treat PTNs as the specification language for system development, and its formal semantics provides intuitions about its relation to various logics used in other modules of our system.

PTNs are closely related to several well-known formalisms which exploit the advantage of graphical representation. One of the best known formalisms are Petri nets [5]. PTNs are a proper subset of Petri nets, but they have been designed in such a way that expressing causal (and thus temporal) dependencies requires only one symbol, which is impossible to obtain using Petri nets. Moreover, the properties of Petri nets have been well studied, so we hope to get more insight into PTNs by establishing a formal correspondence between the semantics of both languages.

Another well designed graphical formalism has been developed by Harel [1]. His *statecharts* seem to have motivations similar to PTNs. Harel also provides a formal (although only operational) semantics for statecharts [2]. However, his approach seems to have one major drawback as compared with ours (at least within our application area). Statecharts are well suited for describing entities fitting the automata theory paradigm. This implies that one has to distinguish an input alphabet (set of events), states of described entity, and a transition function mapping current state and some input value to the new state (and possibly to the output). In our case we want to mix dependencies of the form *input* $\rightarrow$ *state* with dependencies *state* $\rightarrow$ *input*, where *state* correspond to some state of a (possibly intelligent) agent, and *input* is a description of current state of the environment. In the automata theory paradigm, the set of input letters, and the set of output letters are totally unrelated. Of course, very close links tie both formalisms, so it might be interesting to look at their relationship in more systematic way.

The paper consists of four sections. Section 2 informally introduces PTN language primitives, and presents in parallel the underlying ontology. In section 3 the ways of combining PTN primitives into meaningful descriptions of complex systems are presented. Finally, in section 4, we show how a PTN can be reformulated as a Petri net.

2 PTN Language Primitives

Let's begin with a bit of ontology. We assume that the world consists of objects (some of which we may call *agents*), and that the state of the world (including state of the agent, or agents) may be described by using a set of object properties. Property values may (and usually do) change with time. In most cases one of the following two statements is true:

- the set of property values is finite, and the value function is piecewise constant;
- the set of property values is infinite, and is usually continuous. The property value may change with time, but except some distinguished time points, the change is continuous. So the value function is piecewise continuous.

A set of adjacent time tokens, during which a property *continuously* changes its value (this covers also the case of constant value), will be called a *circumstance*. So, for example, state of a robot performing some primitive[1] operation (as polishing a kitchen sink, or searching for an empty soda can), traffic lights being red, a car following another one (being "close" to it), an airplane following some air corridor, or a computer being hung, all may be characterized by a circumstance.

In PTN language we introduce the notion of a *process*, corresponding to some circumstance. A process will be depicted by a box. We are interested in those moments in which discontinuities of a property occur, because such discontinuities convey important information about current situations (or states of affairs). Being more precise, discontinuities inform us about important change of situation.

We will call such discontinuities *events*. Events are represented in the PTN language by hexagonal boxes. As discontinuities (by definition) may only appear between periods of continuous change of some property, events may

[1] *Primitive* does not necessarily mean simple, but rather "basic", or "not compound" (on assumed level of analysis). So "primitive operation" on one analysis level might appear to be complex operation on some lower level.

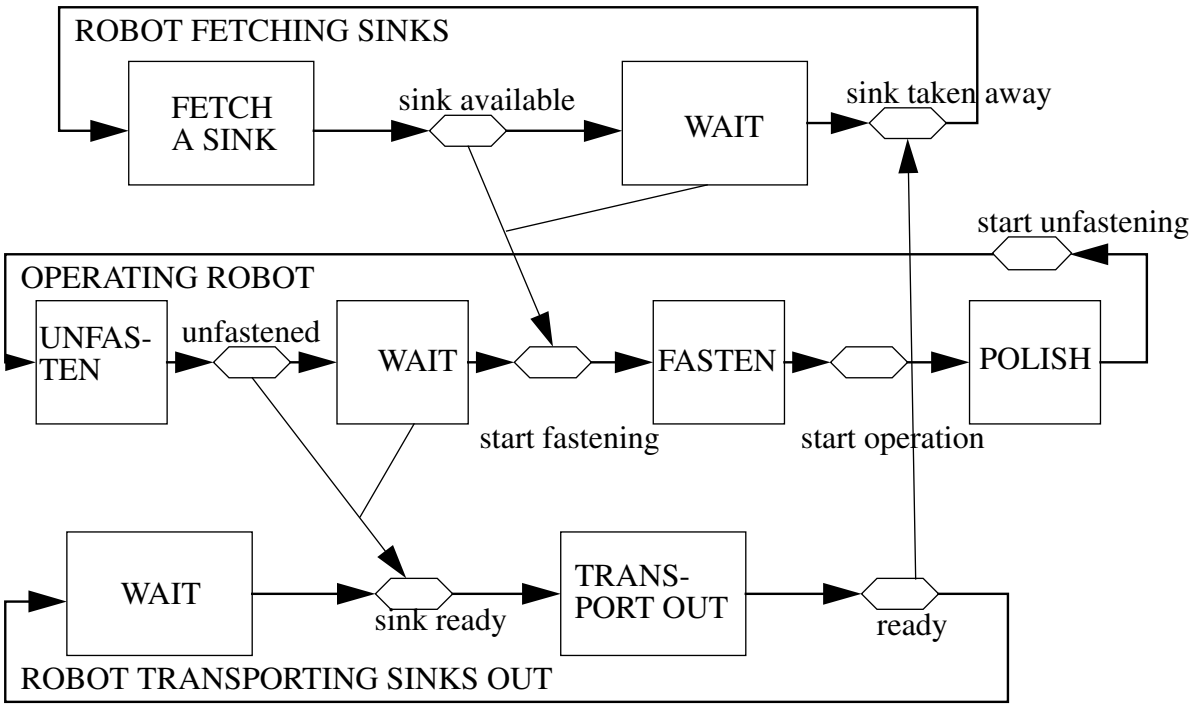

Figure 1: Three cooperating robots.

only appear between two processes. Such a *transition* (from process to event to another process) is denoted by two thick arrows pointing from the first process to the event, and from the event to another process, respectively.

Causal dependencies between discontinuities are represented by *enabling arrows*, shown in figures with thin lines, and pointing from either an event alone, or an event and its subsequent process, to some other event. The first possibility corresponds to the case where one discontinuity forces another one to occur; the second possibility describes the case of enabling.

A simple example of PTN constructed using the introduced primitives appears in fig. 1. The PTN represents here behavior of a system consisting of three robots cooperating during one phase of sink production.

Processes which describe the behavior of some *agent* are called *activities*. Activities which have "inherent termination conditions" (i.e. their termination condition is not explicitly stated in given PTN) are called *actions*. Activity boxes are distinguished by doubling their vertical borders, and actions contain an additional internal box denoting their termination condition. In case of action there is no event following it — it is "encoded" in the termination conditions of the action. However, if one wants this event to enable some other event, then the enabling arrow is drawn directly from the box representing the termination condition. The figure 2 shows usage of activity and action boxes.

Activities and actions may be distinguished in an arbitrary way. They are used to focus attention on processes ascribed to some agent, or agents. In our example a fire brigade has been treated as single agent. Later on we will assume that every PTN cycle containing an activity consists only of activities. The rationale behind this is that if some process (circumstance) is associatied with some property of an agent, then all the subsequent values of that property (thus, all the subsequent processes, or circumstances) are also associated with the same agent.

3 Creating complex descriptions

The main purpose of distinguishing actions is to keep PTNs as simple as possible. Moreover, the actions allow us to draw the enabling arrows according to our intuitions. However, every action box can be seen as a way of encoding an activity and its following event, together with the whole PTN cycle describing the triggering conditions for the event.

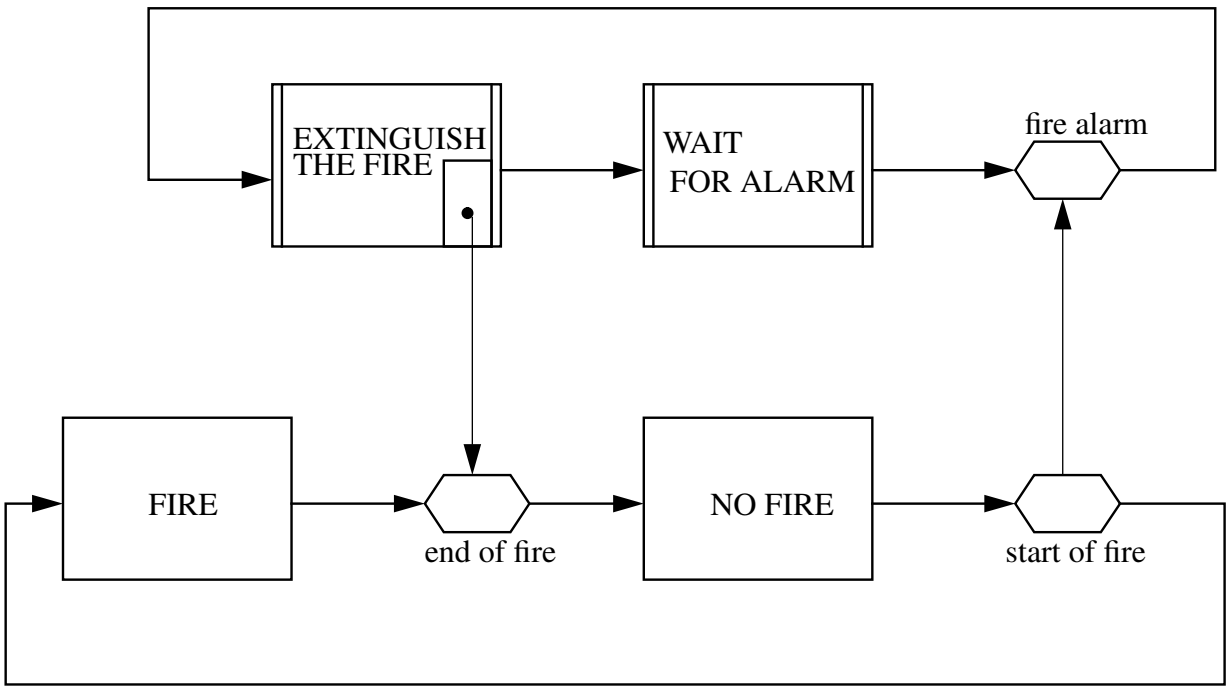

Figure 2: Fire brigade activities.

On fig. 3 we present the way of translating action boxes back into activity boxes enabled from some external cycle. "Inherent termination conditions of an action" we have been speaking about are just the conditions described by this external cycle.

The PTN language allows one to combine enabling arrows using both **and** and **or** connectives. The **and** of two enablings is expressed by two consecutive events on one transition from a process to another process. The **or** may be represented with two different paths leading from one process to another, but is usually depicted by merging the enabling arrows themselves. See fig. 4.

Decision is yet another primitive introduced in the PTN language. It is an action performed by an agent faced by several alternative possibilities. Similarly to all the actions, decisions implicitly refer to some external condition which is not shown in the figure. A decision box is pentagonal, and its usage is presented in figure 5.

In the following example, PTNs have been used to conceptualize the scenario of passing an intersection. Details can be found in [3]. Figures 6 and 7 present two PTNs describing, respectively, a very general overview of passing an intersection, and part of a much more detailed analysis of this scenario. Below an intuitive description of the contents of the depicted PTNs is given.

When an intersection becomes visible (what corresponds to the transition "No intersection in sight" $\rightarrow$ "Intersection visible", the car switches from the "DriveFreely" activity to the "Approach" action. The "Approach" action consists of an appropriate slowing down (up to stopping, if needed), checking the intersection state in order to determine if crossing is possible, and preparing to choose the outcoming road segment (left, right, or straight). After concluding the "Approach" (ie. coming to the meeting point of the road segment we are on, and the intersection) the car switches to the "Pass" action, which concludes passing through the intersection and causes the intersection to vanish. After "Passing" intersection the car switches to the "DriveFreely" activity again, until it meets the next intersection.

It is worth pointing out that the only conceptual difference between the "intersection" cycle, and the "car activity" cycle is that we ascribe the latter to the behavior of some intelligent agent, whereas the former describes the agent's environment. We also note that causal dependencies point both ways, so changes within the environment can cause some behavior change of the agent, and vice versa.

The "Approach" — "Pullup" — "Stop" — "Pass" — "DriveFreely" path (cf. fig. 7) contains the processes (in fact, activities) describing behavior of the *own* car. They correspond to the changing value of some fluent, say "State-of-

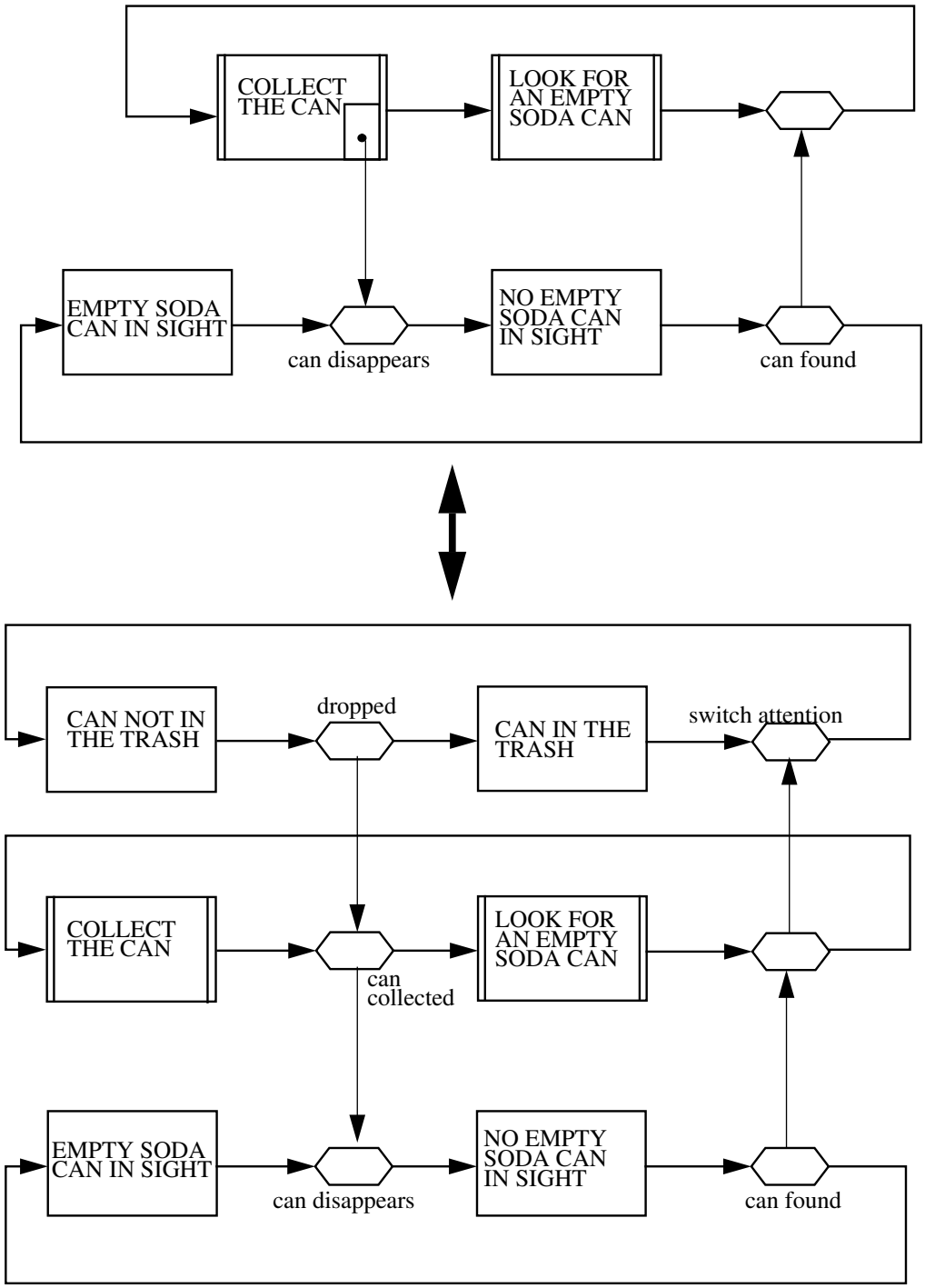

Figure 3: Action as an activity plus additional enabling cycle.

my-own-car", which, in its turn, corresponds to some slot in the state vector containing all the sensory input data.

Because we don't explicitly mark events, but only transitions between processes (thick arrows on the pictures), enabling arrows (thin ones) are drawn between transitions, and not explicitly named events. All the labels in fig. 7 denoting triggering events (called configurations) correspond to different processes in some other part of the PTN describing configurations of crossing cars. These labels should be read as follows:

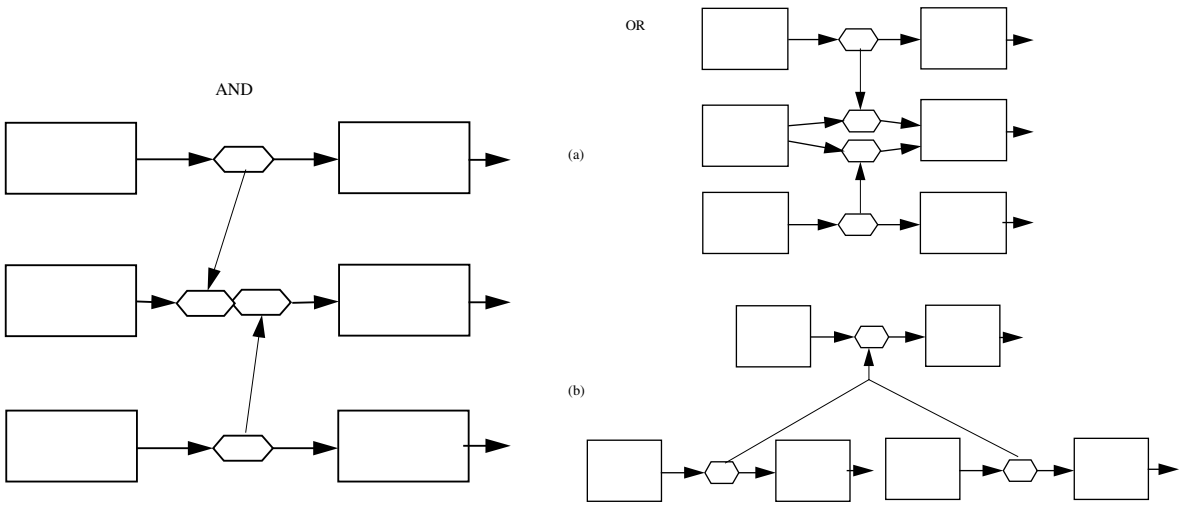

Figure 4: The **and**, and **or** connectives.

A – no crossing cars;

RL[2] (RS, RR, FL, FS, FR, LL, LS, LR) – there is at least one car having the RL (RS, RR, FL, FS, FR, LL, LS, LR) property among the cars approaching the intersection from other directions than the OWN car does.

R (F, L) – there is at least one car with either an RL, RS, or RR (either FL, FS, or FR; either LL, LS, or LR) property among the cars approaching the intersection from other directions than the OWN car does.

It should be stressed that PTNs strongly encourage a hierarchical approach to the task of domain description. We can start with a very rough conceptualization, distinguishing only major properties of the world around us, and their most important values, and then we can successively refine this first approximation, obtaining an accurate (and, what is important in our case, ready to implement) description of (hopefully) all the important interactions among some agent(s) and its (their) environment. Figures 6 and 7 illustrate to some extent this top-down approach, where one box from fig. 6 is split into several ones in fig. 7.

Another advantage of PTNs is the following: they can be used to graphically present and explain the actual contents of a knowledge base. Although this factor is less relevant from a formal point of view, it becomes very important in cases where domain experts, not necessarily fully competent in formal logic, need to be involved in the process of creation, validation, and expansion of a large and complex knowledge base.

4 PTNs vs. Petri nets

PTN is a fully formalized language, with well defined syntax and semantics [4]. However, here we will only show that they can be reformulated as Petri nets (PNs).

More complex PTNs (using the full set of language possibilities) can be rewritten without change of their meaning using only those elements which have been defined in the beginning of the section 2. These elements are: processes (and also activities and actions as their special cases) and events as "building blocks", and transition and enabling arrows as links between these blocks. Therefore we only need to show that these basic elements can be translated into PNs in the desired way. We use the "conditions and events" terminology of Petri nets theory as it directly corresponds to our intuition, however, the pictorial representation used here is the most popular one (see e.g. [5], and is often associated with the "places and transitions" terminology.

[2]In this one, and in the following two-letter labels, the first letter corresponds to the direction the crossing car comes from, the second letter — to the direction it intends to turn. R = right, L = left, S = straight.

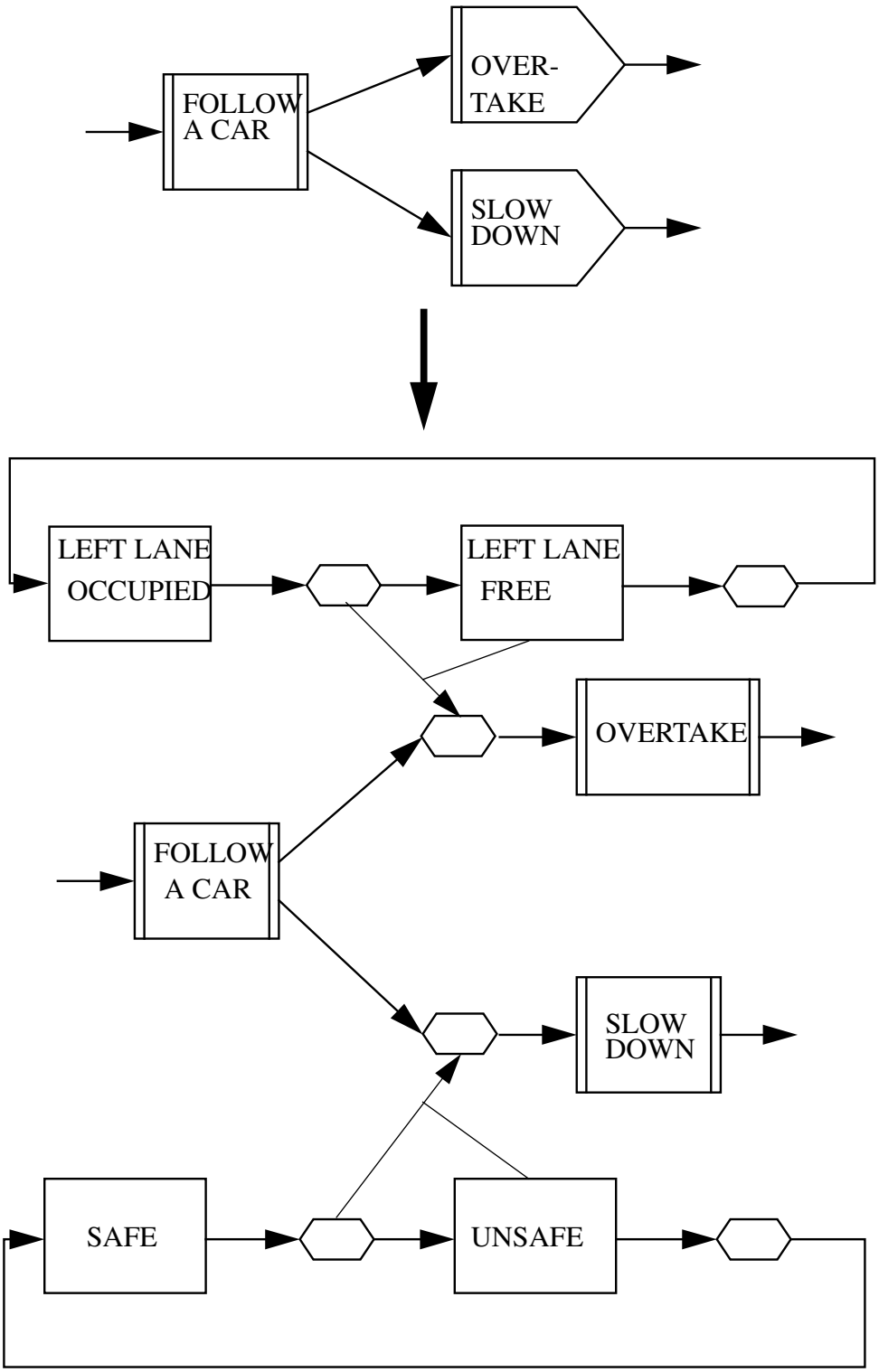

Figure 5: Alternative decisions to be taken by an agent.

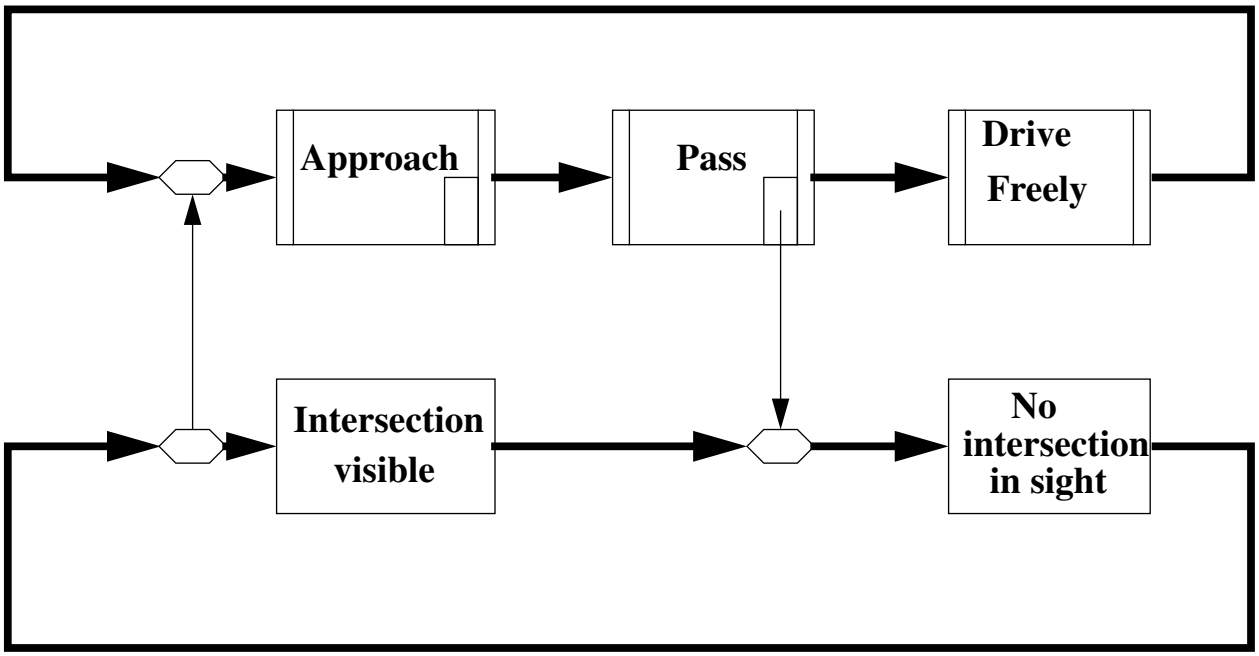

Figure 6: How to pass through an intersection.

Any PTN process is assigned a corresponding PN condition (or state, or place), and every PTN event has one corresponding PN event. PTN transition arrows are also mapped one-to-one into PN arcs pointing from conditions to events, or vice versa. So, for example, a simple PTN as shown in fig. 8, would be translated into the PN shown in fig. 9.

It remains to be shown how the PTN's enabling arrows can be translated into the PN's dependencies. The following two figures present, respectively, a simple PTN with proper enablings only (fig. 10), and its translation into PN (fig. 11). The translation procedure itself is obvious and will not be formulated here. Although a version with the inhibitor arcs would be much simpler, the version without them proves the possibility of using classical PN formalism (it is known that the inhibitor arcs are a substantial extension of the PN formalism).

The last two figures (figs. 12, and 13) present a translation of extended enablings (ie. enabling arrows coming both from an event, and from subsequent process). This concludes the translation algorithm. It can be proved that a PTN, and the PN given by the procedure described above, are behaviorally equivalent.

5 Conclusions

In this paper we have presented an informal introduction to Process Transition Networks — a pictorial language suitable for the conceptualization, and coding of knowledge about complex, quantitatively intractable environments. The word "coding" is justified by the fact that sentences in this language (i.e. PTNs) correspond (can be mechanically translated) to a very restricted temporal logic language RETFL which we are using in our project [3]. The PTN language is equivalent to a subset of classical Petri nets formalism. This suggests one obvious direction of further study: what properties formulated in the Petri nets framework hold for PTNs.

The representational advantages of the language are simplicity, possibility of expressing causal dependencies, ability to describe the behavior of several agents acting in the same environment, modularity, and the possibility of using it for subsequent refinements of an initial conceptualization. These properties have been confirmed during the conceptualization of several non-trivial traffic-behavior scenarios.

One obvious disadvantage is the fact that we introduce yet another language for expressing knowledge about the real world, moreover a very restricted one. However, none of the existing formalisms is suitable for our purpose, that is modelling complex dynamic environments, either because of much generality, or because of lack of expressivity. One

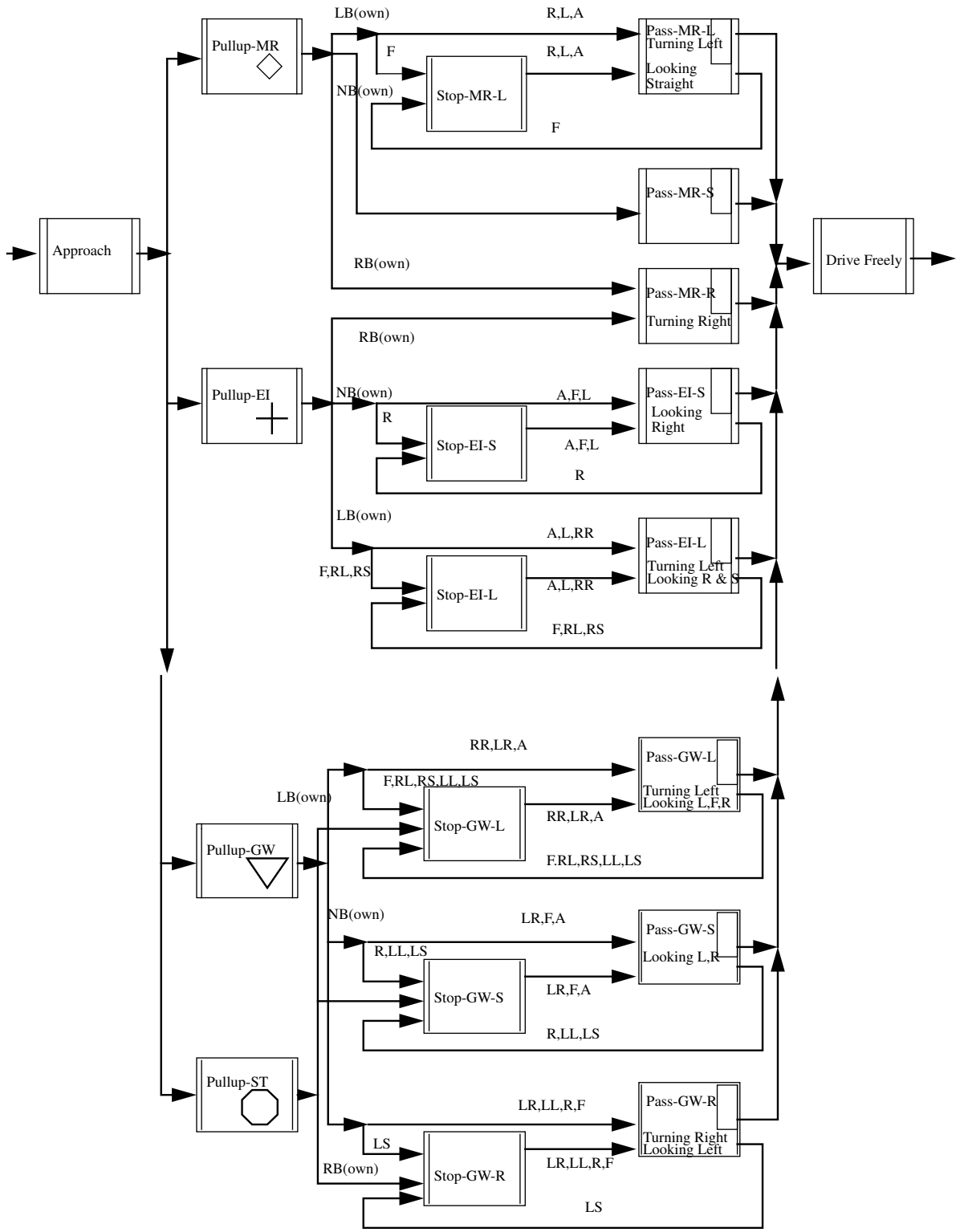

Figure 7: A fragment of "How to pass through an intersection", with more details shown.

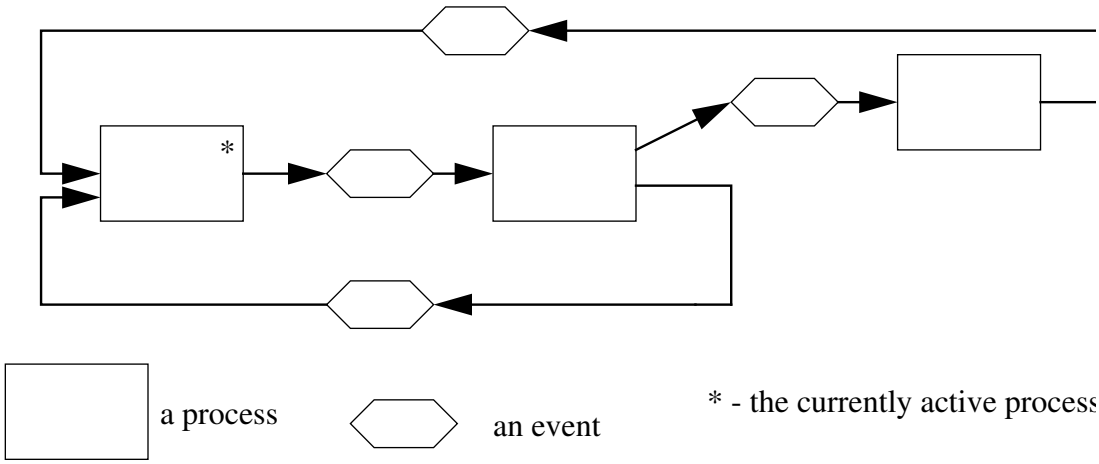

Figure 8: A very simple PTN (with no enabling arrows).

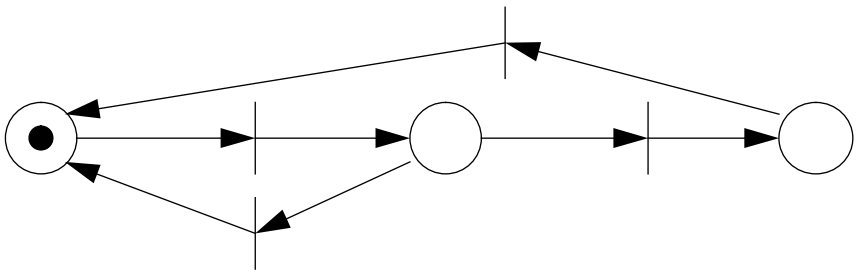

Figure 9: Corresponding Petri Net.

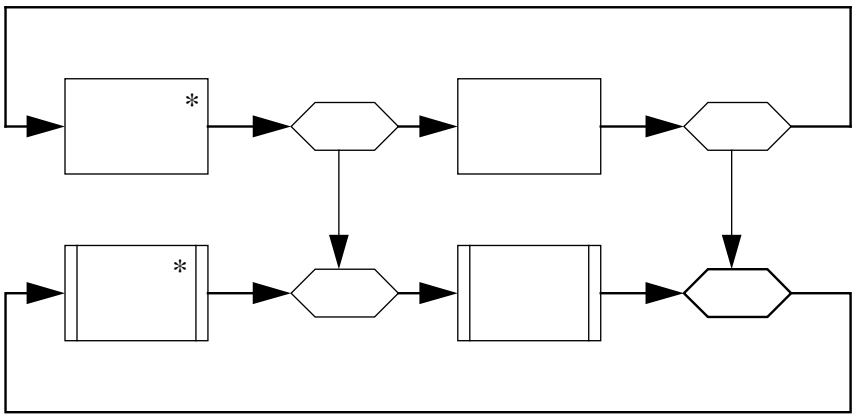

Figure 10: A very simple PTN (with proper enabling arrows).

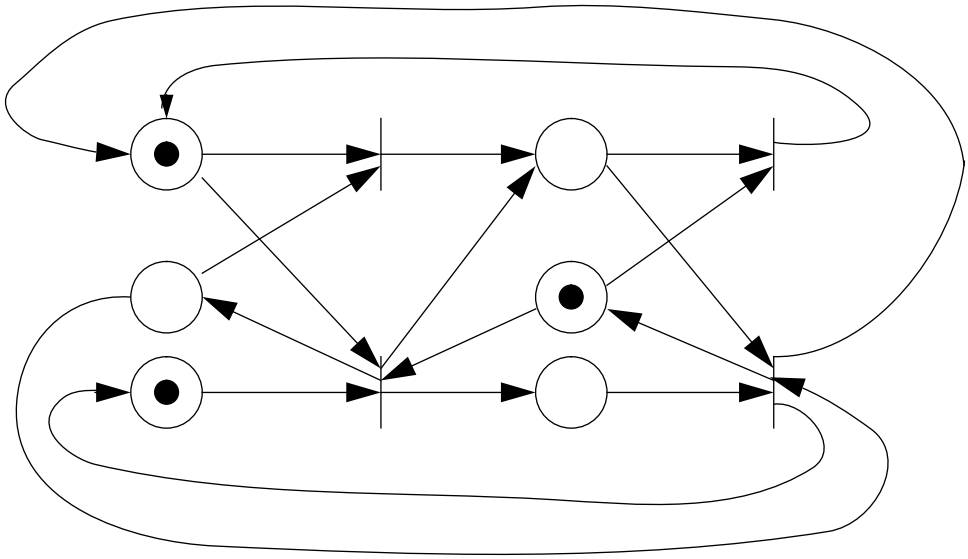

Figure 11: Corresponding Petri Net.

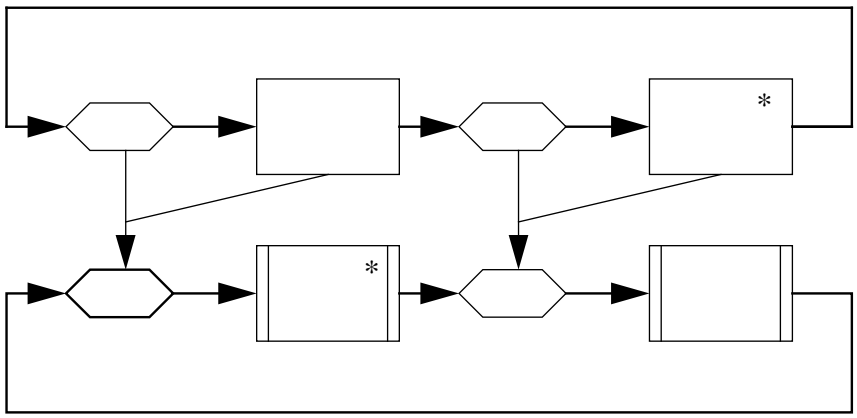

Figure 12: A simple PTN (with extended enabling arrows).

possible alternative would be to use the statechart formalism [1, 2], however its set of automata-theory-based primitives does not quite fit properly to our needs. In our opinion, the communicativity of PTN language, and its ease of use, justifies its introduction.

Acknowledgment

Patrick Doherty helped a lot to improve readability of the paper.

References

[1] David Harel. Statecharts: A visual formalism for complex systems. *Science of Computer Programming*, 8:231–274, 1987.

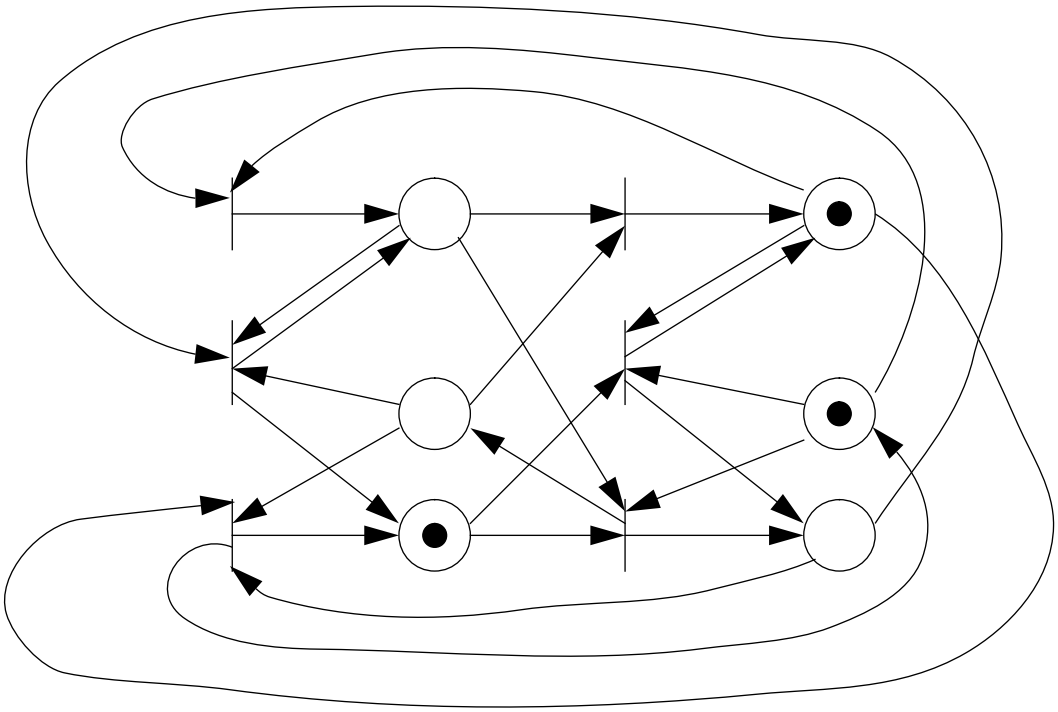

Figure 13: Corresponding Petri Net.

[2] David Harel, Amir Pnueli, J. P. Schmidt, and Rivi Scherman. On the formal semantics of statecharts. In *IEEE Symposium on Logic in Computer Science*, pages 54–64, Ithaca, NY, 1987. IEEE Computer Society Press.

[3] Jacek Malec. How to pass an intersection. In *Proc. of the Prometheus Pro-Art Workshop on Intelligent Co-Pilot*, pages 167–182, Grenoble, France, 1991.

[4] Jacek Malec. Process Transition Networks: A formal graphical knowledge representation tool. In Z. W. Ras and M. Zemankova, editors, *Methodologies for Intelligent Systems, 6th International Symposium*, pages 193–202. Springer-Verlag, October 1991. Lecture Notes in Artificial Intelligence 542.

[5] James L. Peterson. *Petri Net Theory and the Modelling of Systems.* Prentice-Hall, 1981.

[6] Erik Sandewall. Proposal for a ProArt specification platform. Technical Report LAIC-IDA-90-TR18, IDA, Linköping University, 1990.