M.I.T Media Laboratory Perceptual Computing Section Technical Report No. 423
(submitted to ICCV'98) – April 1997

Human Action Detection Using PNF Propagation of Temporal Constraints

Claudio Pinhanez **Aaron Bobick**
20 Ames St. – Cambridge, MA 02139
pinhanez | bobick@media.mit.edu

Abstract

In this paper we develop a representation for the temporal structure inherent in human actions and demonstrate an effective method for using that representation to detect the occurrence of actions. The temporal structure of the action, sub-actions, events, and sensor information is described using a constraint network based on Allen's interval algebra. We map these networks onto a simpler, 3-valued domain (past,now,fut) network — a PNF-network — to allow fast detection of actions and sub-actions. The occurrence of an action is computed by considering the minimal domain of its PNF-network, under constraints imposed by the current state of the sensors and the previous states of the network. We illustrate the approach with examples, showing that a major advantage of PNF propagation is the detection and removal of situations inconsistent with the temporal structure of the action. We also examine a method to increase the robustness of PNF-propagation in the case of faulty sensors.

1 Introduction

The majority of work on the machine understanding of motion sequences has focused on the recovery of the two-dimensional optic flow or the three-dimensional motion of the objects or camera. Recently, however, emphasis has shifted to the interpretation or classification of the observed motion. Bobick [4] provides a taxonomy of motion understanding problems labeled as movements, activities, and actions. One of the central differences between these levels is the degree to which time must be manipulated to recognize the motion. Bobick argues that, in case of human action, it is fundamental to reason about qualitative temporal relationships.

The goal of this paper is to develop a representation for the temporal structure inherent in human actions and to demonstrate an effective method for using that representation to detect the occurrence of actions. The fundamental assumption of our approach is that actions can be decomposed into sub-actions, some of whose occurrence can be directly detected by perceptual methods, and between which there exist a variety of temporal relationships. The task is to determine whether an action — or each of its component sub-actions — is currently happening or not given the input from perceptual sensors. We develop an algorithm, called PNF propagation, that significantly enhances the discriminatory power of sensors by using and propagating temporal constraints.

We start by proposing the use of Allen's temporal primitives ([2]) to describe the richness of the temporal structure of human action. Unlike previous work on action recognition ([24, 16]) the proposed representation (evolved from [17]), enables multiple possibilities for the sequencing of sub-actions, including parallelism and mutual exclusion.

A naive approach would be to map the temporal structure into a constraint satisfaction network ([9]), impose constraints on the occurrence of some actions according to the sensor values, and run constraint satisfaction algorithms to propagate the sensor-derived constraints into the higher-level sub-actions and actions. However, most temporal constraint satisfaction methods require exponential time to obtain results ([25]).

We address this problem by showing a novel method which maps a temporal network into a simpler, discrete-domain network tailored to answer only the basic question of whether the action is currently happening or not. Those networks are called PNF-*networks* because the only values assigned to the network variables past, now, and fut. In this reduced network, propagation of the temporal constraints can be approximately achieved using arc-consistency algorithms ([12, 13, 15]), which are linear in the number of constraints. We demonstrate the effectiveness of the approach on two examples of action detection.

We conclude the paper by examining methods to cope with the brittleness introduced by the use of discrete domains and pure constraint satisfaction. Basically, we propose a method which keeps track of multiple, possible configurations of the temporal structure, enhanced by a heuristic for selection. This is possible because, unlike other representation schemes ([16, 5]), PNF-networks allow compact ways to represent a chain of previous results.

1.1 Previous Work

Most attempts at logical formalization of the semantics of action (e.g. [8, 20, 22, 7]) are grounded in either philosophy or formal logic. These systems are typically not grounded in perceptual primitives.

Some work in the vision community [10, 16, 24] has attempted to incorporate logical definitions of action into perceptual mechanisms. However, these systems are unable to cope with most of the complex time patterns of everyday actions which include external events, simultaneous activities, multiple sequencing possibilities, and mutually exclusive intervals [3]. Kuniyoshi and Inoue ([10]) use finite automata to represent actions performing simple actions to teach a robot. Implicit in the model is the assumption of

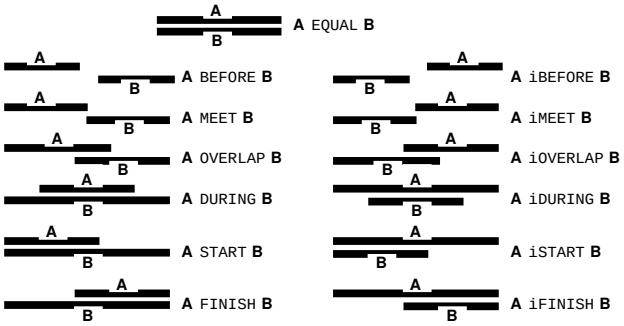

Figure 1: The possible 13 primitive time relationships between 2 intervals [2].

strictly sequential sub-actions, which, although adequate for the mapping into robot primitives, seems to be a strong restriction for the representation of generic human actions.

Similarly, Nagel ([16]) uses *transition diagrams* to represent driver's maneuvers in a highway, but provides no means to represent overlapping activities. Siskind's approach ([23, 24]) uses an *event logic* to represent basic actions, temporarily connected using Allen's primitive relationships. However, besides being a fully exponential modal logic, the proposed temporal propagation method using *spanning intervals* is quite inefficient.

2 Representation of Temporal Structure of Actions

To represent temporal structures we use an *interval algebra constraint network* ([1]), or simply an *IA-network*, which is a constraint satisfaction network ([9]) where the variables correspond to time intervals, and the arcs to binary temporal constraints between the intervals. The temporal constraints are expressed using Allen's interval algebra ([2]) which employs disjunctions of the 13 possible primitive relationships between two time intervals: the relations *equal* (e), *before* (b), *meet* (m), *overlap* (o), *during* (d), *start* (s), *finish* (f), and their inverses, *ib*, *im*, *io*, *id*, *is*, and *if* (see fig. 1 and [2] for the definition of those relationships).

Input Specification

Figure 2 shows the representation for the temporal structure of a "pick-up bowl" action as it is used by the detection system described later in this paper. The interval `pick-up-bowl` corresponds to the period where the action of picking up the bowl is occuring. This action is decomposed into two sub-actions corresponding to the intervals `reach-for-bowl` and `grasp-bowl`.

The relations between those three intervals are defined by the arcs connecting them. The sub-action `reach-for-bowl` is declared to be a time interval which has the same beginning as `pick-up-bowl`, but finishes first — the $\{s\}$ relationship. Similarly, `grasp-bowl` finishes at the same time as `pick-up-bowl` ($\{f\}$). The relationship between `reach-for-bowl` and `grasp-bowl` is defined in more vague terms: they either immediately follow each other or happen in sequence — represented by the disjunction $\{m, b\}$. Implicit in this constraint is the fact that the two sub-actions are mutually exclusive.

Allen's algebra was chosen as the underlying temporal formalism because we consider essential the ability to express mutually exclusive actions; for this reason, we can not use temporal algebras based on relations between endpoints (like [6, 27]), in spite of the better performance of the reasoning methods they provide.

There is also a question of the adequacy of the initial specification of the action, i.e., is the specification sufficiently constrained to recognize the action given the sensors? This problem will be discussed later, but a basic result of the method developed here is the ability to determine when a description is insufficient.

Sensor Specification

The next level of decomposition involves two complementarity predicates (written as $\{m, im\}$) about the physical relation between the bowl and the hands, `bowl-in-hands` and `bowl-out-of-hands`. The fact that reaching for the bowl must happen while the bowl is not in contact with the hands is expressed by the $\{d, f\}$ relationship between `reach-for-bowl` and `bowl-out-of-hands` . Similarly, `bow-in-hands` starts during `grasp-bowl` or just after its end ($\{m, o\}$).

For each of the previous "physical" predicates we can assign simple, low-level detectors (marked by the prefix `DET:`). The first, `DET:hands-close-sta-bowl`, detects if the hands are close to the bowl while the bowl is static and on the table. The other two detectors `DET:bowl-on-table` and `DET:bowl-off-table` identify the presence of the bowl on or off the table. The first two detectors can fire only when the bowl is out of hands while `DET:bowl-off-table` can only happen while the bowl is being held.

Inference of Stronger Constraints

Notice that most of the relationships defined in this example do not involve "deep" common-sense reasoning. For instance, `bowl-in-hands` and `bowl-out-of-hands` are temporally mutually exclusive simply because by definition they represent logically opposite states of the bowl. In [17], it is shown that it is possible to generate the decomposition of actions into sub-actions using "shallow" common-sense reasoning, using a representation scheme based on Schank's *conceptualizations* ([22]). We believe that weak temporal constraints similar to the ones used in this representation of the "pick-up bowl" action can also be automatically generated by a system composed of a dictionary of basic actions and simple reasoning.

Moreover, it is possible to infer stronger temporal constraints between the actions by pre-processing the IA-network using Allen's path-consistency algorithm ([2]). The result of the algorithm is an approximation of the *temporal closure* of the network, that is, the set of all temporal relations which follow from the initial constraints (see [27] for a discussion about how well Allen's algorithm approximate the temporal closure). Figure 3 shows part of the IA-network of "pick-up bowl" after path-consistency, where the intervals corresponding to detectors are omitted for clarity. Notice that path-consistency generates a completely connected graph by determining constraints between every pair of intervals. Also, it tightens the temporal relationships of the network, as, for instance, when it propagates the original constraint that `reach-for-bowl` is followed by `grasp-bowl` ($\{m, b\}$) by

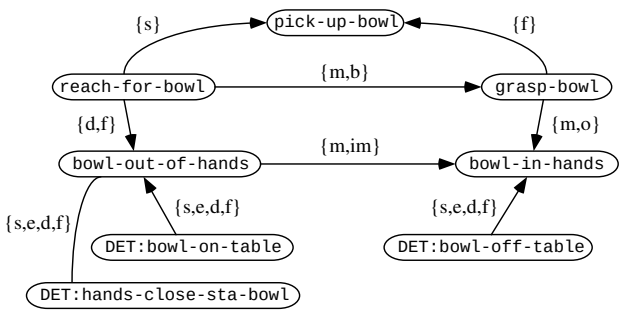

Figure 2: IA-network corresponding to the temporal structure of a "pick-up bowl" action .

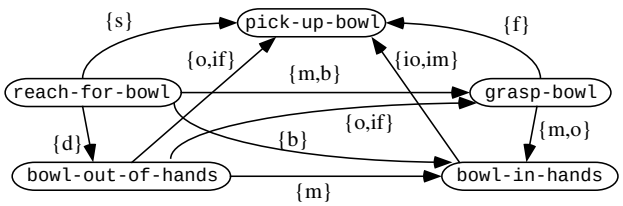

Figure 3: Part of the IA-network of the "pick-up-bowl" action generated by Allen's path-consistency algorithm (the intervals corresponding to detectors are omitted for clarity).

forcing `bowl-out-of-hands` to accept just a *meet* ($\{m\}$)) relationship with `bowl-in-hands` — instead of the original $\{m, im\}$.

3 A New Approach: PNF-Networks

IA-networks are normally used in tasks involving planning or scheduling. Typically, unary constraints are imposed on the intervals, as, for example, by constraining the duration of some of the intervals, or requiring that some of the intervals to be contained in pre-determined intervals (see [14] for an account of different methods).

Using constraint propagation techniques ([14]) it is possible to determine if there is an assignment of time intervals to all the variables $x_1, x_2, \ldots, x_n$ of an IA-network X which satisfy both the original binary constraints between intervals and the unary constraints corresponding to a particular case of the problem. Such a successful assignment is called a *solution* of the IA-network, and if there exists at least one solution, the IA-network is said to be *consistent*. Given a variable x_i, a *feasible value* for x_i is a time interval which belongs to at least one solution of X. The set of all feasible values of a variable x_i is called the *minimal domain* of x_i. Unfortunately, determining the consistency, finding solutions, and calculating the minimal domain in IA-networks is *NP-hard* in the number of constraints ([26]), preventing their use in practical detection problems.

To make the action detection tractable, we develop a method which maps the original IA-network into a simpler network which allows the use of faster, polynomial algorithms to approximately solve the detection problem. The basic idea is that in action detection we are not interested in when the intervals can happen, but only whether they are happening during the current instant of time. This

$\gamma(Q) = \langle r_P, r_N, r_F \rangle$			
	r_P	r_N	r_F
e	P	N	F
b	PNF	F	F
ib	P	P	PNF
m	PN	F	F
im	P	P	NF
o	PN	NF	F
io	P	PN	NF
s	PN	N	F
is	P	PN	F
d	PN	N	NF
id	P	PNF	F
f	P	N	NF
if	P	NF	F

Table 1: The function γ which maps Allen's primitives into PNF-network constraints.

motivates our definition of *PNF-networks*, a temporal constraint network where the only values a variable can assume are "past" (`past`), "now" (`now`), and "future" (`fut`).

3.1 PNF-Networks

Formally, a *PNF-network* is a binary constraint satisfaction network where the domain of the all variables $w_1, w_2, \ldots, w_n$ is the set of symbols $m = \{\texttt{past}, \texttt{now}, \texttt{fut}\}$. To simplify the notation, we define the set M of subsets of m,

$$M = \{\emptyset, \{\texttt{past}\}, \{\texttt{now}\}, \{\texttt{fut}\}, \{\texttt{past}, \texttt{now}\}, \{\texttt{past}, \texttt{fut}\}, \{\texttt{now}, \texttt{fut}\}, \{\texttt{past}, \texttt{now}, \texttt{fut}\}\}$$

whose elements are abbreviated as

$$M = \{\texttt{EMP}, \texttt{P}, \texttt{N}, \texttt{F}, \texttt{PN}, \texttt{PF}, \texttt{NF}, \texttt{PNF}\}$$

Therefore, we can express an unary constraint P_i on a variable w_i by an element of M. For instance, $w_i \in \texttt{PN}$ constrains w_i to be either `past` or `now`. We notate this by saying that $W_i = \texttt{PN}$ is the *restricted domain* of w_i.

A binary constraint P_{ij} between two variables w_i and w_j is a truth matrix which determines which the admissible values are for the pair of variables (w_i, w_j). A compact way to represent the matrix is to use a triad $\langle r_P, r_N, r_F \rangle \in M \times M \times M$ where r_P represents the admissible values for w_j when $w_i = \texttt{past}$, and similarly for r_N and r_F. For example, consider the binary constraint represented by $P_{ij} = \langle \texttt{PN}, \texttt{F}, \texttt{F} \rangle$. According to this constraint, the only values that the pair of variables (w_i, w_j) can assume are $(\texttt{past}, \texttt{past})$, $(\texttt{past}, \texttt{now})$, $(\texttt{now}, \texttt{fut})$, and $(\texttt{fut}, \texttt{fut})$.

Binary constraints between variables in a IA-network can be mapped into binary constraints in a PNF-network. Suppose a pair of variables (w_i, w_j) is constrained in the original IA-network by the relation *meet* ($\{m\}$), that is, the interval w_i is immediately followed by w_j. Intuitively, if w_i is happening now, $w_i = \texttt{now}$, then w_j can only occur in the future, yielding $w_j = \texttt{fut}$. Similarly, if $w_i = \texttt{fut}$, then w_j must also be `fut`. Finally, if it is know that $w_i = \texttt{past}$, the interval w_j must have already started, although it may have finished or not — $w_j \in \texttt{PN}$.

3.2 Projecting IA-networks into PNF-networks

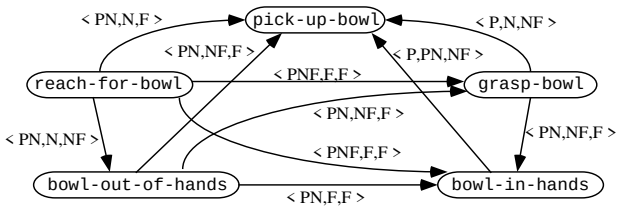

Figure 4: Projection of the IA-network shown in fig. 3 into a PNF-network. The domain of all node variables is PNF.

As we saw above, the relationship *meet* ($\{m\}$) between w_i and w_j can be mapped into the binary constraint $P_{ij} = \langle \mathtt{PN}, \mathtt{F}, \mathtt{F} \rangle$ such as to preserve the temporal ordering contained in the original IA-network. Table 1 displays the function γ which maps the 13 Allen's primitive relationships into their equivalent binary-constraints between PNF symbols. Details about how the function γ preserves the semantics of the original IA-network can be found in [18].

Given a binary constraint in an IA-network, i.e., a set of primitive relationships Q, we can define its projection by the function Γ:

$$\Gamma(Q) = \bigcup_{\lambda \in Q} \gamma(\lambda)$$

where the union of the primitive binary constraints is defined as their component-wise union. For instance, if $Q = \{b, ib\}$, then

$$\begin{aligned} \Gamma(\{b, ib\}) &= \langle \mathtt{PNF}, \mathtt{F}, \mathtt{F} \rangle \cup \langle \mathtt{P}, \mathtt{P}, \mathtt{PNF} \rangle \\ &= \langle \mathtt{PNF}, \mathtt{PF}, \mathtt{PNF} \rangle \end{aligned}$$

Notice that $Q = \{b, ib\}$ is basically a representation for mutually exclusive intervals, and, in this case, the only constraint that can be inferred in the PNF-network is that if the first interval is happening, $w_i = \mathtt{N}$, then the other interval is not happening, $w_j \in \mathtt{PF}$.

Given an IA-network of variables $v_1, v_2, \ldots, v_n$ and the binary constraints Q_{ij} between each pair of variables v_i and v_j, we can project the IA-network into a PNF-network where each variable v_i is mapped into a variable w_i of discrete domain $m = \{\mathtt{past}, \mathtt{now}, \mathtt{fut}\}$; and each binary constraint Q_{ij} is projected into $\Gamma(Q_{ij})$. Figure 4 shows the projection of the IA-network for "pick-up bowl" of fig. 3 into a PNF-network (the detector nodes are omitted for clarity).

3.3 Component Domains

The fundamental part of our action detection method (detailed later) is the computation of the minimal domain of all variables of a special PNF-network where sensor information is represented as unary constraints on the variables. To describe precisely this process, we start by defining some useful concepts.

Given a PNF-network on variables $w_1, w_2, \ldots, w_n$, a *component domain* W of the PNF-network is any combination of restricted domains on the variables w_i, notated by $W = (W_i)_i = (W_1, W_2, \ldots, W_n)$, where each W_i is a subset of $\{\mathtt{past}, \mathtt{now}, \mathtt{fut}\}$. The value of each W_i in a component domain W is called the PNF-*domain* of the variable w_i. We also denote by U the set of all component domains of a PNF-network, $U = M^n$, where n is the number of variables corresponding to the intervals in the PNF-network.

Component domains are a good notation for the representation of unary constraints coming from binary sensors. Typically, we associate with a binary sensor only two values, N if the sensor is on, and PF if the sensor is off. Therefore, we can represent all the information coming from sensors by a component domain $S = (S_i)_i$, where

$$S_i = \begin{cases} \text{value of the } w_i\text{–sensor} & \text{if } w_i \text{ is a sensor} \\ \mathtt{PNF} & \text{otherwise} \end{cases}$$

Generally, a component domain can be seen as a representation for a complete set of unary constraints on the variables of a PNF-network.

3.4 PNF-Restriction

Given a PNF-network and a component domain W, consider the problem of finding the minimal domain of the network by imposing the components of W as unary constraints on the network's variables. The result of the computation can also be represented by a component domain called the *restriction of* W, $\mathcal{R}(W)$, where each component $\mathcal{R}(W)_i$ represents the minimal domain of the variable w_i. We can see the restriction function $\mathcal{R} : U \to U$ as "removing" the elements of each original variable domain W_i which do not appear in at least one solution. If the minimal domain of any variable becomes the empty set, EMP, then we say that the component domain $\mathcal{R}(W)$ is *collapsed*.

Computing the minimal domain in constraint satisfaction networks is, in general, a *NP-hard* problem. However, the constraint satisfaction network literature lists many cases where special characteristics of the network reduce the complexity of the problem ([6, 14]). In our experiments, we have been employing an *arc-consistency*algorithm ([12]) to compute an approximation of the restriction function $\mathcal{R}$, as described in the appendix.

The result of arc-consistency algorithm always contains all the solutions of a constraint satisfaction network. Moreover, in our use of that algorithm, we have never found a situation where the arc-consistency algorithm has produced a component domain larger than the PNF-restriction. In other words, we have some reason to believe that computing PNF-restriction is in fact a simpler problem than the computing of the minimal domain of a general constraint satisfaction network, and achievable in linear time by the arc-consistency algorithm of the appendix. Presently we are trying to determine whether this conjecture is true.

4 Action Detection Using PNF Propagation

When using PNF-restriction for action detection, we consider the computed minimal domain of each variable to determine whether the action may be happening. If the minimal domain of the interval is N, we can say that the action is happening; if it is P, F, or PF the action can be said to be not happening; otherwise (PNF), its state is indeterminate.

PNF-restriction deals exclusively with determining feasible options of an action *at a given moment of time*. However, information from the previous time step can be used to constrain the occurrence of actions in the next instant.

For example, after an interval is determined to be in the past, it should be impossible for it to assume another PNF-value, since the action is over. Similarly, if the current value of the interval is now, in the next instant of time it can still be now, or the corresponding action might have ended, when the interval should be past. To capture these ideas, we define a function that time-expands a component domain into another.

4.1 Time Expansion

We define a *time expansion function*, $\mathcal{T} : U \rightarrow U$, that considers a component domain W^t at time t and computes the smallest component domain W^{t+1} at time $t+1$ that satisfies the intuitive meanings of past, now, and future. We start by defining a time expansion function for each element of $m = \{\texttt{past}, \texttt{now}, \texttt{fut}\}$, $\mathcal{T}_m : m \rightarrow M$. A natural choice is the time expansion function $\mathcal{T}_m : m \rightarrow M$

$$\mathcal{T}_m(\texttt{past}) = \texttt{P} \quad \mathcal{T}_m(\texttt{now}) = \texttt{PN} \quad \mathcal{T}_m(\texttt{fut}) = \texttt{NF}$$

Given the function that time-expands elements $\mathcal{T}_m$, we define the function that expands the elements of M, $\mathcal{T}_M : M \rightarrow M$ as being the union of the results of $\mathcal{T}_m$,

$$\mathcal{T}_M(\Phi) = \bigcup_{\phi \in \Phi} \mathcal{T}_m(\phi)$$

and the time expansion of a component domain W, $\mathcal{T} : U \rightarrow U$, as the component-wise application of $\mathcal{T}_M$ on a component domain $W = (W_i)_i$, $\mathcal{T}(W) = (\mathcal{T}_M(W_1), \mathcal{T}_M(W_2), \ldots, \mathcal{T}_M(W_n))$.

4.2 PNF Propagation

PNF *propagation* is a method to detect the occurrence of actions which is based on intersecting the information from the sensors with the time expansion of the component domain representing all the past information.

Formally, given the PNF-network corresponding to all actions, sub-actions, and detectors, PNF propagation determines the minimal domain of each interval at each time t, represented by the component domain W^t, called the *state of the* PNF-*network* at time t. Sensor information is gathered in the component domain S^t where all states are PNF except those corresponding to perceptual sensors which are assigned the sensor's corresponding value as explained before.

The initial component domain W^0 represents an initial state of total ignorance, $W^0 = (\texttt{PNF})_i$. After that, we determine W^t by time-expanding the previous state W^{t-1} — and therefore determining the possible sequences for that state —, intersecting it to the information from the sensors S^t, and applying restriction to remove logical impossibilities due to temporal constraints. That is

$$W^t \leftarrow \mathcal{R}(\mathcal{T}(W^{t-1}) \cap S^t)$$

It is necessary to time expand the component W^{t-1} before intersecting it with the perceptual information S^t, since between instant $t-1$ and t actions may have ended or begun. Using the past information is a fundamental part of PNF propagation, as shown in the following experimental results.

5 Results

We have been running experiments on three different actions, "pick-up bowl", "wrapping chicken", and "mixing ingredients", using footage from our previous research on automatic TV cameras in cooking shows ([17]). The temporal structure of each action is codified manually into an IA-network. The IA-networks are pre-processed using Allen's path-consistency algorithm (revised in [27]) and then converted into PNF-networks.

To test the PNF formulation, we manually extract the values for the sensors by watching a video of the action being performed. We also determine the interval where every action and sub-action is actually happening, and use the information to evaluate the performance of the action detection method. Using manually extracted information allows the simulation of sensors with different levels of correctness.

5.1 Perfect Sensors

Let's consider the detection of actions using the PNF propagation method as described above. In this subsection we presume perfect sensors; following we develop fault-tolerant versions. To illustrate some characteristics and strengths of the method, we start examining some results for the detection of the action "pick-up bowl" shown in fig. 5. The top part of the figure displays the temporal diagrams for the PNF-domains of the detectors (marked as DET:) and the true state of all other intervals (marked as TRUE:), for a particular instance of the action of picking up a bowl. The data were obtained manually from a video depicting the action. The diagram employs different symbols for each PNF-domain, under the convention that the bottom line represents the fut state, the middle represents now, and the top, past (see the legend at the bottom of fig. 5).

Fig. 5a shows the results when the detection process uses the description of the "pick-up bowl" action exactly as given in fig. 2. Basically, only the physical events related to intervals bowl-in-hands and bowl-out-of-hands are recognized. The main action, pick-up-bowl, is never fully detected. However, in the initial period the method determines that the action may have started, but it is not finished (by the value NF). Similarly, after DET: bowl-off-table becomes N, it is detected that the action is happening or has already happened (PN).

The problem is that the definition of "pick-up bowl" of fig. 2 have constraints that are too weak, although always true. We intentionally construct a weak example because it illustrates clearly some of the extra assumptions that are needed to link events detected by simple sensors to actions and sub-actions.

For example, one of the problems is that the original definition of "pick-up bowl" lacks a causal link between detecting that the bowl is not on the table and the result of the act of grasping. This can be accomplished by determining the relationship between DET: bowl-off-table and grasp-bowl to be sequential and mutually exclusive $\{b, m\}$. Part b of fig. 5 shows that, with the addition of such relation, the end of pick-up-bowl is detected. Notice that the causal link assumes that the only cause for movement of the bowl is a grasping action by the agent of grasping. Movements of the bowl caused by other agents,

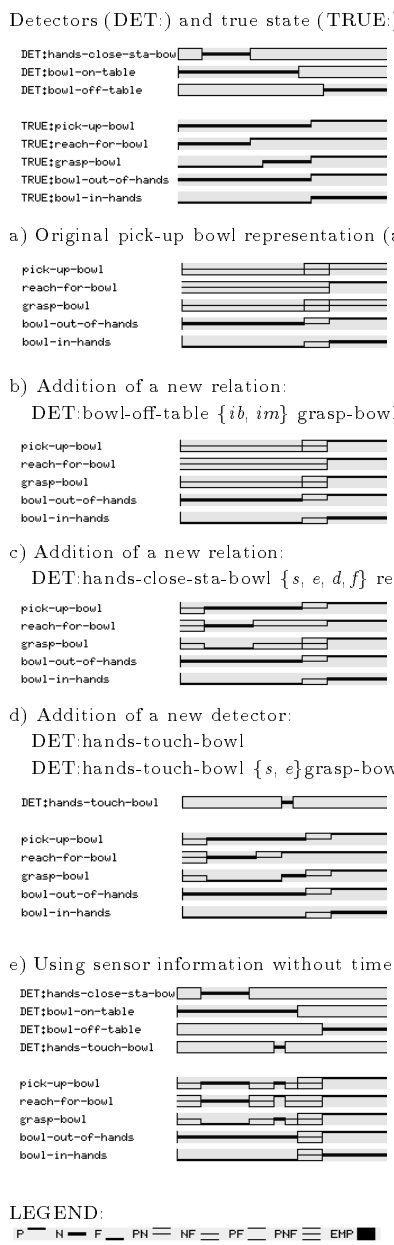

Figure 5: Influence of the temporal structure on the detection of the action "pick-up bowl".

objects (the bowl could be pushed by a spoon), or forces (gravity) are not considered in the context.

To detect the beginning of `pick-up-bowl`, it is necessary that the action description includes some causal relationship about the beginning of the sub-action `reach-for-bowl`. A possible way is to indicate that the proximity between hands and bowl (as detected by `DET:hands-close-sta-bowl`) is an indicator for the occurrence of `reach-for-bowl`. Again, notice that by doing this, we are assigning a relationship which may not be always true. However, given the simplicity of our sensors (and

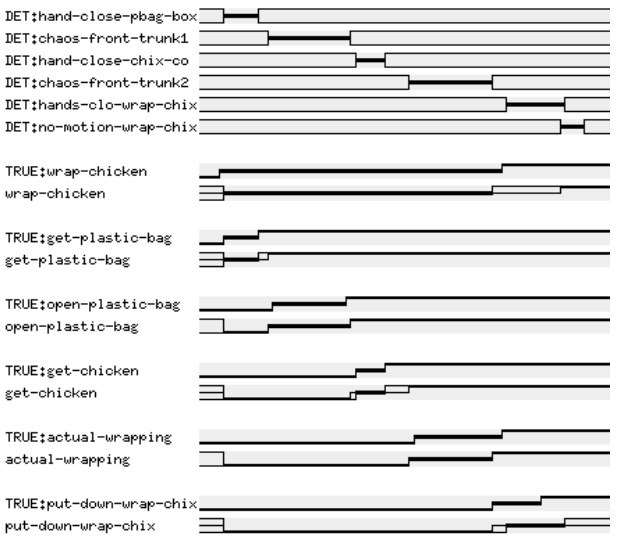

Figure 6: Detection of the action "wrapping chicken".

of most state-of-art vision-based algorithms), such "intentional" links are necessary to detect higher level actions. The results, shown in part c, display the almost perfect detection of `pick-up-bowl` and `reach-for-bowl`.

Finally, if we also want to detect the beginning of the occurrence of `grasp-bowl`, a new detector is necessary. This is shown in part d of fig. 5, which displays the temporal diagram of a new sensor, `DET:hands-touch-bowl` which fires precisely when the hand touches the bowl. In this last case, the states of the intervals are known in most times, and are correct (compare to the `TRUE:` diagram at the top of the figure).

Part e of fig. 5 shows the importance of the information from the previous instant of time on the strength of `PNF` propagation. In this case, W^t is computed solely on the information from the sensors, $W^t = \mathcal{R}(S^t)$. Comparing fig. 5e with part d, we can see a distinct degradation in the results. The main reason is that after a cause for an interval being in the `now` state ceases to exist, the system still considers that the interval can still happen in the `fut` (compare `pick-up-bowl` in both cases).

Figure 6 illustrates the detection of a more complex action, wrapping chicken with a plastic bag, which involves 25 sub-actions and 6 sensors. The constraints were designed to resemble the result of an automatic inference system operating on a compact description of the action based on Schank's conceptualizations ([22]). In [21], Rieger discusses the implementation of such an inference system; in a previous work we have implemented a simpler system able to operate in the context of the cooking show domain ([17]).

Figure 6 displays the true and the recognized state of the main action and of five sub-actions, each of them with a level of complexity similar to the "pick-up bowl" shown above. All the sensors are very simple: proximity between hands and the box containing plastic bags (`DET:hand-close-pbag-box`) and the plate containing chicken (`DET:hand-close-chix-co`), chaotic movement

in front of the subject's trunk (DET:chaos-front-trunk), and absence of motion in the area of the wrapped chicken (DET:no-motion-wrap-chix). Notice that the main action and the 5 sub-actions in the example are correctly detected most of the time.

5.2 Detection with At Most One Wrong Sensor

Implicit in the PNF propagation method as described above is the assumption that the sensors are always correct. When computing the current component domain W^t using the previous domain W^{t-1} (time-expanded), only the possible domains allowed by the previous values of the sensors, S^{t-1}, are considered. For instance, if a sensor erroneously set up an interval to past, the proposed method has no way to recover from the error.

To overcome this limitation, we are using a scheme where the detection system keeps track of some of the different possibilities derivated from considering the sensors not to be perfect. In this paper we describe a method that assumes that at most one sensor is wrong, and guarantees the recovery in such cases. The price for this ability to tolerate errors is an increase in space and time requirements, and a decrease in detection power.

Our approach relies in the construction, for each instant of time t, of a set of *error-tolerant domains*, Ω^t. Typically, there is total ignorance at time t_0, $\Omega^0 = \{(\texttt{PNF})_i\}$. Let's also call S_0^t the values of the l sensors in a PNF-network at instant t. Consider the component domains $S_1^t, S_2^t, \ldots, S_l^t$ where S_j^t, $j = 1, 2, \ldots, l$, is obtained by "inverting" the value of the j-th sensor in S_0^t.

If we want Ω^t to include all possible threads of events considering that at most one sensor is wrong at any given time, we can define the set of error-tolerant domains, Ω^t, as follows:

$$\Omega^t = \{\mathcal{R}(\mathcal{T}(\omega) \cap S_i^t) \text{ not collapsed } \mid \text{ for all } i = 0, 1, 2, \ldots, l \text{ and } \omega \in \Omega^{t-1}\}$$

In other words, the set of error-tolerant domains contains all non-collapsed restrictions of the combinations between previous error-tolerant domains and current values of the sensors, with at most one error allowed.

A potential problem is that Ω^t can increase exponentially by this method. In our tests, however, we found that the large number of repetitions and collapsed component domains kept the size of Ω^t almost constant. A possible explanation can be related to the observation ([11]) that the number of consistent IA-networks is very small if the number of nodes is greater than 15. Currently, we are also developing methods based in the certainty factor of the sensor to avoid the expansion of highly improbable component domains.

The method describe above guarantees that the progression of sets of error-tolerant domains $\Omega^0, \Omega^1, \ldots$, always contains the correct component domain if *at most one sensor is wrong each instant of time*. Therefore, under this constraint, we can assure that if we declare the current values of the intervals to be

$$W^t = \bigcup_{\omega \in \Omega^t} \omega$$

then each W_i^t contains the right answer. However, computing W^t in this manner is over-conservative: many times, most of the intervals are assigned the PNF value.

To address this problem, we propose to decouple the building of the set of error-tolerant domains from the choice of current values for the intervals. It is important to keep all the set of error-tolerant domains as complete as possible. However, we determined that a good policy is to make the current value W^t based on the actual readings of the sensors, considering all the different possibilities of past errors.

Basically, we define W^t to be the union of restriction of all previous domains intersected only with the current value of the sensors,

$$W^t = \bigcup_{\omega \in \Omega^{t-1}} \mathcal{R}(\mathcal{T}(\omega) \cap S_0^t)$$

Of course, if this quantity ends up being empty, then we use the over-conservative method for computing W^t.

In summary, the set Ω^t is computed always considering that one of the sensors might be wrong (plus the previous set Ω^{t-1}), but the current state considers just that the sensors could be wrong in the past, but not in the present. This was found to be a good compromise between accuracy and ability to recover from errors.

5.3 General Error-Tolerant Detection

In practice, we need to be able to recover, at least partially, even in the case of more than one simultaneous error. Instead of explicitly modeling sensory component domains with more than one error — which leads to an exponential increase in time and computation—, we opted for a more conservative (and less computationally intensive) approach. Our basic concern are the situations where all the current sensor readings allow no consistent interpretation with the past information, that is, $\Omega^t = \emptyset$. In this case, we use an heuristic for recovery which disregards completely past information, considering only the possible implications of current sensor information,

$$\Omega^t = \{\mathcal{R}(S_i^t) \text{ not collapsed } \mid \text{ for all } i = 0, 1, 2, \ldots, l\}$$

and, if this is still an empty set, we define the set of error-tolerant domains to be the one with the least information possible, $\Omega^t = \{(\texttt{PNF})_i\}$.

Figure 7 shows examples of the results obtained by combining the methods described above. As a measure of comparison, we provide in fig. 7a the results obtained assuming perfect sensors (the same as in fig. 5d). Using the methods described above, the processing of the same data produces the results in fig. 7b, where there are intervals of time where the system is unable to decide whether the actions are happening or not. Surprisingly, when we introduce errors in the detectors DET:hands-close-sta-bowl and DET:bowl-off-table (but never at the same time), there is an increase in the detection accuracy, as show in fig. 7c. This can be explained by considering that when a sensor has a false positive, its corresponding value of N in the PNF-domain has the power of produce lots of contradictions, pruning the set of error-tolerant domains in an early stage.

Figure 8 shows other results, obtained by adding erroneous information into the sensors of the "wrapping

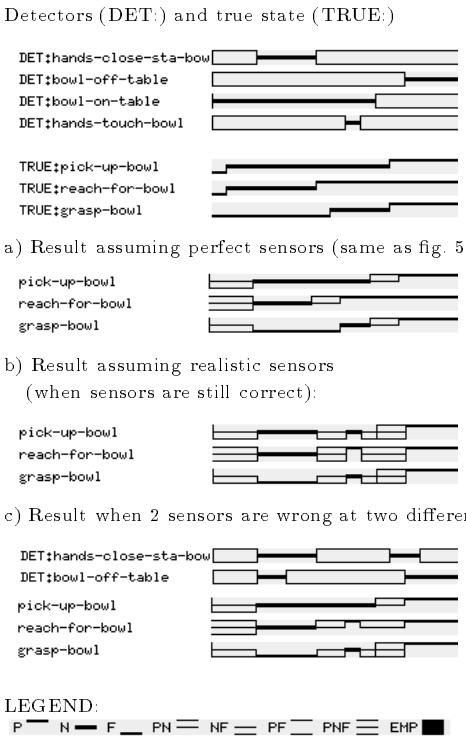

Figure 7: Influence of sensor errors on the detection of the action "pick-up bowl".

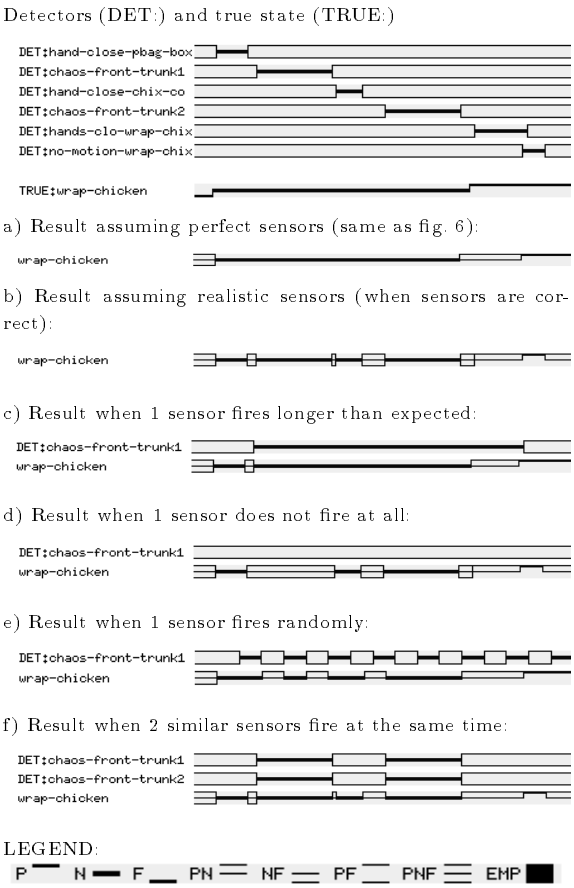

Figure 8: Influence of sensor errors on the detection of the action "wrapping chicken".

chicken" action of fig. 6. In fig. 8a, we repeat the result of fig. 6 for comparison. Part b shows the result when all the information from the sensors is correct, but employing the error recovery system described above. Basically, the result is the same except when all the sensors are off (`PF`). This degradation on performance when sensors are off is examined further in fig. 8c and d, where we observe the effects of false positives and false negatives of the same sensor, `DET:chaos-front-trunk1` on the detection of the action `wrap-chicken`. Notice that a false `now` value increases the detection, while a false `PF` value gives rise to segments of complete ignorance (`PNF`). Figure 8e and f examine two other cases, showing the robustness of the method to repeated errors and to confusion between similar sensors.

6 Final Remarks

This work provides a method — `PNF` propagation — to efficiently exploit the temporal constraints inherent to human action to increase the detection power of simple sensors. The method is based on the simplification into a `PNF`-network of the IA-network representing the temporal structure of an action. To detect the occurrence of actions, we consider the sensor values and the past information as constraints on the variables of the `PNF`-network, and calculate an approximation of the minimal domain using an arc-consistency algorithm. `PNF` propagation is shown to have a good performance when the sensors are assumed to be perfect.

The heuristic method provided in the last section for coping with faulty sensors is still under development. Our basic goal is to incorporate explicitly the certainty coefficients from the sensors into the process of computing the set of error-tolerant domains and in the selection of the current state. It is possible to see our current approach as a method where all the sensors have the same failure probability ϵ, assumed to be very small. Therefore, the probability of double failure becomes almost zero, justifying the sole consideration of the single-failure sensor domains S_j^t. Moreover, the selection of the current state W^t can be seen as taking the union of the possibilities with highest probability: the component domains which assume that the current state is right have a considerably higher probability of occurrence, since ϵ is small.

Moreover, we can also use the sensor certainty coefficients to avoid a potential exponential explosion in the number of elements in the set of error-tolerant domains Ω^t. Currently, the explosion does not occur because of the large number of identical states generated, and also because about 50% of the component domains are collapsed after restriction. Theoretically, however, there is no reason for such a limitation, and an obvious approach would be to make the error-tolerant domains in Ω^t to be restricted to a fixed number of highest-probable error-tolerant domains.

```
Input:  a PNF-network with variables x_1, x_2, ..., x_n,
          and binary constraints P_ij
        W, a component domain, W = (W_i)_i,
          representing unary constraints in the variables
Output: AC-R(W),
          a component domain that is arc-consistent
Algorithm:
    initialize queue with all variables x_i
      where W_i ≠ PNF                                   (1)
    W̄ ← W                                               (2)
    while queue ≠ ∅                                     (3)
        x_{i_o} ← pop(queue)                            (4)
        for each variable x_i                           (5)
            X ← F(W̄_{i_o}, P_{i_o i})                   (6)
            when W̄_i ≠ W̄_i ∩ X                          (7)
                W̄_i ← W̄_i ∩ X                           (8)
                push(x_i, queue)                        (9)
    return W̄                                            (10)
```

Figure 9: An arc-consistency algorithm to compute an upper bound for the restriction of a component domain W, AC-$\mathcal{R}(W)$.

Probabilistic methods for visual action recognition have been proposed in the computer vision community ([5]). Unlike that approach, we believe that it is important to exploit the fact that logical impossibilities prevent the occurrence of some sequences of sub-actions, and that is necessary to incorporate such constraints into vision systems designed to recognize human action. As shown by the previous results, the addition of a single temporal constraint between two intervals can dramatically increase the recognition capability of the detection system.

We are aware of some limitations of the approach. The first, obvious one, is if the computation of PNF-restriction is in fact *NP-hard*, that is, if we determine that the algorithm based on arc-consistency is too weak for detection purposes. However, we do know from our tests that PNF-restriction actually reduces significantly a component PNF-domain.

A second limitation refers to the expressive capabilities of using intervals and their temporal relationships to represent human actions. For instance, in our work in "XxxxxXxxxx" ([19]), we realized the need of provisions to represent cyclic actions which are not met even by Allen's temporal logic. One possible approach to allow cycles is to modify the time expansion function of past to $\mathcal{T}_m(\texttt{past}) = \texttt{PNF}$ in some especial conditions.

A An Algorithm to Approximately Compute the PNF-Restriction

To compute an approximation of the restriction of a component domain W, $R(W)$, we have been using an algorithm for arc consistency which is guaranteed to contain $\mathcal{R}(W)$. We call this algorithm AC-$\mathcal{R}(W)$. Fig. 9 shows this algorithm, in fact a version of the arc-consistency algorithm AC-2 proposed in [12] and adapted to the component domain notation. The algorithm uses the function $\mathcal{F}$, that, given a variable domain X and a PNF constraint $P_{ij} = \langle r_P, r_N, r_F \rangle$ between variables x_i and x_j, returns the possible domain of x_j when x_i assumes values in X. In order to define $\mathcal{F}$, we first define the function f

$$f(x, \langle r_P, r_N, r_F \rangle) = \begin{cases} r_P & \text{if } x = \texttt{past} \\ r_N & \text{if } x = \texttt{now} \\ r_F & \text{if } x = \texttt{fut} \end{cases}$$

$\mathcal{F}$ is the union of $f(x, P)$ over all the elements x in the PNF-domain λ,

$$\mathcal{F}(\lambda, P) = \bigcup_{x \in \lambda} f(x, P)$$

The first step of the algorithm consists in detecting which variables of a component domain W are different from PNF, and queuing all those variables for further expansion. Then $\overline{W}$, the component domain to be returned, is initialized identically to the input W.

The core of the algorithm is a loop which ends when the queue is empty. In each cycle, one variable x_{i_o} at state $\overline{W}_{i_o}$ is examined. For each variable x_i an auxiliary variable X is assigned the possible values for the domain of x_i given the present domain of x_{i_o}, computed by the function $\mathcal{F}$ described above. In steps 7–9, if necessary, the domain of x_i is actualized with the intersection of its previous domain and X, and the modified variable is pushed into the queue.

Since the result of arc-consistency always contains all the solutions of a constraint satisfaction network, it is clear that $\mathcal{R}(W) \subseteq$ AC-$\mathcal{R}(W)$. The basic argument shown in [12] proves that the worst case complexity of the algorithm is linear in the number of constraints.

References

[1] James F. Allen. Maintaining knowledge about temporal intervals. *Communications of the ACM*, 26(11):832–843, November 1983.

[2] James F. Allen. Towards a general theory of action and time. *Artificial Intelligence*, 23:123–154, 1984.

[3] James F. Allen and George Ferguson. Actions and events in interval temporal logic. *Journal of Logic and Computation*, 4(5):531–579, 1994.

[4] Aaron F. Bobick. Movement, activity, and action: The role of knowledge in the perception of motion. *Royal Society Mtg Knowledge-based Vision: Mechanisms and Applications*, 1997.

[5] Matthew Brand, Nuria Oliver, and Alex Pentland. Coupled hidden Markov models for complex action recognition. In *Proc. of CVPR'97*, Puerto Rico, USA, 1997.

[6] Rina Dechter, Itay Meiri, and Judea Pearl. Temporal constraint networks. *Artificial Intelligence*, 49(1–3):225–233, May 1991.

[7] David Israel, John Perry, and Syun Tutiya. Actions and movements. In *Proc. of the 12th IJCAI*, pages 1060–1065, Sydney, Australia, August 1991.

[8] Ray Jackendoff. *Semantic Structures*. The M.I.T. Press, Cambridge, MA, 1990.

[9] Vipin Kumar. Algorithms for constraint-satisfaction problems: A survey. *AI Magazine*, 13:32–44, 1992.

[10] Y. Kuniyoshi and H. Inoue. Qualitative recognition of ongoing human action sequences. In *Proc. of IJCAI'93*, pages 1600–1609, 1993.

[11] Peter B. Ladkin and Alexander Reinefeld. Arc and path consistency revisited. *Artificial Intelligence*, 57(1):105–124, September 1992.

[12] A. K. Mackworth. Consistency in networks of relations. *Artificial Intelligence*, 8(1):99–118, 1977.

[13] Alan K. Mackworth. The complexity of some polynomial network consistency algorithms for constraint satisfaction problems. *Artificial Intelligence*, 25:65–74, 1985.

[14] Itay Meiri. Combining qualitative and quantitative constraints in temporal reasoning. *Artificial Intelligence*, 87(1–2):343–385, November 1996.

[15] Roger Mohr and Thomas C. Henderson. Arc and path consistency revisited. *Artificial Intelligence*, 28(2):225–233, March 1986.

[16] Hans-Hellmut Nagel. A vision of 'vision and language' comprises action: An example from road traffic. *Artificial Intelligence Review*, 8:189–214, 1995.

[17] Claudio S. Pinhanez and Aaron F. Bobick. Approximate world models: Incorporating qualitative and linguistic information into vision systems. In *AAAI'96*, pages 1116–1123, Portland, Oregon, August 1996.

[18] Claudio S. Pinhanez and Aaron F. Bobick. PNF Calculus: A method for the representation and fast recognition of temporal structure. Technical Report 389, M.I.T. Media Laboratory Perceptual Computing Section, September 1996.

[19] Claudio S. Pinhanez, Kenji Mase, and Aaron F. Bobick. Interval scripts: A design paradigm for story-based interactive systems. In *CHI'97*, pages 287–294, Atlanta, Georgia, March 1997.

[20] Steve Pinker. *Learnability and Cognition*. The M.I.T. Press, Cambridge, MA, 1989.

[21] Charles J. Rieger III. Conceptual memory and inference. In *Conceptual Information Processing*, chapter 5, pages 157–288. North-Holland, 1975.

[22] Roger C. Schank. Conceptual dependency theory. In *Conceptual Information Processing*, chapter 3, pages 22–82. North-Holland, 1975.

[23] Jeffrey Mark Siskind. *Naive Physics, Event Perception, Lexical Semantics, and Language Acquisition*. PhD thesis, M.I.T, Dept. of Electrical Engineering and Computer Science, January 1992.

[24] Jeffrey Mark Siskind. Grounding language in perception. *Artificial Intelligence Review*, 8:371–391, 1994.

[25] Peter van Beek. Reasoning about qualitative temporal information. *Artificial Intelligence*, 58(1–3):297–326, December 1992.

[26] Marc Vilain and Henry Kaut. Constraint propagation algorithms for temporal reasoning. In *Proc. of AAAI'86*, pages 377–382, Philadelphia, Pennsylvania, 1986.

[27] Marc Vilain, Henry Kautz, and Peter van Beek. Constraint propagation algorithms for temporal reasoning: A revised report. In Daniel S. Weld and Johan de Kleer, editors, *Readings in Qualitative Reasoning About Physical Systems*, pages 373–381. Morgan Kaufmann, San Mateo, California, 1990.