Constraint Nets: A Unitary Model for Hybrid Concurrent Systems

Ying Zhang and Alan K. Mackworth

Department of Computer Science
University of British Columbia
Vancouver, B.C., Canada V6T 1Z2
E-mail: zhang@cs.ubc.ca
Phone: (604)822-3731, Fax: (604)822-5485

Abstract

Hybrid systems are systems consisting of a non-trivial mixture of discrete and continuous components, such as a controller realized by a combination of digital and analog circuits, a robot composed of a controller and a physical plant, or a robotic system consisting of a robot coupled to a continuous environment.

Hybrid systems are more general than traditional real-time systems. The former can be composed of continuous subsystems in addition to discrete or event-controlled modules. Some models of hybrid computation have recently been proposed. In this paper, we present a different approach, taking a hybrid system as a general dynamic system.

Our Constraint Net model (CN) is capable of modeling and analyzing dynamic behaviors in hybrid systems. CN captures the most general structure of dynamic systems so that systems with discrete as well as dense time, and asynchronous as well as synchronous event structures, can be modeled in a unitary framework. CN provides multiple levels of abstraction so that a system can be developed hierarchically; therefore, the dynamics of the environment as well as the dynamics of the plant and the dynamics of computation and control can be modeled and then integrated. Since CN explicitly represents locality, we can use it to explore true concurrency in distributed systems. With a rigorous formalization in abstract algebra, CN provides a programming semantics for the design of hybrid real-time embedded systems. CN also serves as a foundation for the specification and verification of robotic systems.

1 Motivation and Introduction

A robot is generally composed of a controller which is embedded in a physical plant. A robotic system [22], the integration of a robot and its environment, is a typical hybrid system, consisting of a non-trivial mixture of discrete and continuous components [11]. The overall behavior of a robot is emergent from interactions among various components in the controller, the plant and the environment.

Much work has been done recently on introducing real-time concepts into formal models of concurrency [6]. For example, Alur and Dill [2] developed the theory of timed automata to reason about timed behaviors. Henzinger et al. [9] incorporated time into an interleaving model of concurrency in which upper and lower bounds on time delay are associated with each transition. Various real-time extensions of process algebras [16] have been proposed to model the relative speed of processes. The Timed Petri Net model [4] was introduced to specify and verify real-time systems. None of these models, however, are able to represent continuous change.

The traditional models for continuous (resp. synchronous discrete) dynamic systems are differential (resp. difference) algebraic equations. However, most advanced control systems involve asynchronous processes on computer networks in addition to analog or synchronous digital circuits. Differential (difference) algebraic equations are not adequate for modeling these systems.

Some effort has been made recently to develop models for hybrid systems by generalizing timed transition systems to phase transition systems [11, 15] in which computations consist of alternating phases of discrete transitions and continuous activities. A hybrid system specification using Z was described in [20]. A duration calculus based on continuous time was developed [7] in which integrators can be applied to predicates over a time interval. The use of weakest-precondition predicate transformers in the derivation of sequential, process-control software was discussed in [13].

Our approach to developing a model for hybrid systems is based on the following arguments. Instead of adding a model of time onto an existing model of concurrency, a model of dynamics should be developed in the first place [17]. Instead of fixing a model with particular data types, a model of dynamics should support abstract time structures and abstract data types. Using this approach, we have developed Constraint Nets (CN), a unitary model for hybrid dynamic systems.

The major influences on the Constraint Net model are Ashcroft's Operator Net model [3] and Lavignon and Shoham's Temporal Automaton model [10].

The Operator Net model, abstracted from Lucid [19], is defined on continuous algebras using fixpoint theory. The most attractive feature of this model is its independence of any particular algebra. Given a continuous algebra which specifies data types and basic operations, a sequence (continuous) algebra is obtained on which an operator net can be defined. LUSTRE [5], a development based on Lucid, is a real-time programming language, in which sequences are interpreted as time steps. In addition, LUSTRE introduces clocks, so that any expression is evaluated at its clock's sampling rate.

The Temporal Automaton model is a step towards modeling causal functions in multiple time domains. The Temporal Automaton model provides explicit representation of process time, symmetric representation of a machine and its environment, and aggregation of individual machines to form a machine at a coarser level of granularity. However, there remain untackled problems in modeling continuous change and event control.

As in the Operator Net model, CN is defined on continuous algebras using fixpoint theory. As does LUSTRE, CN introduces reference time structures and clocks, but the reference time of CN can be dense and the clocks of CN denote events [18]. As in the Temporal Automaton model, transductions are introduced as an abstraction of causal functions, but our definition of transductions is generalized to include continuous activities and event-driven transitions. In this definition, a transduction is a transformational process from a tuple of input traces to an output trace. Furthermore, environments and machines are represented in a unitary framework. As do both operator nets and temporal automata, constraint nets provide composite structure and multiple levels of abstraction. Unlike both operator nets and temporal automata, constraint nets introduce 'locations' so that distributed memories are explicitly modeled. In contrast to most concurrency models which are inherently non-deterministic, CN is a deterministic model; non-determinism can be modeled by constraint nets with hidden input locations.

In summary, the major contributions of the Constraint Net model are: (1) by introducing clocks, CN models various time structures and coordination among components with different time structures; (2) developed on abstract algebras, CN supports abstract data types and functions; (3) with a rigorous formalization, CN provides a programming semantics for the design of hybrid real-time embedded systems; (4) by symmetrically modeling plants and environments as well as control, CN serves as a foundation for the specification and verification of robotic systems [21].

The rest of the paper is organized as follows. Section 2 presents the syntax of constraint nets. Section 3 introduces the algebraic formalization of time and dynamics. Section 4 gives the semantics of constraint nets using fixpoint theory of continuous algebras. Section 5 illustrates CN via two examples: the cat and mouse problem and a car-like maze traveler. Section 6 concludes this paper and proposes future research directions.

2 The Structure of Constraint Nets

In this section, we present the syntax of constraint nets and characterize the composite structure and modularity of the model.

2.1 Syntax

A *constraint net* is a triple $CN \equiv \langle Lc, Td, Cn \rangle$, where Lc is a set of *locations*, each of which is associated with a *sort*; Td is a set of *transductions*, each of which is associated with a tuple of *input ports* and an *output port*, of certain sorts; Cn is a set of directed *connections* between locations and ports of transductions of the same sort. Topologically, a constraint net is a bipartite graph where locations are represented by circles, transductions are represented by boxes and connections are represented by arcs, each from a port of a transduction to a location or vice versa, with the restriction that (1) there is at most one connection pointing to each location, (2) each port of a transduction connects to a unique location and (3) no location is isolated. A location is an *input* iff there is no connection pointing to it otherwise it is an *output*. For a constraint net CN, the set of input locations is denoted by $I(CN)$, the set of output locations is denoted by $O(CN)$. A constraint net is *closed* iff there are no input locations otherwise it is *open*. For example, Figure 1, where $f_{\mathcal{T}}$ is a pointwise extension of a function f and δ is a unit delay, is an open net representing a state transducer: $s(0) = s_0, s(n+1) = f(i(n), s(n))$. Figure 2 is a closed net representing a differential equation $\dot{s} = f(s)$.

2.2 Modules

A *module* is a triple $CN(I, O)$ where CN is a constraint net, $I \subseteq I(CN)$ and $O \subseteq O(CN)$ are subsets of the input and output locations of CN respectively; I and O define the *interface* of the module. A module $CN(I, O)$ is *open* iff $I \neq \emptyset$ otherwise it is *closed*. A complex module can be hierarchically

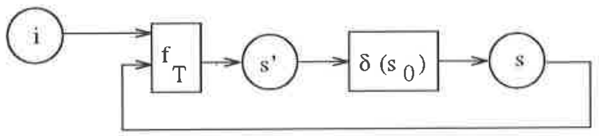

Figure 1: The constraint net representing a state transducer

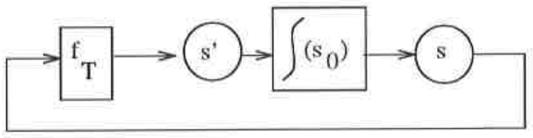

Figure 2: The constraint net representing $\dot{s} = f(s)$

constructed from simple ones.

There are several operations that can be applied to modules to obtain a new module. The first is *composition*, which combines two modules into one whose interface is the union of the two interfaces. The second is *coalescence*, which coalesces a group of locations with the same sort in the interface of a module into an individual location, with the restriction that at most one of the coalesced locations is an output location. The third is *hiding*, which deletes a set of locations from the interface.

In addition, we define three combined operations using the basic operations. The first is *cascade connection* which connects two modules in serial. The second is *parallel connection* which connects two modules in parallel. The third is *feedback connection* which connects an output of the module to an input of its own [14].

3 An Algebraic Theory of Dynamics

In this section, we define the semantics of locations and transductions of constraint nets, with an algebraic formalization of time structures, traces and transductions.

3.1 Time structures

Generalizing [10], we define a *time structure* as a metric space [12] $\langle \mathcal{T}, d \rangle$ where $\langle \mathcal{T}, \leq_{\mathcal{T}} \rangle$ with $\perp_{\mathcal{T}}$ as the least element is a total order, and $d : \mathcal{T} \times \mathcal{T} \rightarrow \mathcal{R}^+$ is a metric, such that $\forall t_0 \leq_{\mathcal{T}} t_1 \leq_{\mathcal{T}} t_2, d(t_0, t_2) = d(t_0, t_1) + d(t_1, t_2)$. Given a time structure $\langle \mathcal{T}, d \rangle$, the *measurement function* $m : \mathcal{T} \rightarrow \mathcal{R}^+$ is defined as $m(t) =_{def} d(\perp_{\mathcal{T}}, t)$. Obviously $m(\perp_{\mathcal{T}}) = 0$ and $m(t_1) < m(t_2)$ iff $t_1 <_{\mathcal{T}} t_2$.

The *precision* of a time structure $\langle \mathcal{T}, d \rangle$ is the minimum distance between any two time points, i.e.

$$p(\mathcal{T}) =_{def} \inf\{d(t_1, t_2) | \forall t_1 \neq t_2 \in \mathcal{T}\}.$$

A time structure $\mathcal{T}$ is *discrete* iff $p(\mathcal{T}) > 0$ otherwise it is *dense*. A time structure $\mathcal{T}$ is *finite* iff $|T|$ is finite. Let $\mathcal{N}$ be the set of natural numbers. Some examples of time structures are defined as follows.

1. $\mathcal{T} = \{n | n \in \mathcal{N}, 0 \leq n \leq 5\}, m(n) = n$ is a finite time structure.

2. $\mathcal{T} = \{n | n \in \mathcal{N}\}, m(n) = n$ is a discrete time structure.

3. $\mathcal{T} = \{r | r \in \mathcal{R}^+, 0 \leq r \leq 1\}, m(r) = r$ is a dense time structure.

4. $\mathcal{T} = \{n | n \in \mathcal{N}\}, m(n) = \frac{2^n - 1}{2^n}$ is a dense time structure.

5. $\mathcal{T} = \{\perp\} \cup \{n\}_{n \in \mathcal{N}}, m(\perp) = 0, m(n) = \frac{1}{2^n}$ is a dense time structure.

A time structure $\langle \mathcal{T}, d \rangle$ may be related to another time structure $\langle \mathcal{T}_r, d_r \rangle$ by a reference time mapping h where (1) $h : \mathcal{T} \rightarrow \mathcal{T}_r$ is strictly monotonic, i.e. $t <_{\mathcal{T}} t'$ implies $h(t) <_{\mathcal{T}_r} h(t')$ and (2) the least element is preserved, i.e. $h(\perp_{\mathcal{T}}) = \perp_{\mathcal{T}_r}$. $\mathcal{T}_r$ is a *reference time* of $\mathcal{T}$, and $\mathcal{T}$ is a *sampled time* of $\mathcal{T}_r$.

Generalizing [11], we define the *time domain* of a time structure $\langle \mathcal{T}, d \rangle$ as the completion of $\mathcal{T}$, $\mathcal{T}^\infty$. $\mathcal{T}^\infty$ is an algebraic *cpo* with $\{[\perp_{\mathcal{T}}, t] | t \in \mathcal{T}\}$ as the set of compact elements [8]. For example if $\mathcal{T} = \mathcal{R}^+$, $\mathcal{T}^\infty = \{[0, t], [0, t) | t \in \mathcal{R}^+\} \cup \{\infty\}$. The need for this definition will be demonstrated later in the definition of traces. Here, we define previous time functions using the time domain. Function "pre" denotes the moment preceding the current time. Let $pre : \mathcal{T} \rightarrow \mathcal{T}^\infty \cup \{\emptyset\}$,

$$pre(t) =_{def} \{t' | t' \in \mathcal{T}, t' <_{\mathcal{T}} t\}.$$

For example if $\mathcal{T} = \mathcal{R}^+$, $pre(5) = [0, 5)$. Furthermore, we can define the time previous to the current time by an interval $\delta \geq 0$ as: $-\delta : \mathcal{T} \rightarrow \mathcal{T}^\infty \cup \{\emptyset\}$,

$$t - \delta =_{def} \{t' | t' <_{\mathcal{T}} t, d(t, t') \geq \delta\}.$$

Obviously $t - 0 = pre(t)$.

3.2 Traces

A *variable domain* is a domain whose carrier set can be recursively defined as follows:

- a flat partial order is a variable domain;
- a product of variable domains is a variable domain.

A metric space $\langle A, d\rangle$ is defined for variable domain A. For any sequence $\langle a_n\rangle$ in A, if the limit does not exist then $\lim_{n\to\infty} a_n =_{def} \perp_A$ when A is a flat partial order; $\lim_{n\to\infty}\langle a_n^1, a_n^2\rangle =_{def} \langle \lim a_n^1, \lim a_n^2\rangle$ when $A \equiv A_1 \times A_2$ is a product.

A *variable trace* on a time structure $\langle \mathcal{T}, d\rangle$ is a mapping from $\mathcal{T}$ to a variable domain A: $v : \mathcal{T} \to A$; it is *deterministic* iff $\forall t \in \mathcal{T}, v(t) \neq \perp_A$. For example if $\mathcal{T} = \mathcal{R}^+$ and $A = \overline{\mathcal{R}} = \{\perp\} \cup \mathcal{R}$, $v = \lambda t.sin(t)$ is a deterministic variable trace, and $v = \lambda t.Ce^{kt}$ where $C, k \in \mathcal{R}$ is also a deterministic variable trace.

A variable trace $v : \mathcal{T} \to A$ can be extended to the time domain $v : \mathcal{T}^\infty \to A$ as follows. If $t^\infty \in \mathcal{T}^\infty$ is compact $v(t^\infty) =_{def} v(\bigvee_{\mathcal{T}} t^\infty)$. Otherwise let $m^* = \sup\{m(t) | t \in t^\infty\}$, $v(t^\infty) =_{def} \lim_{m(t)\to m^*} v(t)$. For the previous example, $\sin(\infty) = \lim_{t\to\infty} sin(t) = \perp$, and if $k < 0$, $Ce^{k\infty} = \lim_{t\to\infty} Ce^{kt} = 0$. A variable trace $v : \mathcal{T} \to A$ is *nonintermittent* iff $v(t^\infty) = \perp_A$ implies $\forall t'^\infty \supset t^\infty, v(t'^\infty) = \perp_A$ otherwise it is *intermittent*.

An *event trace* is a nonintermittent variable trace with variable domain $B = \{\perp, 0, 1\}$. An event trace $e : \mathcal{T} \to B$ generates a sampled time structure $\langle \mathcal{T}_e, d\rangle$ of $\langle \mathcal{T}, d\rangle$ if $e \neq \lambda t. \perp$. Formally, $\mathcal{T}_e \subseteq \mathcal{T}$ is defined as: $\mathcal{T}_e =_{def} \{\perp_{\mathcal{T}}\} \cup \{t \in \mathcal{T} | e(t) \neq \perp, e(t) \neq e(pre(t))\}$ if $e \neq \lambda t. \perp$ and $\mathcal{T}_e =_{def} \emptyset$ otherwise, i.e., each left-discontinuous point of the event trace defines a time point (see Figure 3).

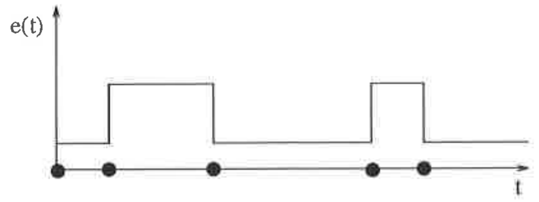

Figure 3: An event trace: each dot depicts a time point

A variable trace can be transformed into another with a different time structure. Let $\mathcal{T}_r$ be a reference time structure of $\mathcal{T}$ with the reference time mapping h. The *sampled variable trace* of

$v : \mathcal{T}_r \to A$ onto $\mathcal{T}$ is a variable trace $\underline{v} : \mathcal{T} \to A$, $\underline{v} =_{def} \lambda t.v(h(t))$. On the other hand, the *interpolated variable trace* of $v : \mathcal{T} \to A$ onto $\mathcal{T}_r$ is a variable trace $\overline{v} : \mathcal{T}_r \to A$,

$$\overline{v} =_{def} \lambda t. \begin{cases} \bot_A & \text{if } \forall t' \in \mathcal{T}, t >_{\mathcal{T}_r} h(t') \\ v(\{t' | h(t') \leq_{\mathcal{T}_r} t\}) & \text{otherwise.} \end{cases}$$

3.3 Transductions

A *transduction* is a mapping from a tuple of input traces to an output trace, $F_{\mathcal{T}_0,\mathcal{T}_1,\ldots,\mathcal{T}_n} : \mathcal{V}_{\mathcal{T}_1}^{A_1} \times \ldots \times \mathcal{V}_{\mathcal{T}_n}^{A_n} \to \mathcal{V}_{\mathcal{T}_0}^{A_0}$ where $\mathcal{V}_{\mathcal{T}_i}^{A_i}$ is the set of variable traces with time structure $\mathcal{T}_i$ and variable domain A_i, which satisfies the causal relationship between its inputs and the output, i.e. if the inputs are the same up to a certain time point, the outputs will be the same at that time. Formally, if $\mathcal{T}_r$ is a reference time structure for all $\mathcal{T}_i$ with the reference mapping h_i, then for any pair of input traces v, v', and $t_0 \in \mathcal{T}_0$: if $\forall i \forall t_i,\ h_i(t_i) \leq_{\mathcal{T}_r} h_0(t_0) \to v_i(t_i) = v_i'(t_i)$ then

$$F_{\mathcal{T}_0,\mathcal{T}_1,\ldots,\mathcal{T}_n}(v_1, \ldots, v_n)(t_0) = F_{\mathcal{T}_0,\mathcal{T}_1,\ldots,\mathcal{T}_n}(v_1', \ldots, v_n')(t_0).$$

In general, a transduction is a transformational process. For instance, a temporal integration is a transduction, and any state transducer defines a transduction from input traces to state traces. Clearly, transductions are closed under composition.

Let $\mathcal{T}_r$ be a reference time structure of $\mathcal{T}$. If $F_\mathcal{T}$ is a transduction whose variables have the same time structure, the *interpolated transduction* from $\mathcal{T}$ to $\mathcal{T}_r$ is defined as $\overline{F_\mathcal{T}}(v_1, \ldots, v_n) =_{def} \overline{F_\mathcal{T}(\underline{v_1}, \ldots, \underline{v_n})}$. Similarly, the *sampled transduction* from $\mathcal{T}_r$ to $\mathcal{T}$ can be defined as $\underline{F_{\mathcal{T}_r}}(v_1, \ldots, v_n) =_{def} \underline{F_{\mathcal{T}_r}(\overline{v_1}, \ldots, \overline{v_n})}$.

There are two basic types of transductions: pointwise extensions and delays. Given $f : A_1 \times \ldots \times A_n \to A_0$, a *pointwise extension* of f onto a time structure $\mathcal{T}$ is a transduction $f_\mathcal{T} : \mathcal{V}_1 \times \ldots \times \mathcal{V}_n \to \mathcal{V}_0$:

$$f_\mathcal{T}(v_1, \ldots, v_n) =_{def} \lambda t.f(v_1(t), \ldots, v_n(t)).$$

There are two types of delays: unit delays and transport delays. A *unit* delay is a transduction from $\mathcal{V}_\mathcal{T}^A$ to $\mathcal{V}_\mathcal{T}^A$. Formally, let $init \in A$ be the output value at the start time point, a unit delay in the time structure $\mathcal{T}$ is:

$$\delta_\mathcal{T}^A(init)(v) =_{def} \lambda t. \begin{cases} init & \text{if } pre(t) = \emptyset \\ v(pre(t)) & \text{otherwise.} \end{cases}$$

A *transport* delay is a transduction from $\mathcal{V}^A_\mathcal{T}$ to $\mathcal{V}^A_\mathcal{T}$. Formally, let *init* $\in A$ be the output value at the start period of time, $\delta > 0$ be the time delay, a transport delay in the time structure $\mathcal{T}$ is:

$$\Delta^A_\mathcal{T}(\delta)(init)(v) =_{def} \lambda t. \begin{cases} init & \text{if } t - \delta = \emptyset \\ v(t-\delta) & \text{otherwise.} \end{cases}$$

In addition to basic types of transductions which operate in a given time structure, an *event-driven* transduction is a transduction with an extra input, an event trace; it operates at each event point and the output value holds between two events. The additional event trace input of an event-driven transduction is called the *clock* of the transduction. Formally, let $\mathcal{E}_\mathcal{T}$ be the set of all event traces on time structure $\mathcal{T}$, and $F_{\mathcal{T}_e} : \mathcal{V}^I_{\mathcal{T}_e} \to \mathcal{V}^O_{\mathcal{T}_e}$ be a transduction on the time structure $\mathcal{T}_e$ produced by an event trace e. We define an event-driven transduction on time structure $\mathcal{T}$ as $F^e_\mathcal{T} : \mathcal{E}_\mathcal{T} \times \mathcal{V}^I_\mathcal{T} \to \mathcal{V}^O_\mathcal{T}$, $F^e_\mathcal{T}(e, v) =_{def} \overline{F_{\mathcal{T}_e}}(v)$. Various transductions can be defined for event synchronization [18].

3.4 Σ-dynamics

Finally, with preliminaries established, we can characterize the domain structure for constraint nets. We have defined time structures which are abstraction of time. Together with abstract data types, we can define abstract dynamics.

Abstract data types can be defined by Σ-variable domains. Let $\Sigma \equiv \langle S, F \rangle$ be a signature which contains a distinguished nullary function symbol for each sort s, Ω_s. A Σ-*variable domain* is a triple $\langle \{A_s\}_{s\in S}, \{\leq_{A_s}\}_{s\in S}, \{f^A\}_{f\in F}\rangle$ where $\langle \{A_s\}_{s\in S}, \{\leq_{A_s}\}_{s\in S}\rangle$ is a S-sorted variable domain, f^A is continuous for each $f \in F$ and Ω^A_s is $\perp_{A_s}$ [8].

Given a Σ-variable domain A and a time structure $\mathcal{T}$, a Σ-dynamics $\mathcal{D}(\mathcal{T}, A)$ is a triple $\langle \mathcal{V}, \leq_\mathcal{V}, \mathcal{F}\rangle$ where

- $\mathcal{V} \equiv \{\mathcal{V}^{A_s}_\mathcal{T}\}_{s\in S} \cup \mathcal{E}_\mathcal{T}$ where $\mathcal{V}^{A_s}_\mathcal{T} = \{v | v : \mathcal{T} \to A_s\}$ is the set of variable traces and $\mathcal{E}_\mathcal{T}$ is the set of event traces with the reference time structure $\mathcal{T}$;

- $\leq_\mathcal{V} \equiv \{\leq_{\mathcal{V}^{A_s}_\mathcal{T}}\}_{s\in S} \cup \{\leq_e\}$ is the set of partial orders on variable traces, $v_1 \leq_{\mathcal{V}^{A_s}_\mathcal{T}} v_2$ iff $\forall t \in \mathcal{T}, v_1(t) \leq_{A_s} v_2(t)$ and $e_1 \leq_e e_2$ iff $\forall t \in \mathcal{T}, e_1(t) \leq_B e_2(t)$;

- $\mathcal{F} \equiv \mathcal{F}_\mathcal{T} \cup \mathcal{F}^e_\mathcal{T}$ where $\mathcal{F}_\mathcal{T} \equiv \{f^A_\mathcal{T}\}_{f\in F} \cup \{\delta^{A_s}_\mathcal{T}(init)\}_{s\in S, init\in A_s} \cup \{\Delta^{A_s}_\mathcal{T}(\delta)(init)\}_{s\in S, \delta>0, init\in A_s}$ is the set of pointwise extensions, unit delays and transport delays, $\mathcal{F}^e_\mathcal{T} \equiv \{F^e | F \in \mathcal{F}_\mathcal{T}\}$ is the set of event-driven transductions.

Theorem 3.1 *Given a Σ-dynamics $\mathcal{D}(\mathcal{T}, A)$ consisting of a triple $\langle \mathcal{V}, \leq_{\mathcal{V}}, \mathcal{F}\rangle$ then (1) $\langle \mathcal{V}, \leq_{\mathcal{V}}\rangle$ is a multi-sorted cpo (complete partial order), and (2) each transduction in $\mathcal{F}$ is continuous.*

Proof: (Sketch) Variable domains are algebraic *cpos*. The set of all functions to a *cpo* is also a *cpo*. The set of event traces is also a *cpo*. It is easy to check that $f_{\mathcal{T}}^{A}$ is continuous. Delays and event-driven transductions are continuous given the properties of variable domains and event traces. □

4 The Semantics of Constraint Nets

In this section, we present a denotational semantics for constraint nets based on fixpoint theory.

4.1 Fixpoint semantics

A Σ-dynamics $\mathcal{D}(\mathcal{T}, A) \equiv \langle \mathcal{V}, \leq_{\mathcal{V}}, \mathcal{F}\rangle$ is the *semantic domain* of a constraint net $CN \equiv \langle Lc, Td, Cn\rangle$ iff each location in Lc with sort s denotes a variable trace $v \in \mathcal{V}_{\mathcal{T}}^{A_s}$; each transduction in Td is a composition of transductions in $\mathcal{F}$ with at most one clock; each clock denotes an event trace $e \in \mathcal{E}_{\mathcal{T}}$.

Each connection relates an input/output port of a transduction with a location. Therefore, the semantic representation of a constraint net is a set of equations, where each left-hand side is an individual output location and each right-hand side is an expression composed of transductions and locations: $\vec{o} = \vec{F}_{\mathcal{T}}(\vec{i}, \vec{o})$ where $\vec{o}$ is the tuple of output locations, $\vec{i}$ is the tuple of input locations and $\vec{F}_{\mathcal{T}}$ is the tuple of transductions.

Theorem 4.1 *For any constraint net with a Σ-dynamics as its semantic domain, there is a least fixpoint of $\vec{o} = \vec{F}_{\mathcal{T}}(\vec{i}, \vec{o})$.*

Proof: (Sketch) Since each transduction is continuous, $\vec{F}_{\mathcal{T}}$ is a continuous transduction from $\vec{\mathcal{V}}_i \times \vec{\mathcal{V}}_o$ to $\vec{\mathcal{V}}_o$. According to fixpoint theory [8], there is a continuous transduction $\mu.\vec{F}_{\mathcal{T}} : \vec{\mathcal{V}}_i \to \vec{\mathcal{V}}_o$ which is the least fixpoint of this equation. □

The semantics of a constraint net CN is defined as: $[[CN]] =_{def} \mu.\vec{F}_{\mathcal{T}}$ where $\mu.\vec{F}_{\mathcal{T}} : \vec{\mathcal{V}}_i \to \vec{\mathcal{V}}_o$ is the least fixpoint of the set of equations of CN. We write $\vec{o} = [[CN]](\vec{i})$ as the *semantic equation* of CN. The semantics of a module $CN(I, O)$ is defined as: $[[CN(I, O)]] =_{def} [[CN]][O]$ where $\vec{F}[O]$ is the subset of tuples of $\vec{F}$ restricted to O.

For example, a state transducer (Figure 1) is composed of a pointwise extension of a *state transition function* f and a unit delay. A state transducer defines a continuous transduction from $\mathcal{V}_{\mathcal{T}}^{I}$ to $\mathcal{V}_{\mathcal{T}}^{Q}$ if $f : I \times Q \to Q$ is a continuous function and I and Q are variable domains. The semantic representation of a state transducer is: $v_{ns} = f_{\mathcal{T}}(v_i, v_s), v_s = \delta(s_0)(v_{ns})$ i.e. $v_s = \delta(s_0)(f_{\mathcal{T}}(v_i, v_s))$. The semantic equation of the state transducer is: $v_s = (\mu.F)(v_i)$ where $F = \delta(s_0) \circ f_{\mathcal{T}}$.

For a complex system with many components, the semantics of the whole system can be obtained from the semantics of its components by deriving the least fixpoint of the set of semantic equations of all the components.

4.2 Infinitesimal delays and differential equations

So far we have no definition for integration, the most important transduction for time structures defined by real intervals. We can define integration by introducing infinitesimal transport delays. A transport delay $\Delta(\delta)(init)$ is called *infinitesimal* iff $\delta > 0$ is an infinitesimal. The *limiting behavior* of a variable trace $v(\delta)$, where δ is a tuple of infinitesimals, is defined as: $v^* =_{def} \lambda t. \lim_{\delta \to 0} v(\delta)(t)$. Let CN be a closed net, i.e. $I(CN) = \emptyset$, $[\![CN]\!][o], o \in O(CN)$ be a variable trace, and δ be the tuple of infinitesimals in CN. The limiting behavior of CN is defined as: $[\![CN^*]\!] =_{def} \lambda o.([\![CN]\!][o])^*$.

Let $\Sigma \equiv \langle \{r\}, \{\Omega, +, \times\} \rangle$ and Σ-dynamics be $\mathcal{D}(\mathcal{R}^+, \overline{\mathcal{R}})$. For simplicity, we use $+$ to denote its pointwise extension $+_{\mathcal{R}^+}$, $-x$ to denote $\lambda t.(-1) \times_{\mathcal{R}^+} x$, and $y - x$ to denote $y + (-x)$ etc. Let $d(\delta)(v) =_{def} v - \Delta(\delta)(0)(v)$, and $q = \Delta(\delta)(init)(q) + \Delta(\delta)(0)(u) \times d(\delta)(v)$ be an open net with infinitesimal delay δ. Let $int(\delta)(init)$ be the least fixpoint of the net; it is a *temporal integrator* iff $v = \lambda t.t$, the identity function. We will use $\int(init)$ to denote $int(\delta)(init)$ with an infinitesimal δ.

For example let us investigate the semantics of the net in Figure 2 with $f = \lambda s.(-s)$. Given Σ-dynamics $\mathcal{D}(\mathcal{R}^+, \overline{\mathcal{R}})$, by starting with $s_f^0(\delta) = \lambda t. \perp$, we have

$$s_f^1(\delta) = int(\delta)(init)(s_f^0(\delta)) = \lambda t. \begin{cases} init & \text{if } t < \delta \\ \perp & \text{otherwise,} \end{cases}$$

$$s_f^2(\delta) = int(\delta)(init)(s_f^1(\delta)) = \lambda t. \begin{cases} init & \text{if } t < \delta \\ init - init \times \delta & \text{if } \delta \leq t < 2\delta \\ \perp & \text{otherwise,} \end{cases}$$

$$\vdots$$

$$s_f^{k+1}(\delta) = int(\delta)(init)(s_f^k(\delta)) = \lambda t. \begin{cases} init & \text{if } t < \delta \\ init - init \times \delta & \text{if } \delta \leq t < 2\delta \\ \vdots & \\ init \times \Sigma_{n=0}^{k}(-1)^n C_k^n \delta^n & \text{if } k\delta \leq t < (k+1)\delta \\ \perp & \text{otherwise.} \end{cases}$$

We have $s_f(\delta) = \bigvee\{s_f^k(\delta)\}$. The limiting behavior of the net is $lim_{\delta \to 0} s_f(\delta)(t) = \lim_{\delta \to 0} init \times \Sigma_{n=0}^{k}(-1)^n \frac{k!}{n!(k-n)!}\delta^n$ where $k = \lfloor \frac{t}{\delta} \rfloor$, i.e. $s(t) = init \times \Sigma_{n=0}^{\infty}(-1)^n \frac{t^n}{n!} = init \times e^{-t}$, which is the solution of $\dot{s} = -s$. We should notice that in general the limiting behavior of a constraint net representing a differential equation $\dot{s} = f(s)$ will correspond to its solution if f satisfies a Lipschitz condition [12]. Integrators are one of the basic types of transductions on time structures of real intervals.

5 Examples: Modeling in Constraint Nets

In this section, we demonstrate constraint net modeling using two examples. One is the cat and mouse problem and the other is a car-like maze traveler.

5.1 The cat and mouse problem

The cat and mouse problem was proposed as a sample real-time pedagogical problem at the REX workshop [11, 20]. The problem can be described as follows: at time 0, a mouse starts running from a certain position on the floor in a straight line towards a hole in the wall, which is at a distance X_0 from the initial position. The mouse runs at a constant velocity V_m. After a delay of δ, a cat is released at the same initial position and chases the mouse at velocity V_c along the same path. Will the cat catch the mouse?

The constraint net in Figure 4 models this problem. In this net transductions V_m and V_c are constant traces, representing the constant velocities of the mouse and the cat respectively. Assume the hole is at position 0 and initial position of the mouse and the cat is $-X_0$. Function $*$ is multiplication and $move : \overline{\mathcal{R}} \times \overline{\mathcal{R}} \to B$ is a function defined as: $move =_{def} \lambda x_m x_c.(x_m \geq x_c) \wedge (x_m \leq 0)$. That is, the output value of $move$ is 0 whenever the cat catches the mouse ($x_m < x_c$) or the mouse reaches the hole ($x_m > 0$). The set of equations of this net is: $\dot{x_m} = move(x_m, x_c) \times V_m$, $\dot{x_c} = move(x_m, x_c) \times v_c$, and $v_c = \Delta(\delta)(0)V_c$ with $x_m(0) = x_c(0) = -X_0$.

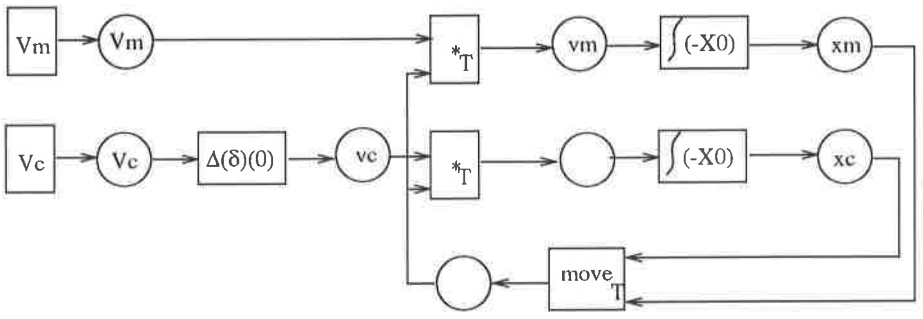

Figure 4: The constraint net for the cat-mouse problem

5.2 The car-like maze traveler

Suppose a maze is composed of separated T-shaped obstacles of bounded size placed in one of four directions on an unbounded plane. A car-like robot with two touch sensors, the forward sensor SF and the right-side sensor SR, is required to traverse the maze from west to east as shown in Figure 5. The robot can move forward and backward, and can make turns; however, its turns are limited by

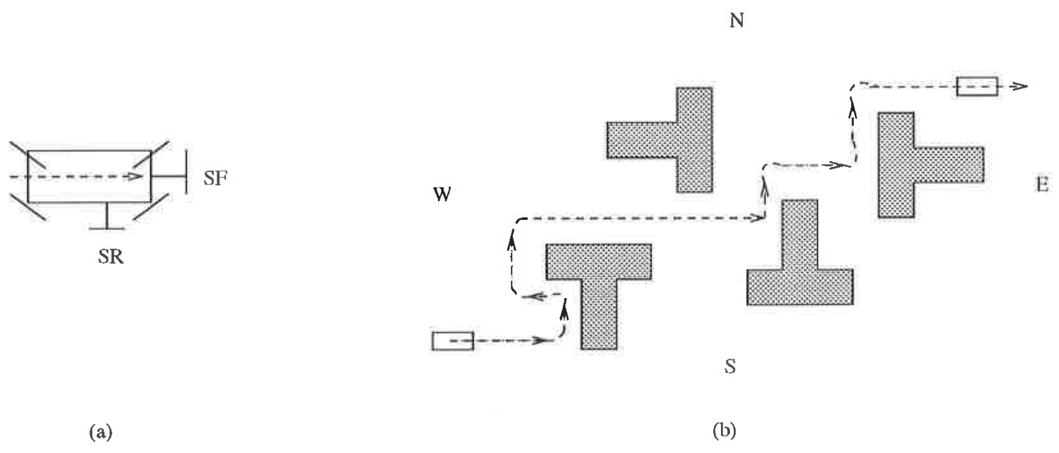

Figure 5: (a) The car-like robot (b) Traveling through a maze

mechanical stops on the steering gear, which turns the front and back wheels symmetrically. Despite these nonholonomic constraints, it can achieve any position and orientation on an unbounded plane. The plant of the robot can be modeled by following equations:

$$\dot{x} = v\cos(\theta), \quad \dot{y} = v\sin(\theta), \quad \dot{\theta} = v\tan(\alpha)/L$$

where v is its velocity, θ is its heading, L is its half-length and α is the current steering angle.

Figure 6 models the robotic system of this maze traveler, where the controller connects sensing signals to motor commands. For example, when the forward sensor SF is on, the robot is facing a wall directly within some distance; when α is positive, the robot is turning left.

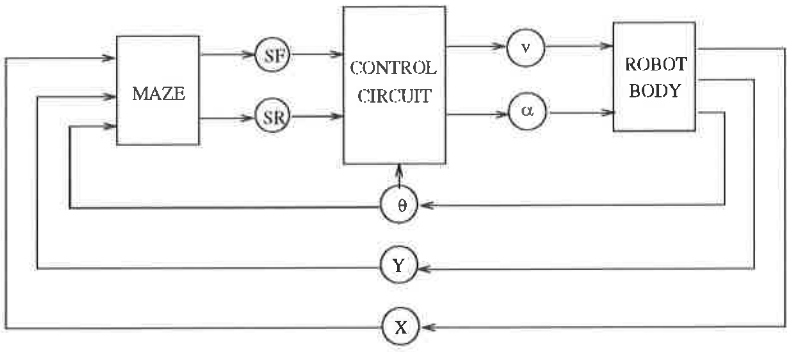

Figure 6: The maze traveler robotic system

The controller of the maze traveler is a state transition system in Figure 7(a) where circles are states with a double circle as the initial state, and arcs with label x/y indicate input x and output y. In Figure 7(a), MF means "move forward" ($\alpha = 0$), TL means "turn left" ($\alpha > 0$) and TR means "turn right" ($\alpha < 0$) [1]. For example, when the state of the robot is moving east and the forward sensor is set to 1, it should turn left and the next state is moving north.

It is clear that we cannot set any fixed sampling rate for this transition system, since we don't know how long it takes to turn to the next heading. Rather, an event-driven transduction is appropriate. There are three types of events: (1) θ becomes $\frac{\pi}{2}k$, (2) SF changes from 0 to 1 and (3) SR changes from 1 to 0. The event "or" of these three events will be the output of the event generator EG to trigger the state transition system TD (Figure 7 (b)).

6 Conclusion and Future Work

We have presented the unitary Constraint Net model for hybrid concurrent systems based on algebraic theory. We have been able to model robotic behaviors with constraint nets which are simulated by

[1]Note that this finite transition system works for T-shaped blocks; in general an infinite memory is needed for counting the number of turns [1]. The velocity is set at a constant value of 1; in reality a PD controller would be used.

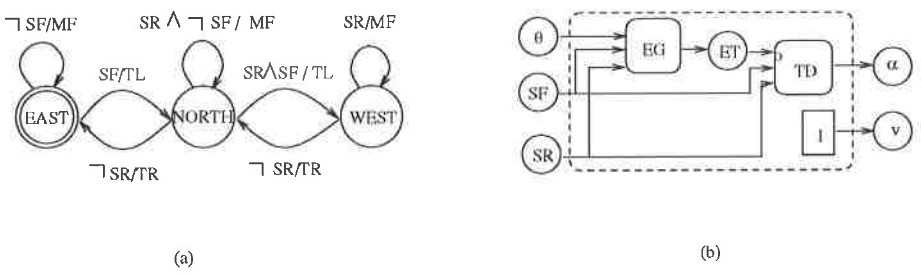

Figure 7: (a) State transition graph (b) The controller

logical concurrent objects [22]. Related work on a real-time temporal logic for the specification and verification of constraint nets was reported in [21]. We are now developing a visual programming, simulation and verification environment, known as ALERT (A Laboratory for Embedded Real-Time systems), based on the Constraint Net model.

References

[1] H. Abelson and A. A. diSessa. *Turtle Geometry: The Computer as a Medium for Exploring Mathematics.* The MIT Press, 1981.

[2] R. Alur and D. Dill. The theory of timed automata. In J.W. deBakker, C. Huizing, W.P. dePoever, and G. Rozenberg, editors, *Real-Time: Theory in Practice*, number 600 in Lecture Notes on Computer Science, pages 45 – 73. Springer-Verlag, 1991.

[3] E.A. Ashcroft. Dataflow and eduction: Data-driven and demand-driven distributed computation. In J. W. deBakker, W.P. deRoever, and G. Rozenberg, editors, *Current Trends in Concurrency*, number 224 in Lecture Notes on Computer Science. Springer-Verlag, 1986.

[4] B. Berthomieu and M. Diaz. Modeling and verification of time dependent systems using time petri nets. *IEEE Transactions on Software Engineering*, 17(3), March 1991.

[5] P. Caspi, D. Pilaud, N. Halbwachs, and J. A. Plaice. Lustre: A declarative language for programming synchronous systems. In *ACM Proceeding of Principles of Programming Languages*, 1987.

[6] J.W. deBakker, C. Huizing, W.P. dePoever, and G. Rozenberg, editors. *Real-Time: Theory in Practice.* Number 600 in Lecture Notes on Computer Science. Springer-Verlag, 1991.

[7] M. R. Hansen and C. Zhou. Semantics and completeness of duration calculus. In J.W. deBakker, C. Huizing, W.P. dePoever, and G. Rozenberg, editors, *Real-Time: Theory in Practice*, number 600 in Lecture Notes on Computer Science, pages 209 – 225. Springer-Verlag, 1991.

[8] M. Hennessy. *Algebraic Theory of Processes.* The MIT Press, 1988.

[9] T.A. Henzinger, Z. Manna, and A. Pnueli. Timed transition systems. In J.W. deBakker, C. Huizing, W.P. dePoever, and G. Rozenberg, editors, *Real-Time: Theory in Practice*, number 600 in Lecture Notes on Computer Science, pages 226–251. Springer-Verlag, 1991.

[10] J. Lavignon and Y. Shoham. Temporal automata. Technical Report STAN-CS-90-1325, Robotics Laboratory, Computer Science Department, Stanford University, Stanford, CA 94305, 1990.

[11] O. Maler, Z. Manna, and A. Pnueli. From timed to hybrid systems. In J.W. deBakker, C. Huizing, W.P. dePoever, and G. Rozenberg, editors, *Real-Time: Theory in Practice*, number 600 in Lecture Notes on Computer Science, pages 448 – 484. Springer-Verlag, 1991.

[12] E. G. Manes and M. A. Arbib. *Algebraic Approaches to Program Semantics.* Springer-Verlag, 1986.

[13] K. Marzullo, F.B. Schneider, and N. Budhiraja. Derivation of sequential, real-time, process-control programs. In A. M. van Tilborg and G. M. Koob, editors, *Foundations of Real-Time Computing: Formal Specifications and Methods*, pages 40 – 54. Kluwer Academic Publishers, 1991.

[14] M. D. Mesarovic and Y. Takahara. *General Systems Theory: Mathematical Foundations.* Academic Press, 1975.

[15] X. Nicollin and J. Sifakis. From atp to timed graphs and hybrid systems. In J.W. deBakker, C. Huizing, W.P. dePoever, and G. Rozenberg, editors, *Real-Time: Theory in Practice*, number 600 in Lecture Notes on Computer Science, pages 549 – 572. Springer-Verlag, 1991.

[16] X. Nicollin and J. Sifakis. An overview and synthesis on timed process algebras. In J.W. deBakker, C. Huizing, W.P. dePoever, and G. Rozenberg, editors, *Real-Time: Theory in Practice*, number 600 in Lecture Notes on Computer Science, pages 526 – 548. Springer-Verlag, 1991.

[17] C.A. Petri. "Forgotten topics" of net theory. In W. Brauer, W. Reisig, and G. Rozenberg, editors, *Petri Nets: Applications and Relationships to Other Models of Concurrency*, number 255 in Lecture Notes on Computer Science, pages 500 – 514. Springer-Verlag, 1986.

[18] I.E. Sutherland. Micropipeline. *Communication of ACM*, 32(6), June 1989.

[19] W.W. Wadge and E.A. Ashcroft. *Lucid, the dataflow programming language.* Academic Press, 1985.

[20] W. G. Wood. A specification of the cat and mouse problem. In J.W. deBakker, C. Huizing, W.P. dePoever, and G. Rozenberg, editors, *Real-Time: Theory in Practice*, number 600 in Lecture Notes on Computer Science, pages 676 – 686. Springer-Verlag, 1991.

[21] Y. Zhang and A. K. Mackworth. Will the robot do the right thing? Technical Report 92-31, Department of Computer Science, University of British Columbia, 1992.

[22] Y. Zhang and A.K. Mackworth. Modeling behavioral dynamics in discrete robotic systems with logical concurrent objects. In S.G. Tzafestas and J.C. Gentina, editors, *Robotics and Flexible Manufacturing Systems.* Elsevier Science Publishers B.V., 1992.