制約概念に基づくハイブリッドシステムモデリング言語 HydLa

上田 和紀　石井 大輔　細部 博史

時間の経過に伴って状態が連続変化したり，状態や方程式系自体が離散変化したりする系をハイブリッドシステムと呼ぶ．ハイブリッドシステムの記述手段としてはオートマトンやペトリネットを拡張する方法が代表的であるが，不確実値の扱い，演算結果の保証，シミュレーションと検証の統合などの観点から，我々は制約概念を活用した枠組の構築を目指している．本論文では，制約概念に基づくハイブリッドシステムモデリング言語 HydLa を紹介する．その特徴は，制約階層概念の採用によって制約条件を過不足なく与えることを容易にした点と，数学や論理学の記法の活用によって簡明な構文と意味論を与えることができた点である．

Hybrid systems are systems involving continuous changes of states and discrete changes of both states and equations governing their behavior. Description of hybrid systems have typically been made by extending automata and Petri nets, but we are aiming at establishing a constraint-based framework with a view to handling uncertain values, guranteeing the result of computation and unifying simulation and verification. This paper introduces a constraint-based modeling language HydLa for hybrid systems. The two important features of HydLa are: (1) it features constraint hierarchies to facilitate well-defined modelling and (2) it provides simple syntax and semantics by employing concepts and notations of mathematics and logic wherever possible.

1　はじめに

制約プログラミング (constraint programming) は，現実世界の現象や問題を記述した連続・離散領域の等式・不等式制約を，制約伝播に代表される探索技術を駆使して解く記述・求解パラダイムである．

制約プログラミングにおける変数は単一代入であるため，時間またはイベントとともに変化する状態の表現には一見適さない．しかし状態の系列を考えることにすれば，制約プログラミングの枠組に乗せることができる．並行論理プログラミングおよびその一般化である並行制約プログラミングでは，含意 (implication) を同期機構として使い，状態系列を直接間接に表すストリームを計算する．

並行制約プログラミングは，離散的に変化する並行プロセス系に対して高い記述能力をもつが，その自然な拡張によって，連続量や連続変化も統合的に扱える計算モデルや高水準言語を設計することはできないだろうか？ 本研究の目的は，時間の経過に伴って状態が連続変化したり，状態や方程式系自体が離散変化したりする**ハイブリッドシステム**のモデリングの枠組を制約概念に基づいて構築することである．

制約概念を軸とする理由は二つある．一つは，制約概念は，同期機構や条件つき制約（ある条件の成否によって他の条件を有効または無効にする）のような**制御構造**の表現能力をもつからである．もう一つは，連続量は一般に観測や演算の誤差を伴うことから，連続量を伴う系の表現や検証においては，等式だけでなくて不等式や区間（本質的には 2 本の不等式の組）の概念を用いて不確定情報を的確に扱う必要があるからである．

制約概念は強力ではあるが，ハイブリッドシステムのモデリングにおいて制約条件を過不足なく適切

A Constraint-Based Modeling Language for Hybrid Systems.

Kazunori Ueda, Daisuke Ishii, 早稲田大学理工学術院情報理工学科, Dept. of Computer Science and Engineering, Waseda University.

Hiroshi Hosobe, 国立情報学研究所, National Institute of Informatics.

本論文は，日本ソフトウェア科学会第 25 回大会発表論文 (2008) の改訂版である．

に与えることは容易ではない．モデリングを容易にするためには，制約間に優先度を導入した**制約階層** (constraint hierarchy) の考え方が有効であると期待される．本論文では，設計の大枠が固まった制約モデリング言語 HydLa の設計指針，宣言的意味論，記述例を紹介する．

2 関連研究

ハイブリッドシステムの記述体系としてはハイブリッドオートマトン[7] やハイブリッドペトリネット[3] がもっとも良く知られ，組込みシステムや制御系の検証を主な目的として研究が進んできた．しかしこれらは，多数のオブジェクトやその相互作用を扱う系（力学系など）や複雑な法則に従う系など，高い表現能力を要する系の記述に対応できるかどうかが明らかでない．合成可能性 (compositionality) を考慮した理論計算モデルとしては，ハイブリッド I/O オートマトン[10] やハイブリッドプロセス代数[1] がある．しかしハイブリッドシステムの記述への**高水準プログラミング言語**の立場からの研究は多くない．

数少ない例外の一つは，制約プログラミングの枠組でハイブリッドシステムを扱うハイブリッド並行制約プログラミング[6] である．これは，並行制約プログラミングの同期機構をハイブリッドシステムの離散変化の判断機構として活用した枠組である．ハイブリッド並行制約言語 Hybrid CC [5] は処理系が公開され，制御系から計算生物学に至るさまざまな記述例が蓄積されてきている．

Hybrid CC は変数のとりうる値の範囲を**区間制約** (interval constraints) の形で保持する機能や非線形制約の求解機能などの強力な機能の実装を試みている．区間制約は，計算の誤差と，モデリング段階における観測パラメタの不確かさの両方に対応するための技術で，計算結果の信頼性を保証するために不可欠な機能であるが，ハイブリッドシステム特有の離散変化 (discrete change) を区間制約の枠組で適切に扱うことをはじめとして，言語設計と実装の両面で多くの技術課題が残されている．

制約プログラミングからハイブリッドシステムへのアプローチにはほかにも CLP(F) を用いた手法が提案されている [8]．CLP(F) は実数値関数を領域とする制約論理プログラミング言語で，Prolog を拡張するかたちで関数形に関する制約記述機能や Taylor 展開に基づく区間制約求解機能を提供している．しかし，複雑な制御の記述性や，区間制約の存在下での離散変化の計算など，技術課題も多く残されている．

制約階層[2] の研究はユーザインタフェースへの応用から始まる長い歴史をもち，制約階層をもつ制約論理プログラミング言語の提案もある[14]．しかし，ハイブリッドシステムへの適用は未開拓であり，制約階層の意味論も主にユーザインタフェースへの応用を念頭に研究が進められてきた．

理論研究と実装が進みつつある最近のモデリング言語としては KeYmaera [13] がある．KeYmaera は強力な検証体系を特徴とするが，モデル記述は手続き型プログラミングに似た構文を採用しており，宣言型のアプローチをとる HydLa と対照をなす．

ハイブリッドシステムのための言語ではないが，制約概念をもつ手続き型言語 Kaleidoscope'90 [4] は，変数を値の変更履歴のストリームとして定式化している点や制約階層を持つ点など，HydLa といくつかの類似点をもつ．

3 HydLa の概要と設計指針

HydLa はハイブリッドシステムのための宣言型言語であり，与えられた問題の数学的定義をできるだけそのままの形で記述し，その実行や解析を行うことを目標としている．プログラミングを専門としない技術者の利用を目指している点からはモデリング言語と呼ぶのがふさわしいが，本論文では論理学における用語法との混乱を避けるために，HydLa で記述したものをモデルでなくプログラムと呼ぶ．

HydLa プログラムが記述の対象とする系は，一般に複数個の関数 $x_1(t), x_2(t), \ldots (t \geq 0)$ で表現される．HydLa プログラムは，時間の推移とともに連続的もしくは離散的に変化するこれらの関数の挙動に関する制約条件を与える．HydLa プログラム P の宣言的意味は，関数群 $\overrightarrow{x}(t) = \{x_i(t)\}_{i\geq 1}$ が P を充足するという関係，または（同じことであるが）P を満たす $\overrightarrow{x}(t)$ 全体の集合として定義される．

HydLa が制約階層を備える理由は，人工知能分野における知識表現においてデフォルトや例外の概念が

有用である理由と同じである．たとえば床に落下して弾むボールのシミュレーションを考えると，ほとんどの時刻ではボールの速度変化は重力加速度から定まるが（デフォルト），床との衝突のときだけは重力加速度ではなく衝突の方程式から定まる（例外）．床との衝突時点でニュートンの法則を制約条件に加えたままにしておくと，系を定める方程式は全体として充足不可能となってしまう．ボールの軌道が well-defined であるためには各時点の制約条件を過不足なく与えなければならないが，我々の Hybrid CC プログラミングの経験では，ニュートンの法則を無効化すべき時点をユーザが過不足なく列挙することは難しく，デフォルトと例外の考え方で系を表現する方が簡潔になる．

また，実世界のシミュレーションでは，移動や変化に関する情報がない物体は移動や変化を起こさないことにするのが自然である．このことを陽に表現する論理式は，人工知能分野において**フレーム公理** (frame axioms) として知られるが，HydLa においてもフレーム公理を適切かつ容易に定義する機能が重要となる．

4 HydLa の構文

HydLa プログラムの目的は，方程式系（以下，不等式も含むものとする）を定義することと，方程式系に優先度を付与することである．前節で述べた HydLa の設計指針から，方程式系の記述にはできるだけ数学と論理学の記法を利用して、新たに習得しなければならない概念や記法を最小限にすることとした．本論文でも数学記法をそのまま使うこととする．

4.1 名前

HydLa で扱う名前にはいくつかの種類がある．HydLa での標準的記法にしたがって順に説明する．

英小文字から始まる名前は**関数変数**（関数を領域とする変数）を表す．関数変数には単純関数変数と複合関数変数とがある．単純関数変数の領域は，時刻 $t \geq 0$ で定義された実数値関数の集合であって，その具体形が満たすべき制約条件を HydLa プログラムが与える．ただし HydLa では制約は時相論理式で与えるため，時刻を引数として明示的に与えることはない．

複合関数変数は，暗黙の時刻引数以外の引数をとり，これによって同様の性質をもつ一群の関数の集まりを表すことができる．本論文では 6.2 節で簡単な使用例を紹介する．

すべて英大文字の（二文字以上の）名前は**定義名** (defined name) であって，プログラムに命名するものと制約に命名するものの二種類がある．定義名は引数をとることができる．制約を表す定義名は，最終引数として暗黙の時刻引数をもつ．

頭文字だけが英大文字の名前は，述語定義に現れる束縛変数を表す．

最後は数値定数である．数値定数は，その値を常に返す定数関数とみなす．時間変化しない物理定数に名前をつける場合は，$\Box(g = 9.8)$ のような定義を与えればよい．これは $\forall t \geq 0\,(g(t) = 9.8)$ を表す．

4.2 プログラム

HydLa のプログラムは，**制約モジュール**の順序（多重）集合である．制約モジュール間の半順序階層構造は，二つのプログラムの合成において一方に優先度を与えるか否かを指定することで表現する．

$$\begin{aligned}\text{(program)}\quad & P ::= M \mid \mathit{Pname}(\overrightarrow{E}) \\ & \mid P, P \mid P \ll P \\ \text{(module)}\quad & M ::= C\end{aligned}$$

Pname はプログラム定義（プログラムに名前をつけたもの）の呼出しであり，引数を指定することもできる．P_1, P_2 は優先順序をつけない合成である．$P_1 \ll P_2$ は P_2 中の各制約モジュールに P_1 中の各制約モジュールよりも高い優先度を与える合成である．

HydLa の制約は，関係式，制約定義の呼出し，連言，全称量化，含意，$\Box$ (always) からなる時相論理式である．

$$\begin{aligned}\text{(constraint)}\quad C ::= {} & E \mathit{relop} E \mid \mathit{Cname}(\overrightarrow{E}) \\ & \mid C \wedge C \mid \forall X \in Y(C) \\ & \mid C \Rightarrow C \mid \Box C\end{aligned}$$

$E \mathit{relop} E$ の形の制約を**原子制約**と呼ぶ．関係演算子 *relop* の集合は，典型的には等号を含む．プログラム中の制約は，時刻 0 において成立すべき制約を表す．ただし時相演算子 $\Box$ を用いることによって，時刻 0 以降に成立すべき制約を指定することもできる．

HydLa は他の多くの制約モデリング言語と同様，

正確に言えば，制約システムによってパラメタ化された**言語スキーム**であって，HydLa 自身は，記述可能な方程式や不等式のクラスを特に定めない．しかし，微分方程式および離散変化が表現できることは必須条件であるので，式の中では，通常の算術演算子のほかに時間微分，時間シフト，および左極限をとるための演算子が使えるものとする．

(expression)　$E ::=$ 通常の式 $\mid E' \mid E\hat{}E \mid E-$

前述のように，式の中では時刻 t への明示的な言及は行わない．たとえば式 $x+y+1$ の意味は $x(t)+y(t)+1$，x' の意味は $x'(t)$，$x\hat{}5$ の意味は $x(t+5)$，$x-$ の意味は左極限 $\lim_{t'\uparrow t} x(t')$ である．

全称量化は連言の一般化であり，他のプログラミング言語の繰返しに対応する機能を提供する．含意 $C_1 \Rightarrow C_2$ の前件 C_1 は原子制約またはその連言とし，$\Rightarrow$ や $\hat{}$ や $\Box$ を使うことはできないものとする．その理由と目的は，前件は後件のガードであることと，ガードで未来に関することに言及しないようにすることである．

定義，すなわち制約やプログラムへの命名は，以下の構文で行う．HydLa プログラムは 0 個以上の定義を伴うことができる．

$$\begin{aligned}\text{(definition)}\quad D ::= {} & Pname(\overrightarrow{X})\{P\} \\ \mid {} & Cname(\overrightarrow{X}) \Leftrightarrow C\end{aligned}$$

$\overrightarrow{X}$ は束縛変数の列である．定義は必ず成り立つべきものであり，優先度付与の対象ではない．

プログラム定義と制約定義の違いは，前者の中では制約階層を使うことができるが，後者は純粋な論理式であって制約階層は使えない点である．そのかわり，後者が制約とプログラムの両方のレベルで使えるのに対して，前者はプログラムのレベルでしか使えない．第 6 節で例示するが，プログラム定義はハイブリッドシステムの個々の構成要素の挙動を定義するのに適し，制約定義は一つの構成要素の挙動のさまざまな側面を定義するのに適する．

5　宣言的意味論

通常のプログラミング言語と異なり，モデリング言語においては，通常の前向き実行だけでなく，記号実行や逆向き実行など，さまざまな実行方式を併用して系の検証や解析を行う可能性がある．したがってモデリング言語では操作的意味論よりも宣言的意味論が中心的役割を果たす．

プログラム P は，制約モジュールの半順序集合 $\langle U, \ll \rangle$ を定める．ここで U は P を構成する制約モジュールの集合である．制約モジュール M_1, M_2 が $M_1 \ll M_2$ の関係にあるとき，M_2 は M_1 よりも高い優先度をもつが，HydLa の制約階層は，オブジェクト指向言語のクラス階層と同様，優先度が高いものほど階層の下位にあるとみなす．$\mathcal{O}(P)$ で，U の downward closed な部分集合全体の集合を表す．

プログラム S に出現する関数変数を具体的な実数値関数に具体化する制約を c とする（c は S に出現しない関数を含んでいてもよい）．宣言的意味論の目的は，c が S を充足するという関係 $c \models S$ の定義を与えることである．

以下に示す $c \models S$ の定義は，**無矛盾性** (consistency) に基づく制約情報の採用を基本原理としている．つまり，制約モジュール集合の無矛盾かつ downward closed な部分集合のうちで極大なものを時々刻々満たすことを求めている．

$$\begin{aligned}
&\forall t\, \exists H \in \mathcal{O}(S)(&& (1)\\
&\quad c \Rightarrow H(t) && (2)\\
&\quad \wedge\ \neg \exists c'\, \exists H' \in \mathcal{O}(S)(&& (3)\\
&\qquad \forall t' < t\, (c'(t') = c(t')) && (4)\\
&\qquad \wedge\ H' \supsetneq H && (5)\\
&\qquad \wedge\ c' \Rightarrow H'(t))) && (6)
\end{aligned}$$

行 (1) の限量子の順序は，c が制約階層のどの部分まで満たすかが時々刻々変化してもよいことを表している．行 (2) は，c が時刻 t で制約階層のある部分まで満たすことを表している．行 (3) 以下では，時刻 t において c よりも良く制約階層を満たす c' がないことを言おうとしている．ただし，任意の関数制約 c' を許すと，時刻 t でもっと多くの制約を満たせるものは簡単に作れてしまう．そこで「時刻 t より前については c と同じに振る舞う関数の中で」という条件をつけて（行 (4)），H を真に含む集合 H' を（行 (5)），関数制約 c' が時刻 t において満たすことがない（行 (6)）ことを要請している．

上記の定義は，ある量 $x_i(t)$ を，時間の経過方向への制約伝播によって決めることは許しているが，遡及方向への制約伝播は排除している．

上記の意味論がどのように動作するかは，次節の例題を通じて確かめることができる．

6 記述例

6.1 ノコギリ波

現在時刻の小数点以下 ($\in [0,1)$) を返す周期関数 f を HydLa で定義すると次のようになる．

$$\text{INIT} \Leftrightarrow f=0.$$
$$\text{INCREASE} \Leftrightarrow \Box(f'=1).$$
$$\text{DROP} \Leftrightarrow \Box(f-=1 \Rightarrow f=0).$$
$$\text{INIT, (INCREASE} \ll \text{DROP).}$$

INIT は時刻 0 での f の値を制約する．INCREASE は DROP よりも弱い制約で，f の値を，1 になる直前まで増加させる．DROP は，f の値が 1 になろうとする瞬間に 0 にリセットする制約である．

現在時刻の小数点以下を返す関数 f は上記の制約を満たし，逆に，上記の制約をみたす関数は一意に定まる．なお f は INIT と DROP を常に満たし，INCREASE を $t=1,2,3,\ldots$ 以外で満たす．

上記のプログラムの INIT の定義を

$$\text{INIT} \Leftrightarrow 0 \le f < 1.$$

に変更すると何が起きるであろうか？ f の初期値は $[0,1)$ の任意の値となるが，その値を初期値として積分が始まり，値が 1 になろうとすると 0 にリセットされる．つまり値域 $[0,1)$ と傾き 1 をもつノコギリ波全体の集合を表すことになる．特定の t における $f(t)$ の値については $[0,1)$ であることしか言えないが，$f(t)$ の波形に関する諸性質は本プログラムから導くことができるはずである．

システム検証においては，初期値ではなくリセット条件の閾値が幅を持ち，かつ毎回変動する場合にも対応したい．$f(t)$ が 0.9 以上ならばリセットしてもよく，かつ $f(t)$ が 1 を超えてはならないとすると，

$$\text{MAX} \Leftrightarrow \Box(0.9 \le a < 1).$$

を追加した上で 3〜4 行目を

$$\text{DROP} \Leftrightarrow \Box(f-=a \Rightarrow f=0).$$
$$\text{MAX, INIT, (INCREASE} \ll \text{DROP).}$$

と変更すればよい．ここで a は値域 $[0.9,1)$ をとる関数であること以外に明示的な制約はかかっていないが，HydLa はフレーム公理として，関数の右連続性と左連続性を，その関数に対するユーザ定義制約よりも弱い制約（明示された不連続点以外で成り立つべき制約）として標準的に仮定する．したがって初期値 0 から単調に増加して 1 を超える関数は必ず連続関数 a との交点をもつ（中間値定理）．

6.2 踏切

ハイブリッドシステムの例題としてよく使われる踏切の制御を取り上げる．本例題では，2 本の列車，接近および通過検知センサ（踏切からそれぞれ 1000m, 100m 離して設置），コントローラ，遮断機をそれぞれモデル化している．二本の列車が 4000m 離れた地点から 220 秒の間隔をおいて秒速 20m で接近する．

$$\begin{aligned}
&\text{TRAIN}(N, \mathit{Ipos}, \mathit{Vel}) \ \{ \\
&\quad \mathit{train}(N) = \mathit{Ipos} \land \Box(\mathit{train}(N)' = \mathit{Vel}) \\
&\} \\
&\text{SENSOR}(S, \mathit{Pos}, \mathit{Sig}) \ \{ \\
&\quad \Box(\mathit{Sig} = 0) \\
&\quad \ll \forall N \in S(\Box(\mathit{train}(N) = \mathit{Pos} \ \Rightarrow \ \mathit{Sig}=1)) \\
&\} \\
&\text{CONTROLLER} \ \{ \\
&\quad \Box(\mathit{raise}' = 0) \\
&\quad \ll (\mathit{raise} = 1 \\
&\qquad \land \Box(\mathit{app}=1 \ \Rightarrow \ \mathit{raise}\hat{}5 = 0) \\
&\qquad \land \Box(\mathit{exit}=1 \ \Rightarrow \ \mathit{raise}\hat{}5 = 1)) \\
&\} \\
&\text{GATE} \ \{ \\
&\quad \Box(g' = 0) \\
&\quad \ll (g = 90 \\
&\qquad \land \Box(\mathit{raise}=1 \land g<90 \ \Rightarrow \ g' = 10) \\
&\qquad \land \Box(\mathit{raise}=0 \land g>0 \ \Rightarrow \ g' = -10)) \\
&\} \\
&\text{TRAIN}(1, 4000, -20), \\
&\text{TRAIN}(2, 4000, -20)\hat{}220, \\
&\text{SENSOR}(\{1,2\}, 1000, \mathit{app}), \\
&\text{SENSOR}(\{1,2\}, -100, \mathit{exit}), \\
&\text{CONTROLLER, GATE.}
\end{aligned}$$

train は列車番号を第 1 引数，時刻を（暗黙の）第 2 引数にとる複合関数 (4.1 節) である．SENSOR はイベント検知のための制約で，いずれかの列車がセンサ上を通過した時点だけ *Sig* の値を 1 にする．本論文に出現するほとんどの関数は右連続であるが，イベ

ントを表現する *Sig* は例外である．CONTROLLER はイベントを受け取ると 5 秒遅れて遮断機に制御信号を送り，GATE は制御信号にしたがって遮断機の開閉を行う．CONTROLLER は，SENSOR からのイベント情報を保持する双安定フリップフロップとして機能する．

最後の 5 行で，システム全体を構成要素から組み上げている．制約プログラミングにおいてはこのように，構成要素の並行合成は，各構成要素が課する制約の連言をとることで実現できる．構成要素間の通信は共有の関数変数 *app*, *exit*, *raise* を使って行っている．並行制約プログラミングと同様，一方のプロセスが関数変数に対する制約を発行 (tell) し，他方がその値を観測 (ask) することで通信を実現している．

7 まとめと今後の課題

制約モデリング言語 HydLa の設計目標はハイブリッドシステムの簡潔な記述である．それを可能にする宣言型記述の枠組と制約階層の意味論を提案し，記述例を紹介した．本研究は長期間の予備研究に立脚しているが，HydLa の具体形が定まってからは日が浅い．今後も多くの例題を記述してその記述性や設計の妥当性を確かめる必要がある．特に，ハイブリッドシステムには Zeno をはじめとするさまざまな病理現象が知られている [11] [12]．HydLa の設計の一つの動機は，病理現象をもつシステムの記述や解析の見通しを良くすることにあり，その点からの検討も進めている．

HydLa はモデリング言語であり，その実行は通常の前向き実行，すなわち系のシミュレーションだけではなくさまざまな方法を考える必要がある．通常実行においても，誤差を許容して速度を追求したい場合と結果の正当性を保証したい場合とがある．さらに後者には大きく分けて，区間制約に基づく方法 [9] と記号実行に基づく方法とがあり，それらの統合が必要であると考えている．検証においてもモデル検査手法が使える場合と演繹的手法が必要になる場合とがある．我々は HydLa の設計と並行してこれらの要素技術の検討を重ねてきたが，今後はプロトタイプ実装の作成を急ぎ，通常実行と検証の両面からの検討を進めてゆく予定である．

謝辞 HydLa 言語の着想や設計，早稲田大学上田研究室の大谷順司，笹嶋唯，露崎浩太，廣瀬賢一各氏，および上田研究室描画班 OB 各氏との関連諸技術に関する討論に支えられながら進んできた．本研究の一部は，科学研究費補助金（基盤研究 (B)20300013）の補助を得て行った．

参 考 文 献

[1] Bergstra, J. A. and Middleburg, C. A., Process Algebra for Hybrid Systems. *Theor. Comput. Sci.*, Vol. 335, No. 2–3 (2005), pp. 215–280.

[2] Borning, A., Freeman-Benson, B. and Wilson, M. Constraint Hierarchies. *Lisp and Symbolic Computation*, Vol. 5, No. 3 (1992), pp. 223–270.

[3] David, R. and Alla, H.. On Hybrid Petri Nets. *Discrete Event Dynamic Systems*, Vol. 11, No. 1–2 (2001), pp. 9–40.

[4] Freeman-Benson, B., Kaleidoscope: Mixing Objects, Constraints, and Imperative Programming. In *Proc. OOPSLA/ECOOP'90*, 1990, pp. 77–88.

[5] Gupta V., Jagadeesan, R., Saraswat, V. and Bobrow, D., Programming in Hybrid Constraint Languages. In *Hybrid Systems II*, LNCS 999, Springer-Verlag, 1995, pp. 226–251.

[6] Gupta, V, Jagadeesan, R., Saraswat, V., Computing with Continuous Changes. *Sci. Comput. Program.*, Vol. 30, No. 1–2 (1998), pp. 3–49.

[7] Henzinger, T. A., The Theory of Hybrid Automata. In *Proc. LICS'96*, 1996, pp. 278–292.

[8] Hickey, T. J. and Wittenberg, D. K., Rigorous Modeling of Hybrid Systems Using Interval Arithmetic Constraints. In *Proc. HSCC 2004*, LNCS 2993, Springer-Verlag, 2004, pp. 402–416,

[9] Ishii, D., Ueda, K. and Hosobe, H., An Interval-based Approximation Method for Discrete Changes in Hybrid cc. In *Trends in Constraint Programming*, F. Benhamou, et al. (Eds.), ISTE, Ltd., London, 2007, pp. 245–255.

[10] Lynch, N., Segala, R. and Vaandrager, F., Hybrid I/O Automata. *Inf. Comput.*, Vol. 185, No. 1 (2003), pp. 159–157.

[11] Mosterman, P. J., Hybrid Dynamic Systems: Modeling and Execution. In *Handbook of Dynamic System Modeling*, P. A. Fishwick (Ed.), Chapman & Hall/CRC, 2007, Chapter 15.

[12] 大野善之，石井大輔，上田和紀，数式処理・Quantifier Elimination を用いたハイブリッドシステムの Zeno 状態の導出手法．人工知能学会第 22 回全国大会論文集，2008，1D1-3.

[13] Platzer, A. and Quesel, J.-D., KeYmaera: A Hybrid Theorem Prover for Hybrid Systems. In *IJCAR 2008*, LNCS 5195, Springer-Verlag, 2008, pp. 171–178.

[14] Wilson, M. and Borning, A., Hierarchical Constraint Logic Programming. *J. Log. Program.*, Vol. 16, No. 3 (1993), pp. 277–318.