2014 年度 修士論文

リスト記法の導入と動的な制約集合導出によるハイブリッド制約言語 HydLa とその処理系のスケーラビリティ向上

提出日： 2015 年 1 月 24 日
指導 ： 上田 和紀 教授

早稲田大学大学院 基幹理工学研究科
情報理工学専攻

学籍番号： 5113B031-0
河野 文彦

概要

コンピュータ制御されたシステムは世の中に多く存在している．そのようなシステムの中でも実世界の情報を利用しながら実世界と相互作用するシステムが増えてきている．実世界と相互作用するシステムの例として自動車の衝突被害軽減ブレーキシステムなどが挙げられる．

実世界と相互作用するシステムの多くはハイブリッドシステムとみなすことができる．ハイブリッドシステム [13] とは時刻の経過に伴って状態や方程式が離散変化したり，状態が連続変化したりする動的システムであり，物理学や制御工学の多くの問題に適用できる．

ハイブリッドシステムモデリング言語 HydLa は制約階層 [6] に基づく宣言型の言語であり，制約を用いたモデルの簡潔な記述を特徴としている．HydLa では宣言された制約の優先度に従う制約の集合（解候補モジュール集合）のうち各時刻で極大無矛盾な集合を満たす変数の軌道がそのモデルの軌道となる．

HydLa の処理系 HyLaGI は数式処理による計算誤差のないシミュレーションや記号実行によるパラメータを含むモデルのシミュレーションなどの特徴を有している．

オブジェクトの数が多いハイブリッドシステムの挙動は自明でなく，そのようなモデルの挙動を求めることは重要である．例えば自動車の衝突被害軽減ブレーキシステムにおいて多数の自動車が存在する場合，各自動車がブレーキによって速度変化することで他の自動車に影響を及ぼすため，すべての車の挙動を正確に求めることは難しい．

HydLa や HyLaGI でオブジェクト数の多いモデルを扱うと，HydLa では全オブジェクトの制約を記述することによる記述量の増加が問題となり，HyLaGI では解候補モジュール集合の数が指数的に増加することによる実行時間の増加が問題となる．

本研究では HydLa にリスト記法を導入することで，HydLa の記述量の削減を行った．導入したリストの内包記法を用いることによって複数のオブジェクトに関する同じ性質を持つ制約（各オブジェクトの初期値に関する制約など）を 1 つのリストで表現できるようになり，オブジェクト数 n の HydLa プログラムの記述量を最大で $O(n^2)$ から $O(1)$ に削減することができた．

またシミュレーション中に必要となる解候補モジュール集合を動的に導出することで，オブジェクト数 n 個の HydLa プログラムに対して解候補モジュール集合の数を最大で $O(5^n)$ から $O(1)$ に削減することができ，解候補モジュール集合に関する処理の時間計算量を最大で $O(5^n n \log n)$ から $O(n)$ に抑えることができた．

Abstract

There are many computer controlled systems in the world. Among such systems, the number of systems interacting with the real world is increasing. For instance, automated brake systems in vehicles interact with the real world.

Most of these systems can be regarded as hybrid systems. Hybrid systems are dynamical systems including discrete changes and continuous changes. Hybrid systems apply to many problems of physics and control engineering.

HydLa is a declarative hybrid systems modeling language based on constraint hierarchies. We can model hybrid systems easily by using constraints. The trajectories of a model written in HydLa are trajectories of variables satisfying a maximally consistent candidate set of constraints at each time. Candidate sets are constraint sets which satisfy constraint hierarchies.

HyLaGI, an implementation of HydLa, can perform simulation without calculation errors by formula manipulation and can simulate parameterized models by symbolic execution.

The trajectories of hybrid systems with many objects are not clear. It is important to calculate such trajectories. In automated brake systems, it is difficult to calculate the behavior of all vehicles, because changing the velocity of a vehicle may affect other vehicles.

However, when we write and execute hybrid systems with many objects in HydLa, we must write constraints on all objects, and HyLaGI takes very long time when it simulates large HydLa programs because the number of candidate sets is exponential.

In this research, we first introduced list notation into HydLa to reduce the amount of description. We reduced the amount of description of a HydLa program with n objects from $\mathrm{O}(n^2)$ to $\mathrm{O}(1)$ by representing constraints on a series of objects with the same form, such as constraints on the initial values of each object, using list comprehension.

Secondly, we reduced the number of candidate sets generated for a model with n objects from $\mathrm{O}(5^n)$ to $\mathrm{O}(1)$ by dynamical generation of candidate sets. As a result, we reduced the time complexity of the processing of candidate sets from $\mathrm{O}(5^n n \log n)$ to $\mathrm{O}(n)$.

目次

図目次

表目次

第 1 章

はじめに

コンピュータ制御されたシステムは世の中に多く存在している．そのようなシステムの中でも実世界の情報を利用しながら実世界と相互作用するシステムが増えてきている．実世界と相互作用するシステムの例として自動車の衝突被害軽減ブレーキシステム [8] や飛行機の衝突回避システム [2] などが挙げられる．

実世界と相互作用するシステムの多くはハイブリッドシステムとみなすことができる．ハイブリッドシステム [13] とは時刻の経過に伴って状態や方程式が離散変化したり，状態が連続変化したりする動的システムである．ハイブリッドシステムは物理学や制御工学の多くの問題に適用することができる．例えば物理学では床や壁で跳ねるボールやビリヤード，スイッチ付きの電気回路などが挙げられる．制御工学では前述の自動車の衝突被害軽減ブレーキシステムや飛行機の衝突回避システムをはじめ，倒立振子の安定化制御や水槽の推移制御などが挙げられる．

ハイブリッドシステムをモデリングする言語として早稲田大学上田研究室で HydLa[19] が提案されている．HydLa は制約階層 [6] に基づく宣言型のハイブリッドシステムモデリング言語であり，制約を用いたモデルの簡潔な記述を特徴としている．HydLa では宣言された制約の優先度に従う制約の集合（解候補モジュール集合）のうち各時刻で極大無矛盾な集合を満たす変数の軌道がそのモデルの軌道となる．

我々は HydLa の処理系として HyLaGI[15] を開発している．HyLaGI は数式処理による計算誤差のないシミュレーションや記号実行によるパラメータを含むモデルのシミュレーション，非決定実行による定性的に異なる全解軌道の算出などを特徴としている．

オブジェクトの数が多い大規模なハイブリッドシステムの挙動は自明でなく，そのようなモデルの挙動を求めることは重要であると考えられる．例えば，自動車の衝突被害軽減ブレーキシステムにおいて多数の自動車が存在する場合，各自動車がブレーキによって速

度変化することで他の自動車に影響を及ぼすため，すべての車の挙動を正確に求めることは難しい．

しかし HydLa や HyLaGI でオブジェクト数の多いような大規模なモデルを扱うことを考えると，HydLa ではすべてのオブジェクトに関する制約を記述することによる記述量の増大が問題となり，HyLaGI では解候補モジュール集合の数の指数的増加による実行時間の増大が問題となる．例えば図 1.1 に挙げる非常に単純なモデルを考える．

図 1.1　1 次元上のビリヤード

図 1.1 は 1 次元上に複数のボールが存在しており，初期状態では 1 つのボールのみが動いており，他のボールと衝突をしてボールの運動の状態が変わっていくモデルである．このモデルのボールの数を n としたときに HydLa では $O(n^2)$ の数の制約を記述しなければならない．またこのモデルにおける解候補モジュール集合の数は 2^n 個であり，その処理に指数オーダーの実行時間がかかる．

そこで HydLa の記述量の増大を防ぐために HydLa にリスト記法を導入し，HyLaGI の実行時間の増大を防ぐために動的な制約集合の導出を行い探索する制約の集合の数を削減した．

本論文では背景としてハイブリッドシステムと HydLa，HyLaGI について説明したのち，動的な制約集合の生成手法および導入したリスト記法の説明を行う．

1.1　論文構成

本論文の構成は以下の通りである．第 2 章，第 3 章，第 4 章でそれぞれ研究の背景となるハイブリッドシステム，HydLa，HyLaGI について紹介する．第 5 章では HydLa での記述量の削減のために HydLa に導入するリスト記法の構文や記述例について説明し，リスト記法による HydLa での記述量の削減効果について評価する．第 6 章では HyLaGI の実行時間の削減のための動的な解候補モジュール集合の導出手法のアルゴリズムおよびそのアルゴリズムの正当性の証明や HyLaGI の実行アルゴリズムに加える変更点について説明し，動的な解候補モジュール集合導出による HyLaGI の実行時間削減効果について評価する．第 7 章で関連研究に触れ，第 8 章でまとめと今後の課題について述べる．

第 2 章

ハイブリッドシステム

ハイブリッドシステム [13] とは時刻の経過に伴って状態や方程式が離散変化したり，状態が連続変化したりする動的システムである．ハイブリッドシステムは物理学や制御工学の多くの問題に適用することができる．例えば物理学では床や壁で跳ねるボールやビリヤード，スイッチ付きの電気回路などが挙げられる．制御工学では自動車の衝突被害軽減ブレーキシステムや飛行機の衝突回避システム，倒立振子の安定化制御や水槽の推移制御 [11] などが挙げられる．

2.1　ハイブリッドシステムの例

この節ではいくつかの例を取りあげて詳しく説明する．以降ハイブリッドシステムの説明に用いる図は上から下にかけて時刻が経過しており，各段階における方程式が右側に書いてある図となっている．

まずは第 1 章で HydLa の記述量や HyLaGI の時間計算量を示すために使用した図 1.1 の 1 次元上のビリヤードの例について説明する．

1 次元上のビリヤードが時刻の経過に伴って状態が変化する様子を示した図を図 2.1 に示す．　図 2.1 のもっとも上の状態が 1 次元ビリヤードの初期状態を表している．初期状態ではボール 1 つのみが移動している状態である．上から 2 つ目の状態はボールが連続的に運動している様子を表している．3 つ目の状態では 1 つ目と 2 つ目のボールが衝突したときにそれらのボールの速度が離散的に変化する．最も下の状態がボールが衝突して速度が離散変化した後に連続的にボールが運動している様子を表している．1 次元上のビリヤードは状態が離散変化や連続変化しており，ハイブリッドシステムとしてみなすことができる．

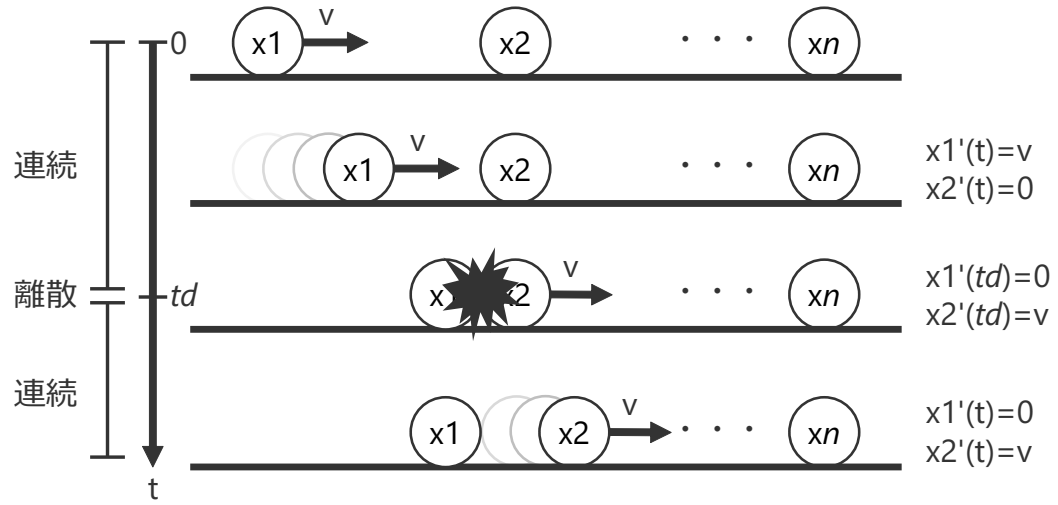

図 2.1　1 次元上のビリヤード（時刻の経過による各状態の図）

次に図 2.2 に 1 次元上で自動車が走行しているモデルを示す．　図 2.2 のもっとも上に

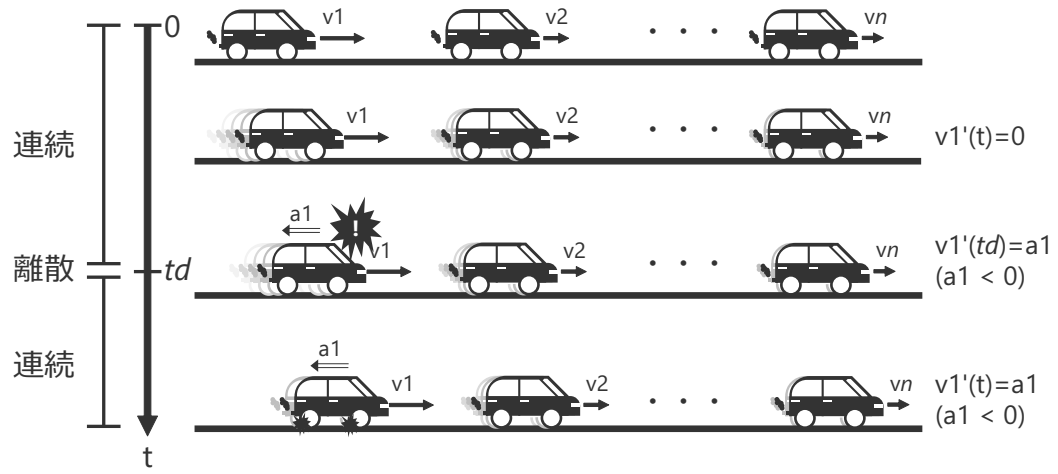

図 2.2　1 次元上での自動車の車間距離制御

ある状態が初期状態であり各自動車がそれぞれの速度で走行している．上から 2 つ目の状態ではしばらくの間，各自動車が等速で走行している状態を表している．上から 3 つ目の状態では最も左に書かれている自動車が車間距離が近いと判断し，減速し始める状態を表している．最後の状態では最も左の自動車のみが減速しつつ走行している状態を表している．自動車の交通モデルも状態が離散変化したり連続変化したりしているためハイブリッ

ドシステムとみなすことができる．

最後に水槽の推移制御のモデルを図 2.3 に示す．　図 2.3 のもっとも上の状態が初期状

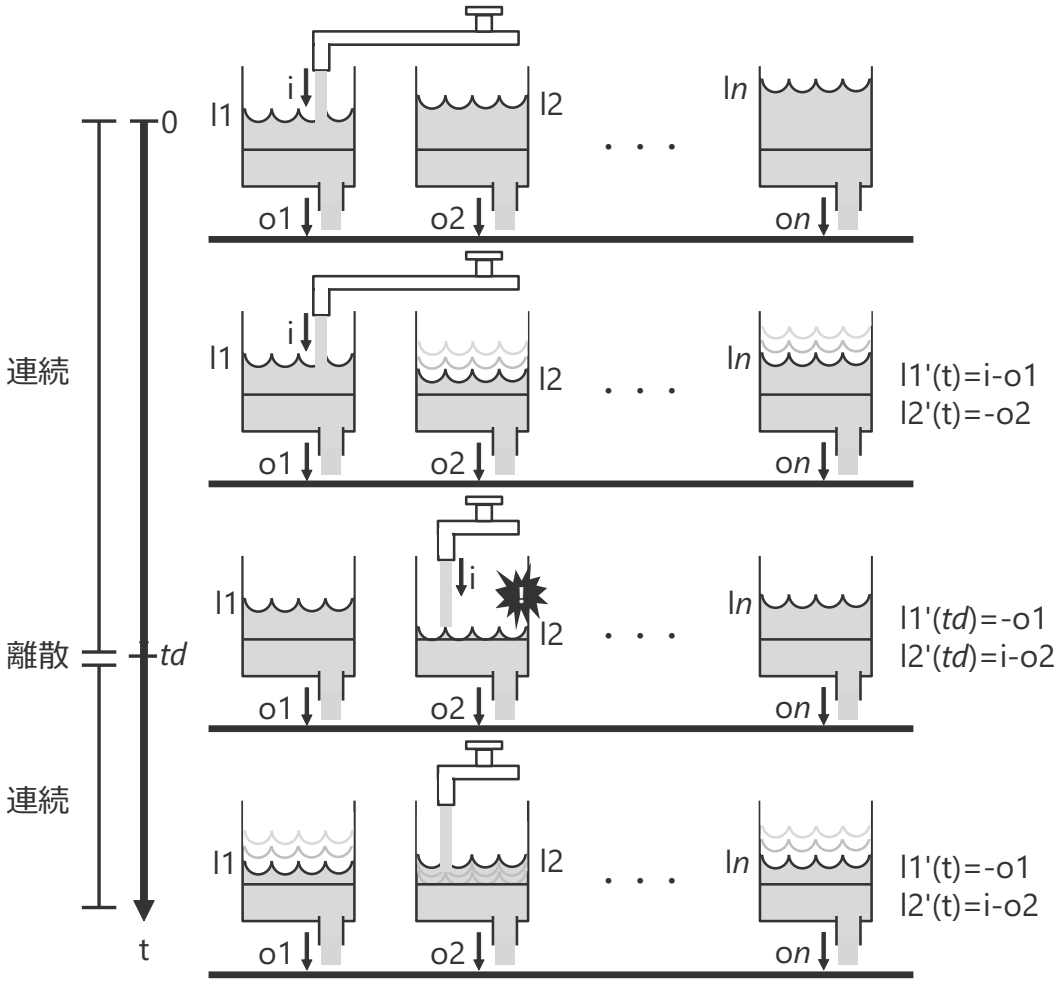

図 2.3　水槽の推移制御

態であり，各水槽から水が一定量ずつ流出しており，最も左の水槽にのみ水が流入されている状態を表している．2 つ目の状態では水位が一定量ずつ連続的に変化している様子を表している．3 つ目の状態では左から 2 つ目の水槽の水位が閾値に到達したため蛇口の位置が変化し 2 つ目の水槽に水が流入するように状態が離散的に変化する．最後の状態は水の流入状態が変化した状態で水位が連続的に変化している状態を表している．水槽の推移制御も時刻の経過に伴って状態が離散変化と連続変化を起こしておりハイブリッドシステムとみなすことができる．

図 2.1 や図 2.2，図 2.3 のモデルは 6.6 節の動的な解候補モジュール集合による実行時

間削減効果の評価および 5.3 節のリスト記法の導入によるプログラム記述量の評価に用いる.

第 3 章

ハイブリッドシステムモデリング言語 HydLa

HydLa[19] とは制約階層 [6] に基づく宣言型のハイブリッドシステムモデリング言語である.

HydLa は制約を用いることで与えられた数式をほとんどそのまま記述することができ, ハイブリッドシステムを簡潔にモデリングすることができる. HydLa で記述されたモデルのことを HydLa プログラムと呼ぶ.

3.1 HydLa の構文

本論文で HydLa 処理系 HyLaGI[15] が対象とする HydLa の構文を図 3.1 に示す.

(hydla program)	H	:=	(DF . | DC .)*
(definition)	DF	:=	$Dname(\vec{X})\{DC\}$ | $Cname(\vec{X})$ <=> C
(declaration)	DC	:=	M | $Dname(\vec{E})$ | DC,DC | DC << DC | (DC)
(module)	M	:=	C
(constraint)	C	:=	A | $Cname(\vec{E})$ | C /\ C | G => C | [] C | (C)
(guard)	G	:=	A | G /\ G | G \/ G | (G)
(atomic constraint)	A	:=	E ($relop$ E)+
(expression)	E	:=	通常の式 | E' | E^E | E-

図 3.1 HydLa の構文

図 3.1 は文献 [19] で記述されている HydLa の構文とは以下の点で異なっている．

- ガード条件に論理和を含めることができる
- 存在量化子 ∃ を使用できない
- 原子制約に 3 つ以上の数式を等号または不等号によって繋げて記述することができる
- 各構文規則の名前が異なっている

HydLa プログラムは (hydla program) で定義される構文で記述される．

図 3.1 の 3 行目の (declaration) が表す構文で HydLa プログラムで用いる制約の優先度を宣言する．制約の優先度は記号<< をによって記述を行う．例えば A<< B は制約 A よりも制約 B の方が優先度が高いことを意味する．制約の優先度の宣言は記号<< による優先度の記述を記号, で列挙することによって記述する．(declaration) は記号, や記号<< を含まない 1 つの制約の記述を許しているが，これはその制約と優先度関係にある制約が存在しないことを示している．制約の優先度については 3.2 節で詳しく説明する．

宣言された制約の優先度に含まれる制約をモジュールと呼び，図 3.1 の 4 行目の (module) がモジュールに対応している．

図 3.1 内に出現する *Dname* は定義された制約の優先度の名前を表している．

Cname は定義された制約の名前を表しており，*Cname* の引数の $\vec{X}$ は束縛変数の列である．

HydLa プログラム内に出現する変数はすべて時刻に関する関数であり，例えば HydLa プログラム中に変数の x を記述した場合，その x は $x(t)$ を表す．

直後に記号–が付随する式はその式の左極限値を表し，直後に記号’ が付随する式はその式の時間に関する微分を表す．

3.2 HydLa の宣言的意味

HydLa でモジュール間に優先度を与えるときには記号, と記号 << を用いる．記号 << は 2 つのモジュール間に優先度を与える推移的な二項関係である．例えばモジュール M_1，M_2 に記号 << を用いて M_1<< M_2 のように優先度を定義したとき，M_1 よりも M_2 の方が優先度が高いことを意味する．一般に M_1<< M_2 と優先度を定義した場合 M_1 がモデルのデフォルトの動作を表す制約であり，M_2 がモデルの例外的な動作を表す制約となる．また，自身より優先度の高いモジュールが存在しないモジュールを Required なモジュールと呼ぶ．与えられたモジュール R が Required であるかどうか判断する関数

IsRequired(R) を式 3.1 で定義する．*IsRequired*(R) が True を返す場合にモジュール R は Required なモジュールとなる．

$$IsRequired(R) := \neg\exists M(R << M) \tag{3.1}$$

HydLa での優先度の宣言は記号 << が与える二項関係または 1 つのモジュールを記号, を用いて列挙することで記述する．1 つのモジュールが記述されているとき，そのモジュールと優先度関係にあるモジュールが存在しないことを明記していることを意味する．

HydLa では略記法を許しており，記号 << を複数用いて 3 つ以上のモジュールを繋げた記述や（）を用いた記述が可能である．記号 << を複数用いて 3 つ以上のモジュールを繋げた場合，隣り合う 2 つのモジュールに優先度関係を与えたものが列挙されているものと解釈する．例えば $M_1 << M_2 << M_3$ であれば $M_1 << M_2, M_2 << M_3$ と解釈される．（）を用いた記述を行った場合，記号，と記号 << をそれぞれ算術演算子の + と $\times$ に対応する優先順位で処理を行う．例えば $(M_1, M_2) << M_3$ であれば $M_1 << M_3, M_2 << M_3$ と解釈される．

定義されたすべての優先度を満たすモジュールの集合を解候補モジュール集合と呼ぶ．解候補モジュール集合は式 3.2 と式 3.3 を満たす集合 *MS* である．

$$\forall M_1, \forall M_2(M_1 << M_2 \wedge M_1 \in MS \Rightarrow M_2 \in MS) \tag{3.2}$$

$$\forall M(IsRequired(M) \Rightarrow M \in MS) \tag{3.3}$$

HydLa プログラムに出現する各変数の振る舞いのうち，各時刻における解候補モジュール集合内の極大無矛盾集合を満たすものが HydLa の宣言的意味となる．

3.3 HydLa の記述例

天井に向かって投げ上げる質点のモデルを HydLa で記述したものを図 3.2 に示す．

```
INIT(x) <=> 9<x<11 /\ x'=0.
FALL(x) <=> [](x''=-10).
BOUNCE(x) <=> [](x-=15 => x'=-(4/5)*x'-).

INIT(y), FALL(y) << BOUNCE(y).
```

図 3.2 天井に質点を投げ上げる HydLa プログラム

1 行目では質点の初期値に関する制約 INIT を定義している．INIT は引数付きの制約であり，引数として受け取った変数 x についての制約となる．2 行目，3 行目で定義している制約も同様に引数付きの制約である．HydLa のモデルに出現する変数はすべて時刻に関する関数であり，記号’は時刻に関する微分である．さらに記号 [] のつかない制約は初期時刻でのみ成立する制約であるため，INIT の定義は式 3.4 を意味している．

$$9 < \mathtt{x}(0) /\backslash \mathtt{x}(0) < 11 /\backslash \frac{d\mathtt{x}(0)}{dt} = 0 \tag{3.4}$$

2 行目では質点にかかる重力に関する制約 FALL を定義している．記号 [] は □ を表しており時刻 t が 0 以上で成立する制約であることを意味する．さらに直後に記号-が付いた変数は左極限値を意味している．従って FALL の定義は式 3.5 を意味する．

$$\forall t \geq 0 \left(\frac{d^2\mathtt{x}(t)}{dt^2} = -10 \right) \tag{3.5}$$

3 行目では質点が天井に衝突したときの制約 BOUNCE を定義している．記号=>によってガード条件付制約を表現しているため BOUNCE は式 3.6 を意味する．

$$\forall t \geq 0 \left(\lim_{t_p \to t-0} \mathtt{x}(t_p) = 15 \Rightarrow \frac{d\mathtt{x}(t)}{dt} = \lim_{t_p \to t-0} \frac{d\mathtt{x}(t_p)}{dt} \right) \tag{3.6}$$

最後の行では 1 行目から 3 行目で定義した制約間の優先度を宣言している．各制約には変数 y を引数として渡している．図 3.2 では制約 INIT と優先度関係をもつ制約はなく，制約 FALL は制約 BOUNCE よりも優先度が低いことを表している．

3.4　本論文で使用するプログラム

本論文での動的な解候補モジュール集合の導出（6 章）およびリスト記法の導入（5）で用いるモデルの HydLa プログラムを示す．

1 つ目に図 2.1 のモデルでボールが 3 つの場合のモデルを示す．

図 3.3 のプログラムでは各ボールに大きさはなく，質量がすべて等しく，すべての衝突は完全弾性衝突であり，すべてのボールは 2 枚の壁に挟まれている．制約 INIT は各ボールの初期位置および初期速度に関する制約である．制約 COLLISION はボール同士の衝突により，各ボールの速度が変化するという制約である．制約 UNIFORM は各ボールが等速度運動をするという制約である．すべてのボールは 2 枚の壁に挟まれており制約 WALL は各ボールと壁との衝突によってボールの速度が変化する制約である．ボール同士の衝突やボールと壁の衝突によって速度が変化する瞬間には衝突による速度変化の制約

```
INIT(b,b0,vb0) <=> b=b0 /\ b'=vb0.
COLLISION(b1,b2) <=> [](b1-=b2- => b1'=b2'- /\ b2'=b1'-).
UNIFORM(b) <=> [](b''=0).
WALL(b) <=> [](b-=-1 \/ b-=10 => b'=-b'-).

INIT(x1,0,1), INIT(x2,2,0), INIT(x3,6,0),
(UNIFORM(x1), UNIFORM(x2)) << COLLISION(x1,x2),
(UNIFORM(x1), UNIFORM(x3)) << COLLISION(x1,x3),
(UNIFORM(x2), UNIFORM(x3)) << COLLISION(x2,x3),
UNIFORM(x1) << WALL(x1), UNIFORM(x2) << WALL(x2),
UNIFORM(x3) << WALL(x3).
```

図 3.3 ボールが 3 つの 1 次元ビリヤード（図 2.1）の HydLa プログラム

COLLISION や WALL とボールが等速運動をするという制約 UNIFORM を同時に満たすことができない．図 2.1 のモデルでは衝突の際には速度変化をすることを想定しているので COLLISION および WALL は UNIFORM よりも優先度を高く記述している．

次に図 2.2 のモデルで自動車 3 台の場合のモデルを図 3.4 に示す．

```
INIT(x,x0,v0) <=> x = x0 /\ x' = v0 /\ x'' = 0.
CONSTANT <=> [](distance = 100) /\ [](maxV = 25) /\ [](minV = 15).
DISTANT(x1,x2) <=>
    [](x1'- < maxV- /\ x2- - x1- >= distance * 6/5 => x1'' = 1).
NEAR(x1,x2) <=>
    [](x1'- > minV- /\ x2- - x1- <= distance * 4/5 => x1'' = -1).
UNIFORM(x) <=> [](x'' = 0).
COLLISION(x1,x2) <=> [](x1- = x2- => [](x1' = 0 /\ x2' = 0)).

CONSTANT.
INIT(x1,100,20), INIT(x2,200,20), INIT(x3,300,19),
UNIFORM(x1) << (DISTANT(x1,x2), NEAR(x1,x2)) <<
    COLLISION(x1,x2),
UNIFORM(x2) << (DISTANT(x2,x3), NEAR(x2,x3)) <<
    (COLLISION(x1,x2), COLLISION(x2,x3)),
UNIFORM(x3) << COLLISION(x2,x3).
```

図 3.4 自動車が 3 台の場合の車間制御（図 2.2）の HydLa プログラム

図 3.4 のプログラムでは各自動車の大きさがないものとし，車間距離を distant に保とうと各自動車が自車の速度を距離に応じて制御するプログラムである．制約 INIT は各自動車の初期位置および初期速度に関する制約である．制約 CONSTANT は定数としてプロ

グラム中で使用する車間距離 distant，自動車の最高速度 maxV，自動車の最低速度 minV に関する制約である．制約 UNIFORM は各自動車が等速度運動をするという制約である．制約 DISTANT は前方車両との距離が離れた場合（車間距離が distant $\times$ 6/5 以上）の場合自車を加速させる制約である．制約 NEAR は前方車両との距離が離れた場合（車間距離が distant $\times$ 4/5 以下）の場合自車を減速させる制約である．制約 COLLISION は前後の車両と衝突すると以降は常に停止する状態になるという制約である．制約 DISTANT と制約 NEAR は車間距離に応じて自動車の加速度を変化させる制約であり，自車と前方車両との距離が遠すぎたり近すぎたりする場合には等速運動をする制約の UNIFORM と DISTANT や NEAR を同時に満たすことはできない．図 2.2 のモデルでは距離に応じて自動車の速度を変化させるので DISTANT および NEAR を UNIFORM よりも優先度を高く記述している．また制約 COLLISION は衝突したときに以降の速度を常に 0 にするという制約であり，衝突して事故が起きた際に再度動き出すことがないようにモデリングするために車間距離に応じて加速度を変化させる制約の DISTANT と NEAR よりも高い優先度で記述している．

次に図 2.3 のモデルで水槽が 3 つの場合のモデルを図 3.5 に示す．

```
INIT(x,xinit,vinit) <=> x = xinit /\ x' = vinit.
CONSTANT <=> [](inflow = 3).
LINEAR(x) <=> [](x'' = 0).
INFLOW(x1,r1, outflow) <=> [](x1- = r1- => x1' = inflow - outflow).
NO_INFLOW(x1,r1,x2, outflow) <=>
    [](x1- = r1- & x2'- = inflow - outflow => x2' = -outflow).

CONSTANT,
INIT(x1, 1, inflow - 1), INIT(x2,2+2/2,-1-((2-1)/17)),
INIT(x2,2+3/2,-1-((3-1)/17)),
LINEAR(x1) << INFLOW(x1, 1, 1+((1-1)/17)),
LINEAR(x2) << INFLOW(x2, 1, 1+((2-1)/17)),
LINEAR(x3) << INFLOW(x3, 1, 1+((3-1)/17)),
LINEAR(x1) << NO_INFLOW(x2, 1, x1, 1+((1-1)/17)),
LINEAR(x1) << NO_INFLOW(x3, 1, x1, 1+((1-1)/17)),
LINEAR(x2) << NO_INFLOW(x1, 1, x2, 1+((2-1)/17)),
LINEAR(x2) << NO_INFLOW(x3, 1, x2, 1+((2-1)/17)),
LINEAR(x3) << NO_INFLOW(x1, 1, x3, 1+((3-1)/17)),
LINEAR(x3) << NO_INFLOW(x2, 1, x3, 1+((3-1)/17)),
```

図 3.5　水槽が 3 つの場合の水槽の水位制御（図 2.3）の HydLa プログラム

図 3.5 のプログラムでは全水槽から水が流出しており，1 つの水槽にのみ水の流入があるプログラムである．ある水槽 ti の水位 xi が閾値に達した場合，水の流入がその水槽 t

に切り替わる．制約 INIT は各水槽の初期水位および水位の増減速度に関する制約である．制約 CONSTANT は定数としてプログラム中で使用する水の単位時間あたりの流入量 inflow に関する制約である．制約 LINEAR は水位の単位時間あたりの変位量は一定であるという制約である．制約 FLOW は水槽 ti の水位が閾値に達したときに水槽 ti に水が流入するという制約である．制約 NO_INFLOW は水槽 ti の水位が閾値に達し，水槽 tj に水が流入している状態のときに水槽 tj に対する水の流入を止める制約である．制約 FLOW と制約 NO_INFLOW によって同時に水が流入する水槽を 1 つに限定している．制約 FLOW と制約 NO_INFLOW は水位に応じて水が流入する水槽（水槽の水位の単位時間あたりの変位量）を変化させる制約であり，水が流入する水槽が切り替わるときに水位の単位時間あたりの変化量が一定であるという制約の LINEAR と INFLOW や NO_INFLOW を同時に満たすことはできない．水位が閾値に達したときに水が流入する水槽を変化させるために INFLOW および NO_INFLOW を LINEAR よりも優先度を高く記述している．

最後にボールが 3 つの 1 次元ビリヤードのモデルのボール同士の衝突時における反発係数に非決定性を持たせた HydLa プログラムを図 3.6 に示す．

```
INIT(b,b0,vb0) <=> b=b0 /\ b'=vb0.
COLLISION1(b1,b2) <=> [](b1-=b2- => b1'=b2'- /\ b2'=b1'-).
COLLISION2(b1,b2) <=> [](b1-=b2- => b1'=(4/5)*b2'- /\ b2'=4/5*b1'-).
UNIFORM(b) <=> [](b''=0).
WALL(b) <=> [](b-=-1 \/ b-=2*4 => b'=-b'-).

INIT(x1,0,1), INIT(x2,2*2-2,0), INIT(x3,2*3-2,0),

UNIFORM(x1) << (WALL(x1), ((COLLISION1(x1,x2), COLLISION2(x1,x2),
        COLLISION1(x1,x3), COLLISION2(x1,x3)) << $TRUE)).
UNIFORM(x2) << (WALL(x2), ((COLLISION1(x1,x2), COLLISION2(x1,x2),
        COLLISION1(x2,x3), COLLISION2(x2,x3)) << $TRUE)).
UNIFORM(x3) << (WALL(x3), ((COLLISION1(x1,x3), COLLISION2(x1,x3),
        COLLISION1(x2,x3), COLLISION2(x2,x3)) << $TRUE)).
```

図 3.6 ボールが 3 つの非決定的なビリヤードの HydLa プログラム

図 3.3 と図 3.6 の違いはボール同士の衝突によるボールの速度変化の制約が 2 つ定義されていることである．2 つのボール同士の衝突の制約 COLLISION1(x1,x2) および COLLISION2(x1,x2) の違いは反発係数の値であり，それぞれの制約の優先度を同じものとして宣言し，$TRUE（HyLaGI の組み込みの値であり論理値の True を表す）よりも弱い優先度として宣言することでそれぞれのボール同士の衝突の制約は同じ優先度かつ

Required でないモジュールとなるため，ボール同士が衝突する際に極大無矛盾集合が複数存在し，非決定的な振る舞いをすることになる．

第 4 章

HydLa 処理系 HyLaGI

HyLaGI[15] とは数式処理に基づいた HydLa の処理系である．入力として HydLa プログラムを受け取り，与えられた HydLa プログラムを満たす変数の軌道群（解軌道）を出力する．HyLaGI は以下の特徴を備えている．

- 数式処理により誤差のないシミュレーションが可能
- 記号実行によりパラメータを含むモデルをシミュレーション可能
- 非決定実行により定性的に異なる解を同時に算出可能

HyLaGI は C++ で記述されており約 36,000 行規模のプログラムとなっている．HyLaGI はバックエンドとして Mathematica[20] と REDUCE[4] を用いている．

4.1 HyLaGI の実行アルゴリズム

HyLaGI の実行アルゴリズムを図 4.1 に示す．

HyLaGI は離散変化の計算フェーズである Point Phase(PP) と連続変化の計算フェーズである Interval Phase(IP) をシミュレーション時刻が HyLaGI 実行時に指定した時刻になるまで繰り返し計算することでシミュレーションを行う．PP の処理は図 4.1 の 5 行目から 11 行目に対応し，IP の処理は図 4.1 の 12 行目から 19 行目に対応している．

以下図 4.1 のアルゴリズムについて説明する．*SolveCH* は入力の HydLa プログラムに記述されたモジュールの優先度の宣言を用いてすべての解候補モジュール集合を求める関数である．求めたすべての解候補モジュール集合は *TopologicalSort* によって集合の包含関係に基づいてソートされ *MS* に代入される．*MaxModuleSet* によって全解候補モジュール集合のうち要素数が最大の集合 M_{all} に代入し，*GetVariables* によっては HydLa

Input: HydLa プログラム $HydLaProgram$，シミュレーション終了時刻 $MaxT$

1: MS := $TopologicalSort(SolveCH(HydLaProgram))$
2: M_{all} := $MaxModuleSet(MS)$
3: V := $GetVariables(HydLaProgram)$
4: T := 0; S := true; CP := true; E := $\emptyset$
5: **while** $T <_{CP} MaxT$ **do**
6: // PP
7: S := $SubstituteMinTime(S,T)$
8: $(S,CP,E,\_,\_,\_)$:= $CalculateMCS(S,MS,E,CP,T,CheckConsistencyPP)$
9: **if** S = false **then**
10: break
11: **end if**
12: (S,CP) := $AddParameters(S,CP,V)$
13: // IP
14: (S,CP,E,M,A_+,A_-) := $CalculateMCS(S,MS,E,CP,T,CheckConsistencyIP)$
15: S := $SolveDifferentialEquation(S)$
16: **if** S = false $\vee \neg\ IsUnique(S,V)$ **then**
17: break
18: **end if**
19: $(MinT,CP)$:= $GetElement(CompareMinTime($
$\{FindMinTime(S \wedge CP \Rightarrow g) \mid (g \Rightarrow c) \in A_-\}$
$\cup \{FindMinTime(S \wedge CP \Rightarrow \neg\ g) \mid (g \Rightarrow c) \in A_+\}$
$\cup \{FindMinTime(S \wedge CP \wedge M_-) \mid M_- \in (M_{all} \setminus M)\}$
$\cup \{FindMinTime(S \wedge CP \wedge \neg\ M_+) \mid M_+ \in M\}$
$\cup \{(MaxT - T, \text{true})\}))$
20: T := $MinT + T$
21: **end while**

図 4.1 HyLaGI の非決定実行アルゴリズム（文献 [15] より引用）

プログラム内に出現するすべての変数の集合を V に代入している．以下の処理をシミュレーション時刻 T が与えられたシミュレーション終了時刻 $MaxT$ に達するまで繰り返す．HydLa プログラム内に出現する変数は時刻に関する関数であり，*SubstituteMinTime* によって出現する全変数に現在時刻を代入する．次に *CalculateMCS* によって解候補モジュール集合の中から極大無矛盾集合を求め，そのときの制約や記号定数に関する条件などを得る．*CalculateMCS* の詳細を図 4.2 に示す．得られた制約が false であった場合シミュレーションを終了する．*CalculateMCS* の結果，値が一意に定まらない変数が出現した場合に *AddParameters* によってパラメータを導入しその変数を置き換える．次に処理が IP に移る．最初に *CalculateMCS* によって解候補モジュール集合の中から極大無矛盾集合を求め，そのときの制約や記号定数に関する条件などを得る．得られた制約が false

であるか値が一意に定まらない変数が出現した場合にはシミュレーションを終了する．その後，*FindMinTime* によって与えられた各条件を満たす最小の正の時刻を求め，得られた各時刻を *CompareMinTime* によって比較し，*GetElement* によって得られた最小時刻とそのときのパラメータを得る．最後にシミュレーション時刻を現在時刻に得られた最小時刻を加算することで更新し，5 行目に処理が戻る．このときシミュレーション時刻が終了時刻を超えた場合にはループを抜けシミュレーションが終了する．

Input: 制約ストア S，解候補モジュール集合のリスト MS，展開済み always 制約の集合 E，記号定数に関する条件 CP，現在時刻 T，無矛盾性判定関数 *CheckConsistency*
Output: 制約ストア，記号定数に関する条件，展開済み always 制約の集合，成り立つガード条件の集合，成り立たないガード条件の集合

1: **for** $M \in MS$ **do**
2: **if** $T > 0$ **then**
3: M := *EliminateNotAlways(M)*
4: **end if**
5: $(S_{tmp}, E_{tmp}, CP, A_-, A_+)$:= *CalculateClosure*$(S, M, CP, E, CheckConsistency)$
6: **if** $S_{tmp} \neq$ false **then**
7: **return** $(S_{tmp}, CP, E_{tmp}, M, A_-, A_+)$
8: **end if**
9: **end for**
10: **return** $(\text{false}, CP, E, \emptyset, \emptyset, \emptyset)$

図 4.2 CalculateMCS（文献 [15] より引用）

4.2 HyLaGI の実行例

HyLaGI で図 3.2 の HydLa プログラムをシミュレーションしたときの出力を示す．図 3.2 のモデルでは質点の初期位置に幅があるため，図 3.2 のモデルの解軌道は以下のものが考えられる．考えられるそれぞれの解軌道に対応する HyLaGI の出力は各項目に括弧で記述された番号の図が対応する．

- 質点が天井に接する（図 4.4）
- 質点が天井に衝突する（図 4.5）
- 質点が天井にとどかない（図 4.6）

図 3.2 のプログラムの解軌道を図示したものを図 4.3 に示す．

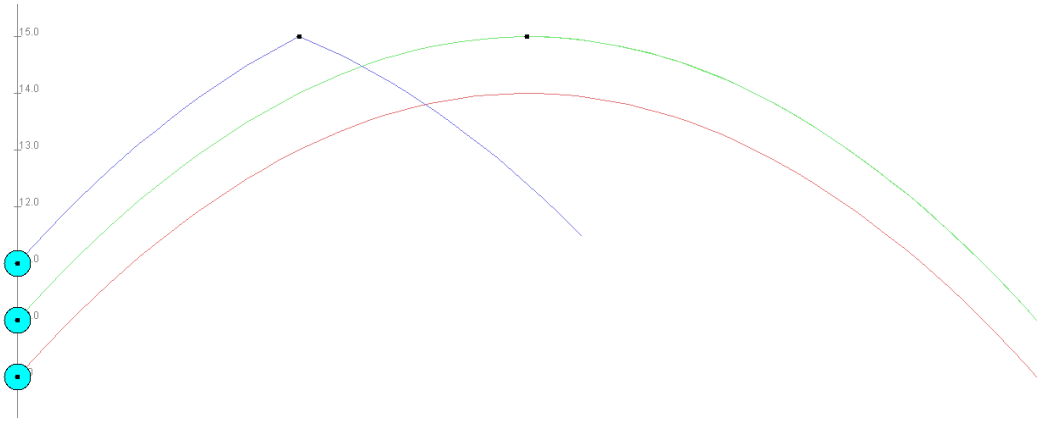

図 4.3 図 3.2 のプログラムの解軌道

```
---------parameter condition---------
p[y, 0, 1]       : (9, 11)
---------Case 1---------
---------1---------
---------PP---------
unadopted modules: {}
t        : 0
y        : p[y, 0, 1]
y'       : 10
y''      : -10
---------IP---------
unadopted modules: {}
t        : 0->1+(-1)*(-2+p[y, 0, 1]*1/5)^(1/2)
y        : t*(t+(-2))*(-5)+p[y, 0, 1]
y'       : (t+(-1))*(-10)
y''      : -10
---------2---------
---------PP---------
unadopted modules: {}
t        : 1+(-1)*(-2+p[y, 0, 1]*1/5)^(1/2)
y        : 15
y'       : 0
y''      : -10
---------IP---------
unadopted modules: {}
t        : 1+(-1)*(-2+p[y, 0, 1]*1/5)^(1/2)->inf
y        : (-5)*(-3+(t+(-1)+(-2+p[y, 0, 1]*1/5)^(1/2))^2)
y'       : (-10)*(t+(-1)+(-2+p[y, 0, 1]*1/5)^(1/2))
y''      : -10
---------parameter condition---------
p[y, 0, 1]       : 10
# time reached limit
```

図 4.4 質点が天井に接する場合の出力

```
---------Case 2---------
---------1---------
---------PP---------
unadopted modules: {}
t       : 0
y       : p[y, 0, 1]
y'      : 10
y''     : -10
---------IP---------
unadopted modules: {}
t       : 0->1+(-1)*(-2+p[y, 0, 1]*1/5)^(1/2)
y       : t*(t+(-2))*(-5)+p[y, 0, 1]
y'      : (t+(-1))*(-10)
y''     : -10
---------2---------
---------PP---------
unadopted modules: {FALL().}
t       : 1+(-1)*(-2+p[y, 0, 1]*1/5)^(1/2)
y       : 15
y'      : 5^(-1/2)*(-8)*(-10+p[y, 0, 1])^(1/2)
---------IP---------
unadopted modules: {}
t       : 1+(-1)*(-2+p[y, 0, 1]*1/5)^(1/2)->inf
y       : 36+t^2*(-5)+18*(-2+p[y, 0, 1]*1/5)^(1/2)
          +t*(10+(-18)*(-2+p[y, 0, 1]*1/5)^(1/2))+p[y, 0, 1]*(-13)/5
y'      : 10+t*(-10)+(-18)*(-2+p[y, 0, 1]*1/5)^(1/2)
y''     : -10
---------parameter condition---------
p[y, 0, 1]      : (10, 11)
# time reached limit
```

図 4.5 質点が天井に衝突する場合の出力

```
---------Case 3---------
---------1---------
---------PP---------
unadopted modules: {}
t       : 0
y       : p[y, 0, 1]
y'      : 10
y''     : -10
---------IP---------
unadopted modules: {}
t       : 0->inf
y       : t*(t+(-2))*(-5)+p[y, 0, 1]
y'      : (t+(-1))*(-10)
y''     : -10
---------parameter condition---------
p[y, 0, 1]      : (9, 10)
# time reached limit
```

図 4.6 質点が天井にとどかない場合の出力

第 5 章

リスト記法の導入

オブジェクト数の多い問題を HydLa でモデリングするときにはすべてのオブジェクトについての制約を記述しなければならずプログラムの記述量が増えてしまう。

例えば図 2.1 で示した 1 次元上のビリヤードの例ではボールの数 n に対し各ボールについての初期値制約を n 個，各ボールの衝突の制約を 2 つのボールの全組み合わせ数である ${}_nC_2$ 個，各ボールと壁の衝突の制約を n 個，各ボールが等速運動をするという制約を n 個記述しなければならない。

そこでオブジェクト数の多い問題に対する HydLa プログラムの記述量の増大を防ぐためにリストを用いた記法を HydLa に導入する。本章では新たに HydLa に導入する記法およびその使用例について説明する。

5.1 導入する構文

図 5.1 にリストを導入した HydLa の構文を示す。

新たな構文では制約の優先度定義を要素とするリスト（priority list，PL）と式を要素とするリスト（expression list，EL）を導入する。

PL は要素をすべて列挙する記法（$\{MP_1, MP_2, \cdots, MP_n\}$），リストの内包記法を利用した記法（$\{MP \mid LC_1, LC_2, \cdots, LC_n\}$）および定義したリスト名（$PLname$）によって記述することができる。例えば内包記法について {INIT(i) | i in {1,2,3,4} } と記述した場合には {INIT(1),INIT(2),INIT(3),INIT(4)} を意味する。

EL は PL の要素の制約の優先度に対応する部分を式 E に変更した記述に加え，2 点リーダを用いた記法で記述することができる。2 点リーダを用いた記法はでは変数を含まない式と名前が数字で終わる変数を用いることができる。変数を含まない式を用いた記述

(hydla program)	H	:=	(DF . | DC .)*
(declaration)	DC	:=	M | PL | $PL[E]$ | DC, DC | DC << DC | (DC)
(definition)	DF	:=	$MPname(\vec{X})\{MP\}$ | $Cname(\vec{X})$ <=> C
		|	$ELname$:= EL | $PLname$:= PL
(list condition)	LC	:=	$MPname$ in PL | $Variable$ in EL | E =!= E
(list size)	LS	:=	| (PL | EL) |
(priority list)	PL	:=	{MP (, MP)*} | {MP | LC (, LC)*} | $PLname$
(module priority)	MP	:=	M | $MPname(\vec{E})$ | MP,MP | MP << MP | (MP)
(module)	M	:=	C
(constraint)	C	:=	A | $Cname(\vec{E})$ | C /\ C | G => C | [] C | (C)
(guard)	G	:=	A | G /\ G | G \/ G | (G)
(atomic constraint)	A	:=	E ($relop$ E)+
(expression list)	EL	:=	{E (, E)*} | {E | LC (, LC)*} | $ELname$
		|	{RE..RE}
(range expression)	RE	:=	変数を含まない式 | 名前が数字で終わる変数
(expression)	E	:=	通常の式 | E' | E^E | E- | $EL[E]$

図 5.1　リストを導入した HydLa の構文

では {1*2+1..5}（{3,4,5}）や { j \| i in {1,2}, j in {i+1..4}}（{2,3,4, 3,4}）などの記述が可能である。この記述での i はリスト {1,2} の要素を列挙するための一時変数であり，{i+1..4} が評価される際には i に 1 と 2 が代入され i+1 は変数を含まない式になる。名前が数字で終わる変数を用いた記述では最後の数字部分以外の名前が等しい必要があり，{x1..x3} のような記述を行う。{x1..x3} は {x1,x2,x3} を表す。

リストの要素にアクセスする際には [E] を用いる。あるリスト L に対して $L[E]$ と記述した場合に L 内の E 番目の要素を表す。

リストの要素数を取得する記法としてあるリスト L に対して $|L|$ と記述する。例えばリスト L が {x1,x2,x3} で定義されているとき $|L|$ は 3 を意味する。

制約の優先度のリストの導入に伴って宣言部の構文に変更を加える。宣言部の要素の単位としてモジュール M 以外に制約の優先度リスト PL およびその要素 $PL[E]$ を記述可能とする。優先度リスト PL を宣言部に記述した場合には PL 内の全要素を記号，を用いて列挙したものと解釈する。

5.2　リスト記法を用いた HydLa プログラムの例

本節では導入したリスト記法を用いた HydLa プログラムの例を示す。動的な解候補モジュール集合導出の例を図 3.3 の 1 次元ビリヤードの例を用いて示す。INIT はボールの数分である INIT(x1), INIT(x2), INIT(x3), INIT(x4) の 4 種類, COLLISION は全ボールから 2 つのボールを取り出す組み合わせ数である COLLISION(x1,x2)，COLLISION(x1,x3)，COLLISION(x1,x4)，COLLISION(x2,x3)，COLLISION(x2,x4)，COLLISION(x3,x4) の 6（$= {}_4C_2$）種類，WALL はボールの数分である WALL(x1)，WALL(x2)，WALL(x3)，WALL(x4) の 4 種類，UNIFORM はボールの数分である UNIFORM(x1)，UNIFORM(x2)，UNIFORM(x3)，UNIFORM(x4) の 4 種類が記述されている。

次に導入したリスト記法を用いた図 3.3 と等価な HydLa プログラムを図 5.2 に示す。

```
X := {x1..x4}

INIT(b,b0,vb0) <=> b=b0 /\ b'=vb0.
COLLISION(b1,b2) <=> [](b1-=b2- => b1'=b2'- /\ b2'=b1'-).
UNIFORM(b) <=> [](b''=0).
WALL(b) <=> [](b-=-1 \/ b-=|X|+2 => b'=-b'-).

INIT(X[1],0,1), {INIT(X[i],i*2,0) | i in {2..|X|}},
{(UNIFORM(X[i]), UNIFORM(X[j])) << COLLISION(X[i],X[j])
                                  | i in {1..|X|-1}, j in {2..|X|}},
{UNIFORM(X[i]) << WALL(X[i]) | i in {1..|X|}}.
```

図 5.2　リスト記法を利用した 1 次元ビリヤードのプログラム

図 3.3 と図 5.2 の違いを以下に示す。

- 1 行目に各ボールの位置を表す変数のリストを定義する
- 制約 WALL の定義における条件部の値が 10 から |X|+2 に変更
  - 壁の位置を固定（10 に）するとボールの数が増えた時にボールが壁の外に出てしまうため，ボールの個数 |X| に応じた値に変更している
- 2 つ目から 4 つ目のボールに関する初期値の制約の宣言をリストを用いた記法に変更
  - リストの内包記法で変数 i を用いることによって X[2] から X[4] に対応する初期値制約を宣言している

- 等速運動の制約 UNIFORM とボール同士の衝突の制約 COLLISION の優先度の宣言をリストを用いた記法に変更
  - 変数 i と j を用いてボールの全組み合わせについて UNIFORM と COLLISION の優先度の宣言をしている。
- 等速運動の制約 UNIFORM とボールと壁との衝突の制約 WALL の優先度の宣言をリストを用いた記法に変更

リスト記法を導入する前の記法（図 3.3）ではボールの数を n とすると初期値制約 INIT を n 個，ボール同士の衝突の制約 COLLISION とボールの等速運動の制約 UNIFORM の優先度の宣言を ${}_nC_2$ 個，ボールと壁との衝突の制約 WALL とボールの等速運動の制約 UNIFORM の優先度の宣言を n 個記述する必要がある。一方リスト導入後の記法（図 5.2）ではボールの数を n とすると最初に定義するボールの位置を表す変数のリストを {x1..xn} とするだけでボール n 個の 1 次元ビリヤードのプログラムを記述できるため，初期値制約 INIT を 1 個，INIT に関するリストを 1 個，ボール同士の衝突の制約 COLLISION とボールの等速運動の制約 UNIFORM の優先度の宣言に関するリストを 1 個，ボールと壁との衝突の制約 WALL とボールの等速運動の制約 UNIFORM の優先度の宣言のリストを 1 個記述すればモデルを表すことができる。

5.3 評価

図 3.3 のプログラムを導入したリスト記法を用いて記述したものはリスト記法を用いた HydLa プログラムの例で示した図 5.2 である。図 3.3 以外のプログラムとして図 3.4 と図 3.5 のプログラムを導入したリスト記法を用いて記述した結果をそれぞれ図 5.3 と図 5.4 に示す。

リスト記法を導入することによるオブジェクト数 n 個のプログラムを記述するために必要な制約の記述数の変化を表 5.1 に示す。ここでの必要な制約の記述数は制約の優先度の宣言部に出現する制約の数のことを表し，同じ制約が複数回記述されている場合には記述回数分だけ数え上げるものとする。またリストの内包記法で記述された制約があった場合には制約の部分のみをカウント対象とする。

例えば図 3.3 のプログラムでは UNIFORM(x1) は 4 個記述されているが，これは 4 つと数え上げ，総成約数は 30 個であると数える。図 5.2 のプログラムでは {INIT(X[i],i*2,0) | i in {2..|X|}}の部分は 1 つと数え，総成約数は 7 個であると数える。

```
X := {x1..x4}.

INIT(x,x0,v0) <=> x = x0 /\ x' = v0 /\ x'' = 0.
CONSTANT <=> [](distance = 50) /\ [](maxV = 30) /\ [](minV = 10).
DISTANT(x1,x2) <=>
    [](x1'- < maxV- /\ x2- - x1- >= distance * 6/5 => x1'' = 5).
NEAR(x1,x2) <=>
    [](x1'- > minV- /\ x2- - x1- <= distance * 4/5 => x1'' = -5).
UNIFORM(x) <=> [](x'' = 0).
COLLISION(x1,x2) <=> [](x1- = x2- => [](x1' = 0 /\ x2' = 0)).

CONSTANT.
{ INIT(X[i],100*i,20) | i in {1..|X|-1} }, INIT(X[|X|],100*|X|,19),
{ UNIFORM(X[i]) <<
    ( DISTANT(X[i],X[i+1]), NEAR(X[i],X[i+1]) ) <<
        (COLLISION(X[i-1],X[i]), COLLISION(X[i],X[i+1]))|
            i in {2..|X|-1} },
UNIFORM(X[1]) << (DISTANT(X[1],X[2]), NEAR(X[1],X[2])) <<
    COLLISION(X[1],X[2]),
UNIFORM(X[|X|]) << COLLISION(X[|X|-1],X[|X|]).
```

図 5.3 リスト記法を用いた自動車の車間制御のプログラム

```
Xs := {x1..x4}.

INIT(x,xinit,vinit) <=> x = xinit /\ x' = vinit.
CONSTANT <=> [](inflow = |Xs|).
LINEAR(x) <=> [](x'' = 0).
INFLOW(x1,r1, outflow) <=>
    [](x1- = r1- => x1' = inflow - outflow).
NO_INFLOW(x1,r1,x2, outflow) <=>
    [](x1- = r1- & x2'- = inflow - outflow => x2' = -outflow).

INIT(Xs[1], 1, inflow - 1), CONSTANT,
{ INIT(Xs[i], 2 + i/2, -1-((i-1)/17) ) | i in {2..|Xs|} }.
{ LINEAR(Xs[i]) << INFLOW(Xs[i], 1, 1+((i-1)/17) ) | i in {1..|Xs|} }.
{ LINEAR(Xs[j]) << NO_INFLOW(Xs[i], 1, Xs[j], 1+((j-1)/17)) |
    i in {1..|Xs|}, j in {1..|Xs|}, i =!= j }
```

図 5.4 リスト記法を用いた水槽の水位制御のプログラム

```
X := {x1..x4}.

INIT(b,b0,vb0) <=> b=b0 & b'=vb0.
COLLISION1(b1,b2) <=> [](b1-=b2- => b1'=b2'- & b2'=b1'-).
COLLISION2(b1,b2) <=> [](b1-=b2- => b1'=4/5*b2'- & b2'=4/5*b1'-).
UNIFORM(b) <=> [](b''=0).
WALL(b) <=> [](b-=-1 | b-=2*|X| => b'=-b'-).

INIT(X[1],0,1),
{ INIT(X[i], 2*i-2, 0) | i in {2..|X|} },
{ UNIFORM(X[i]) << WALL(X[i]) | i in {1..|X|} },
{ (UNIFORM(X[i]), UNIFORM(X[j])) <<
    (COLLISION1(X[i],X[j]), COLLISION2(X[i],X[j])) <<
       $TRUE | i in {1..|X|-1}, j in {i+1..|X|} }.
```

図 5.5 リスト記法を用いた非決定的なビリヤードのプログラム

表 5.1 リスト記法の導入によるオブジェクト数 n 個のプログラムの制約の記述量の変化

モデル	総制約数（導入前）	総制約数（導入後）
ビリヤード（図 2.1）	$3(n^2+n)/2$	7
自動車の車間距離制御（図 2.2）	$6n-3$	14
水槽の水位制御（図 2.3）	$2n^2+n+1$	7
非決定ビリヤード（図 2.1）	$2n^2+2n$	9

表 5.1 を見るとすべての例題において記述量がオブジェクト数 n に依存する値から定数まで減っていることが分かる。図 3.3 のプログラムと図 5.2 のプログラムを比較すると，$n-1$ 個の INIT，${}_nC_2$ 個の UNIFORM と COLLISION の優先度定義および n 個の UNIFORM と WALL の優先度定義がそれぞれ 1 つの内包記法を用いたリストで表されていることが分かる。図 5.2 におけるリストの内包記法で表された INIT は 2 個目のボールから n 個目のボールの初期位置および初期速度の制約を表している。それらのボールの初期速度は 0 であり，初期位置はボールの番号 i に対し $2i$ で表される。仮に各ボールの初期位置にこのような規則性がない場合，`{INIT(X[i],i*2,0) | i in {1..|X|}}`の記述は `{INIT(X[i],P[i],0) | i in {1..|X|}}`（ただしリスト P は図 5.2 における 1 行目のリスト X の定義と同様に `P := {2,5,7,4,8,1}`のような具体値による定義が必要である）のように記述するか，すべての制約を別々に記述する必要がある。すべての制約を別々に

記述した場合には $n-1$ 個の INIT の記述が必要であり，リスト P を用いた記法であっても P の定義であるリストの要素を $n-1$ 個記述しなければならない。

従って n 個のオブジェクトを定数個の制約の記述で表すためにはそれらのオブジェクトがある規則に従っている必要があると考えられる。複数のオブジェクトが存在するモデルにおいて各オブジェクト間に規則性が全くない問題は多くないと考えられるため，複数のオブジェクトが存在するモデルの多くはリストの内包記法によって記述量を減らすことが可能であると考えられる。例えば図 3.3 において各ボールの初期位置に規則性がない場合でも，各ボールの反発係数は等しいものと考えれば UNIFORM や COLLISION，WALL は定数個の記述で済むことになる。

第 6 章

動的な解候補モジュール集合の導出

HydLa プログラムの解軌道を求めるためには解候補モジュール集合の中で極大無矛盾集合となる集合を求める必要がある．

一般に Required でないモジュールの数が多いほど解候補モジュール集合の数が多くなる．Required でないモジュールはモデルの例外の存在するデフォルト動作に関する制約であるので，例外の存在するデフォルト動作が多い問題は解候補モジュール集合が多い問題となる．典型的なデフォルト動作が多い問題はオブジェクトの数が多い問題である．

例えば図 2.1 で示した 1 次元上のビリヤードの例ではボールの数 n に対し解候補モジュール集合の数は 2^n 個となる．しかしシミュレーションに必要な解候補モジュール集合は極大無矛盾な集合のみであり全解候補モジュール集合を算出する必要はない．

HyLaGI は極大無矛盾集合を求めるためにシミュレーションの初めに全解候補モジュール集合を算出している（図 4.1 の 1 行目）．解候補モジュール集合の総数が多い HydLa プログラムを取り扱う場合，全解候補モジュール集合を生成する処理がボトルネックとなってしまう．そこで本章では HyLaGI のシミュレーション中に必要となった解候補モジュール集合のみを動的に生成する手法を示す．

6.1 解候補モジュール集合の無矛盾性

この節では解候補モジュール集合の無矛盾性および無矛盾性についての定理を示す．

解候補モジュール集合はモジュールの集合であり，モジュールは制約から成っている．モジュール集合が無矛盾であるということはモジュール集合内のモジュールに出現する各変数について，その集合の要素の連言を真とする値が存在することと同値である．従ってあるモジュール集合 MS が $\{M_1, M_2, \cdots, M_n\}$ で表されるとき，MS の無矛盾性は式 6.1

で表現することができる．式 6.1 内に出現する $Variables(MS)$ は MS 内のモジュールに出現する変数の集合を表す．

$$\exists V \in Variables(MS)(M_1 \wedge M_2 \wedge \cdots \wedge M_n) \tag{6.1}$$

次に解候補モジュール集合の無矛盾性についての定理を述べる．

定理 6.1.1. $MS1 \subseteq MS2$ であり，かつ矛盾する $MS1$ が存在することと $MS2$ が矛盾することは同値である．

証明. 2 つのモジュール集合 $MS1$，$MS2$ を考える．これらのモジュール集合には包含関係 $MS1 \subseteq MS2$ が成り立っているものとする．$MS1$ を $\{M_1, M_2, \cdots, M_n\}$，$MS2$ を $\{M_1, M_2, \cdots, M_n, \cdots, M_m\}$ と表す．$MS1$ 内のモジュールに出現する変数の集合を $V1$，$MS2$ 内のモジュールに出現する変数の集合を $V2$ とする．$MS1$ の無矛盾性は式 6.2，$MS2$ の無矛盾性は式 6.3 で示される．

$$\exists V \in V1(M_1 \wedge M_2 \wedge \cdots \wedge M_n) \tag{6.2}$$
$$\exists V \in V2(M_1 \wedge M_2 \wedge \cdots \wedge M_n \cdots \wedge M_m) \tag{6.3}$$

まずは $MS1 \subseteq MS2$ であり $MS1$ が矛盾するならば $MS2$ が矛盾することを示す．このとき $MS1$ が矛盾していると仮定すると式 6.2 が false となる．

$MS1$ が矛盾する（式 6.2 が false）という仮定から $V1$ 内に $M_1 \wedge M_2 \wedge \cdots \wedge M_n$ を満たすことの出来ない変数 V_f が存在する．$MS1$ は $MS2$ に包含されているため $V1 \subseteq V2$ が成り立つことから $V2$ 内にも V_f が存在する．V_f は $M_1 \wedge M_2 \wedge \cdots \wedge M_n$ を満たすことができないため，その論理式にいくつかのモジュールを論理積を用いて追加した V_f は $M_1 \wedge M_2 \wedge \cdots \wedge M_n \wedge \cdots \wedge M_m$ も満たすことができない．従って式 6.2 が false になる場合式 6.3 も false となるため，矛盾したモジュール集合 $MS1$ を包含するモジュール集合 $MS2$ は矛盾する．

次に $MS2$ が矛盾するならば $MS1 \subseteq MS2$ であり, かつ矛盾する $MS1$ が存在することを示す．$MS2$ が矛盾するとき $V2$ の内の変数に満たすことのできない論理式 $M_i \wedge \cdots \wedge M_j$ が存在する．このとき満たすことができない各モジュール $M_i, \cdots, M_j$ を要素とする集合 MS_i が存在する．MS_i は矛盾する集合であり，$MS_i \subseteq MS2$ を満たすため，$MS2$ が矛盾するとき，$MS1 \subseteq MS2$ であり矛盾する $MS1$ が存在する．

以上より定理 6.1.1 が成立することが示された． □

6.2 動的な解候補モジュール集合導出手法

HyLaGI は解軌道を計算するために解候補モジュール集合内の極大無矛盾集合を求めている．従って矛盾する集合や極大無矛盾集合に包含される集合の計算を行う必要はない．そこで動的な解候補モジュール集合の導出では未導出な解候補モジュール集合のうち矛盾するか否かが不明な集合であり極大無矛盾集合に包含されない集合の中で包含関係において極大な集合を生成することを考える．

図 6.1 に動的な解候補モジュール集合導出アルゴリズムを示す．

Input: 解候補モジュール集合の集合 MSS, 矛盾したモジュール集合 IS, 極大無矛盾集合の集合 $MCSS$
Output: 解候補モジュール集合の集合 N

```
 1: N := ∅
 2: for all MS∈MSS do
 3:   if MS⊇IS then
 4:     for all I∈IS do
 5:       if ¬∃W∈IS(W<<I ∧ ¬(I << W)) ∧ ¬IsRequired(I) then
 6:         S := MS \ { M | M<<I ∨ M=I }
 7:         if ¬∃MCS∈MCSS (S⊂MCS) then
 8:           N := N∪{S}
 9:         end if
10:       end if
11:     end for
12:   else
13:     N := N ∪ {MS}
14:   end if
15: end for
16: return N
```

図 6.1 動的な解候補モジュール集合導出アルゴリズム

以下，図 6.1 の重要部分の説明を行う．2 行目のループで生成済みであり無矛盾性が未判定である解候補モジュール集合の全要素について処理を行う．3 行目から 6 行目にかけての処理で，矛盾した集合を包含する解候補モジュール集合から式 6.4 を満たすモジュール I について式 6.5 を満たすモジュール M をすべて取り除くことによって新たな解候補モジュール集合を生成している．

$$\neg\exists W \in IS(W << I \wedge \neg(I << W)) \wedge \neg IsRequired(I) \tag{6.4}$$

$$M << I \vee M = I \tag{6.5}$$

7 行目から 8 行目にかけては，生成された解候補モジュール集合が極大無矛盾集合に包含されていない場合には結果として返す解候補モジュール集合の集合 N に生成されたモジュール集合を追加する．12 行目から 13 行目では矛盾した集合を包含していない集合をそのまま N に追加している．

6.3 HyLaGI のアルゴリズムの変更点

動的な解候補モジュール集合の導出を行うために HyLaGI の実行アルゴリズムに変更を加える必要がある．図 4.1 を変更したものを図 6.2 に，図 4.2 を変更したものを図 6.3 に示す．

Input: HydLa プログラム $HydLaProgram$，シミュレーション終了時刻 $MaxT$

1: M_{all} := $GetMaxModuleSet(HydLaProgram)$
2: V := $GetVariables(HydLaProgram)$
3: T := 0; S := = true; CP := true; E := $\emptyset$
4: **while** $T <_{CP} MaxT$ **do**
5: // PP
6: S := $SubstituteMinTime(S,T)$
7: $(S,CP,E,\_,\_,\_)$:= $CalculateMCS(S,\{M_{all}\},E,CP,T,CheckConsistencyPP)$
8: **if** S = false **then**
9: break
10: **end if**
11: (S,CP) := $AddParameters(S,CP,V)$
12: // IP
13: (S,CP,E,M,A_+,A_-) := $CalculateMCS(S,\{M_{all}\},E,CP,T,CheckConsistencyIP)$
14: S := $SolveDifferentialEquation(S)$
15: **if** S = false $\vee \neg$ $IsUnique(S,V)$ **then**
16: break
17: **end if**
18: $(MinT,CP)$:= $GetElement(CompareMinTime($
19: $\{FindMinTime(S \wedge CP \Rightarrow g) \mid (g \Rightarrow c) \in A_-\}$
20: $\cup \{FindMinTime(S \wedge CP \Rightarrow \neg\, g) \mid (g \Rightarrow c) \in A_+\}$
21: $\cup \{FindMinTime(S \wedge CP \wedge M_-) \mid$
22: $M_- \in (M_{all} \setminus M)\}$
23: $\cup \{FindMinTime(S \wedge CP \wedge \neg\, M_+) \mid M_+ \in M\}$
24: $\cup \{(MaxT - T, \text{true})\}))$
25: T := $MinT + T$
26: **end while**

図 6.2 動的な解候補モジュール集合の生成を利用した HyLaGI の非決定実行アルゴリズム

図 4.1 から図 6.2 への変更点を示す．図 4.1 では最初に全解候補モジュール集合を算出

Input: 制約ストア S，解候補モジュール集合の集合 MS，展開済み always 制約の集合 E，記号定数に関する条件 CP，現在時刻 T，無矛盾性判定関数 $CheckConsistency$
Output: 制約ストア，記号定数に関する条件，展開済み always 制約の集合，成り立つガード条件の集合，成り立たないガード条件の集合

```
1: while MS ≠ ∅ do
2:   M := GetMaximalSet(MS)
3:   if T > 0 then
4:     EliminateNotAlways(M)
5:   end if
6:   (S_tmp,E_tmp,CP,A_-,A_+) :=
7:    CalculateClosure(S,M,CP,E,CheckConsistnecy)
8:   if S_tmp ≠ false then
9:     return (S_tmp,CP,E_tmp,M,A_-,A_+)
10:  end if
11:  IM := GetInconsistentMS(M)
12:  for all I ∈ IM do
13:    MS := GenerateNewMSS(MS,I,_)
14:  end for
15: end while
16: return (false,CP,E,∅,∅,∅)
```

図 6.3 動的な解候補モジュール集合の生成を利用した $CalculateMCS$

していたが，図 6.2 では最初に HydLa プログラム内で宣言されたモジュールすべてを要素とする解候補モジュール集合 M_{all} を求めている．すべてのモジュールを要素として持つモジュール集合は式 3.2 と式 3.3 の後件をすべてのモジュールについて満たすため解候補モジュール集合となる．この変更に伴い $CalculateMCS$ に引数として渡していた全解候補モジュール集合のリストの部分を M_{all} のみを要素とする集合を渡すように変更する．

図 4.2 から図 6.3 への変更点を示す．HyLaGI の非決定実行アルゴリズムに変更が加わったことにより，入力として受け取る解候補モジュール集合の集合 MS が全解候補モジュール集合を要素とする集合から全モジュールを要素とする解候補モジュール集合のみを要素とする集合となっている．変更前のアルゴリズムでは 1 行目で MS 内の全要素について計算を行っていたが，変更後は動的に解候補モジュール集合を生成するため MS が空集合でない間，計算を続ける．変更後のアルゴリズムの 11 行目に出現する関数 $GetInconsistentMS(M)$ はモジュール集合 M の中で矛盾した集合を得る関数である．12 行目，13 行目では 11 行目で得た矛盾した集合を利用して新たな解候補モジュール集合の生成を行う．矛盾した集合を得る手法については文献 [12] を参照してほしい．極大無矛盾集合は同時に複数個存在する場合があり，全解探索ではそのすべてを求める．全解探索を行う場合には図 6.3 において 27 行目で返す結果を保持しておき，保持されているすべ

ての M（27 行目の第 4 要素）を 23 行目の第 3 引数に渡し，29 行目で保持しておいた結果をすべて返せばよい．

6.4　動的な解候補モジュール集合の導出例

動的な解候補モジュール集合導出の例を図 2.1 の 1 次元ビリヤードの例を用いて示す．1 次元ビリヤードの HydLa プログラムは図 3.3 に示されている．

図 3.3 の HydLa プログラムの全解候補モジュール集合における集合の包含関係による半順序をハッセ図で表したものを図 6.4 に示す．　図 6.4 のハッセ図の各ノードが解候補

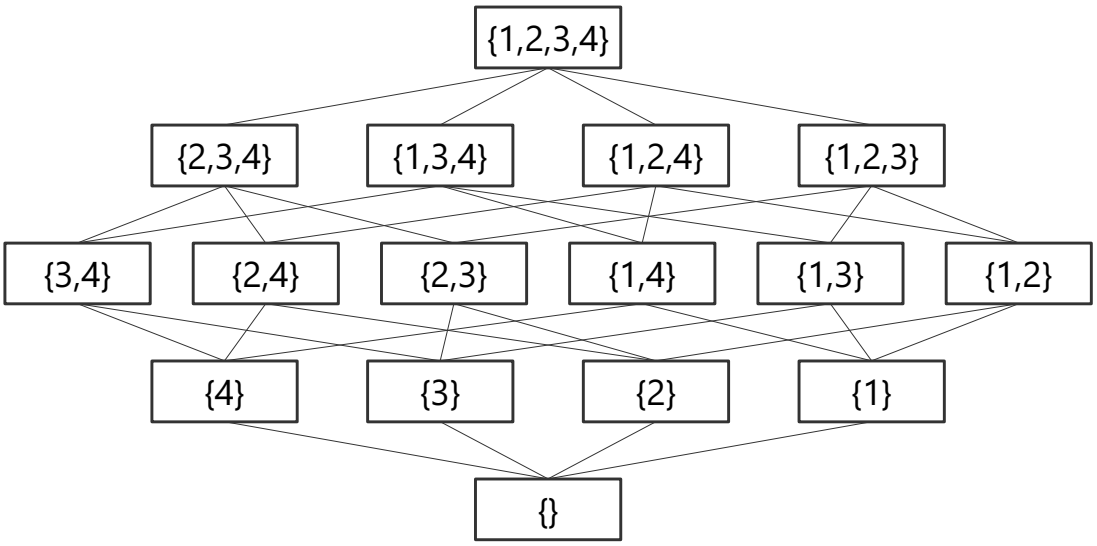

図 6.4　図 3.3 の HydLa プログラムの全解候補モジュール集合のハッセ図

モジュール集合であり，集合の要素となっている数字 i は UNIFORM(xi) を表している．INIT や COLLISION，WALL はすべて Required なモジュールであるため全解候補モジュール集合の要素であるので図 6.4 では省略している．以降数字のみが要素のモジュール集合を記述したときは各数字 i に対応する UNIFORM(xi) と Required なモジュールがすべて要素となっている集合を表すものとする．

図 3.3 の HydLa プログラムにおいて 1 つ目のボール（x1）と 2 つ目のボール（x2）が衝突した瞬間の極大無矛盾集合を算出する例を示す．衝突によって 1 つ目のボールの速度が変化するため 1 つ目のボールの等速運動の制約 UNIFORM(x1) と衝突の制約 COLLISION(x1,x2) が矛盾する．また 2 つ目のボールについても同様にして UNIFORM(x2) と COLLISION(x1,x2) が矛盾する．従って 1 つ目のボールと 2 つ目のボールが衝突する瞬

間には {UNIFORM(x1),COLLISION(x1,x2)} および {UNIFORM(x2),COLLISION(x1,x2)} を包含しない集合が無矛盾な集合となる．図 6.4 内の無矛盾な集合のみをピックアップした図を図 6.5 に示す．　図 6.5 内の集合のうち極大な {3,4} が極大無矛盾集合となる．

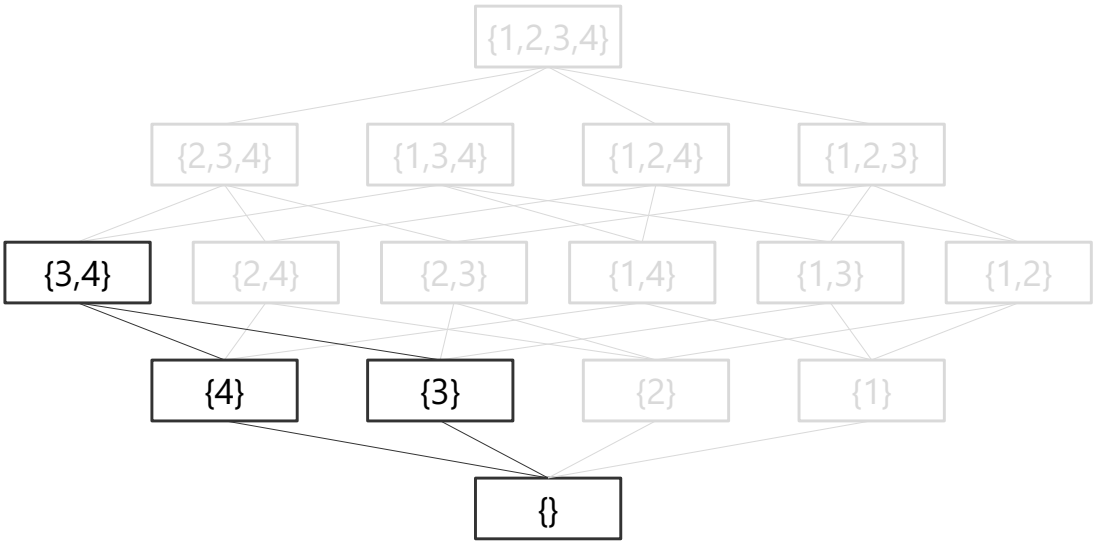

図 6.5　無矛盾な解候補モジュール集合

図 6.6 が動的な解候補モジュール集合導出における初期状態となる．初期状態では全モジュールを要素とする集合 {1,2,3,4} のみが導出されている．

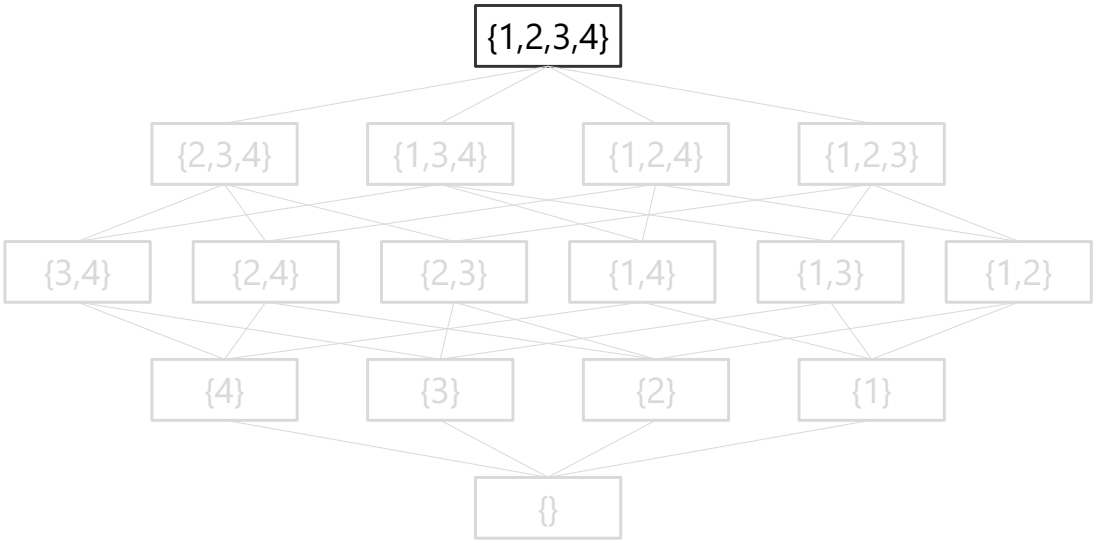

図 6.6　動的な解候補モジュール集合導出の初期状態

初期状態の {1,2,3,4} の無矛盾性判定を行い，{UNIFORM(x1),COLLISION(x1,x2)} および {UNIFORM(x2),COLLISION(x1,x2)} が矛盾した集合として特定できたとする．あるモジュール集合の無矛盾性判定をしたときに，どのモジュールが矛盾するかを判定する方法については文献 [12] を参照してほしい．まずは {UNIFORM(x1),COLLISION(x1,x2)} を矛盾する集合として与える．図 6.1 のアルゴリズムの入力の MSS の初期状態の要素は {1,2,3,4} のみである．図 6.1 の 6 行目の条件を満たすモジュール I は UNIFORM(x1) である．M<< UNIFORM(x1) となすモジュール M が存在しないため図 6.1 の 7 行目で新たに生成される集合は $\{1,2,3,4\} \setminus \{1\} = \{2,3,4\}$ である．従って図 6.1 のアルゴリズムを 1 度適用した後のハッセ図は図 6.7 となる．

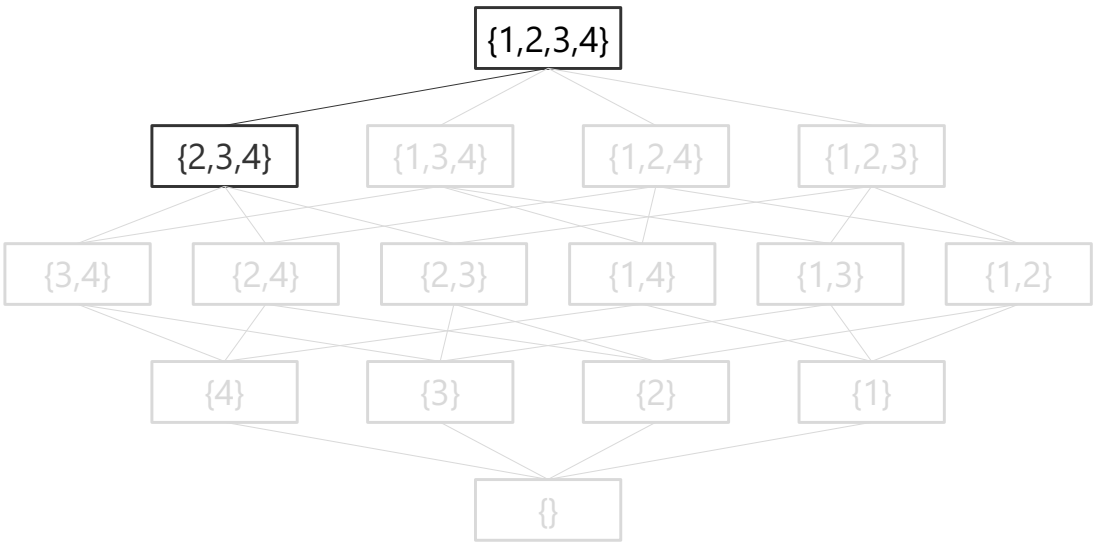

図 6.7　動的な解候補モジュール集合導出の経過

次に矛盾した集合として {UNIFORM(x2),COLLISION(x1,x2)} を与える．このときの MSS の要素は {2,3,4} のみである．1 度目の適用と同様にして {3,4} が生成される．そのときのハッセ図は 6.8 となる．{3,4} はすべての矛盾した集合を包含しない集合のうち極大なものなので極大無矛盾集合となる．

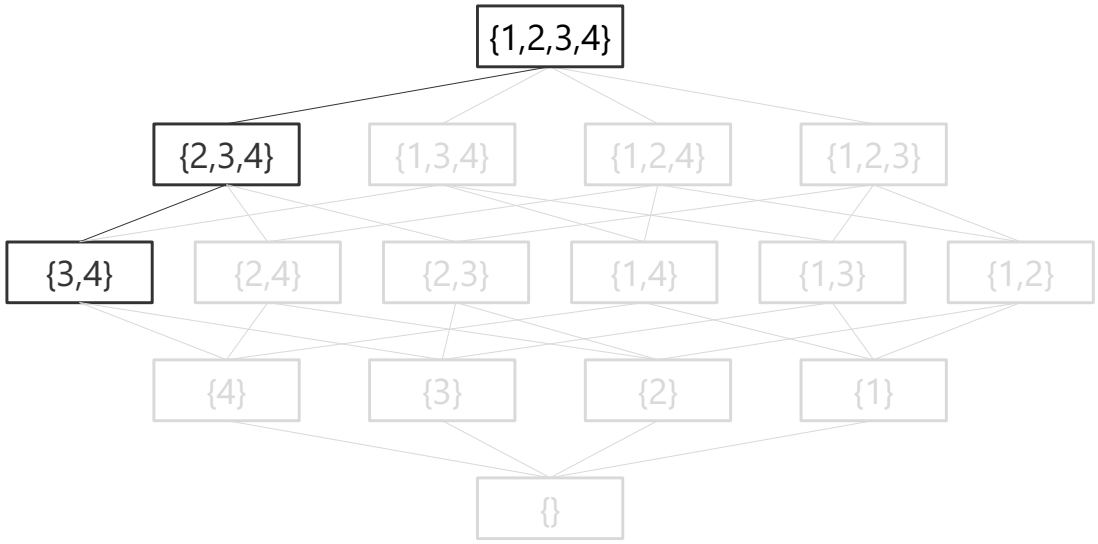

図 6.8　動的な解候補モジュール集合導出の最終結果

6.5　動的な解候補モジュール集合生成アルゴリズムの正当性の証明

この節では図 6.1 のアルゴリズムの正当性を証明する．まずは証明に使用する式を以下に示す．式 6.6，式 6.7 は解候補モジュール集合 MS が満たさなければならない式である．

$$\forall M_1, \forall M_2 (M_1 << M_2 \land M_1 \in MS \Rightarrow M_2 \in MS) \tag{6.6}$$
$$\forall M (IsRequired(M) \Rightarrow M \in MS) \tag{6.7}$$

式 6.8，式 6.9 は図 6.1 のアルゴリズムに関係する式である．図 6.1 のアルゴリズムは入力された解候補モジュール集合から式 6.8 を満たす矛盾する集合 IS 内のモジュール I について式 6.9 を満たすモジュール M をすべて取り除くことで新たな集合を生成する．

$$\neg \exists W \in IS (W << I \land \neg (I << W)) \land \neg IsRequired(I) \tag{6.8}$$
$$M << I \lor M = I \tag{6.9}$$

図 6.1 のアルゴリズムの正当性を証明するために補題を 1 つ示す．

補題 6.5.1. 図 6.1 の解候補モジュール集合導出のアルゴリズムは解候補モジュール集合を生成する

証明. モジュールの集合が解候補モジュール集合であるためには式 6.6 と式 6.7 を満たしている必要がある．生成された集合が式 6.6 と式 6.7 を満たすことを示す．

まずは生成された集合が式 6.6 を満たすことを示す．生成元の集合は解候補モジュール集合であるため，式 6.6 を満たす．新たな集合は生成元の集合からモジュール M を取り除くことで生成される．モジュール M が取り除かれるとき $M \in MS$ が false となる．この変化によって式 6.6 を満たさなくなる場合は取り除くモジュール M が式 6.10 を満たすときのみである．

$$\exists W(W << M \wedge W \in MS) \tag{6.10}$$

これは式 6.6 の $M_2 \in MS$ が false になり，その前件が true となることに対応する．除かれるモジュールは式 6.9 を満たすすべてのモジュール M である．$M_1 = I$ を満たす M_1 に着目すると，$M_2 << I$ を満たすモジュール M_2 もすべて除かれるため，$M_2 \in MS$ は false となり式 6.10 は満たされない．次に $M_1 << I$ を満たす M_1 に着目する．記号 << は推移律を満たすため，$M_2 << M_1$ を満たすモジュール M_2 は $M_2 << I$ も必ず満たす．従って $M_2 << M_1$ を満たすモジュール M_2 はすべて除かれるため $M_2 \in MS$ が false となるため式 6.10 が満たされない．以上より集合を生成する際に取り除くすべてのモジュールは式 6.6 を満たさないため，生成された集合は式 6.6 を満たす．

次に生成された集合が式 6.7 を満たすことを示す．生成元の集合は解候補モジュール集合であるため式 6.7 を満たす．生成された集合が式 6.7 を満たさなくなる場合は取り除くモジュール M のうち $IsRequired(M)$ を true とするモジュールが存在する場合である．$M_1 << I$ を満たすモジュール M_1 は $IsRequired(M_1)$ を false となる．$M_1 = I$ を満たすモジュール M_1 は式 6.8 より I が $\neg IsRequired(I)$ が満たされている必要があるため，$IsRequired(M_1)$ は false となる．従って取り除くすべてのモジュール M が $IsRequired(M)$ を false とするため，生成された集合は式 6.7 を満たす．

以上より，生成されたすべての集合が式 6.6 と式 6.7 を満たすため生成される集合は解候補モジュール集合となる．　□

次に図 6.1 の正当性の証明を行う．図 6.1 のアルゴリズムの正当性は定理 6.5.1 で示される．

定理 6.5.1. 図 5.1 の解候補モジュール集合導出アルゴリズムは解候補モジュール集合であり極大無矛盾な集合が存在する場合，必ずその集合を導出する

証明. 図 6.1 の初期入力はすべてのモジュールを要素とする集合である．従って最初に生成される集合 GS は式 6.8 を満たす矛盾した集合 IS の要素 I について式 6.9 を満たすす

べてのモジュール M 以外のモジュールを要素とする集合である．

GS にモジュールを追加することで GS を包含する解候補モジュール集合のうち極小の集合を生成することを考える．GS に追加できるモジュールは GS の要素でないモジュールであり，そのようなモジュールは式 6.9 を満たすモジュールである．

M<< I を満たすモジュール M を GS に追加することを考える．M のみを GS に追加した場合には式 6.6 を満たすことができないため解候補モジュール集合になり得ないので I も同時に追加する必要があるが，I を GS に追加する際に M を追加しなくても式 6.6 を満たすため，その集合は GS を包含する極小な解候補モジュール集合ではない．

$M = I$ を満たすモジュール M を GS に追加することを考える．ここで出現する I は式 6.8 を満たすため，式 6.11 も満たす．

$$\neg\exists W \in IS(W << I \wedge \neg(I << W)) \tag{6.11}$$

式 6.11 は式 6.12 と同値である．

$$\forall W \in IS(\neg(W << I) \vee I << W) \tag{6.12}$$

式 6.12 内の $\neg(W << I)$ を満たすモジュール W は式 6.9 の M を W にした式を満たさないため GS に含まれるモジュールである．

式 6.12 内の $\neg(W << I)$ を満たさず，$I << W$ を満たすモジュール W は式 6.9 の M を W にした式を満たすため GS には含まれないが，GS に I のみを追加した場合には式 6.6 を満たさないため解候補モジュール集合になり得ないので I と同時に W も GS に追加する必要がある．逆に W のみを GS に追加しても式 6.6 を満たさないため I を同時に追加する必要がある．

以上より，GS を包含する極小の集合は矛盾の集合 IS の全要素を含んでいるため，GS は矛盾した集合 IS を包含しない極大の集合となる．定理 6.1.1 により IS を包含する解候補モジュール集合は矛盾するため，GS が包含しない解候補モジュール集合は IS を包含しており，定理 6.1.1 よりそれらの集合は矛盾することが分かる．生成した集合に他の矛盾した集合が含まれている場合には図 6.1 のアルゴリズムを繰り返し適用することによって，解候補モジュール集合内に矛盾集合を 1 つも包含しない集合が存在すれば，矛盾集合を 1 つも包含しない極大な集合を生成でき，定理 6.1.1 よりその集合が極大無矛盾集合となる．　□

6.6 評価

評価実験を行った実行環境を表 6.1 に示し，評価を行った例題についての説明を表 6.2 に示し，各例題の実行条件を表 6.3 に示す．

表 6.1 実行環境

OS	Ubuntu 14.04 LTS
CPU	Inter(R) Xeon(R) 2.50GHz, Quad-core × 2 (8CPUs)
Memory	8Gbyte
Backend	Mathematica 9.0

表 6.2 測定例題

例題	説明	プログラム
ビリヤード	1 次元上にボールが並ぶビリヤード	図 3.3
自動車の車間距離制御	前方の車との距離に応じて速度を制御	図 3.4
水槽の水位制御	各水槽の水位を水の流入によって制御	図 3.5
ビリヤード（非決定）	衝突時の反発係数が非決定的なビリヤード	図 3.6

表 6.3 例題の実行条件

例題	オブジェクト数	タイムアウト時間	フェーズ数
ビリヤード	4 個–300 個（ボール）	1800 秒	20
自動車の車間制御	4 個–300 個（自動車）	1800 秒	20
水槽の水位制御	4 個–300 個（水槽）	1800 秒	20
ビリヤード（非決定）	4 個–300 個（ボール）	3600 秒	6

解候補モジュール集合に関する処理は大きく 3 つに分けられる．

初期化 宣言されたモジュールの優先度をパースし，解候補モジュール集合を扱うクラスを初期化する

矛盾時　解候補モジュール集合 IS が矛盾した際に IS を包含する解候補モジュール集合を探索の候補から外し，次の解候補モジュール集合の計算を行う

無矛盾時　解候補モジュール集合 CS が無矛盾だった際に CS が包含する解候補モジュール集合を探索の候補から外し，次の解候補モジュール集合から外し，次の解候補モジュール集合の計算を行う（非決定実行時）

6.6.1　以前の手法で通常実行を行った結果に対する考察

解軌道が決定的なビリヤードの例題，車間制御の例題および水位制御の例題を以前の手法で実行したときの結果をそれぞれ図 6.9，図 6.10，図 6.11 に示す．

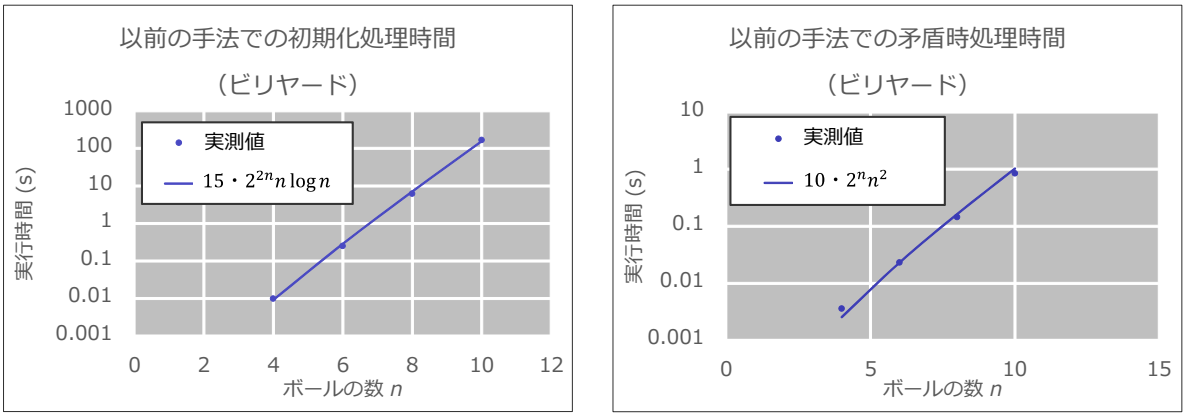

図 6.9　以前の解候補モジュール集合導出手法によるビリヤードの例題の処理時間

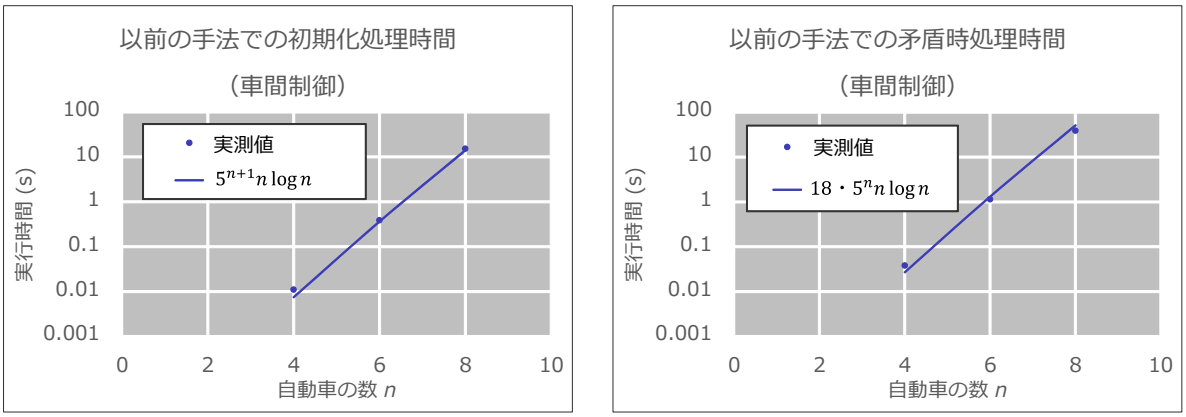

図 6.10　以前の解候補モジュール集合導出手法による車間制御の例題の処理時間

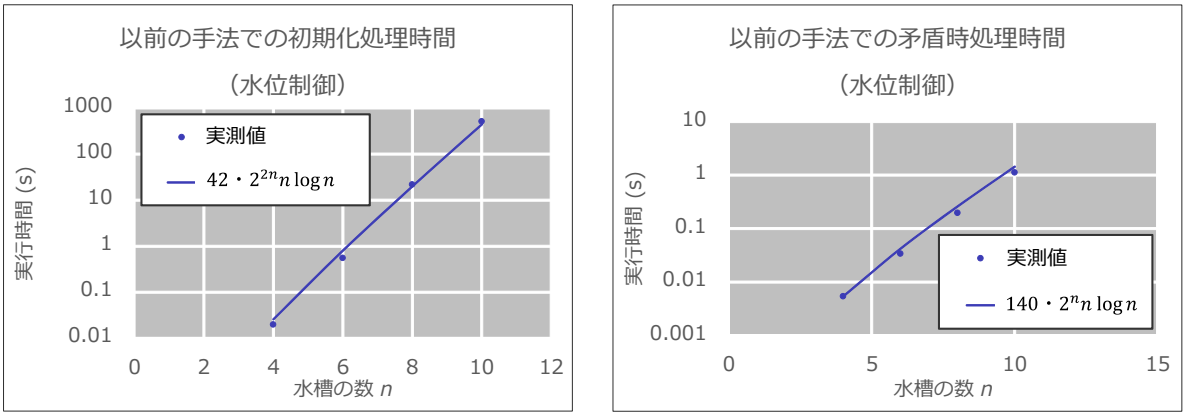

図 6.11　以前の解候補モジュール集合導出手法による水位制御の例題の処理時間

初期化処理に関する考察

まずは解候補モジュール関係の初期化処理について考察する．以前の解候補モジュール集合導出手法ではシミュレーション開始前にモジュールの優先度の宣言部を読み込みつつ解候補モジュール集合の導出を行う．例えば `A,(B<<C)` というモジュールの優先度の宣言があった場合，まず `B<<C` の処理により {C} と {B,C} が生成される．次に `A,(B<<C)` を処理することで {A,C} と {A,B,C} が生成される．`A,(B<<C)` の記述における解候補モジュール集合は {A,C} と {A,B,C} であるが，最終的な解候補モジュール集合を生成するまでに {C} と {B,C} を生成しているため，以前の手法による初期化処理において生成されるモジュール集合の数はそのプログラムにおける解候補モジュール集合の数と一致しない．全解候補モジュール集合を生成するアルゴリズムの詳細は文献 [10] を参照してほしい．

初期化処理で生成されるモジュール集合数およびモジュール集合生成時間の平均を図 6.12 に示す．

図 6.12 を見るとモジュール集合の生成数がビリヤードおよび水位制御の例題でおよそ $O(4^n)$，車間制御の例題でおよそ $O(5^n)$ 個となっている．ビリヤードおよび水位制御の例題の解候補モジュール数は 2^n であり，車間制御の例題の解候補モジュール集合数 5^n であるため，ビリヤードと水位制御の例題では解候補モジュール集合数の 2 乗の数のモジュール集合が，車間制御の例題においては解候補モジュール集合数に比例する数のモジュール集合が生成されていることが分かる．また各例題においてモジュール集合を 1 つ生成するためにかかる平均時間はおよそ $O(n\log n)$ であることが分かる．モジュール集合の生成では今までに得られた 2 つのモジュール集合を優先度に従ってマージすることに

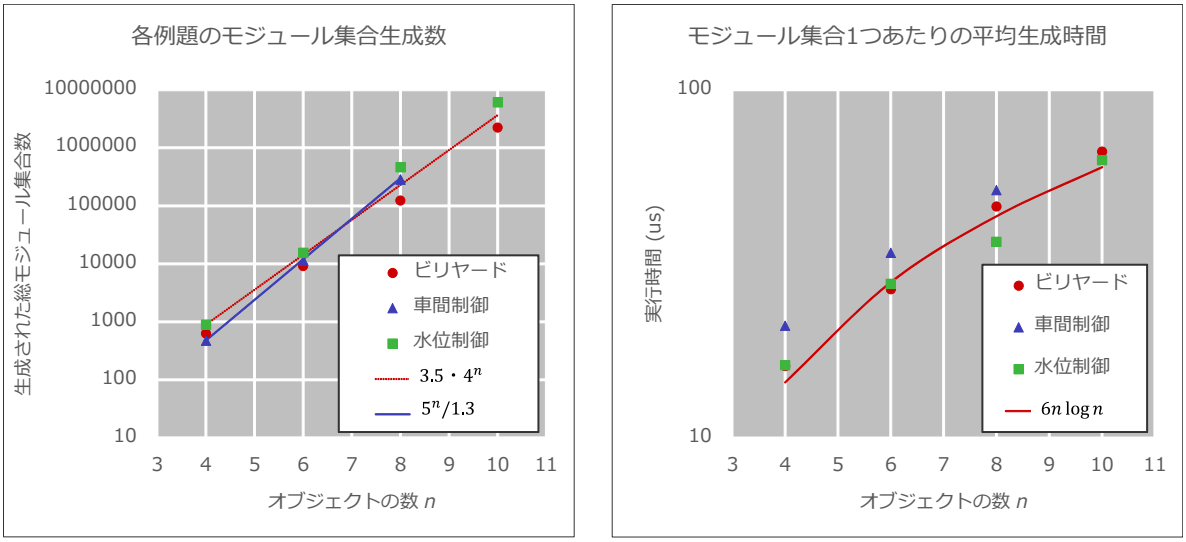

図 6.12　シミュレーション前に生成されるモジュール集合数

よって生成する．2 つのモジュール集合のマージはモジュール集合 MS_1 に別のモジュール集合 MS_2 を挿入することによって行われる．MS_1 に MS_2 を挿入するときの時間計算量はそれぞれの要素数を S_1，S_2 とすると $O(S_2 \log S_1)$ となるため，モジュール集合を 1 つ生成するためにかかる平均時間がおよそ $O(n \log n)$ となると考えられる．以前の手法における解候補モジュール集合の初期化処理のボトルネックはモジュール集合生成であるため，初期化処理は生成されたモジュール集合数 × モジュール集合の生成時間となり，各例題の初期化処理における時間計算量はビリヤード，水位制御が $O(4^n n \log n)$，車間制御が $O(5^n n \log n)$ となっていることが分かる．

矛盾時の処理に関する考察

次に解候補モジュール集合関係の矛盾時の処理について考察する．以前の解候補モジュール集合導出手法ではモジュール集合 IS が矛盾したときに全解候補モジュール集合から IS を包含するモジュール集合を除外する．矛盾時の処理において全解候補モジュール集合から除外された総モジュール集合数と 1 つのモジュール集合を除外するための平均時間を図 6.13 に示す．

図 6.13 と各例題の実行結果を比べると，実行結果の計算量は除外する解候補モジュール集合数と 1 つの解候補モジュール集合を除外するためにかかる平均時間の積の計算量と一致することが分かる．図 6.13 を見ると除外する解候補モジュール集合はそれぞれ全解候補モジュール集合数と比例していることが分かる．ビリヤードと水位制御の問題はモ

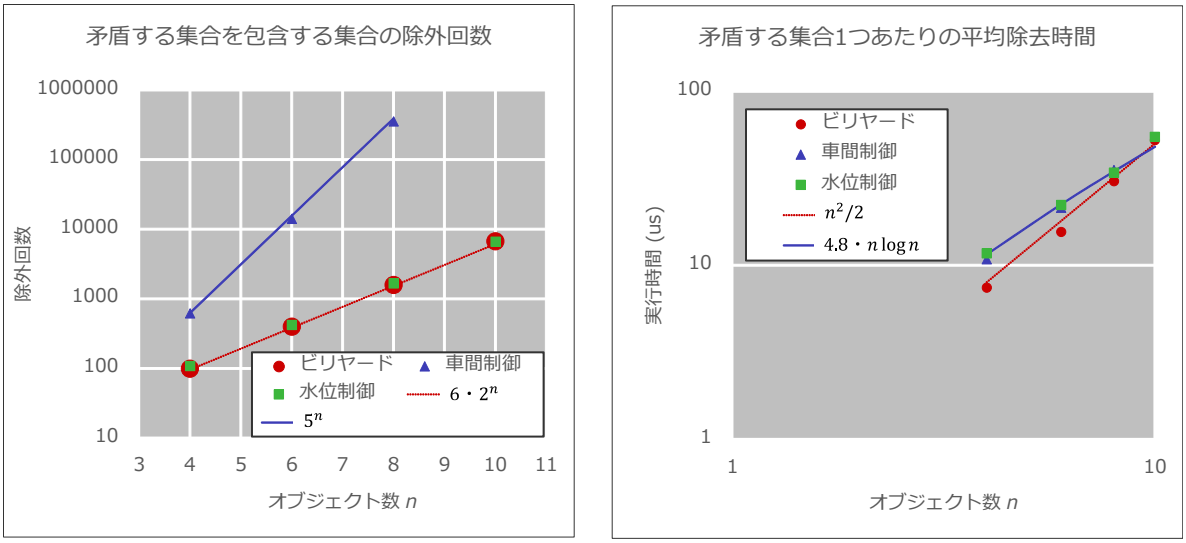

図 6.13　シミュレーション前に生成されるモジュール集合数

ジュール数および解候補モジュール集合数はオブジェクト数 n に対し，それぞれ $\mathrm{O}(n^2)$ と $\mathrm{O}(2^n)$ で一致しているため 1 つの解候補モジュール集合を除外するための平均時間の時間計算量は一致すると考えられるが図 6.13 を見ると一致していない．解候補モジュール集合の除去は解候補モジュール集合数 n に対して $\log n$ の時間計算量で計算が可能であるため，解候補モジュール集合数が 2^n であるビリヤードおよび水位制御，解候補モジュール集合数が 5^n である車間制御の例題において解候補モジュール集合の除去にかかる時間の計算量は $\mathrm{O}(n)$ になるはずである．図 6.13 の縦軸の単位は μs であるため最大でも約 100μs であり，十分なデータがないため正確な時間計算量を知るためにはさらにオブジェクト数を増やして測定する必要があると考えられる．

6.6.2　動的な導出手法で通常実行を行った結果に対する考察

解軌道が決定的なビリヤードの例題，車間制御の例題および水位制御の例題を動的な導出手法を用いて実行したときの結果をそれぞれ図 6.14，図 6.15，図 6.16 に示す．

初期化処理に関する考察

まずは解候補モジュール集合の初期化処理について考察する．動的なモジュール集合導出手法の初期化処理ではモジュールの優先度の宣言を読み込みその記述からモジュール間の優先度データを生成し，優先度の記述に出現するすべてのモジュールを要素とする集合

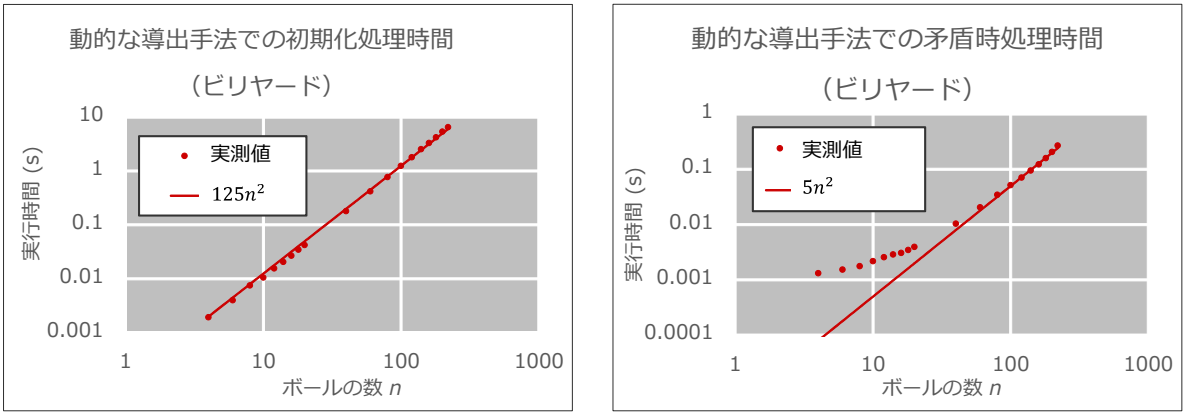

図 6.14　動的な解候補モジュール集合導出手法によるビリヤードの例題の処理時間

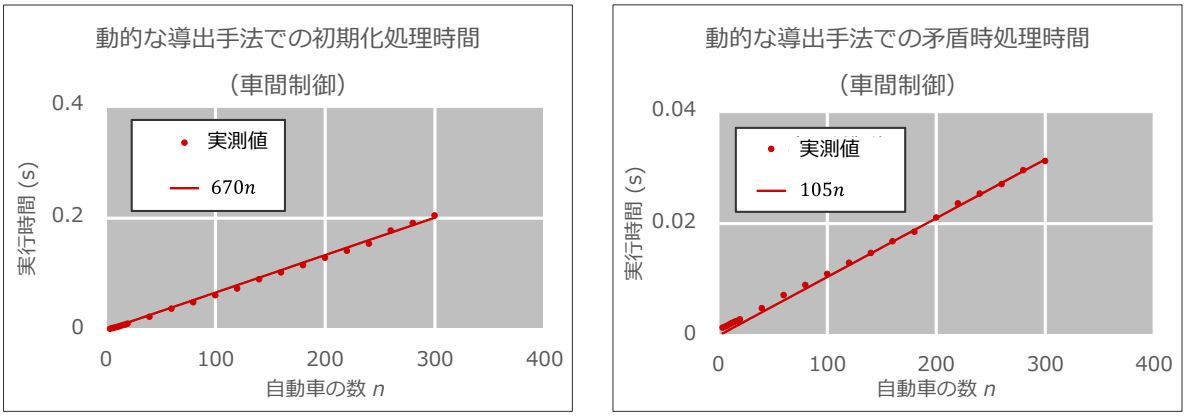

図 6.15　動的な解候補モジュール集合導出手法による車間制御の例題の処理時間

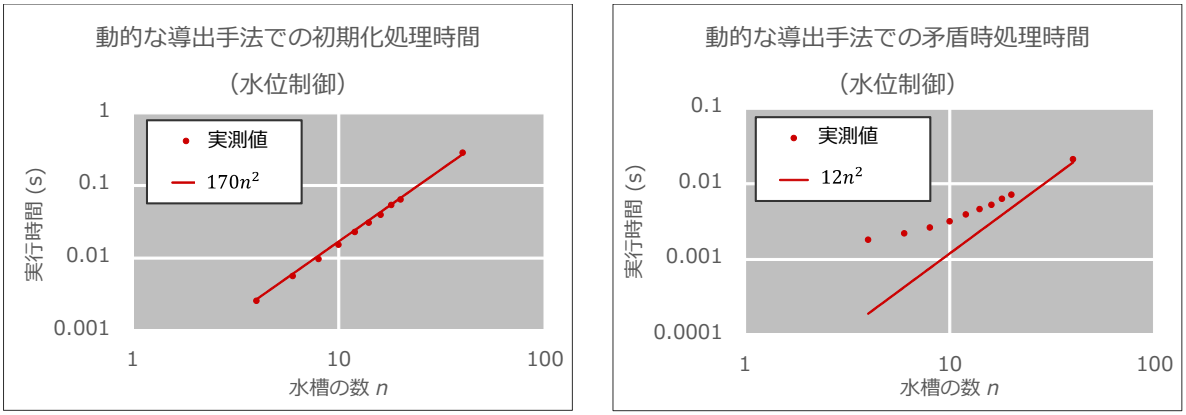

図 6.16　動的な解候補モジュール集合導出手法による水位制御の例題の処理時間

を初期状態の解候補モジュール集合として生成する．例えば A<<B<<C という優先度の記述があった場合にまず A<<B から A は B よりも優先度が低く，B は A よりも優先度が高いというデータを生成する．このとき初期状態の解候補モジュール集合は {A,B} となる．次に A<<B<<C から B は C よりも優先度が低く，C は B よりも優先度が高いというデータを生成し，先ほど生成したデータとマージすることによって B は A よりも優先度が高く，C よりも優先度が低いモジュールであるというデータを保持する．このとき初期状態の解候補モジュール集合は {A,B,C} となる．

動的な解候補モジュール集合導出手法における初期化処理の優先度データ生成回数および生成時間を図 6.17 に，優先度データのマージ回数およびマージ時間を図 6.18 に，初期状態の解候補モジュール集合の生成回数および生成時間を 6.19 に示す．

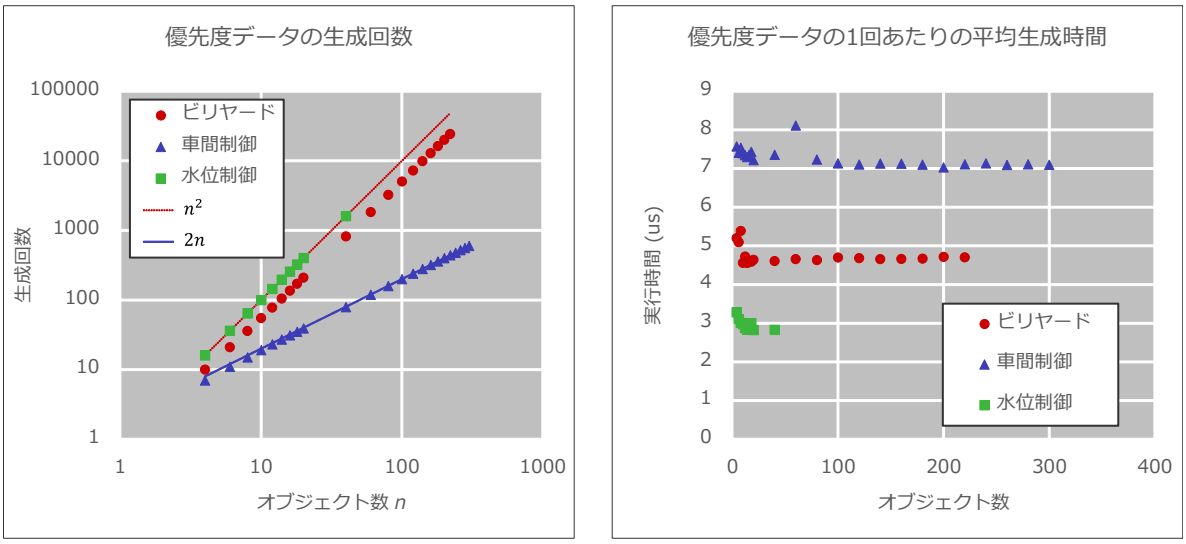

図 6.17　動的な解候補モジュール集合導出手法による優先度データ生成時間

図 6.17，図 6.18，図 6.19 を見ると優先度データ生成回数はビリヤードおよび水位制御において $O(n^2)$ であり，車間制御では $O(n)$ である．優先度データは各モジュールに関して生成を行うため優先度生成回数は総モジュール数に依存する数になる．実際にビリヤードと水位制御での総モジュール数は $O(n^2)$ であり，車間制御では $O(n)$ である．これは優先度データのマージおよび初期解候補モジュール集合の生成の回数についても同様であり，ビリヤードと水位制御で $O(n^2)$，車間制御で $O(n)$ である．

優先度データの生成は’,’ を介せず直接 << で繋がっているモジュールについて行われる．今回用いた通常実行の例題では << で直接つながっているモジュールの数はオブジェクトの数に依存していないため優先度データの生成にかかる時間はオブジェクトの数に対

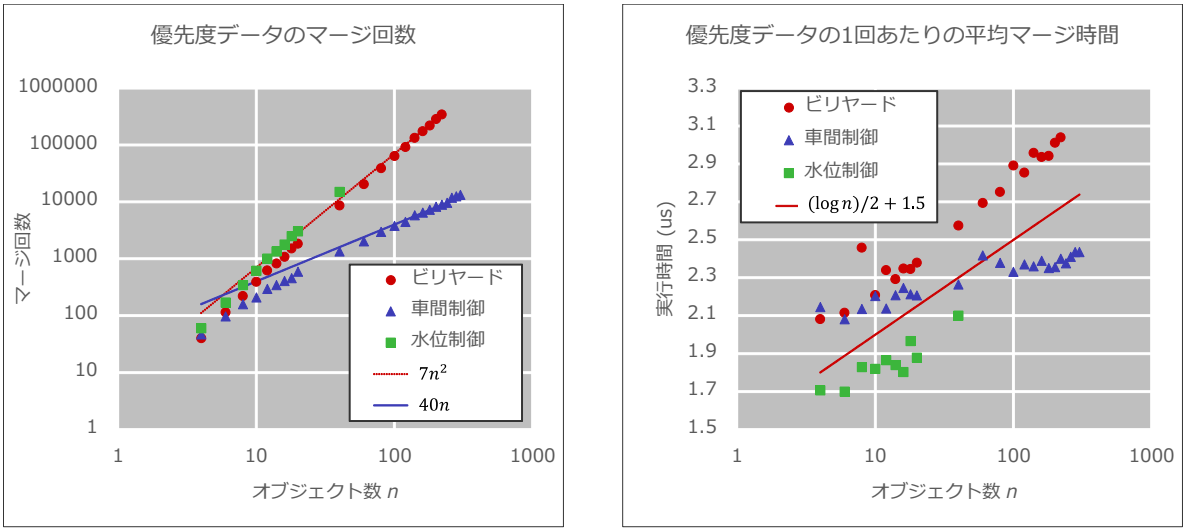

図 6.18　動的な解候補モジュール集合導出手法による優先度データのマージ時間

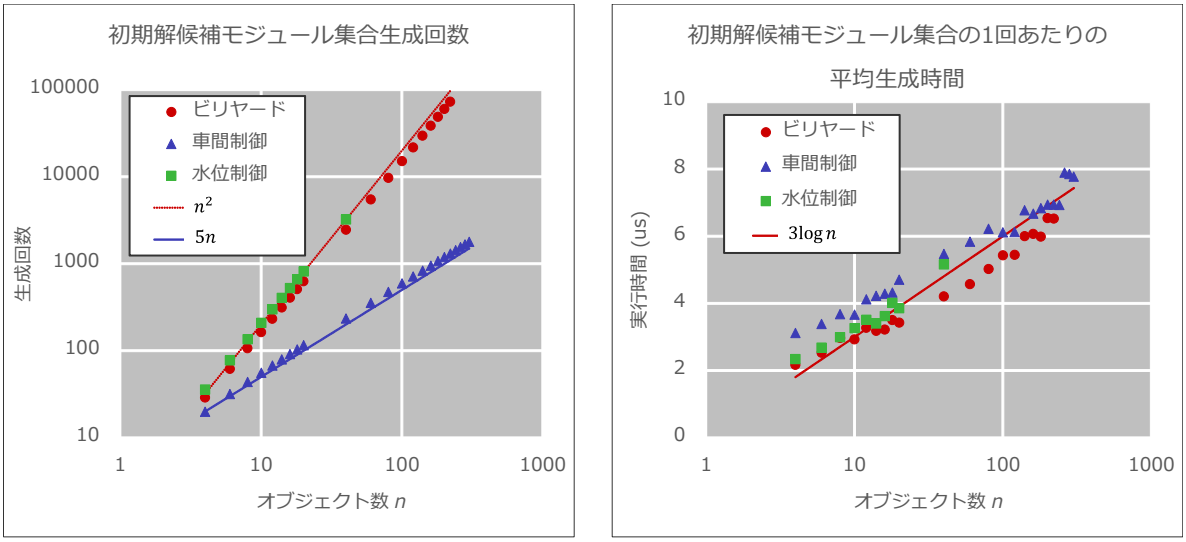

図 6.19　動的な解候補モジュール集合導出手法による初期解候補モジュール集合生成時間

して O(1) である．

ビリヤードおよび水位制御では 1 つのモジュールに対し優先度関係にあるモジュールが最大で $O(n^2)$ 個であり，車間制御では O(1) である．優先度データのマージはデータの要素数に対し対数オーダーで処理を行うことが可能であるため，優先度データのマージ時間はビリヤードと水位制御の例題で $O(\log n)$，車間制御の例題では O(1) となり，これは図 6.18 のデータとほぼ一致する．

初期解候補モジュール集合の生成時間は例題中に出現するモジュール数の対数時間で実行できるため，すべての例題で初期解候補モジュール集合の生成時間は $O(\log n)$ となる．

以上のことをまとめるとビリヤードと水位制御の例題の初期化処理時間の時間計算量は $O(n^2 \log n)$ であり，車間制御の例題では $O(n \log n)$ となると考えられる．しかし測定したデータではビリヤードと水位制御の例題の初期化処理時間は $O(n^2)$ であり，車間制御の例題では $O(n \log n)$ である．図 6.17，図 6.18，図 6.19 を見ると各処理の平均時間はすべて $10\mu s$ 以下と非常に短いため，関数呼び出しや測定データの出力などの影響が大きく，解候補モジュール集合の各処理は測定を行った範囲では全体の実行時間に影響を及ぼしていないと考えられる．従ってオブジェクト数をさらに増やした場合に初期化処理の時間計算量はビリヤードと水位制御の例題では $O(n^2 \log n)$ となり，車間制御の例題では $O(n \log n)$ となると考えられる．

矛盾時の処理に関する考察

次に解候補モジュール集合関係の矛盾時の処理について考察する．動的な解候補モジュール集合の導出手法では図 6.1 のアルゴリズムに従って新たな解候補モジュール集合を生成する．

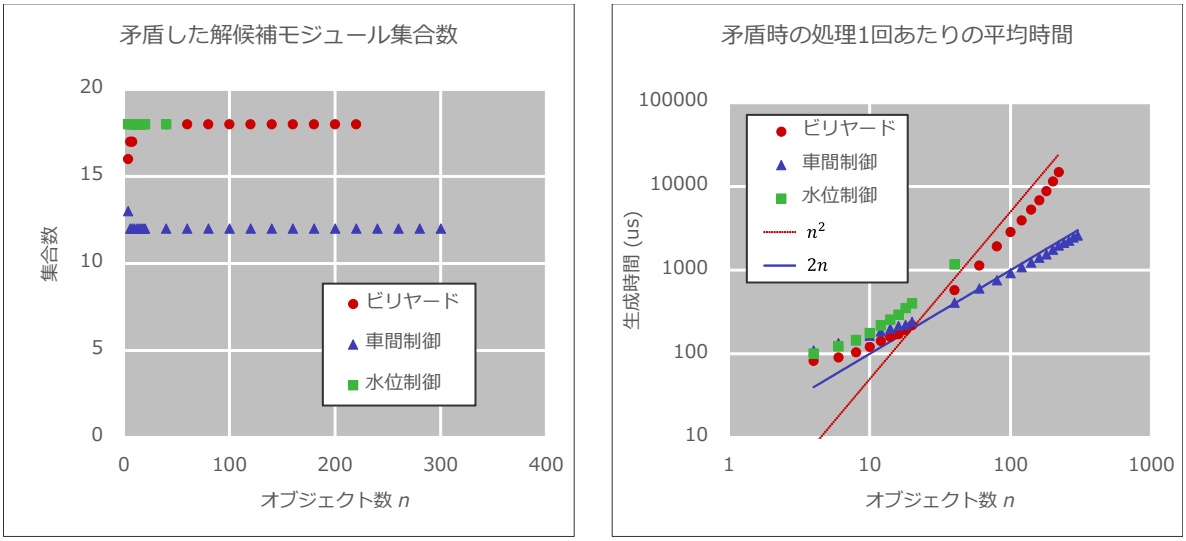

図 6.20　動的な解候補モジュール集合導出手法による矛盾時の処理時間

図 6.20 を見ると解候補モジュール集合が矛盾した回数はすべての例でオブジェクト数に依存せずに一定となっている．Required なモジュールはすべての解候補モジュール集合の要素となるため集合の包含関係のチェックの際に Required なモジュールにつ

いて考慮する必要がないため動的な解候補モジュール集合導出手法では最適化のために Required なモジュールと Required でないモジュールを分けて保持している．すべての例題で Required でないモジュールの数が $O(n)$ であり，ビリヤード，水位制御の例題では Required なモジュールの数が $O(n^2)$ であり車間制御の例題では $O(n)$ である．

図 6.1 のアルゴリズムでモジュール数に依存する処理は以下の処理である．

- Required のモジュールのチェック（図 6.1 の 5 行目）（図 6.21）
- 新たな解候補モジュール集合の生成（図 6.1 の 6 行目）（図 6.22）
- 生成済みの解候補モジュール集合の集合に生成した集合を挿入（図 6.1 の 8 行目）（図 6.23）

Required のモジュールのチェックは対象の矛盾した集合から Required なモジュールをすべて取り除くことによって実行している．今回測定した例題では矛盾した集合の要素数はモジュール数に依存しないため除去時間の時間計算量は Required なモジュールの要素数に比例する値となる（図 6.21）．

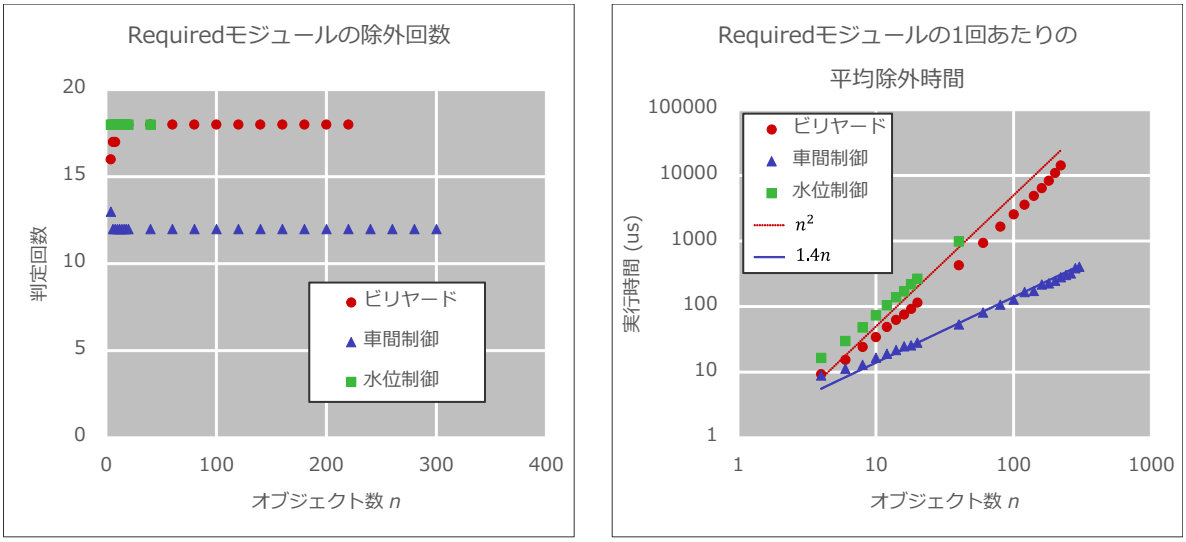

図 6.21　動的な解候補モジュール集合導出手法による矛盾時の処理における Required モジュールの判定時間

新たなモジュール集合の生成は対象となるモジュール集合から条件を満たすモジュールを取り除くことで行われる．対象となるモジュール集合は Required でないモジュールの集合であり，今回測定した例題では取り除くモジュールはオブジェクト数に依存しないためモジュール集合の生成の時間計算量は Required でないモジュールの集合数の対数オー

ダーとなる（図 6.22）.

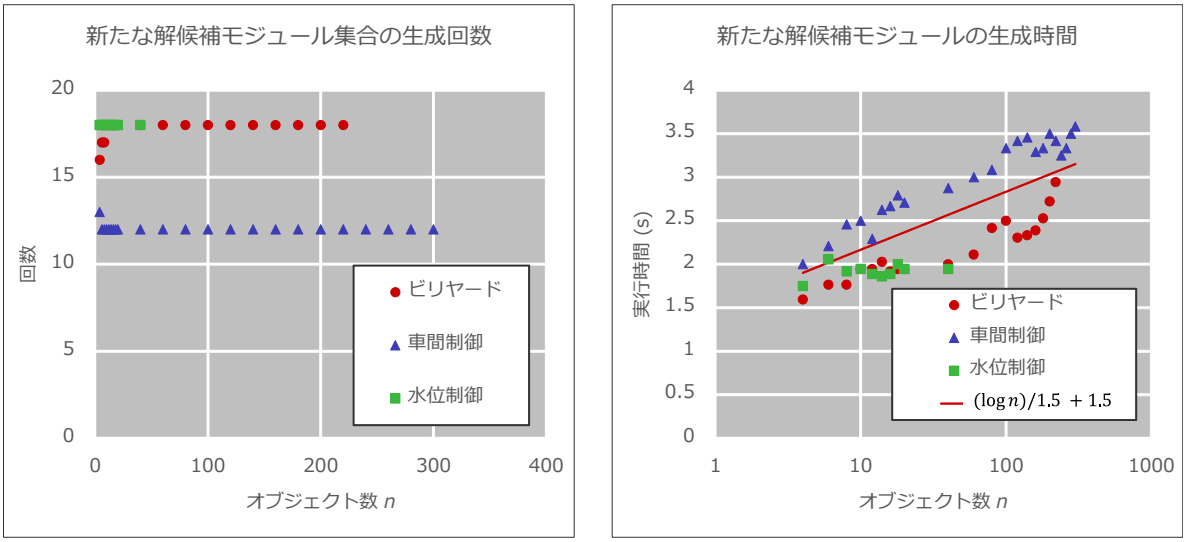

図 6.22　動的な解候補モジュール集合導出手法による矛盾時の処理における新たなモジュールの生成時間

今回測定した例題では図 6.22 で分かるように生成される解候補モジュール集合の数はオブジェクト数によらずに一定である．従って生成した集合を挿入する処理には生成済みの解候補モジュール集合の集合内に同じ集合が存在するかどうかをチェックする時間がかかり，その時間計算量は Required でないモジュール数に比例する（図 6.23）.

以上のことをまとめると動的な解候補モジュール集合導出手法では Required モジュールの除去に最も大きな時間計算量がかかりビリヤードおよび水位制御の例題で $O(n^2)$ であり，車間制御で $O(n)$ となる．実際に図 6.14，図 6.15，図 6.16 を見ると測定結果と理論的な時間計算量は一致している．

6.6.3　非決定実行を行った結果に対する考察

以前の手法では図 3.6 のプログラムの実行がタイムアウトする前に終了したものがボールの数 4 つのときのみであったため，以前の手法についてはオブジェクト数に対する時間計算量が測定できない．従って以前の手法を用いた非決定実行に関しては時間計算量の議論はせずに動的な解候補モジュール集合導出手法によって同じプログラムを実行した結果と比較する．比較した結果を図 6.24 に示す．図 6.24 を見るとすべての処理の実行時間がおよそ 1/1000 になっていることが分かる．

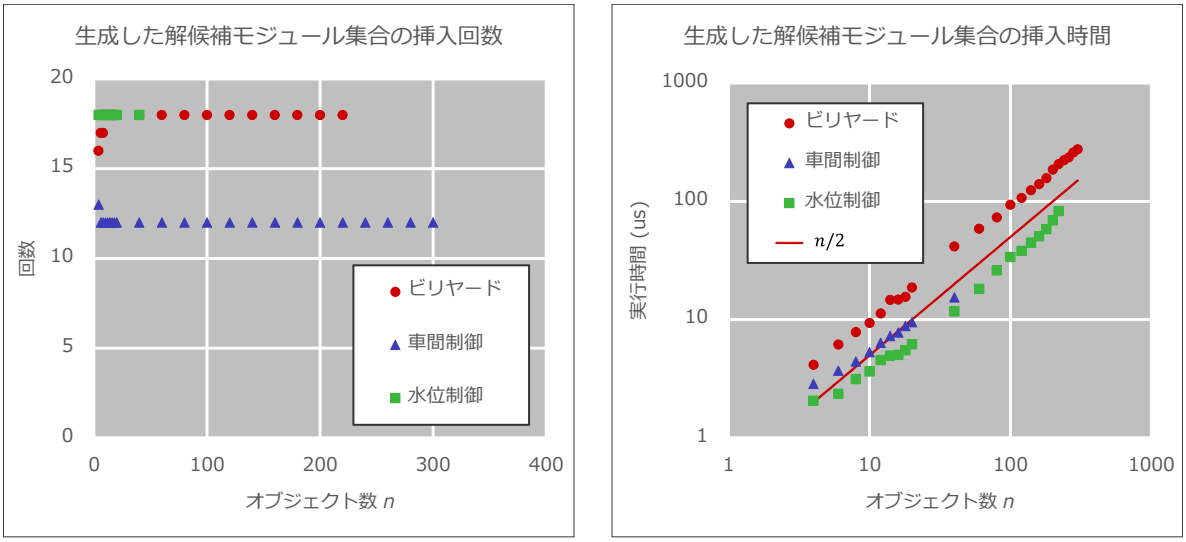

図 6.23　動的な解候補モジュール集合導出手法による矛盾時の処理における生成した集合の挿入時間

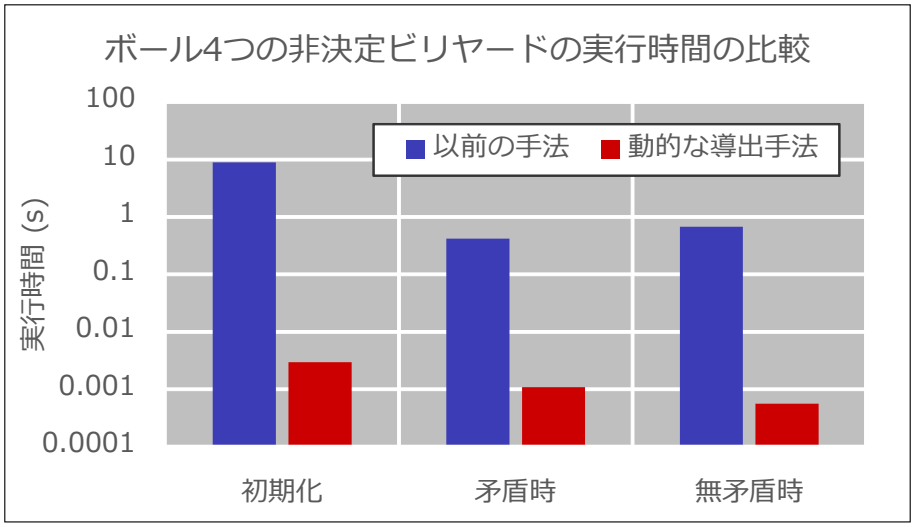

図 6.24　ボールの数 4 つの非決定ビリヤードの実行時間

次に動的な解候補モジュール集合導出手法で非決定的なビリヤードを実行した結果を図 6.25 に示す.

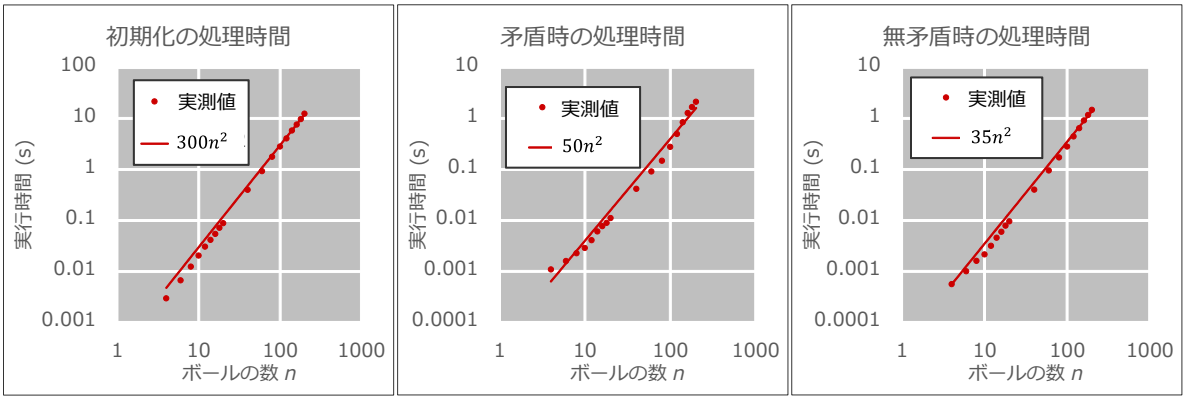

図 6.25　動的な解候補モジュール集合導出手法による非決定ビリヤードの実行結果

初期化処理に関する考察

非決定的なビリヤードを実行した際の動的な解候補モジュール集合導出手法での初期化処理の実行時間を図 6.26 に示す．動的な解候補モジュール集合導出手法での初期化処理では非決定的でないビリヤードと非決定的なビリヤードで違いがないため初期化処理の時間計算量は非決定的でないビリヤードと一致する．

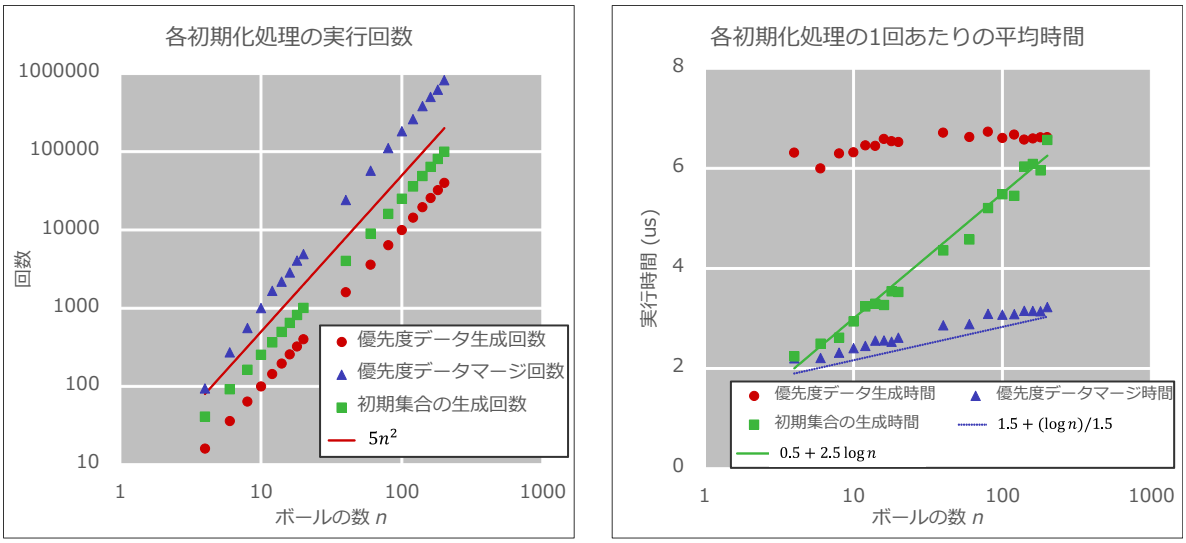

図 6.26　動的な解候補モジュール集合導出手法による非決定ビリヤードの初期化処理

矛盾時の処理に関する考察

非決定的なビリヤードを実行した際の動的な解候補モジュール集合導出手法での矛盾時の処理の実行時間を図 6.27 に示す．非決定的でないビリヤードと非決定的なビリヤードの動的な解候補モジュール集合導出手法での矛盾時の処理に影響を及ぼす要素の違いは Required なモジュールの数が $O(n^2)$ から $O(n)$ となり Required でないモジュールの数が $O(n)$ から $O(n^2)$ になっていることである．非決定なビリヤードにおける矛盾時の処理の詳細は非決定的でないビリヤードと一致する．

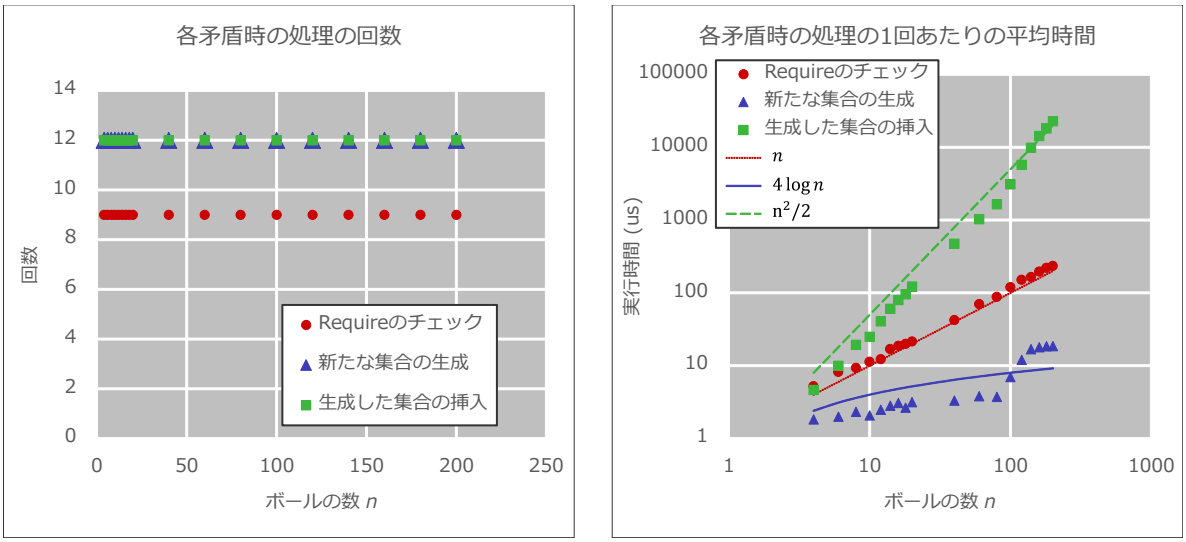

図 6.27　動的な解候補モジュール集合導出手法による非決定ビリヤードの矛盾時処理

無矛盾時の処理に関する考察

最後に非決定的なビリヤードを実行した際の動的な解候補モジュール集合導出手法での無矛盾時の処理の実行時間を図 6.28 に示す．無矛盾時の処理は大きく次の 2 つの処理に分けることができる．

- 生成済みの解候補モジュール集合が無矛盾なモジュール集合に包含されるかどうかを判定する
- 無矛盾なモジュール集合に包含される解候補モジュール集合を生成済みの解候補モジュール集合の集合から除外する

非決定なビリヤードの例題では生成される解候補モジュール集合はボールの数によらず一定である．生成済みの解候補モジュール集合の要素数および無矛盾なモジュールの要素数は非決定なビリヤードの例題に出現する全モジュール数に依存し $O(n^2)$ となる．包含性の判定時間と生成済みの解候補モジュール集合の集合から除外する際に同じ解候補モジュール集合かどうかの判定時間の時間計算量は除外するモジュール集合の要素数に依存し $O(n^2)$ となる．

従って非決定的なビリヤードを実行した際の動的な解候補モジュール集合導出手法での無矛盾時の処理の時間計算量は $O(n^2)$ となる．

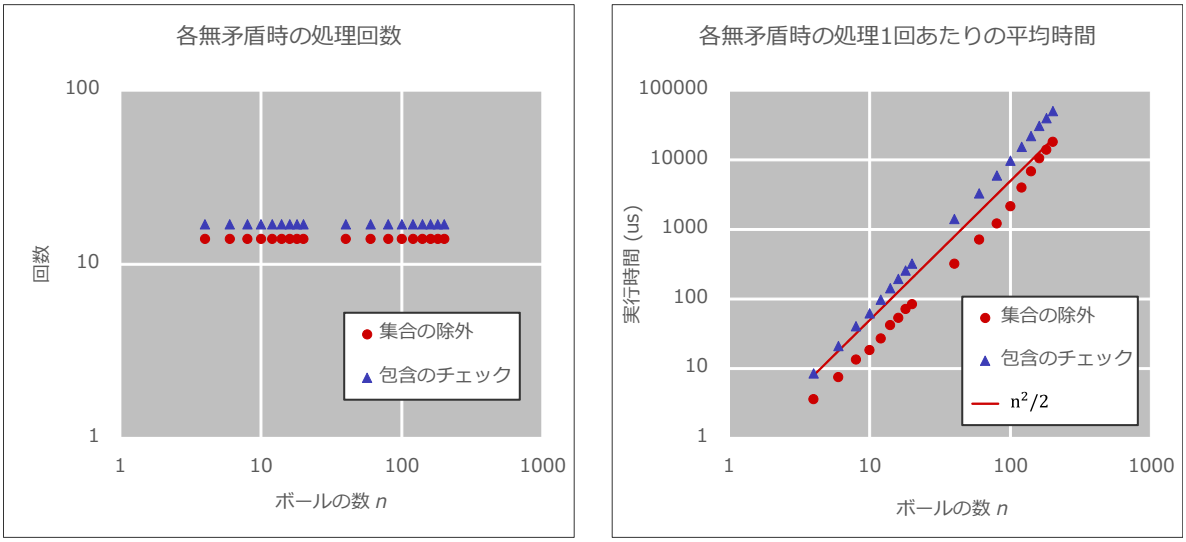

図 6.28　動的な解候補モジュール集合導出手法による非決定ビリヤードの無矛盾時処理

6.6.4　評価結果のまとめ

今回の評価で測定した例題のオブジェクト数 n 個の場合の解候補モジュール集合の数を表 6.4 に示す。

今回の評価で測定した通常実行の例題における時間計算量の比較を表 6.5 に示す．

非決定ビリヤードの例題では以前の手法ではタイムアウトによりボールが 4 つの場合の例題しか測定できなかったため，ボールが 4 つの場合の実行時間の比較を表 6.6 に示す．

動的な解候補モジュール集合導出手法を利用した際の非決定ビリヤードの例題の時間計算量を表 6.7 に示す．

表 6.4 各例題におけるオブジェクト数 n 個の場合の解候補モジュール集合の数

例題	解候補モジュール集合の数
ビリヤード	2^n
車間制御	$2 \times 5^{n-1}$
水位制御	2^n
非決定ビリヤード	$1 + n + \sum_{i=2}^{n} (2^{i(i-1)}{}_nC_i)$

表 6.5 通常実行の例題に対する時間計算量

	ビリヤード		車間制御		水位制御	
	before	after	before	after	before	after
初期化	$O(4^n n\log n)$	$O(n^2)$	$O(5^n n\log n)$	$O(n)$	$O(4^n n\log n)$	$O(n^2)$
矛盾時	$O(2^n n^2)$	$O(n^2)$	$O(5^n n\log n)$	$O(n)$	$O(2^n n\log n)$	$O(n^2)$

表 6.6 非決定実行ビリヤードでボールが 4 つの場合の実行時間

	before	after
初期化	8957337 (μs)	2956 (μs)
矛盾時	421192 (μs)	1091 (μs)
無矛盾時	678107 (μs)	563 (μs)

表 6.7 動的な解候補モジュール集合導出手法を使用した場合の非決定実行ビリヤードの時間計算量

初期化	$O(n^2)$
矛盾時	$O(n^2)$
無矛盾時	$O(n^2)$

第 7 章

関連研究

7.1 KeYmaera

KeYmaera[3] はハイブリッドシステムのための証明器である．KeYmaera はハイブリッドシステムの正当性や安全性，制御性，反応性，活性といった性質を証明することができる．KeYmaera は HyLaGI と同様にパラメータを含むシステムの扱いが可能であり，数式処理による計算も可能である．

KeYmaera におけるモデリング言語としてハイブリッドプログラムという手続き型言語に似た構文を持つ言語を利用しているところが HydLa と異なり，KeYmaera の目的はシミュレーションでなく検証であることが HyLaGI と異なっている．

7.2 HybridCC

HybridCC[5] はハイブリッドシステムを記述するための並行制約プログラミング言語である．HybridCC は制約を用いたハイブリッドシステムの簡潔な記述が可能である．

現状の HybridCC の処理系では区間演算による計算を行っているがモデルの離散変化を正確に計算できない場合がある．一方 HyLaGI では数式処理によりモデルの離散変化を正確に計算できる点が HybridCC と異なる点である．

7.3 B-Prolog

B-Prolog[1] は制約論理プログラミングシステムである．B-Prolog は Prolog にいくつかの拡張を加えた高性能な実装である．Prolog は宣言型の論理プログラミング言語であ

る．B-Prolog では宣言的にリストを構成するための構造としてリストの内包記法がある．B-Prolog でのリストの内包記法は次の構文で記述できる．

$$[T : E_1 \texttt{ in } D_1,\ \ldots,\ E_n \texttt{ in } D_n,\ \mathit{LocalVars}, \mathit{Goal}]$$

例えば L `@=` `[`X `:` X `in` $1..5$`]` . は $L = [1,2,3,4,5]$ を意味する．

Prolog は汎用的な目的で提案されているのに対し，HydLa はハイブリッドシステムをモデリングすることが目的であり，微分方程式などを用いて時刻に対して連続的に変化する変数を容易に表すことができる点が B-Prolog と異なる．

7.4 Visual Prolog

Visual Prolog[16] は Prolog に基づいた制約プログラミング言語である．Visual Prolog もリストの内包記法が可能であり，リストの内包記法は次の構文で記述される．

$$[\ \mathit{Term}\ \|\ \mathit{Term}\ ]$$

例えば [X || $X = \mathit{getMember\_nd}(L),\ X$ `mod` $2 = 0$] は L 内の要素であり，偶数である X を要素とするリストを表す．

Visual Prolog は Windows 向けのアプリケーションを作成するのに用いられることを想定しているのに対し，HydLa はハイブリッドシステムをモデリングすることが目的であり，微分方程式などを用いて時刻に対して連続的に変化する変数を容易に表すことができる点が Visual Prolog と異なる．

7.5 インクリメンタルな解候補導出

HydLa における解候補モジュール集合の導出手法としてインクリメンタルな解候補導出手法が文献 [17] で提案されている．文献 [17] でのアルゴリズムではモジュール数 n 個の解候補モジュール集合が矛盾した際にモジュール数 $n-1$ 個の解候補モジュール集合をすべて導出する．一方，本論文で紹介した動的な解候補モジュール集合導出アルゴリズムでは矛盾した制約の情報を用いて必要な解候補モジュール集合のみを導出している点が異なる．

7.6 leaf pruning

ある制約の集合のすべての部分集合から極大無矛盾集合を求めるときの最適化手法として leaf pruning が提案されている [14]．leaf pruning では全部分集合をノードとする包含関係に基づくエッジを持つスパニング木を生成し，部分木内の最も小さな集合が矛盾した場合にはその部分木内のノードはすべて矛盾することを利用して計算する集合を枝狩りする手法である．

leaf pruning で対象としている制約集合はある制約集合の全部分集合であり本論文で述べた極大無矛盾集合の生成手法で対象としている制約集合は制約の優先度に違反しない部分集合のみである点が異なる．

第 8 章

まとめと今後の課題

8.1 まとめ

本論文ではボールが多数あるビリヤードや自動車が多数存在する場合の自動車の車間制御の問題などオブジェクトの数が多い問題に対して HydLa の記述量や HyLaGI の実行時間が増大してしまうのに対し，HydLa にリスト記法を導入することで HydLa の記述量を減らし，動的な解候補モジュール集合導出手法によって HyLaGI の実行時間の削減を行った．

今回使用した例題でオブジェクト数を n としたときにリストの導入により HydLa での制約の記述量を $O(n)$ または $O(n^2)$ から $O(1)$ に削減することができ，動的な解候補モジュール集合導出手法によって HyLaGI における解候補モジュール集合の処理の時間計算量を最大で $O(5^n \log n)$ から $O(n)$ に削減することができた．

8.2 今後の課題

8.2.1 リスト記法を利用したシミュレーション

HyLaGI はリスト記法を導入したプログラムを処理する際にリストを用いない記法で記述された等価なプログラムと同等の処理を行う．例えば HyLaGI における図 5.2 のプログラムの処理と図 3.3 の処理は完全に一致する．つまり {INIT(X[i],2*i,0) | i in {2..|X|}} は INIT(x2,4,0), INIT(x3,6,0), INIT(x4,8,0) と書かれているものとして扱われる．従って {INIT(X[i],2*i,0) | i in {2..|X|}} の計算では式 8.1 の計算を行うこ

とになる.

$$\begin{aligned} \mathtt{x2}(0) = 4 \wedge \frac{d\mathtt{x2}}{dt}(0) = 0 \\ \wedge\ \mathtt{x3}(0) = 6 \wedge \frac{d\mathtt{x3}}{dt}(0) = 0 \\ \wedge\ \mathtt{x4}(0) = 8 \wedge \frac{d\mathtt{x4}}{dt}(0) = 0 \end{aligned} \tag{8.1}$$

従ってリスト記法によって制約の記述量を減らしても計算する制約の数を減らすことはできない.

{INIT(X[i],2*i,0) | i in {2..|X|}} の制約をそのまま記述したものが式 8.2 となる.

$$\mathtt{x}i(0) = 2 * i \wedge \frac{\mathtt{x}i}{dt}(0) = 0 \quad (i = 2, 3, 4) \tag{8.2}$$

式 8.2 と式 8.1 は等価であるため式 8.1 を計算する代わりに式 8.2 の計算を行っても同じ結果を得られると考えられる. ボールの数を n としたときに式 8.1 の原子制約（論理和および論理積を含まない制約）は $2n - 2$ 個になる. 一方式 8.2 では i の範囲が 2 から n になるだけで, 原子制約は 2 個になる. これは制約 INIT に限らず, COLLISION や WALL, UNIFORM についても同様に計算が必要な制約の数を減らすことができると考えられる.

謝辞

3 年間にわたりご指導くださった上田和紀教授に心より感謝いたします．日頃の研究や出張時にお世話になりました細部博史先生や石井大輔さん，上田研研究室 HydLa 班の皆様に感謝いたします．研究について多大なるご協力をいただき，研究以外の研究室生活でもお世話にならない日はなかった松本翔太さんに深く感謝いたします．在学中には遅くまで，そして卒業してからも遊んでくださった宮原和大さん，谷口直輝さん，和田亮さん，よく雑談をしてくれた青山龍一君，研究テーマが近く，海外出張や学会で一緒に行動することが多かった小林輝哉くん，そしてお昼にいろいろなお店に連れて行ってくれた和田努くんに感謝いたします．最後に常に支えてくださった家族に感謝いたします．

2015 年 1 月　河野 文彦

参考文献

[1] Afany Software : B-PROLOG, 2014, http://www.picat-lang.org/bprolog/

[2] Andre, P. Edmund M, C. : Formal Verification of Curved Flight Collision Avoidance Maneuvers: A Case Study, Springer Berlin Heidelberg, 2009, pp.547–562.

[3] Andre, P., Jan-David, Q. : KeYmaera: A Hybrid Theorem Prover for Hybrid Systems, In IJCAR 2008, LNCS 5195, Springer-Verlag, 2008, pp.171–178.

[4] Anthony, C. A., REDUCE, http://reduce-algebra.com/, 2014.

[5] Bjorn, C., Vineet, G. : Hybrid cc with interval constraints, in Proc. HSCC'98, LNCS 1386, Springer, 1998, pp.80–94.

[6] Borning, A., Freeman-Benson, B., Wilson, M. : Constraint Hierarchies, Lisp and Symbolic Computation, Vol.5, 1992, pp.221–268.

[7] Carloni, L., Passerone, R., Pinto, A., Sangiovanni-Vincentelli, A. L. : Languages and Tools for Hybrid Systems Design, Foundations and Trends in Design Automation, Vol.1, No.1, 2006, pp.1–204.

[8] Eric, C., Andreas, E., Mattias, B. : Collision Warning with Full Auto Brake and Pedestrian Detection - a practical example of Automatic Emergency Braking, In Intelligent Transportation Systems (ITSC) 2010 13th International IEEE Conference, IEEE, 2010, pp.155–160.

[9] Gupta, V., Jagadeesan, R., Saraswat, V., Bobrow, D. : Programming in Hybrid Constraint Languages, in Hybrid Systems II, LNCS 999, Springer, 1995, pp.226–251.

[10] 廣瀬賢一, 大谷順司, 石井大輔, 細部博史, 上田和紀 : 制約階層によるハイブリッドシステムのモデリング手法, 日本ソフトウェア科学会 第 26 回大会, 2D-2, 2009.

[11] Karl, H. J., John, L., Shankar, S. : Modeling of hybrid systems. Control Systems, Robotics and Automation, from Encyclopedia of Life Support Systems (EOLSS

), 2002.

[12] 河野文彦, 小林輝哉, 松本翔太, 上田和紀：数式処理に基づくハイブリッドシステムシミュレータ Hyrose の大規模モデルシミュレーションに向けた拡張, 日本ソフトウェア科学会 第 31 回大会, 2014.

[13] Lunze, J. : Handbook of Hybrid Systems Control: Theory, Tools, Applications, Cambridge University Press, 2009.

[14] Malouf, R. : Maximal Consistent Subsets, Computational Linguistics, Vol.33, No.2, 2007, pp.153–160.

[15] 松本翔太, 上田和紀 : ハイブリッド制約言語 HydLa の記号実行シミュレータ Hyrose, コンピュータソフトウェア, Vol.30, No.4, 2013, pp.18–35.

[16] Prolog Development Center A/S : Visual Prolog, 2014, http://www.visual-prolog.com/

[17] 清水悠一：ハイブリッドシステムモデリング言語 HydLa 処理系におけるインクリメンタルな解候補導出, 早稲田大学基幹理工学部, 卒業論文, 2010.

[18] 上田和紀, 細部博史, 石井大輔：ハイブリッド制約言語 HydLa の宣言的意味論, コンピュータソフトウェア, Vol.28, No.1, 2011, pp.306–311.

[19] 上田和紀, 石井大輔, 細部博史：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, SSV2008（第 5 回システム検証の科学技術シンポジウム）, 2008.

[20] Wolfram Research, Inc., Mathematica, http://www.wolfram.co.jp/products/mathematica/index.html, 2014

発表文献

[1] 河野文彦, 小林輝哉, 松本翔太, 上田和紀： 数式処理に基づくハイブリッドシステムシミュレータ Hyrose の大規模モデルシミュレーションに向けた拡張, 日本ソフトウェア科学会 第 31 回大会, 2014, 登壇発表（13 pages）.

[2] 小林輝哉, 河野文彦, 松本翔太, 上田和紀： フェーズ間の制約差分情報および制約-変数間の依存関係を用いた HydLa 処理系の最適化, 人工知能学会第 28 回全国大会, 4C1-2, 2014, 登壇発表（共著, 4 pages）

[3] 松本翔太, 河野文彦, 上田和紀： ハイブリッド制約言語 HydLa を用いたハイブリッドシステムの解析, 人工知能学会第 28 回全国大会, 3O1-3, 2014, ポスター（共著, 3 pages）

[4] 河野文彦, 松本翔太, 上田和紀： 制約の静的解析を用いた HydLa 処理系の最適化, 人工知能学会第 27 回全国大会, 2J1-1, 2013, 登壇発表（4 pages）

付録

動的な解候補モジュール集合導出処理のための主要関数のソースコードを示す.

新たなモジュール集合生成処理（図 6.1 のアルゴリズム）の関数

```
1  void IncrementalModuleSet::generate_new_ms(const module_set_set_t&
       mcss, const ModuleSet& ms){
2    HYDLA_LOGGER_DEBUG("%% inconsistent module set : ", ms.get_name
         ());
3    std::string for_debug = "";
4    for( auto mit : mcss ){
5      for_debug += mit.get_name() + " ,";
6    }
7    HYDLA_LOGGER_DEBUG("%% maximal consistent module set : ",
         for_debug);
8    // result set of module set
9    module_set_set_t new_mss;
10   ModuleSet inconsistent_ms = ms;
11   inconsistent_ms.erase(required_ms_);
12   for( auto lit : ms_to_visit_ ){
13     // if ``lit'' include inconsistent set, ``lit'' is
           inconsistent. Then new module set is generated
14     if(lit.including(inconsistent_ms)){
15       // get vector of removable module set
16       std::vector<ModuleSet> rm = get_removable_module_sets(lit,
             inconsistent_ms);
17       for(auto removable_module_set : rm){
18         // remove removable module set from ``lit''.
19         ModuleSet new_ms = lit;
20         new_ms.erase(removable_module_set);
```

```
21          bool checked = false;
22          for( auto mcs : mcss ){
23           // if ``new_ms'' is included by maximal consistent set,
                 then ``checked'' becomes true
24            if(mcs.disjoint(new_ms)){
25              checked = true;
26              break;
27            }
28          }
29         // if ``checked'' is false, add ``new_ms'' to ``new_mss''
30          if(!checked){
31            new_mss.insert(new_ms);
32            HYDLA_LOGGER_DEBUG("%% new_ms : ", new_ms.get_name());
33          }
34        }
35      }else{
36        // if ``lit'' does not include inconsistent module set, ``
              lit'' remains
37        if(!check_same_ms_generated(new_mss, lit)){
38          new_mss.insert(lit);
39        }
40      }
41    }
42    // update ``ms_to_visit_'' by generated module sets
43    update_by_new_mss(new_mss);
44  }
```

図 6.1 のアルゴリズムの 6 行目における $\{ M \mid M<<I \vee M=I \}$ の算出関数

```
1   std::vector<ModuleSet> IncrementalModuleSet::
        get_removable_module_sets(ModuleSet &current_ms, const
        ModuleSet &ms)
2   {
3     HYDLA_LOGGER_DEBUG("%% candidate modules : ", ms.get_name(), "\n
          ");
4     // ``ms'' is inconsistent set
5     std::vector<ModuleSet> removable;
6
7     for( auto it : ms )
```

```
8   {
9     bool has_weaker = false;
10    for( auto weaker : weaker_modules_[it] )
11    {
12      if(ms.find(weaker) != ms.end())
13      {
14        has_weaker = true;
15        break;
16      }
17    }
18    if(has_weaker) continue;
19    // one of the removable module set
20    ModuleSet removable_for_it;
21    // a vector which has all modules which is weaker than ``it''
22    std::vector<module_t> childs;
23    childs.push_back(it);
24
25    while(childs.size() > 0)
26    {
27      // pop the top of childs
28      auto roop_it = childs.front();
29      removable_for_it.add_module(roop_it);
30      // to prevent recheck, erase ``childs'' front
31      childs.erase(childs.begin());
32      // if there are modules which is weaker than ``roop_it''
33      if(weaker_modules_.find(roop_it) != weaker_modules_.end())
34      {
35        for( auto wmit : weaker_modules_[roop_it] )
36        {
37        // ``wmit'' is a module which is weaker than ``roop_it''
38        // if ``wmit'' is included by current module set
39          if(current_ms.find(wmit) != current_ms.end())
40          {
41          // push ``wmit'' to ``childs''
42            childs.push_back(wmit);
43          }
44        }
45      }
46    }
```

```
47        if(same_modules_.count(it)) removable_for_it.insert(
              same_modules_[it]);
48        removable.push_back(removable_for_it);
49      }
50
51      // string for debug
52      std::string str = "";
53      for(auto it : removable)
54      {
55        str += it.get_name();
56        str += " , ";
57      }
58      HYDLA_LOGGER_DEBUG("%% removable modules : ", str, "\n");
59      return removable;
60    }
```