# 代数仕様言語Maudeを用いた制約オートマトンの実現

中島 震†

Lightweight Formal Methods（LFM）は，形式手法の考え方の 1 つであって，不具合を系統的に発見するために自動解析ツールを用いる方法である．自動解析を可能にするために，探索範囲を有界にするなど解析の部分性を容認する．また，対象を静的なデータ構造関係あるいは動的な振舞い仕様のいずれかに制限する．本稿では制約オートマトンの考え方を採用して，静的なデータ構造関係と動的な振舞い仕様を統合する LFM ツールを提案する．特に，書き換え論理に基づく代数仕様言語 Maude を用いる具体的な表現と解析の方法を示す．

# Maude-based Implementation of Constraint Automata

Shin Nakajima†

The notion of Lightweight Formal Methods (LFM) is a modest approach to practical uses of the formal methods for systematic search of potential errors. In order to be automatic, an LFM tool focuses on either the static structuring of data or the dynamic behavioral specification. This paper proposes to use constraint automata to combine the two aspects. In addition, the constraint automata are encoded in Maude, an algebraic specification langauge Maude based on rewriting logic. The advanced features of Maude offer many benefits for the automatic analyses.

## 1. はじめに

Lightweight Formal Methods（以下，LFM と略す）と呼ぶ考え方[13]が，形式手法を実際に適用する方法として注目を集めている．記述対象に不具合がないことを数学的に証明するという以前からの形式手法と異なり，不具合を系統的に発見するために自動解析ツールを用いる方法である．自動的に行うために探索範囲を有界にするなど，部分的な解析に終わることが多い．同時に特定の観点からの記述に限定せざえるをえない．表現力を犠牲にしても自動的な解析を実現する．

ソフトウェアのデザインは広範な特徴を持ち，静的なデータ構造関係や動的な振舞い仕様などを含むが，個々の LFM ツールは特定の観点だけを扱う．たとえば，Alloy[14]はデータの静的な構造の取扱いが中心であり，他方，モデル検査法[6]は制御の流れを中心とする振舞い解析に有効である．自動解析という LFM の良さを生かし，多様な特徴を取り扱うことが可能な形式仕様と解析の方法が望まれる．

本稿では制約オートマトンと呼ぶ考え方を提案する．記号的な表現法を用いることで，静的なデータ構造関係をモデル検査法などの状態解析法の枠内で取り扱う方法を提案する．既存のモデル検査ツールに比べて，データの側面が重要となる対象の取扱いが容易になる．さらに，制約オートマトンを書き換え論理に基づく代数仕様言語 Maude[7),17)]で表現する．特に，LTL モデル検査[9]など Maude が提供する高度な解析機能を用いることで制約オートマトンの自動解析を簡便に行えることを示す．

以下，2 章で本研究の背景と動機を整理する．3 章で制約オートマトンの考え方を説明する．4 章で制約オートマトンを Maude で表現する方法を述べる．5 章で Alloy 本[14]記載の「ホテル鍵問題」を対象として Maude を用いた制約オートマトンの記述を解析の例を紹介する．6 章で提案方式について考察する．

## 2. 研究の背景と動機

### 2.1 Lightweight Formal Methods

Lightweight Formal Methods（LFM）[13]は記述対象が正しいことを厳密な証明を諦めて，不具合を系統的に発見することに主眼をおく．

モデル検査法に基づくツールは，並行システムの振舞いを表現した有限の状態空間を探索する．通常のテ

† 国立情報学研究所
National Institute of Informatics

スト技法を用いる方法に比べると不具合の発見方法として系統的であり信頼できる．しかし，不具合がないことを証明する目的に合わないことが多い．最初に提案された CTL モデル検査法[6] などモデル検査アルゴリズム自身は健全かつ完全である．一方，実際の利用法という観点からは一種のデバッグツールとしての有用性が高い．対象を有限の状態空間に限定するために，事前に抽象化された部分的なシステム記述に対する解析になってしまうからである．また，有界モデル検査法（BMC）[2] は，当初からシステムに内在する不具合の系統的な発見ツールとして考案された．さらに，Alloy [14] は静的なデータ構造関係を本質的に 1 階述語論理の式で表現し，自動化ツールとするために当該の式を満たすモデルを決められた有限スコープ内で探索する方法を採用した．

LFM ツールはうまく使えば高度な「デバッガ」として有用である．しかし，対象の表現は特定の性質に限定される．特に，既存のモデル検査ツールはデータの側面を適切に扱えないことが多い．Alloy は時間点を明示する仕様の書き方を採用することで処理流れに関する性質を解析する．しかし，通常のアプリケーションに関わる性質と時間点の記述とが混在するため複雑な記述になるという欠点がある．

静的なデータ関係の側面と動的な振舞い仕様とを統合するような LFM ツールが望まれる．表現力の豊かさと自動化の可能性のバランスが大切である．

### 2.2 制約概念を中心とするモデリング法

制約概念はシステムを構成するオブジェクト間の相互作用を表現するモデル化手法として有用である[15]．相互作用を制約式の集合によって宣言的に表す一方，制約解消や制約充足を行うツールを用いることで制約処理を自動化することができる．たとえば，Alloy [14] は静的な情報構造の表現方法として，さらに，解析の方法として，制約概念を用いている．先に述べたように，Alloy は検査アルゴリズムを決定可能とするために有界探索の方法を採用した．

データ制約の方法を活用することでシステムのデータに関わる側面を記号的に表現することが可能になる．そのため，データの側面を扱うために制約解消系を統合したモデル検査ツールの研究がある[1),3)〜5),8),22)]．制約を用いることで無限のデータドメインの値を記号的に表現する．想定するアプリケーション領域の特徴を書き表すために十分な表現力を持ち，同時に，決定可能な制約解消系が存在するような制約言語をうまく選ぶことが要点である．

さらに，状態空間の解析という観点からは，アプリケーション制約を満たさない部分を探索範囲から外すことによって探索効率を向上させることができるという長所もある．このようなアプリケーションの性質を活用した状態空間探索の効率化法は Bogor [21] のようなモデル検査法フレームワークでも採用されている．

ある程度広い範囲のアプリケーションに対して有効であって，かつ，制約充足性判定が決定可能な制約言語を考案することは重要である．しかし，本稿では，モデリング記法が 1 つの制約言語を提供するのではなく，必要に応じて制約言語をプラッグインするという考え方にたつ．すなわち，アプリケーション分析の結果として得られたドメイン専用の制約言語を組み込めるようにする．これは，CCS を基本的な振舞い記述に用いる一方，暗号方式などを表現するデータに関わる理論をパラメトリックにする CryptoCCS [16] と同様な考え方である．

本稿では，データの側面が重要となるシステムの振舞い仕様を解析する LFM ツールを実現するという目的に対して，プラッグイン可能な制約言語という考え方を持つ有限状態遷移のモデリング記法，すなわち，制約オートマトンを考える．

### 2.3 Maude を用いる形式仕様

従来より，代数仕様の技術は，高度なデータ構造を数学的に厳密に，かつ簡明に表現する方法として有用であることが知られている．たとえば，OBJ 系の形式仕様言語[11] では，リストや木構造の抽象データ型を帰納的に定義し，等式論理によって解釈を与える．したがって，ソフトウェアシステムのデータを中心とする静的な側面を厳密に規定することができる．

さらに，Meseguer の書き換え論理[17] の考え方を持つ代数仕様言語では，動的な振舞いも表現可能である．たとえば，Nakajima ら[18] は，静的な側面であるオブジェクト図と動的なコラボレーション図を組み合わせたオブジェクト指向デザインを CafeOBJ によって表現できることを示した．実行可能な CafeOBJ 記述を得ることが主眼であるためツール支援としては機能確認に重点が置かれていた．書き換え論理を用いることで構造ならびに振舞いの両方の側面を表現できることを示したといえる．

Meseguer の研究グループが開発した Maude[7] はメンバシップ等式論理と書き換え論理に基づく代数仕様言語である．Maude 仕様はモジュール定義の集まりからなる．等式仕様からなるモジュールは関数モジュール（functional module）`fmod` と呼ばれ，OBJ の等式論理を拡張したメンバシップ等式論理で意味を与えられる．システムモジュール（system module）`mod` は

書き換え規則と等式仕様の双方を持つことができ，書き換え論理で解釈される．抽象データ型は `fmod`，状態遷移システムは `mod` として作成する．

Maude は書き換え規則が生成する状態空間を解析する方法として，幅優先探索に基づく到達性解析法や線型時相論理 LTL（Linear Temporal Logic）モデル検査ツール[9] を提供している．これらの機能を活用した研究として JavaFAN [10] がある．また，Ogata and Futatsugi [20] は，帰納的に定義されたデータ構造が仕様記述ならびに解析で重要な役割を果たす例で Maude が有用であったことを報告している．

本稿では，Maude を用いて制約オートマトンを表現する方法を提案する．基本的な方針は以下のとおりである．第 1 に，チャネル通信機構を有する有限状態オートマトンをシステムモジュールの仕様記述として作成する．第 2 に，制約言語を関数モジュールの仕様記述として作成する．制約言語モジュールのインタフェースの役割を果たすオペレーションを明確に決定することで，同一インタフェースを持つ異なる制約言語モジュールをシステムモジュールにプラッグイン可能とする．

本稿で提案する制約言語の考え方は，制約論理プログラミング CLP でデータ独立オートマトン[24] を定義した方法[22] に影響を受けた．Arbab ら[1] はデータ独立オートマトンを制約の考え方で定義し，また Maude を用いる表現方法を与えた．本稿の制約オートマトンという用語は基本的な考え方を簡便に表現しているので，文献 1) から借用した．しかし，制約オートマトンの定式化（3 章）ならびに Maude 上での表現方法（4 章）や有効な使い方（5 章）は独立に考案したものである．

なお，制約言語を関数モジュールとして表現するので，制約言語の評価によって書き換え規則が生成する状態空間に影響を与えることはない．したがって，制約言語の導入は，モデル検査法などでつねに問題となる状態空間の大きさに悪い影響を与えない．

## 3. モデリング記法

### 3.1 制約オートマトン

本稿で提案するモデリング記法は制約概念を持つ有限状態オートマトンであり，制約オートマトンと呼ぶ．**図 1** は制約オートマトンの基本となる Mealy 型状態遷移システム[23] の断片を示す模式図である．状態はシステムの実行制御ポイントに加えて，変数を管理する大域的なメモリならびに大域的な制約集合から構成される．ここで制約集合は，別途定義された制約言語の式の集まりである．

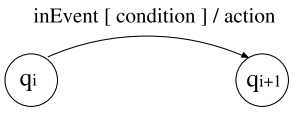


**図 1** Mealy 型状態遷移システム
Fig. 1 Mealy-style state transition system.

状態遷移ラベルには以下の情報を付加する．

- `inEvent`
  当該遷移を可能にする入力イベントを示す．
- `condition`
  当該遷移の発火条件を与えるガード条件を示す．ガード条件は大域的なメモリが保持する変数値を参照することができる．図 1 に示した遷移は，入力イベント `inEvent` が存在し，かつ `condition` が真となるときに限り，発火してゆき先状態に移る．ところが，ある状態で参照する制約集合は保持する制約式が互いに矛盾するなどの理由によって充足不能となる場合がある．制約集合を満たさない状態に達したため，これ以降の遷移は無用である．そのためには，遷移発火条件の一部として，制約充足可能性検査 `satisfiable` を行えばよい．先に述べたガード条件を `guard` と表すと，遷移発火条件は `[ guard and satisfiable ]` となる．
- `action`
  状態遷移にともなって実行するアクションを示す．遷移先の状態に実行の効果を与える．アクションには新しいイベントの生成，メモリ内変数値の更新，などを含む．
  さらに，`action` は制約集合を操作することもできるとする．遷移前の状態において充足可能であれば遷移先に伝播する．また，状態遷移にともなって新しい制約式を追加することも可能とする．
  次に示す例は変数値の更新と新しい制約式追加を表す `action` の具体的な記述例である．

  ```
  x:=x-1; { y!=z }
  ```

  変数 `x` は代入を表すシンボル (`:=`) で更新されることを示す．また，`{` と `}` で囲んだ新たな制約式 `y!=z` を追加することを示す．ここで，制約集合は保持する要素の制約式の論理積と見なした．

上記の基本的な定義では，制約充足可能性の検査はすべての遷移について行うようになっている．しかし，制約言語の性質によって充足性判定の手間が大きいため，毎回の検査は避けたい．そこで，不要な検査を避けるために，新しい制約を追加するときだけに充足性検査を行う．新たな制約式 `C` を制約集合 `X` に追加する

関数 add(X,C) を定義した．既存の制約集合 X は冗長でなく，かつ，充足可能と仮定する．

```
fun add(X,C)
= if unsat(X,C) then fail
  else if redundant(X,C) then X else (C and X)
```

式 (C and X) は C と X の論理積（∧）を示す．

上記で，述語 unsat(X,C) は X に C を追加することで充足不能となる場合に真となる．さらに，述語 redundant(X,C) は既存の制約集合 X に C を追加しても新たな情報が増えない場合，すなわち，X に対して C が冗長である場合に真となる．これら 2 つの述語は，使っている制約言語ごとに定義可能であると仮定する．

関数 add(X,C) は述語 unsat(X,C) が真となるとき，制約集合が充足不能であることを示す特別なシンボル fail を返す．したがって，状態遷移発火の検査にともなって行うべき処理 satisfiable は単に fail シンボルの有無を調べるだけでよい．関数 add(X,C) は新しい制約を追加するときのみに評価するので，上に述べた場合に発生する状態遷移ごとの処理負荷を軽減することができる．

### 3.2 プラッガブル制約言語

本稿の制約オートマトンはアプリケーションドメインに特化した制約言語をプラッグインするという点でパラメトリックになっている．制約言語が満たすべき性質は先に述べた 2 つの述語 unsat(X,C) と redundant(X,C) とが決定可能であるということである．すなわち，充足可能判定が決定可能であり，いくつかの簡単化規則が定義できればよい．逆に，これら 2 つの述語がプラッガブル制約言語の制約オートマトンからみた「インタフェース」となる．以下，具体的に説明するために，簡単な制約言語を対象としてこれら 2 つの述語を定義する．

ここでは等号と非等号（u = v，u ≠ v）のプリミティブ制約を持つ制約言語を考える．制約集合はこれらの論理積である．記号=と ≠ が交換律を満たすとすると，u と v を要素として，制約言語の性質を以下の規則で表現することができる．

(1) u = u
(2) u = v, v = w ⇒ u = w
(3) u = v, v ≠ w ⇒ u ≠ w
(4) u ≠ u ⇒ fail
(5) u = v, u ≠ v ⇒ fail

述語 unsat(X,C) は上記の規則から容易に作成することができる．また，述語 redundant(X,C) は新たに追加する C がすでに X 中にあるか否かを検査するように定義しておけばよい．4.3 節でこの制約言語の Maude 表現を示す．

## 4. Maude を用いる表現

### 4.1 表現方法の概要

Maude の仕様はモジュール定義の集まりである．各モジュールはシグネチャと等式仕様あるいは書き換え規則を持つ．シグネチャはソート記号とオペレータ記号の宣言からなる．等式仕様を持つモジュールは関数モジュール fmod であり，メンバシップ等式論理で意味解釈を与えられる．システムモジュール mod は等式仕様と書き換え規則を持つことができ，書き換え論理で意味づけされる．抽象データ型は fmod として，状態遷移マシンは mod を用いて表現する．

Maude で制約オートマトンを実現する際には以下の点に留意する．第 1 に有限状態オートマトンの表現方法，第 2 にオートマトン間の通信方法，第 3 に制約言語の意味定義，である．

第 1 の点に関しては状態遷移を表現するために適切な（条件付き）書き換え規則を定義した mod モジュールを作成すればよい．Maude の標準的な仕様記述方法[17] に従えば容易に作成することができるが，後の解析を考慮して，到達可能な状態空間が有限になるように注意することが必要である．

第 2 の通信方法に関しては制約オートマトンがイベントを送受信すると考える（図 1 参照）．ランデブ通信や非同期バッファ付き通信など多様な通信機構が考えられるが，いずれも Maude の標準的な方法[17] に従えばよい．

第 3 の制約言語については Maude は制約の概念を持たないので多少の工夫が必要になる．しかし，抽象データ型を導入することができるので，ドメイン特化制約言語を定義することは難しくない．すなわち，抽象データ型の仕様記述法に従って制約言語の構文を帰納的に定義する．次に，制約言語の意味を構文の帰納的な構造に従って与えればよい[11]．ただし，ここでは，言語の実行意味を定義するのではなく，充足可能性の判定と簡単化規則が「意味定義」を与えることになる．本稿では先に述べた 2 つの述語 unsat と redundant を定義することにする．

### 4.2 オートマトン・モジュール

まず，制約オートマトンを Maude で表現する具体的な方法を説明する．Maude で状態遷移マシンを表現する標準的な方法に従って，KERNEL mod モジュールを定義する．システムを構成するオートマトン，メモリ，制約集合，イベントバッファなど，すべての要素を持つ大域的な状態空間を導入する．ここではオートマト

ン以外の要素は他の fmod モジュールで定義されていると仮定する．たとえば，fmod モジュール MEMORY はソート記号 Memory ならびに必要なシグネチャと等式仕様を定義する．その中で CONSTRAINT-SET は制約言語を定義するモジュールで，後に詳細に説明する．

```
mod KERNEL is
  sorts Proc Soup .
  protecting NAME-MODE .
  protecting MEMORY .
  protecting CONSTRAINT-SET .
  protecting EVENTS .
  subsorts Event  Memory < Soup .
  subsorts Proc ConstraintSet < Soup .

  op none : -> Soup .
  op _ _ : Soup Soup -> Soup
              [assoc comm id: none] .
  ...
endm
```

ソート記号 Soup はシステム全体の実行スナップショットを表現する．Soup を Event，Proc，ConstraintSet，Memory の共通スーパソートとしているので，これらの要素を保持管理することができる．さらに，2 つの引数 Soup を結合する並置オペレータ (_ _) には [assoc comm id: none] 属性を付加して宣言し，このオペレータが交換律と結合律を満たし，また定数 none を単位元として持つことを示した．アプリケーション機能を書き表す特定の状態遷移マシンを表現するためには，KERNEL モジュールを取り込んだモジュール，たとえば，FOO を作成する．

```
mod FOO is
  extending KERNEL .

  op [_,_] : Qid Mode -> Proc .
  ...
endm
```

KERNEL モジュールはソート記号 Proc を定義するだけで特に具体的な項を与えなかった．FOO モジュールでは，アプリケーションの振舞いを記述するために適切な情報を持つ生成子オペレータ [_,_] を定義する．

Proc ソートの生成子 [_,_] は必ず次の 2 つの引数を持つとする．第 1 引数はソート Qid のデータで当該状態遷移マシンに付与したユニークな識別子を表す．また第 2 引数はソート Mode のデータで，図 1 の $q_i$ に相当する状態識別子を表す．さらに，アプリケーションの振舞いを表現するために必要な固有の情報を適宜定義する．5.2 節に，そのような Proc 生成子の例を示す．

状態遷移モジュール FOO は，状態遷移を表す書き換え規則を持つ．この書き換え規則はソート Soup に対して定義されており，=>記号の左辺にある Soup 項を右辺の Soup 項に書き換えることを示す．Soup がシステムの実行スナップショットを表現する情報であるので，状態遷移，メモリの書き換え，制約集合の変更，イベントバッファの変化，などを一挙に表すことができる．

図 1 に模式的に示した遷移関係は，次のような書き換え規則のテンプレートとして表すことができる．

```
var M : Memory . var N : Qid .
var X : ConstraintSet .
crl [ N, m ] M X e
     =>  [ N, m' ] after(M) add(X, c) e'
if  satisfiable(X) and guard(M, g) .
```

ここで，小文字の識別子は特定ソートの項を表すとし，e と e' は Event 項，m と m' は Mode 項，c は制約式，g はガード条件を指す．書き換え規則は，条件付き（crl）であって，2 つの条件が満たされるときに，次に示す動作を行う．第 1 に，メモリは書き換えられて after(M) になる．第 2 に，新しい制約式 c が現在の制約集合 X に追加される．第 3 に，新しいイベント e' が生成される．満たすべき条件は if 以降に書かれており，制約集合が充足可能であって，かつ，現在のメモリ M が保持する値に対してガード条件 g が真になることを示す．以上は説明のためのテンプレートである．具体的な記述例を 5.2 節に示す．

### 4.3 制約言語モジュール

制約言語を定義するのは fmod モジュール CONSTRAINT-SET であり，制約言語の構文と意味を与える．構文的な側面についてはプリミティブ制約をオペレータ（op）として宣言する．次に意味を 2 つのオペレータ simplify と satisfiable についての等式仕様で与える．オペレータ unsatisfiable のほうが satisfiable よりも定義が簡明になることが多い．また，関数 simplify は制約集合を等価であって要素数の少ない制約集合に書き換える．

```
fmod CONSTRAINT-SET is
  sorts Constrait ConstraintSet .
  subsort Constraint < ConstraintSet .
  protecting LOGICAL-VALUE .

  op nil : -> ConstraintSet .
  op _ _ : ConstraintSet ConstraintSet
         -> ConstraintSet [assoc comm id: nil] .
  op satisfiable : ConstraintSet -> Bool .
  op unsatisfiable : ConstraintSet -> Bool .
  op simplify : ConstraintSet -> ConstraintSet .
  op add : ConstraintSet Constraint
                              -> ConstraintSet .
  ...
endfm
```

さらに，個々の制約言語では，適切なプリミティブ制約ならびに意味を与えるための等式仕様を具体的に

作成する．以下では，3.2 節に示した簡単な制約言語の定義を例として示す．

3.1 節で議論したように，ここでは，関数 add(X,C) による方法を採用するので，2 つのオペレータ unsat(X,C) と redundant(X,C) を定義することを考える．

第 1 に satisfiable オペレータは，充足不可能を示す fail シンボルを用いて簡単に定義できる．

```
op fail:  -> Constraint .

var X : ConstraintSet .
eq satisfiable(fail) = false .
eq satisfiable(X) = true [owise] .
```

属性 [owise] を持つ等式は他の等式が適用可能でない場合に選択されることを示す．すなわち，fail を持たなければ true とすることを示している．

第 2 に，2 つのプリミティブ制約に対応して 2 つの定数オペレータを導入する．交換律が成り立つことをオペレータ属性（[comm]）により明示した．

```
op eqX:  Value Value -> Constraint [comm] .
op neqX: Value Value -> Constraint [comm] .
```

第 3 に，オペレータ unsat(X,C) に関係する等式仕様を作成するのであるが，3.2 節のルールからただちに得ることができる．

```
var X : ConstraintSet . var C : Constraint .
vars U V W : Value .

eq unsat(X, neqX(V,V)) = true .
eq unsat(eqX(U,V) X, neqX(U,V)) = true .
eq unsat(neqX(U,V) X, eqX(U,V)) = true .
eq unsat(eqX(U,V) X, neqX(V,W))
               = unsat(X, neqX(U,W)) .
eq unsat(eqX(U,V) X,  eqX(V,W))
               = unsat(X,  eqX(U,W)) .
eq unsat(X, C) = false [owise] .
```

1 番目の等式仕様はルール (4) に対応する．以下，2 番目と 3 番目の等式がルール (5) であり，その他はルール (1) と (3) が表す推移関係を表現するための等式仕様である．

オペレータ redundant(X,C) の仕様は新しい制約 C が X にあるか否かを検査するだけである．

```
eq redundant(X, eqX(V,V)) = true .
eq redundant(eqX(U,V) X, eqX(U,V)) = true .
eq redundant(neqX(U,V) X, neqX(U,V)) = true .
eq redundant(X, C) = false [owise] .
```

上記の等式仕様は，Maude が提供する associative-commutative パターンマッチの機能を活用している．先に示したように制約集合 ConstraintSet の並置オペレータ (_ _) に [assoc comm id: nil] 属性を与えたので，ConstraintSet 項に対するパターンマッチはプリミティブ制約の具体的な並び順序にかかわらず網羅的に行われる．この機構のおかげで等式仕様を簡明に直観的に表現することができる．

## 5. ホテル鍵問題

本章では，制約オートマトンの考え方に基づくモデリング技法を使って，Maude による記述と解析の方法を具体的に説明する．例題は Alloy 本[14] で取り扱っている「ホテル鍵問題」である．データ制約と振舞い仕様の双方が重要な役割を果たす面白い例である．

### 5.1 問題記述

Alloy 本[14] の 185 ページに記載されている「ホテル鍵問題」を余計な解釈が入らないように英文のまま示す．

> *Hotel Room Locking Problem* The idea is that the hotel issues a new key to the next occupant, which recodes the lock, so that previous keys will no longer work. The lock is a simple, stand-alone unit (usually battery-powered), with a memory holding the current key combination. A hardware device, such as a feedback shift register, generates a sequence of pseudo random numbers. The lock is opened either by the current key combination, or by its successor; if a key with the successor is inserted, the successor is made to be the current combination, so that the old combination will no longer be accepted.

この問題は，単純であるが，データ制約に関わる性質と同時に実行系列に関する振舞い仕様の両方が絡む興味深い特徴を持つ．

オリジナルの Alloy 記述[14] が示すとおり，制約概念が，データの関係記述と解析の双方の点で中心的な働きをする．鍵の値が全順序制約を満たすことが Alloy 記述の要点であり，また，Alloy による自動解析の基本となっている．

一方，顧客がチェックアウト後にも部屋に入ることがあるかを検査するためには一連のオペレーション系列からなる振舞い仕様に対して性質を調べる必要がある．Alloy 記述では，時間点を明示的に仕様記述に書き表すという仕様スタイルを採用することで振舞い仕様を解析している．しかし，振舞い仕様を解析するために追加した時間点に関わる記述がホテル鍵のアプリケーションに関わる部分と入り組んでおり，そのため，Alloy 記述の可読性が悪い．

本稿では制約オートマトンを用いることで，両方

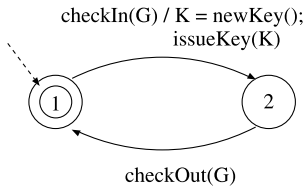


**図 2** フロントデスクの振舞い
Fig. 2 Front desk automaton.

の側面を明確に区別し書き分けることができること，ならびに，Maude の高度な解析機能を用いることで，Alloy と同様に自動解析が可能なことを具体的に議論する．

### 5.2 制約オートマトンによる記述

以下の記述では，ホテルの部屋は 1 つしかないと仮定する．議論したい側面に関しては一般性を失わない．

#### 5.2.1 フロントデスク

**図 2** はフロントデスクの状態遷移図である．イベント `checkIn` を受け取ると，新しい鍵を生成し，同じ顧客からの `checkOut` イベントを待つ状態に遷移する．以下に，フロントデスクの Maude 記述を示す．

```
mod FRONT-DESK is
  extending KERNEL .  protecting KEY-VALUE .

  op [_,_|_,_] : Qid Mode Key Qid -> Proc .
  op r : Proc Key Qid -> Soup .

  var S G1 G2 : Qid . var I K : Key .

  rl checkIn(S,G2) [S,fst|I,G1]
     => r([S,fst|I,G2],newKey(I),G2) .
  rl checkOut(S,G1) [S,snd,|I,G1]
     => [S,fst,|I,G1] .

  eq r([S,fst|I,G1],K,G2)
     = [S,snd|K,G1] issueKey(G2,K,'room) .
endm
```

FRONT-DESK では `[_,_|_,_]` を 2 つのアプリケーション引数を持つ `Proc` の生成子として定義した．第 1 引数（`Key`）は現在の鍵値を，第 2 引数（`Qid`）は最後にチェックインした顧客の識別子を保持する．オペレータ記号 `r` は遷移に対応する書き換え規則を簡明に表現するために導入した補助関数である．図 2 の 2 つの遷移に対応して 2 つの書き換え規則を定義した．いずれも制約集合にかかわらない．

詳細な定義を省略したオペレータ記号 `newKey` は呼ばれるごとに新しい鍵値を返す．具体的な鍵値を生成するのではなく，全順序関係を満たす記号的な値を返す．全順序関係を満たすという制約条件は後に仕様記述の解析を行う際に指定する．

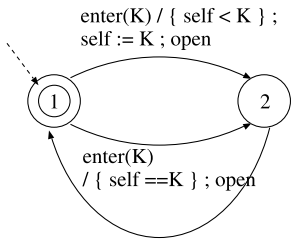


**図 3** 部屋ロックの振舞い
Fig. 3 Room lock automaton.

#### 5.2.2 部屋のロック

**図 3** は部屋のロックの振舞いを示すオートマトンである．状態 1 から 2 へは鍵操作の 2 つの場合に対応して 2 つの遷移を示した．

変数 `self` を現在の鍵値とするとき，遷移ラベル

```
enter(K) / { self < K }; self := K; open
```

は，`enter(K)` イベントが到着したときに遷移が発火すること，さらに，アクションとしては，新しい順序制約（`self < K`）の追加，変数の更新（`self := K`），新たなイベント（`open`）の生成を行うことを示している．

もう一方の遷移は同じイベントによって発火するが，異なるアクションを持ち，新たな等号制約とイベント生成を行う．

```
enter(K) / { self == K }; open
```

これら 2 つの遷移は同一の入力イベント（`enter(K)`）をとるのでいずれの遷移が発火するかが非決定的に決まると考える．

部屋ロックの `Proc` 項は現在の鍵値を保持する必要があるので，生成子の Maude オペレータ表現は次のようになる．

```
op [_,_|_] : Qid Mode Key -> Proc .
```

第 3 引数が変数 `self` として参照した鍵値である．

#### 5.2.3 制 約 言 語

次に制約言語を定義する．鍵値が全順序関係を満たすことを要求されているので，以下では，等号関係，非等号関係に加えて，大小関係 $<$ (*less-than*) を追加した制約言語を定義する．そのためには，3.2 節で示したルールに，$<$ に関わるルールを追加する．

まず，大小関係 $<$ が推移的で，かつ，非対称であることを示すルールを導入する．

(6) $u < v, v < w \Rightarrow u < w$
(7) $u < u \Rightarrow$ fail
(8) $u < v, v < u \Rightarrow$ fail

次に，等号関係と組み合わせたルールを定義する．

(9) $u < v, u = w \Rightarrow w < v$

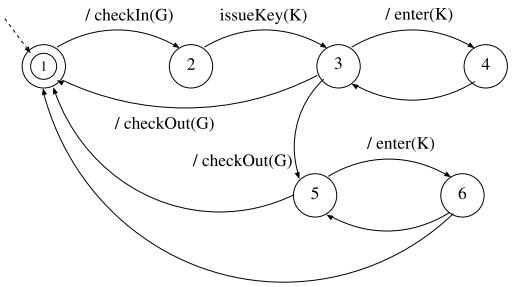


**図 4** 顧客の振舞い
Fig. 4 Guest automaton.

(10) u < v, v = w ⇒ u < w
(11) u < v, u = v ⇒ fail

Maude で表現するためには，先に定義した CONSTRAINT-SET モジュールにプリミティブ制約に対応する < 新たな生成子 ltX を追加する．

```
op ltX : Value Value -> Constraint .
```

さらに，上記のルールから，unsat(X,C) と redundant(X,C) についての等式仕様を追加して，ltX を取り扱えるようにする．

#### 5.2.4 顧客の振舞い

**図 4** は顧客の振舞いを示す．通常の振舞いは状態 1 から 4 までの 4 つの状態で表現できる．状態 3 から 5 にある/checkOut(G) ラベルの遷移を追加すると部屋への侵入に関する振舞いを追加することになる．ここで，侵入とは，チェックアウト後に，返却しなかった鍵で部屋のロックを開けることである．最近のホテルでは，プラスティック製カードの使い捨て鍵を使っているためチェックアウト時に返却する必要がない．したがって，チェックアウト後であってもカード鍵を使用して部屋に入ろうとすることができるのである．

顧客を表現する Proc 項は 2 つのアプリケーション固有の引数を持つので，生成子は次のようになる．

```
op [_,_|_,_] : Qid Mode Key Qid -> Proc .
```

Key の引数はチェックイン時に受け取った鍵の値を示し，第 4 引数の Qid は部屋の識別子を指す．

先に制約オートマトン間の通信はイベント送受信で行うとした．そのため，顧客が部屋のロックを解錠する動作もイベントで行うが，この通信はランデブでなくてはならない．すなわち，部屋のロックは状態 1 から 2（図 3）へ，また，顧客は状態 3 から 4 に同時に変化する．

Maude でランデブ通信を表現するためには，enter イベントに関して複数オブジェクトがからむ相互作用の方法を用いる．以下に，顧客と部屋のロックが絡む複数オブジェクト相互作用の記述例を示す．

```
vars S G : Qid . vars K I : Key .

crl enter(S,K,G) [S,fst|I] [G,fth|K,S] X
=> [S,snd|K] [G,thd|K,S] add(X,ltX(I,K))
if satisfiable(X) .

crl enter(S,K,G) [S,fst|I] [G,fth|K,S] X
=> [S,snd|I] [G,thd|K,S] add(X,eqX(I,K))
if satisfiable(X) .
```

上記の 2 つの書き換え規則は図 3 の状態 1 から 2，ならびに，図 4 の状態 3 から 4 への遷移に相当する．図 3 の状態 1 から 2 への遷移を実現したものなので，5.2.2 項で説明したように制約式の追加を行うアクションを持つように表現した．

### 5.3 Maude を用いる解析

テストデータを与えての仕様実行の機能以外に，Maude は書き換え規則が構築する状態空間を解析する方法を提供している．幅優先探索に基づく到達性解析と LTL モデル検査法である．一般に，この Maude 機能を用いて自動解析が可能であるためには，到達可能な状態空間が有限でなくてはならない．

ホテル鍵問題の Maude 記述を解析する際，状態空間が有限となるようにした．すなわち，3 名の顧客が順次到着し，前の顧客がチェックアウトしてからチェックインするような全体制御を行う「環境」オートマトンを追加定義した．顧客の数は任意であるが，興味ある振舞いを示す最小の数と考えて 3 名とした．

最初に，鍵値に関する制約の考え方が有効であることを確認するためにテスト実行を行った．簡単のため通常の振舞いを示す，すなわち，図 4 の状態 3 から 5 の遷移を除去した顧客を対象とした．この顧客の振舞いに対して Maude の frew コマンドを用いることで，少なくとも 1 つの意図どおりの実行経路が存在することを確認することができる．

テスト実行の際に，初期条件として次の制約式を与えた．

```
ltX('K0,'g1) ltX('g1,'g2) ltX('g2,'g3)
```

ここで'K0 は鍵の特別な初期値を示し，他は順次顧客に与えられる鍵の値を記号的に表現する．上記の制約式は，合計 4 つの鍵値が全順序関係にあることを示す．コマンド frew は非決定的な書き換え規則を公平に選択することで実行し，停止時の最終状態を調べると期待どおりの振舞いを示したことが分かる．

次に，Maude の search コマンドを用いた状態空間解析の効果を示す．上記と同様，通常の振舞いを示す顧客を対象とするが，上記とは異なる初期制約を与えた場合にどのような結果を導くかを調べる．ここでは，鍵値は互いに異なるという弱い制約条件を与える

ことにした．すなわち，全順序の条件を与えていない．

Maude の search コマンドは幅優先探索を行うもので，以下のように，initSoup として与えた Soup 項から探索を開始して targetSoup を探す．

```
search initSoup =>+ targetSoup .
```

ここでは，targetSoup として，3 名の顧客が順次チェックイン・チェックアウトし終わるという状態を指定した．結果は 8 通りの実行経路を求めて，その中には，上記のテスト実行による結果と同一のものを含んでいた．探索結果として得られた状態記述には鍵値に関する制約集合も含む．すなわち，上記のテスト実行では初期条件として与えた全順序制約式を逆に探索結果として求めることができたといえる．

3 番目に時相論理の式で性質を表現し LTL モデル検査による解析を行った．今回は，顧客として，図 4 に示すようなチェックアウト後にも部屋に「侵入」しようとする振舞いを対象とした．初期制約としては先のテスト実行に用いた全順序関係を与えた．

LTL 式により性質を表現するためには，適切な原始命題を導入する必要がある．たとえば，命題 checkedIn(G) は図 4 の状態 2 で真となる．Maude では原始命題をオペレータ記号 |= に対する等式仕様として与える．たとえば，checkedIn(G) は次のように定義する．

```
vars G S : Qid . var K : Key . var Rest : Soup .
eq ([G,snd|K,S] Rest) |= checkedIn(G) = true .
```

このようにして定義した原始命題を用いることで，LTL 式によって振舞いに関わる性質を表現することが可能になる．記号 ~ が論理否定 ¬，/\ が論理積 ∧ とするとき，LTL の式

```
[](~<>unsat ->
      (checkedIn('g2) /\ ~<>entered('g1)))
```

は，制約が充足不能にならない限り，チェックアウトした顧客 1（'g1）は，顧客 2 がチェックインした後，部屋に入れないことをことを示す．特に，制約が充足不能になった場合を表す原始命題 unsat を用いた．

モデル検査ツールはこの性質に対する反例を返し，顧客 2 のチェックイン後も顧客 1 が部屋に入れる場合，すなわち，「侵入」があることを示す☆．Alloy 本に示されているように，この不具合は，顧客がチェックインするイベントと最初に部屋に入るイベントを 1 つの不可分なものとして，他のイベントがインタリーブしないようにすることで修正できる．

LTL モデル検査の 2 番目の例として，チェックインした顧客は，そのうち部屋に入るという性質を考える．

```
[](~<>unsat /\ checkedIn('g1) -> <>entered('g1))
```

この場合もモデル検査ツールは反例を返し，部屋に 1 度も入らないままにチェックアウトする場合があることが分かった．ナンセンスであるが図 4 の顧客はそのような振舞いをするのである．

## 6. おわりに

本稿では状態遷移システムとデータ制約の考え方を統合した制約オートマトンを採用し，代数仕様言語 Maude を用いる具体的な表現と解析の方法を示した．また，Alloy 本記載の「ホテル鍵問題」の記述と性質検証に応用して，制約オートマトンの特徴を議論した．文献 19) に制約オートマトンを用いて分散ソーティング・アルゴリズムの解析を行った事例がある．

Maude は表現力が高いと同時に，高度な状態空間解析法を提供している．そのため，制約オートマトンのような新しいモデリング記法を考える際に，繁雑なツール実現の手間に煩わされないで，モデリング記法の検討に専念できる．以下，提案した方法について，3 つの観点から長所と短所を考察する．

第 1 に，「ホテル鍵問題」の記述と解析の具体例をもとに，モデリング方法の観点から考察する．本稿の方法では鍵値を具体的に与えなくても状態空間の解析ができた．一方，通常のモデル検査ツール[6] を用いる場合，鍵値をツールの入力仕様言語が提供するプリミティブデータを用いて表現しなければならない．鍵値は全順序関係に従うので，たとえば，SPIN[12] を用いる場合，普通は int 型などを用いる．

さらに，制約オートマトンを用いる場合，制約概念を用いることで，対象システムの記述と制約条件を分離することができた．いわば，部分的に規定された初期状態から開始して，モデル検査を行うことに相当する．分離することにより，同一のシステム記述に対して，いろいろな制約条件のもとでの解析を行うことができる．そのような例を 5.3 節に示した．本稿で用いた制約は記号的に表現した鍵値に対する $=$ や $<$ の条件だけであるが，代数仕様の方法を用いることで，より複雑な構造的なデータに関わる制約も表現できると考えている．

第 2 に，特定の制約言語を用いるのではなく，プラッグイン可能とする考え方について考察する．アプリケーション分析の結果として得られるドメイン専用の制約言語を用いることで表現力と決定可能性のバランスを確保するという利点がある反面，適切な制約言語を設計するという負荷が発生する．

☆ Alloy では制約ベースの方法を用いて振舞いを解析し，同様な不具合を見つけている[14]．

実際，適切な制約言語を設計することは，特に，決定可能性の観点から難しい．本稿で用いた制約は単純なもので，本質的に命題論理で表現可能である．他のアプリケーションではより表現力の高い制約言語が必要になり，モデル検査ツールと統合するいくつかの研究がある[3)~5)]．いくつかの標準的な制約言語をライブラリとして用意しておくことが必要であると考えている．

第 3 に，自動的な解析を行うためには，システムは有限でなければならず，これは仕様の作成者が保証すると考えた．本稿の制約オートマトンは有限状態である．一方，Maude は性質検証に必要な状態空間が有限であれば，対象そのものの状態空間は無限になってもよい．したがって，Maude を解析エンジンとして用いる方法では，有限性に対する条件を緩和することができる．

システムが有限状態になるだけでは不十分であり，制約集合の大きさが有限あるいは適切な上限値を持たなければならない．本稿で用いたような単純な制約言語であっても，制約集合が無限に大きくなると解析は停止しない．

5.2 節の例題は，上記 2 つの面で「有限」であるので，自動的な解析が可能であった．すなわち，システムを構成するオートマトンは有限状態であり，かつ，新たな `ltX` 制約を無限に生成することはない．

有限にしなければならないということは，自動検査を可能にすることと表裏一体である．一般に，システムの検証の過程で，有限性を示すことが必要になることが多く，また，有限性の証明が難しい場合も多い．本稿の立場は，2 章で論じたように，LFM の精神に従って，自動化ツールを用いた系統的な不具合発見の方法を考察することである．

**謝辞** 本研究の一部は科学研究費補助金基盤研究（C）17500028 の助成を受けた．

## 参 考 文 献

1) Arbab, F., Baier, C., Rutten, J.J. and Sirjani, M.: Modeling Component Connectors in Reo by Constraint Automaton, *Proc. FLOCASA'03*, ENTCS, Vol.97, pp.25–46 (2004).
2) Bierre, A., Cimatti, A., Clarke, E.M. and Zhu, Y.: Symbolic Model Checking without BDDs, *Proc. TACAS'99*, pp.193–207 (1999).
3) Bultan, T., Gerber, R. and Pugh, W.: Model-Checking Concurrent Systems with Unbounded Integer Variables: Symbolic Representations, Approximations and Experimental Results, *ACM TOPLAS*, Vol.21, No.4, pp.747–789 (1999).
4) Chan, W., Anderson, R., Beame, P. and Notkin, D.: Combining Constraint Solving and Symbolic Model Checking for a Class of Systems with Non-linear Constraints, *Proc. CAV'97*, pp.316–327 (1997).
5) Choi, Y., Rayadurgam, S. and Heimdahl, M.: Automatic Abstraction for Model Checking Software Systems with Interrelated Numeric Constraints, *Proc. ESEC/FSE 2001*, pp.164–174 (2001).
6) Clarke, E., Grumberg, O. and Peled, D.: *Model Checking*, The MIT Press (1999).
7) Clavel, M., Duran, F., Eker, S., Lincoln, P., Marti-Oliet, N., Meseguer, J. and Talcott, C.: The Maude 2.0 system, *Proc. 14th RTA*, pp.76–87 (2003).
8) Delzanno, G. and Podelski, A.: Model Checking in CLP, *Proc. TACAS'99*, pp.223–239 (1999).
9) Eker, S., Meseguer, J. and Sridharanarayanan, A.: The Maude LTL Model Checker, *Proc. 4th WRLA*, ENTCS, Vol.71 (2002).
10) Farzan, A., Chen, F., Meseguer, J. and Rosu, G.: Formal Analysis of Java Programs in JavaFAN, *Proc. CAV 2004*, pp.501–505 (2004).
11) Goguen, J. and Malcolm, G.: *Algebraic Semantics of Imperative Programs*, The MIT Press (1996).
12) Holzmann, G.: *The SPIN Model Checker*, Addison-Wesley (2004).
13) Jackson, D. and Wing, J.: Lightweight Formal Methods, *IEEE Comput.*, Vol.29, No.4, pp.21–22 (1996).
14) Jackson, D.: *Software Abstractions*, The MIT Press (2006).
15) Marriott, K. and Stuckey, P.J.: *Programming with Constraints*, The MIT Press (1998).
16) Martinelli, F.: Towards an Integrated Formal Analysis for Security and Trust, *Proc. FMOODS 2005*, pp.115–130 (2005).
17) Meseguer, J.: Software Specification and Verification in Rewriting Logic, *Models, Algebras and Logic of Engineering Software*, pp.133–193, IOS Press (2003).
18) Nakajima, S. and Futatsugi, K.: An Object-Oriented Modeling Method for Algebraic Specifications in CafeOBJ, *Proc. ICSE'97*, pp.34–44 (1997).
19) 中島 震：制約オートマトンを用いたソフトウェア・デザインの記述法，電子情報通信学会信学技報，Vol.106, No.201, pp.1–6 (2006).
20) Ogata, K. and Futatsugi, K.: Analysis of

the Suzuki-Kasami Algorithm with the Maude Model Checker, *Proc. APSEC'05* (2005).
21) Robby, Dwyer, M.B. and Hatcliff, J.: Bogor: An Extensible and Highly Modular Model Checking Framework, *Proc. 9th ESEC/11th FSE* (2003).
22) Sarna-Starosta, B. and Ramakrishnan, C.R.: Constraint-Based Model Checking of Data-Independent Systems, *Proc. ICFEM 2003* (2003).
23) Wieringa, R.J.: *Design Methods for Reactive Systems*, Morgan Kaufmann (2003).
24) Wolper, P.: Expressing Interesting Properties of Programs in Propositional Temporal Logic, *Proc. POPL'86* (1986).

(平成 19 年 3 月 23 日受付)
(平成 19 年 7 月 3 日採録)

**中島 震**（正会員）

1979 年東京大学理学部物理学科卒業．1981 年東京大学大学院理学系究科修士課程修了．同年 NEC 入社．同社 R&D グループ，法政大学を経て，2004 年情報・システム研究機構国立情報学研究所教授．2005 年から総合研究大学院大学教授を併任．この間，1988〜1989 年米国オレゴン大学客員研究員，2001〜2007 年科学技術振興機構 PRESTO および SORST 研究員，2004〜2007 年北陸先端科学技術大学院大学 JJREX 客員教授を兼務．学術博士（東京大学）．ソフトウェアの形式仕様と検証に関する研究に従事．2002 年度山下記念研究賞，2003 年度日本ソフトウェア科学会論文賞受賞．日本ソフトウェア科学会，ACM 各会員．