ソフトウェア工学 83－6
（1992. 2. 6）

代数的仕様による制御装置仕様の記述と検証

山本純一　　大須賀昭彦　　本位田真一
(株) 東芝 システム・ソフトウェア技術研究所

本稿では、代数的仕様記述言語の記述力と検証能力の評価を目的としたソフトウェア仕様の記述実験について述べる。この実験では半導体ウェハエッチング装置の制御システムを例題としている。まず、「装置の状態」というソートを導入し、装置に対する各操作および装置制御の仕様を状態から状態への関数として記述する。次に、現在試作中の代数的仕様記述支援環境 Metis-AS を用いた定理の自動証明によって、各操作の正しさと制御仕様全体の正しさの検証を試みる。この記述実験の結果、記述対象とした範囲のエッチング装置制御システム仕様が記述でき、定理自動証明による効果的な検証が行なえることが確認できた。

AN ALGEBRAIC SPECIFICATION AND VERIFICATION OF A MACHINE CONTROL SYSTEM

Junichi Yamamoto　　Akihiko Ohsuga　　Shinichi Honiden

Systems & Software Engineering Laboratory, Toshiba Corporation

70, Yanagi-cho, Saiwai-ku, Kawasaki-shi, Kanagawa, 210 Japan

This paper reports on an experiment describing a software specification using an algebraic specification language, whose purpose is to estimate the descriptiveness of the language and the automatic verifiability for the specification. The target of description is a control system of an etching machine for semiconductor wafers. First, a sort 'the state of the machine' is introduced, and the specifications of each operation and the control system for the machine are described as functions on the sort. Then, the correctness of each operation and the whole specification are verified by an automated theorem proving method. An algebraic specification-support system Metis-AS is used in the verification. As a result, the specification for the system can be described, and it is confirmed that the specification can be verified effectively.

1 はじめに

ソフトウェアの仕様を形式的に記述するために様々な形式的仕様記述言語が提案されている。一般に形式的仕様記述には厳密な仕様記述ができ、また、正しさの検証やプログラムへの自動変換が行なえるなどのメリットがある。

そのような形式的仕様記述言語の一つに代数的仕様記述言語 [2] がある。代数的仕様記述言語では仕様を等式の集合で記述する。等式には「左辺と右辺が等しい」という論理的な意味と「左辺は右辺に変換される」という計算の意味がある。そのため、代数的仕様は等式論理によって数学的に解釈できると同時に、計算の規則として手続き的に解釈することもできる。これにより、仕様の正しさの検証や仕様の実行、プログラムへの自動変換などが可能となる。代数的仕様記述言語はこのように高度な検証能力とプログラムへのつながりの良さを併せ持つため、形式的仕様記述言語の中でも有望なものと考えられている。

しかし、まだ実システムの記述例はそれほど多くなく、特に、実システムの仕様に対する自動検証例は少ない。そこで、我々は代数的仕様の記述力と自動検証の有効性を確認するために、半導体ウェハのエッチング装置の制御システムを例題とした仕様記述・検証実験を行なった。今回の実験では、代数的仕様記述を用いたソフトウェア開発を支援するための支援環境 Metis-AS を利用した。Metis-AS では、仕様の直接実行や自動検証などが行なえる。

第 2 章で代数的仕様記述言語と Metis-AS による仕様検証の概要を述べる。ついで、第 3 章では記述実験の例題となる装置について説明する。第 4 章では例題の仕様記述の説明、第 5 章では Metis-AS による自動検証結果の報告をする。最後に、第 6 章でこの記述実験結果を検討する。

2 支援環境 Metis-AS

この章では、Metis-AS における代数的仕様の記述法、および、仕様の検証方法について説明する。

2.1 Metis-AS における仕様の記述法

代数的仕様記述言語では、まず記述する世界を構成するデータをその種類 (sort) により分類し幾つかの集合に分割する。そして、どのデータとどのデータが等しいのかを等式による公理で定義して、記述する世界の意味を与える。例 2.1 に加算の定義された自然数の代数的仕様を挙げる。

例 2.1
```
object integer has
  sort:
        nat;
  constructors:
        0-->nat;
        suc(nat)-->nat;
  function:
        nat+nat-->nat;
  axioms:
        X+suc(Y)==suc(X+Y);
        X+0==X;
end.
```

構成子 (constructor) とは自然数 nat を構成する関数である。一般の代数的仕様では関数を構成子と演算子 (function) に区別しないが、Metis-AS では潜在帰納法を用いた仕様検証を行なうため、ソートを構成する関数を明確にする必要がある。この構成子に関して仕様は次の 2 つの条件で表される定義原理を満たさなければならない [1] :

1. 全ての定項はある構成子項と同等関係にある。
2. 相異なる構成子項の間に同等関係は成立しない。

2.2 Metis-AS による仕様の検証

Metis-AS の主な機能は

- 仕様の構文解釈
- 項書換えによる仕様の実行
- 仕様の完備化
- 定理証明による仕様の自動検証
- 仕様の Prolog プログラムへの自動変換

などである。以下で Metis-AS を用いた仕様の検証方法を概説する。

2.2.1 仕様の構文解釈

仕様の構文解釈では、宣言された関数の型と公理に書かれた関数の引数ソートおよび値ソートとの整合性の確認、および、関数や公理の宣言洩れなどのチェックが行なわれる。仕様中にこれらのエラーが存在した場合、そのエラーの発生箇所が指摘される。

2.2.2 項書換えによる仕様の実行

公理の等式を左辺から右辺への書換え規則とみなすことで、代数的仕様を直接実行できる [3]。これによって記述の正しさをある程度確認できる。例 2.1 の記述においては、

```
  suc(0)+suc(0)
→ suc(suc(0)+0)      1 つ目の公理より
→ suc(suc(0))        2 つ目の公理より
```

などの計算 (1 + 1 = 2 を表す) が行なえる。

2.2.3 仕様の完備化

自然数の例では等式を左辺から右辺への項書換え規則とみなすだけで、完備な項書換え系となる。しかし、一般にはこれだけで完備な項書換え系が得られるとは限らない。Metis-AS には Knuth-Bendix の完備化アルゴリズム [3, 4] が備わっており、自動的に完備な項書換え系が得られる。完備化の際、仕様が定義原理を満たしているかどうかが確認される (ただし、全ての定項がある構成子項と等しい、という点に関しては簡易的な確認のみ) 。

2.2.4 定理証明による仕様の自動検証

Metis-AS には潜在帰納法 [1] を拡張した自動検証アルゴリズムが備わっており、数学的帰納法を必要とするものまでを含めて等式で記述された定理の証明が自動的に行なえる。検証の際の仕様の宣言的意味は始代数で与えられる [2]。始代数とは変数を含まない項の集合の要素のうち、仕様の公理から等しいことが導かれる要素は同一とみなした代数である。

仕様が満たすべき性質を等式で記述し、その等式の成立を証明することで仕様の正しさを検証することができる。例えば、例 2.1 の加算が結合法則を満たすことを検証するには、定理

X+(Y+Z)==(X+Y)+Z

を Metis-AS に与え、この成立を自動証明させればよい。

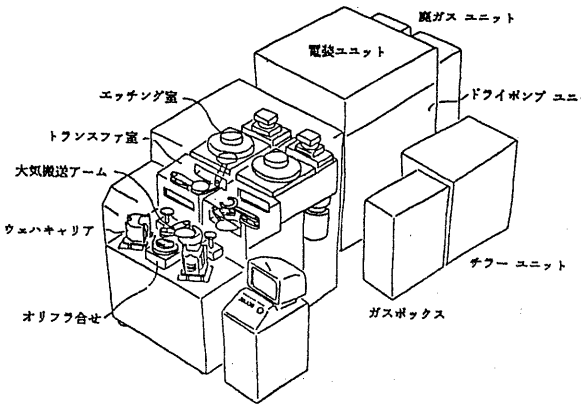

図 1: エッチング装置

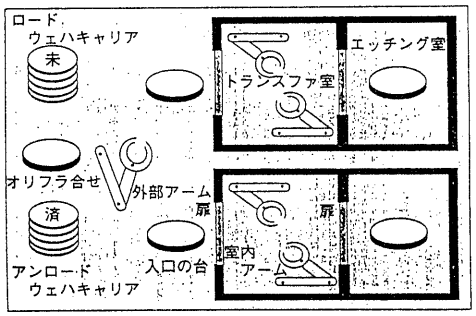

図 2: エッチング装置略図

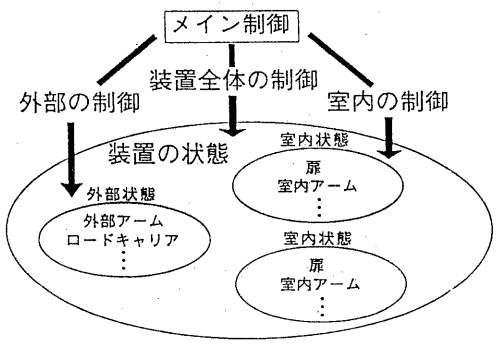

図 3: エッチング装置制御仕様の構成

3 半導体ウェハのエッチング装置

仕様記述実験の例題として半導体ウェハのエッチングを行なう装置（図1）の制御を取り上げた。この装置にはエッチングを行なう部屋が二組あり、それぞれ、内部はトランスファ室とエッチング室に分かれている。各部屋の中にはアームが二つずつあり、また、部屋の境には扉が付いている。部屋の外部には１つの大気搬送アーム（外部アーム）とオリフラ合せ、エッチング前のウェハ・エッチング後のウェハを置くためのウェハキャリアがある。

複数枚のエッチング前ウェハをロードウェハキャリアに置いて作動させると、ウェハを外部アームでエッチング室に運び、エッチングを行ない、完了したウェハを再びアームで取り出す、という動作をし、最終的には全てのウェハのエッチングが行われる。ただし、エッチングを行なうには真空状態が必要なため、扉の開閉や気圧の上下などの動作も必要となり、１枚のウェハは次の順序でエッチングされる（図２参照）。

1. 外部アームがロードウェハキャリアからエッチング前ウェハを１枚取り、オリフラ合せに置く。
2. オリフラ合せでウェハの向きが修正される。
3. 外部アームがオリフラ合せ上のウェハをトランスファ室入口の台の上に置く。
4. 室内アーム１が入口の台上のウェハをトランスファ室に移動する。（トランスファ室を大気圧にし、外部扉を開け、室内アーム１でトランスファ室内にウェハを移動し、外部扉を閉め、トランスファ室を中真空にする。）
5. 室内アーム１がトランスファ室内のウェハをエッチング室に移動する。（内部扉を開け、室内アーム１でエッチング室内にウェハを移動し、外部扉を閉める。）
6. エッチング室を真空にし、エッチングをする。
7. 室内アーム２がエッチング済みウェハをトランスファ室に取り出す。（５の逆）
8. 室内アーム２がトランスファ室からトランスファ室入口の台上にウェハを取り出す。（４の逆）
9. 外部アームが台上のウェハをアンロードウェハキャリアに移動する。

4 仕様の記述

上記エッチング装置制御仕様を代数的仕様で記述する。記述する範囲は前章で述べたウェハの移動に関する制御の部分である。

すでに装置の構成および装置各部の動きは決まっており、ハード面での設計の自由度はない。そこで最初に、ウェハを掴むためのアームや部屋の扉など装置を構成している各部品の仕様を記述し、次にこれらの部品を組み合わせて装置全体の状態とその基本的な操作の仕様を記述した。そして最後に制御部の仕様を記述した。すなわち、仕様を記述を以下の３段階で記述した：

1. 各部品の仕様
2. 装置の状態に対する各操作の仕様
3. 各操作を組み合わせた装置制御の仕様

なお、例題の装置にはエッチングを行なう部屋が２つあるため、エッチングを行なう部屋の状態と部屋の外部の状態を別々に記述し、部屋の状態２つと外部の状態１つを組み合わせて装置全体の状態を構成した。制御の仕様についても同様に、室内状態の制御、外部状態の制御を組み合わせて装置全体の制御を記述した（図３参照）。

以下で、この段階順に具体的な記述について説明する。

4.1 各部品の仕様

まず、エッチング装置を構成する各部品（外部アーム、エッチング室の扉など）の仕様を記述する。

```
object room_state has
  use:
     booo; in_arm; air; door; e_table; t_table;
  sorts:
     room_state;
  constructors:
     room(in_arm,in_arm,t_table,e_table,air,air,door,door)-->room_state;
     undef_R-->room_state;
  functions:
     open_t_door_R(room_state)-->room_state;
     pre_open_t_door_R(room_state)-->bool;
  axioms:
     open_t_door_R(room(A1,A2,Tt,Et,Ta,Ea,Td,Ed)) ==
               if   pre_open_t_door_R(room(A1,A2,Tt,Et,Ta,Ea,Td,Ed))
               then room(A1,A2,Tt,Et,Ta,Ea,open_door(Td),Ed))
               else undef_R;
     open_t_door_R(undef_R)==undef_R;
     pre_open_t_door_R(room(A1,A2,Tt,Et,Ta,Ea,Td,Ed))==check_high(Ta);
     pre_open_t_door_R(undef_R)==undef_B;
end.
```

図 4: 室内状態の仕様（一部）

ここでは、トランスファ室・エッチング室の「扉」を例に説明する。他の部品に関しても同様である。

この扉は室内の気圧を真空に保つために普段は閉まっており、ウェハの出し入れの際にのみ開く。しかし、この段階では装置の他の部分との関係は考慮せず、扉単体で考える。基本的には、部品の状態を構成子、部品に対する操作を演算子として記述している。

```
object door has
  use:
    bool;
  sorts:
    door;
  constructors:
    open-->door;
    closed-->door;
  functions:
    open_door(door)-->door;
    close_door(door)-->door;
    check_open(door)-->bool;
  axioms:
    open_door(X)==open;
    close_door(X)==closed;
    check_open(open)==true;
    check_open(closed)==false;
end.
```

この扉の仕様では、door というソートが新たに導入される。構成子は扉が取り得る状態を表しており、開いている状態 open、閉じている状態 closed の 2 通りの状態を持つ。演算子によって、扉を開ける操作 open_door、扉を閉める操作 close_door、扉の開閉を調べる操作 check_open の 3 つの操作が表される。公理には、これらの操作の意味（「扉を開く操作をすると扉が開く」など）が記述されている。

4.2 装置の状態に対する各操作の仕様

装置の仕様を記述するため、前節の各部品を組み合わせて「装置の状態」というソートを考える。装置に対する操作は状態から状態への演算子として記述する。

ここでは説明のため、室内状態のごく一部を例に挙げる（図 4）。この仕様は室内状態に対してトランスファ室の扉を開ける操作のみを記述したものである。実際の室内状態の仕様にはこの種の操作が２３個ある。また、エッチング室の扉が開閉などを問い合わせるための論理的な関数が２５個、その他の関数が数個記述されている。

この仕様では装置室内に関わる種々の部品を用いて、room_state という室内状態を表すソートを構成している。room_state の構成子は、関数 room と定数 undef_R である。任意の室内状態は room(_,_,_,_,_,_,_,_) という形（ただし、引数は構成子項）か、または undef_R に等しい。undef_R は未定義状態を表す構成子である。

「トランスファ室の扉を開く」操作は open_t_room_R という演算子で表される。また、この操作を行なう際に室内状態が満たしているべき前提条件は pre_open_t_room_R である。公理には、「トランスファ室の扉を開く操作をすると、トランスファ室の気圧が大気圧であれば扉が開き、そうでなければ未定義状態に陥る」ことが記述されている。なお、ここでは「扉」単体の仕様で定義された open_door を利用している。

なお、この仕様は装置の手動操作が行なえる段階にある。例えば、トランスファ室入口にあるウェハをトランスファ室に運ぶには、「トランスファ室の気圧を大気圧にし、扉を開け、室内アームを室外に移動し、ウェハをつかみ、再び室内アームを室内に移動し、扉を閉める」という動作を順次行なえばよい。

4.3 装置制御の仕様

前段階における個々の操作を適当に組み合わせて装置全体の制御の仕様を記述する。ここでは新たなソートは導入せず、すでにある装置の状態の仕様に制御用の新たな演算子を追加する。説明のために装置制御の最上段の一部を示す（図５）。

主に演算子 trans で装置の制御を行う。trans は装置の状態から装置の状態への演算子で、制御による状態遷移を表す。この演算子に初期状態を入力値として与えると、入力された状態が公理に書かれた遷移規則に従って変形され、最終的には終了状態か未定義状態が出力される。

```
object control has
  use:
      bool; carrier;
      room_state; outside_state; machine_state;
      control_room_state; control_outside_state;
  functions:
      trans(machine_state)-->machine_state;
      synch_ctr(machine_state)-->machine_state;
      parallel_ctr(machine_state)-->machine_state;
      initial_state(carrier)-->machine_state;
  axioms:
      trans(State)==if   final_state(State)
                    then State
                    else trans(synch_ctr(parallel_ctr(State)));
      parallel_ctr(machine(Out,RoomA,RoomB))==
                    machine(outside_ctr(Out),room_ctr(RoomA),room_ctr(RoomB));
      parallel_ctr(undef_M)==undef_M;
      initial_state(N)==machine(initial_O(N),initial_R,initial_R);
end.
```

図 5: 装置制御の仕様

trans による状態遷移では、まず、parallel_ctr により外部および 2 つの室内各々の一連の動作が並行に行なわれる。ついで、外部と室内が同期をとるべき動作（外部と室内とのウェハのやりとり）がなされる。状態が終了状態になるまでこの動作が再帰的に続き、状態が終了状態になったら遷移は終了する。

演算子 outside_ctr,room_ctr,synch_ctr はそれぞれ外部状態、室内状態、装置全体の制御のモジュールで定義されている。initial_state は初期状態を表す演算子で、initial_state(N) は N 枚のウェハがロードキャリアに載っている初期状態を表す。そして、trans(initial_state(N)) がその初期状態に対する装置の実行を表す。

この仕様の意味を考える。7 枚のエッチング trans(initial_state(7)) に対して、

```
trans(initial_state(7))==trans(st)
trans(st)==trans(st')
      ...
trans(st'')==final_state_7
```

という等式が成り立っているとする。

これは、仕様の公理を左辺から右辺への状態遷移規則とみなして手続き的に解釈すれば、7 枚のエッチング前ウェハが存在する初期状態が様々な状態を経て、最後に 7 枚のウェハのエッチングが完了した状態（final_state_7）になったことを意味する。また、宣言的解釈によると、演算子 trans による制御で移り得る状態に演算子 trans を施したものはすべて等しいということを表している。すなわち、

```
trans(initial_state(7))
trans(st)
trans(st')
   ...
final_state_7
```

は全て等しく、final_state_7 が構成子項による代表元であることを意味している。

5 仕様の自動検証

この章では、エッチング装置制御の仕様に対して Metis-AS を用いて行なった自動検証について述べる。

検証項目としては以下の 4 項目を考えた：

1. 全仕様の構文的エラーチェック
2. 装置に対する各操作の誤り検出
3. 各操作を複数組み合わせた一連の動きの誤り検出
4. 装置制御の正常終了

これらについて順次説明する。ただし、4 に関しては後で説明するように、この仕様に対する検証はできなかったため、単純化した装置の仕様に対する検証実験を行なった。

5.1 構文的エラーチェック

仕様を実際に Metis-AS に読み込んだところ、構文的エラーが約 100 件ほど発生した。ここでのエラーは主に関数名の誤りや型の間違いによるものであった。

5.2 装置に対する各操作の誤り検出

現実のエッチング装置制御プログラムにおいては、各操作の行なわれる前提条件はセンサーの入力に関する論理式で与えられており、その論理式の値に応じて装置に対するしかるべき操作が行なわれる。この際、論理式や操作の記述間違いにより装置が正しく動作しないエラーが発生している。そのようなエラーを仕様の段階で発見し排除するために、各操作の前提条件や操作自体の誤り検出を考える。

具体的に検出したい誤りとは「トランスファ室の扉を開く」操作を例にとると、次のようなものである：

1. 操作の前提条件「トランスファ室の気圧が大気圧である」が抜けている。
2. 扉を開く際に、トランスファ室の扉とエッチング室の扉を間違えて記述している。

これらを検出するためには仕様の直接実行という方法もあるが、それだけでは十分ではない。様々な場合について一つ一つ確かめるのは非効率的であるし洩れも生じる。また、現実の装置における物理的な制約から発生する誤りは、仕様実行だけではなかなか発見できないであろう。

そこで定理証明による自動検証を試みる。証明する定理は、「前提条件を満たした状態で各操作を行なった結果は

設計者の意図に合致しており、かつ、物理的に不可能な状態に陥らない」というものである。ここで、「設計者の意図」とは「トランスファ室の扉を開く操作をするとトランスファ室の扉が開く」などで、各操作を表す演算子の前提条件と同じ形式の論理式で与える。また、「物理的に不可能な状態」とは、「2本の室内アームが両方ともエッチング室内に存在する」状態や「エッチング室の扉が開いているにもかかわらず、エッチング室が真空である」状態などであり、これらも論理式で与える。今回の検証では「物理的に不可能な状態」として8種類の状態を考慮した。

具体的には、以下のようにして検証を行なった。まず、物理的に不可能な状態を表す演算子 wrong_state と、各操作に対する設計者の意図を表す演算子 post_関数名 を仕様に追加する（検証用のモジュールを一つ設けて利用した）。そして、次のような定理の証明を行なう：

```
        pre_関数名(S)
    and not wrong_state(S)
    and well_defined(S)
imply
        post_関数名(関数名(S))
    and not wrong_state(関数名(S))
==true
```

ここで、演算子 well_defined とは、引数が未定義状態でないことを確認するための論理演算子である。

この検証により、上述の検出したい誤り例などの検出が可能である。

5.3 各操作を複数組み合わせた一連の動きの誤り検出

今回の仕様では全体の制御を再帰的に行なう際に、まず各操作を組み合わせた一連の操作を行なう演算子を定義している。例えば、トランスファ室入口にあるウェハをトランスファ室に運ぶには、「トランスファ室の気圧を大気圧にし、扉を開け、室内アームを室外に移動し、ウェハをつかみ、再び室内アームを室内に移動し、扉を閉める」という定まった動作を順次行うことになるため、これら一連の操作を一つの演算子として定義している。そこで、この一連の操作が途中で止まったりせずに正しく動作することの確認を行なった。これらは基本的には上述の各操作の誤り検出方法と同様の方法で検証できる。

5.4 装置制御の正常終了

今回は仕様を代数的仕様記述のみで記述しているため、仕様全体の意味の検証が可能である。全体として検証したい項目は、「エッチング前のウェハを任意のN枚装置に与えると、N枚全てのエッチングが完了する」ことである。これを次の二つに分けて考える：

1. もしも装置の制御が終了すれば、全てのエッチングは完了する。
2. 装置の制御は必ず終了する。

これら二つの項目の定理証明による検証を試みた。ところが、制御用の演算子は if_then_else 関数を用いた多くの操作を組み合わせて再帰的に構成されており、そのため、完備化が終了せず検証ができていない（詳細は次章で考察する）。そこで、原理的には検証可能であることを確認するために、例題の装置を非常に単純化した装置の仕様を考えて検証実験を行なった。

図6 が単純化した装置の略図である。この装置にはエッチング室は一つしかないため、制御構造は単純である。また、この図の上ではアームが描かれているが実際にはそのようなものは考慮しておらず、元の仕様の各部品に当たるものも単純化されている。なお、この仕様のサイズは全体で325ステップ、公理数は108である。

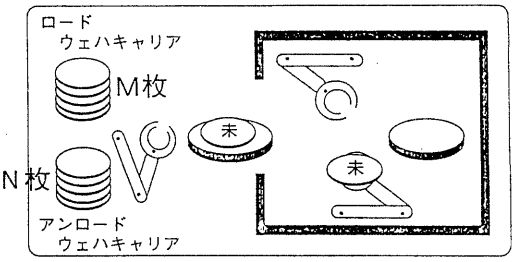

図 6: 単純化した装置

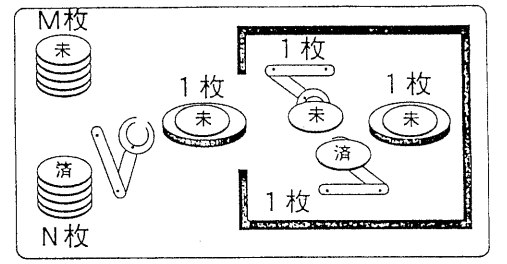

図 7: 途中で止まってしまう状態の例

5.4.1 全てのエッチングの完了

まず、一つ目の項目を考える。この項目は定理

```
trans(initial_state(N))==final_state(N)
```

で表現できる。Metis-AS の定理証明は、定理が仕様の意味する始代数において成立することの証明であり、仕様中の等式の向きは考慮されない。従って、今回の仕様に対してこの定理は「もしも装置の制御が終了すれば、全てのエッチングは完了する」ことのみを表現しており、制御の終了可能性については何も言えない。

単純化した装置に対してこの定理の証明を試みたところ、証明が可能であった。また、故意に誤りを入れた仕様に対して証明をした場合には、(1) 完備化の段階で矛盾が発見されて失敗に終る、(2) 完備化に成功するが定理は不成立、のどちらかによって誤りが指摘された。

5.4.2 装置の終了可能性

続いて、二つ目の項目について考える。

ここでは、制御仕様の公理を左辺から右辺への遷移規則とみなして装置を稼働したとき、「装置が終了状態にたどり着く前に止まってしまわない」ことを検証する。

例えば、エッチング室内に既に3枚のウェハがあるときに外部アームでエッチング室入口にもう1枚ウェハを置いてしまうと、装置はそれ以上動けなくなる（図7）。エッチング前のウェハをエッチング室入口に置く操作の前提条件に「室内に存在するウェハが2枚以内である」という条件が書かれていない場合には、装置は図7の状態に陥って、途中で止まってしまう可能性がある。

制御の演算子は、装置に対する各操作を組み合わせた装置の初期状態から終了状態への関数として記述されている。装置に対する各操作は「前提条件が成立すればある操作を行なう」という形式で記述されている。そこで、「任

	モジュール	構成子	演算子	公理	行数
論理	1	3	5	22	58
部品	11	34	69	202	527
装置の状態	3	6	111	237	748
装置の制御	4	0	38	40	289
合計	19	43	223	501	1622

表 1: 仕様記述結果

意の初期状態から遷移可能な終了状態以外の任意の状態は少なくとも一つの操作の前提条件を満たす」という定理が成り立てば、装置制御の終了可能性が保証される。具体的には次のような定理の証明を行なう：

```
    not final_state(S)
and well_defined(trans(S))
and equal(trans(S),trans(initial_state(N)))
  imply
    pre_f12(S) or pre_f23(S) or pre_f34(S)
 or pre_f45(S) or pre_f52(S) or pre_f26(S)
  == true
```

ここで、f12 などの演算子は単純化した装置における各操作を表し、pre_f12 は f12 という操作の前提条件である。

これにより、上述の誤りの検出が可能であった。制御部の記述の仕方にもよるが、前提条件に上述の条件が書かれていなければ、この定理の不成立か、または、完備化時のエラー発生によって検出される。

6 評価と今後の課題

6.1 記述結果の評価

「装置の状態」というソートを導入し装置に対する操作をこのソート上の演算子として定義することで、今回記述対象とした範囲の仕様記述ができた。表 1に示すように、この仕様は全体で約 1,600 ステップ、500 個の公理、19 のモジュールから成る。

これらの記述には約 100 時間を要した。ただし、今回のような制御仕様の記述に関しては記述指針が全くないため、多くの時間は代数的仕様による記述方針を決めるための試行錯誤に費やされた。実質的な設計・記述に要した時間は約 35 時間である（設計 5 時間、記述 30 時間、検証や修正に費やした時間は含まない）。

6.2 検証結果の評価

Metis-AS を用いた自動検証で見つかった仕様の誤り数および検証に掛かった時間（修正の時間は含まない）は表 2 の通りであった。検証から修正まで全体では約 4 0 時間を要した。また、これらの検証で利用した Metis-AS による検証方法を表 3にまとめておく。

以下で、検証項目毎に検討する。

構文的エラー　表 2に示すように、100 件ほどの構文的エラーが発生している。一般には型の不整合などから仕様の本質的誤りが発見されることもあり得るが、今回の記述実験においては全てケアレスミスによるものである。現在試作中の Metis-AS で記述可能な仕様は単純なモジュールの参照機能のみを持つ非常にプリミティヴなものであり、モジュールのパラメータ化や関数のオーバーローディングなどは許されていない。そのため、各ソート毎に少しづつ異なる似た名前を持つ関数が定義されることになり、混乱が生じたのである。

今後記述力を拡張していく必要があるが、記述の柔軟さと検証能力とはトレードオフの関係にあり、記述力を増していくと検証系に負担がかかる。より柔軟な記述法に対する自動検証方法の確立は今後の課題である。

項目	検証結果	時間
型エラー	約 60 件	
公理・関数の欠如エラー	約 40 件	
各操作の誤り	33 操作中 10 操作に誤り	7
一連の操作の誤り	6 操作中 2 操作に誤り	2
エッチングの完了	未検証	
装置の終了可能性	未検証	
単純化した装置の エッチングの完了	検証可能	3
単純化した装置の 終了可能性	誤り 1 件検出	3

表 2: 仕様検証結果

各操作およびそれらを組み合わせた一連の動きの誤り　各操作の誤りに関する検証を仕様中の全 33 操作に対して行なったところ、10 操作の誤りが検出された。また、各操作を組み合わせた一連の動きに関しては、6 操作中 2 操作から誤りが見つかった。検出された誤りは、ある操作の前提条件におけるエッチング室とトランスファ室の扉の間違い、アームの回転方向の間違い、などである。また、実験のために仕様の中に故意に挿入した誤りも、今回試みた限りでは全て検出可能であった。

これらの誤りは書き間違いや勘違いなど、どちらかと言うと些細な誤りが多い。しかし、これらを修正する際には、自動検証により誤りの存在が指摘されているにも関わらず、誤った箇所がなかなかわからず多くの時間が掛かっている。もしも自動検証によって誤りを指摘されていなければ、仕様の段階で気づくことは困難であろう。この種の誤り検出に自動検証は非常に有効であった。

装置制御の正常終了　前章で述べたように、制御仕様全体に対する検証はできていない。しかしながら、単純化した装置に対する検証実験により自動検証の有効性はある程度確認できた。この検証により全てのエッチングの完了という本質的な性質の保証が得られる。特に、無限通りの状態を持つ装置（装置に与えるウェハの枚数は無限通りである）に関する検証が可能である点は有用である。

6.3 プロセス制御の記述法

今回の仕様は、まず装置の状態に対する個々の操作を記述し、次にそれらを組み合わせて、装置の初期状態から終了状態への演算子として制御仕様を記述した。この演算子は、任意の初期状態が入力されると、装置の各操作に対応する様々な関数を用いて状態遷移を計算し、最後に終了状

	構文解釈	実行	完備化	定理証明
構文解釈エラー	○			
各操作の誤り		○		◎
一連の操作の誤り		○		◎
エッチングの完了			◎	◎
装置の終了可能性			○	◎

○検証に利用可能

◎検証に非常に有用

表 3: Metis-AS による検証方法の利用度

態を出力する。しかし、例えばLOTOSではプロセスの起こる順序の木で仕様の記述をしており、今回の記述の仕方とは明らかに異なる。

今回の記述方法のメリットは、

1. 装置の状態というソートを用いたことで現実世界の装置との対応がうまく付けられ、個々の操作の誤り検出が容易となる。
2. 簡単な定理証明による検証によって、仕様全体としての正しさ（装置制御の正常終了）が原理的には検証可能である。

という点である。特に、潜在帰納法を用いた定理証明により、任意の枚数のウェハが与えられた場合（無限通りの場合がある）の正しさの検証が可能である。

しかし、この記述方法では二つの問題が生じる。

一つ目の問題は、状態遷移の軌跡が残らないという点である。初期状態を入力すると対応する終了状態が得られるだけで、終了状態が得らるまでの状態遷移の軌跡は何も残らない。その遷移の規則は遷移関数が定義されている仕様中に記述されているのだが、遷移過程をより深く解析するために最適な記述法とは言えない。

もう一つの問題は、大きな仕様に対する完備化に莫大な時間が掛かるという点である。図5に示したように、制御による状態遷移を表す演算子 trans は再帰的に定義されており、その定義の中で if_then_else 関数を用いた様々な関数を利用している。これらに完備化アルゴリズムを適用すると、公理中の trans の引数変数 State を具体化した項書換え規則が次々と生成されるため、完備化手続きが発散するわけではないが、変数の具体化が組み合わせ的に爆発してしまい現実的な時間内では手続きが終了しなくなってしまう。そのため、今回の実験では仕様全体に対する自動検証はできなかった。

遷移過程をより深く解析でき、しかも、代数的仕様の持つ定理証明機能を生かせるような記述の仕方を検討しなければならない。

6.4 定理証明の階層化

上述したように、仕様全体に対して完備化を行なうと莫大な数の項書換え規則が生成される。また、これらの規則に出てくる一つ一つの項は非常に長いものとなる。階層的に書かれた仕様の上位のモジュールに対して定理証明を行なう場合、最下位のモジュールまで全ての仕様を用いなければならない。そのため、仕様のサイズが大きい上位モジュールの効率的な検証が不可能になる。

実システムのような大きな仕様を扱うには、仕様の階層構造をうまく反映した定理証明方法が必要である。

6.5 例外の扱い

例外動作に関しては今回の記述の範囲外であり、例外的なことは未定義状態を表す構成子を用いて処理している。例えば、部屋の扉が閉じているときアームを部屋から外へ移動しようとした場合、装置の状態は未定義状態になる。この未定義状態の扱いに関して二つの問題が生じた。

まず、一つ目の問題は記述量の増加である。新たな構成子が増えたことにより、各演算子毎にその構成子に関する公理が必要となる。

もう一つの問題は、あるモジュールであるソートに対して未定義を表す構成子を導入すると、そのモジュールで利用している下位モジュールの別のソートにも未定義構成子を導入する必要が生じることが多々ある、という点である。これにより、(1) 下位モジュールの変更が必要になる、(2) 論理モジュールにも未定義構成子（真でも偽でもない論理値）が導入されるため論理が複雑になり定理の記述が難しくなる、といった問題が生じる。

例外・未定義の扱いは重要な検討課題である。

7 おわりに

本稿では代数的仕様記述言語による記述・検証実験について述べた。その結果、エッチング装置制御システムに対して、今回記述対象とした範囲の仕様が代数的仕様記述言語を用いて記述できた。また、その仕様を現在試作中の支援環境 Metis-AS で処理することにより、効果的な検証が行なえることが確認された。

今後の課題は、検証能力を落さずに記述力を上げること、制御システムなどに対するより適した記述法を検討すること、および、サイズの大きい仕様に対する支援方法を確立することである。

謝辞 エッチング装置制御システムの開発に関する資料の提供および説明をして下さった、(株) 芝浦製作所 相模工場 技術部 浅場部長、(株) 東芝 生産技術研究所 隅田研究主務に深く感謝致します。また、研究の機会と激励をいただいた、(株) 東芝 システム・ソフトウェア技術研究所 西島誠一所長、大筆豊部長に感謝致します。

参考文献

[1] Huet, G. and Hullot, J.-M. Proofs by induction in equational theories with constructors. *J. Comput. Syst. Sci.*, Vol. 25, No. 2, pp. 239–266, 1982.

[2] 稲垣康善, 坂部俊樹. ソフトウェアの代数的仕様記述の理論. In 榎本肇, editor, ソフトウェア工学ハンドブック, pp. 51–91. オーム社, 1986.

[3] 大須賀昭彦. 項書換えシステムと完備化手続き. **bit**, Vol. 20, No. 4, pp. 59–67, 1988. ; または 新しいプログラミング・パラダイム (井田哲雄 編), 共立出版, 第8章 (1989), pp. 165–186.

[4] 坂井公. Knuth-Bendix の完備化手続きとその応用. コンピュータソフトウェア, Vol. 4, No. 1, pp. 2–22, 1987.