ソフトウェア工学 86−15
（1992．7．21）

代数仕様によるプラント制御エキスパートシステムの記述と検証

浦岡 徹　　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

代数的仕様記述を初めとする形式的手法の、実システム開発への適用可能性とその有用性を評価するために、“プラント制御エキスパートシステム”の実現を試みた。開発環境は、並行プログラム開発支援環境MENDELS ZONEである。我々は、関数は代数仕様で、プログラム構造はペトリネットと時相論理で記述し、それらの形式的仕様記述からプログラムを生成した。この開発を通して、形式的手法が実システム開発においても有用であることが確認された。しかし一方で、記述能力拡張などの課題も明らかになった。本稿では、このシステム開発アプローチの中で、特に代数的仕様記述に焦点をあてて報告する。

AN ALGEBRAIC SPECIFICATION AND VERIFICATION OF A PLANT CONTROL EXPERT SYSTEM

Toru Uraoka　Junichi Yamamoto　Akihiko Ohsuga　Shinichi Honiden

Systems & Software Engineering Laboratory, Toshiba Corporation

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

We have developed a plant control expert system using MENDELS ZONE, a development support system for concurrent programs. The objective of this development is to examine the effectiveness of formal methods for development of practical systems. In our system development approach, functions are described by algebraic specification, and module structures are created by Petri nets and temporal logic; then a program is generated from such formal descriptions.

As a result, formal methods are very useful for the development of a practical system, but we recognized several problems of our description framework, too. This paper is focused on algebraic specification in the system development approach.

1 はじめに

代数的仕様記述は、抽象データ型を等式論理に基づき代数として定義する記述法だが、抽象データ型に留まらず広くソフトウェアの仕様記述に用いようと研究されている。

代数的仕様記述は、等式を左から右への項書換え規則とみなすことで、仕様の実行が行なえる。また、Knuth-Bendix アルゴリズムに基づく帰納的定理証明手法 [1],[2],[4] により、仕様の自動検証が可能である。さらに、等式論理から PROLOG への変換の正当性が確認されており [6]、論理型言語への自動プログラム変換が期待できる。以上の、“仕様の実行”、“仕様の検証”、“自動プログラム変換”の可能性から、代数的仕様記述は、形式的仕様記述言語として有望視されている。

我々の研究グループでは、代数的仕様記述支援環境として Metis[5] を開発した。そして、この Metis 環境下で、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・検証実験を行ない、仕様検証の有用性を評価した [7]。さらに、代数仕様を実システム開発に利用するため、Metis と並行プログラム開発支援環境 MENDELS ZONE[3] との統合が図られている。今回、代数的仕様記述を初めとする形式的手法の実システム開発への適用可能性と有効性を評価するために、MENDELS ZONE 環境で“プラント制御エキスパートシステム”を開発した。先のエッチング装置の記述実験が、抽象レベルでの仕様を対象としていたのに対し、今回の開発実験は、仕様からのプログラムの生成を目的としている。本稿では、このシステム開発アプローチの中で、特に代数的仕様記述に焦点をあてて報告する。

2 MENDELS ZONE 上のシステム開発アプローチ

まず、今回の開発事例の中での代数仕様の位置付けを明確にするために、MENDELS ZONE 及びその上でのシステム開発アプローチの概要を紹介しておく。

MENDELS ZONE は、FGCS の研究の一環として試作された並行プログラム開発支援環境で、そのターゲット言語は MENDEL である。新世代コンピュータ技術開発機構 (ICOT) では、汎用の並列プログラミング言語 KL1 を設計し、並列推論マシン PIM の核言語として採用しているが、MENDEL はその KL1 のペトリネットベース拡張言語であり、KL1 への自動変換が可能である。MENDEL のプログラムは、同期をとりつつ並行に動作する複数のオブジェクトから構成される。MENDELS ZONE 上のシステム開発の流れは、大きく分けてオブジェクト生成のフェイズとオブジェクト結合のフェイズからなる [3]。

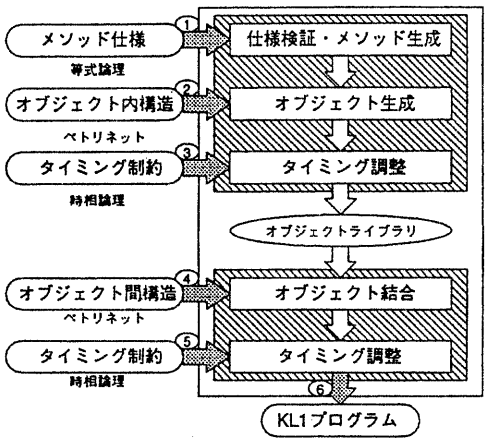

図 1: プログラム生成の流れ

以下では、図 1 中の番号に沿って開発の流れを説明する。

1. オブジェクトのメソッドを関数として捉え、等式論理に基づく代数仕様で記述する。仕様の正しさの検証した後、仕様をメソッドに自動変換する。

2. オブジェクトの内部構造をペトリネット表現で記述する。

3. メソッド間の実行順序に制約がある場合、そのタイミング制約を時相論理に基づく仕様記述言語 TSL で記述する。MENDELS ZONE は、その制約を満たす様にペトリネットの自動調整を行なう。

4. オブジェクトをペトリネットエディタで結合する。

5. オブジェクト間の実行順序に制約がある場合も、メソッド間の場合と同様に記述、調整を行なう。

6. 完成した MENDEL プログラムを、MENDELS ZONE で KL1 プログラムにコンパイルする。

MENDELS ZONE に統合された Metis は、メソッドの仕様記述と生成を支援する。その機能は、以下の項目にまとめられる。

- 仕様の文法チェック、型チェック
- 仕様から項書換え規則の生成
- 項書換えとしての仕様の実行
- 定理証明手法に基づく仕様の検証
- 仕様の論理型言語への自動プログラム変換

3 プラント制御エキスパートシステムの概要

今回の記述対象であるプラント制御エキスパートシステム (以降は、制御システムと呼ぶ) は、現実のプラント制御知識に基づいて動作するエキスパートシステムである。このシステムは、他の研究グループによって KL1 で直接開発されたシステムの一部であり、我々は MENDELS ZONE 環境と形式的手法による開発アプローチを用いてその再実現を試みた。

制御システムの動作は、以下の様にまとめられる。

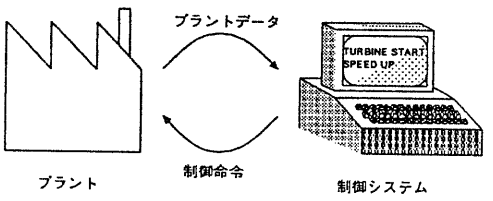

図 2: 制御システムの動作

- プラントの状態を常に監視し、 プラントからのデータをシステム内のプラントデータベースに蓄える。
- 新しく受けたプラントデータとデータベースに蓄えられた 前回値との変化点を検出し、どの制御知識を評価すべきかを決定する。
- 知識を評価し、プラントデータベースに蓄えられた前回評価値と比較して 変化していたら、その知識に対応した命令マクロを展開して、 プラントへの命令送信や新しい知識評価値のプラントデータベースへの 格納を行なう。

以上の動きを繰り返し、知識に従ったプラントの制御を行なう。システムと制御知識は分離していて、知識を入れ換えることで各種の制御モデルが実行できる。

4 オブジェクトとメソッドの抽出

制御システムは、動作の並行性とデータの共有の観点から図 3 の 6 つのオブジェクトに分割される。タイマー部、 DDC 部、アクション部は、内部情報としてそれぞれタイマーテーブル、 DDC テーブル、プラントデータベースと制御知識を持ち、参照や更新を行なう。

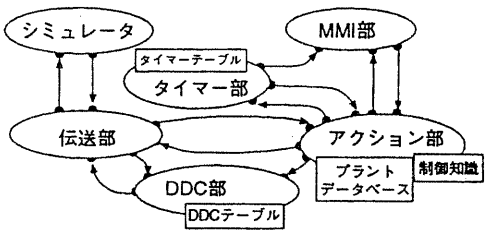

図 3: オブジェクト構成

シミュレータ部 この制御システムは、実際のプラントシミュレータのデータで動作する。 しかし、直接シミュレータに接続するのではなく、 ファイルに保存してあるシミュレータデータを用いる。 シミュレータ部は、そのファイルからデータを読み込み、 伝送部に送信する。 また、伝送部から送られてきた制御命令を表示する。

伝送部 プラントデータを読み込み、 内部表現へ変換し、アクション部と DDC 部へ送る。 シミュレータへ命令を送信する。

アクション部 伝送部から送られるデータとプラントデータベースに保持された 前回値を比較して変化点を検出し、それを基に評価する知識を選択する。 知識を評価し、他のオブジェクトにメッセージを送る。

ＤＤＣ部 アクション部からメッセージを受けると、アクション部とは独立に目標値制御を行なう。 目標値制御とは、特定のプラントデータの値が指定条件を満たすまで、 制御命令を送り続ける操作である。

タイマー部 命令送信を指定秒間遅延させるタイマー処理を行なう。

MMI 部 メッセージの表示、 アクション部の知識評価のモニタリング、 オペレータからのコマンド入力の受け付けを行なう。

個々のオブジェクトについて、さらにその機能をメソッドとして抽出する。例として、アクション部の内部構造を簡略化して図4に示す。

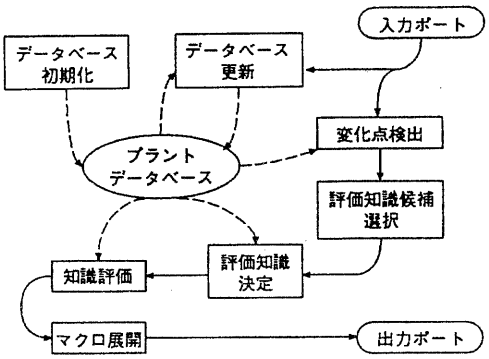

図 4: アクション部の内部構造

オブジェクトは、ポートを通して外部と通信する。メソッドが四角で、データフローが実線矢印で、データベースへの参照と更新が破線矢印で表現されている。

以降の節では、図4中に現れているデータベース更新関数(renew)を例に、代数仕様によるメソッドの仕様記述を説明する。

5 メソッドの代数的仕様記述

代数仕様の特徴は、そのシンプルさと厳格な型にある。その記述は、型(sort)宣言、構成子(constructor)宣言、関数(function)宣言、関数の満たす公理(axiom)からなる。構成子とはその型を構成するもっとも基本的な関数を指す。

データベース更新関数の代数仕様の前に、その入出力データの型の定義する必要がある。まず、idを型として宣言し、データ名、知識名、システムフラグ名を構成子としてidに登録する。登録は、以下の様な定数宣言によって行なう。

定数名 → 型名

idには最終的に200以上のID名を登録した。

```
sort: id;
constructors:
   i10z10 --> id;
   i10z99 --> id;
   i30z10 --> id;
      .......
      .......
```

データがとり得る5つの値's','n','q','p','c' を value として宣言する。

```
sort: value;
constructors:
   s --> value;    n --> value;
   q --> value;    p --> value;
   c --> value;
```

データはidとvalueの組を作る関数で表現する。その参照関係はuseというキーワードで明示する。関数宣言は、

関数名(引数型名,···,引数型名) → 型名

という形で、引数の型と関数の型を明示する。

```
sort: data;
use: id, value;
constructor:
   data(id,value) --> data;
```

データリストは、dataを基にnilとconsで構成される。

```
sort: dlist;
use: data;
constructors:
   nil --> dlist;
   cons(data,dlist) --> dlist;
```

以上の定義から、データリストは例えば以下の様に表現される。

```
   cons(data(i10z10,s),
   cons(data(i30z11,q),
   cons(data(t70z20,n),nil)))
```

データベース更新メソッドは、伝送部などから送られてくるデータでプラントデータベースを更新する関数である。データベースに登録されたデータが前回値と比べ変化した時は、データベース側の値を更新し、データベース側にまだ登録されていないデータが送られてきた時は、新たにそのデータを付け加える。ところで、このシステムにおけるデータベースとは、データや知識評価の前回値が登録されたもので、IDをキーとする検索と更

新ができれば十分である。そこで、ブラントデータベースを単なるデータのリスト表現とみなす。それによりデータベース更新関数は、2つのデータリストを引数にとり、一方のリストで他方のリストを更新したデータリストを出力する関数とみなせる。

以下に、データベース更新関数の定義を示す。axiomsというキーワードの下に、関数間で満たされるべき公理を記述する。この公理が関数の振舞いを決定する。関数replaceは、データベース更新関数の定義に必要な補助関数である。第二引数のデータリストを第一引数のIDで検索し、その値を第一引数の値に置き換える。ただし、第一引数のIDが第二引数のデータリストに含まれない場合、このデータを加える。renewの定義に現れる関数eq_idはidの等号、関数if_dlistはdlistを返す条件分岐だが、その仕様記述は省略する。

データベース更新関数renewの定義

```
functions:
   replace(data,dlist) --> dlist;
   renew(dlist,dlist) --> dlist;
axioms:
   replace(D,nil) == cons(D,nil);
   replace(data(ID1,V1),cons(data(ID2,V2),DL))
   == if_dlist(eq_id(ID1,ID2),
         cons(data(ID1,V1),DL),
         cons(data(ID2,V2),
            replace(data(ID1,V1),DL)));

   renew(nil,DB) == DB;
   renew(cons(D,DL),DB)
   == replace(D,renew(DL,DB));
```

以上の定義を簡単に書き下すと、

1. 関数replaceは、検索すべきデータリストが空なら、第一引数のデータのみからなる一要素リストを返す。
2. そうでなければ、データリストのトップのデータと第一引数のIDを比べ、等しければ値を置き換え、等しくなければ値をそのままにして同じ処理を残りのリストに繰り返す。
3. 関数renewは、第一引数のデータリストの全データについて、第二引数のデータベースに対するreplaceを行なう。

6 仕様の検証

メソッドの仕様が記述できたら、それをMetis環境でKnuth-Bendix完備化手続きにかける。完備化とは、等式として書かれた仕様中の公理から項書換え規則を生成する手続きである。生成された項書換え規則で書換えられた任意の項は、必ずただ一つの正規形に到達する。完備化の後に、記述者は仕様の満たすべき性質を検証項目として与える。記述された仕様上でその検証項目が成立しているかどうかが、Metis 環境で検証できる。Metisの記述法では、関数の中からその型を構成する基本的関数を構成子として別に宣言したが、その存在により仕様の検証がより効率的に行なえる[4]。

先のデータベース更新関数の仕様に対しては、次の様な検証項目が考えられる。

検証項目(1)

> IDがデータリストDLに含まれるならば、DL内のIDの値とDLでデータベースDBを更新した後のIDの値は等しい。

```
element(ID,DL) imply
eq_v(search(ID,DL),search(ID,renew(DL,DB)))
   == true;
```

検証項目(2)

> IDがデータリストDLに含まれないのならば、データベースDB内のIDの値は、DLによる更新によって変化しない。

```
not element(ID,DL) imply
eq_v(search(ID,DB),search(ID,renew(DL,DB)))
   == true;
```

ただし、検証項目に現れた以下の関数の仕様記述は省略する。

element : 第一引数の ID が第二引数のデータリストに含まれているかを判定する述語
search : 第二引数のデータリストを第一引数の ID で検索してその値を返す関数
eq_v : value の等号
imply : データ型 bool 上で定義されている。A imply B == (not A) or B;

検証手続きは Knuth-Bendix 完備化手続きの応用だが、その共通の問題点は、手続きが必ずしも停止しないことである。データベース更新関数について与えた二つの検証項目についても、このままでは手続きが発散してしまう。しかし、検証手続きの発散に何らかのパターンが見い出せる場合、補題を加えることで検証手続きが停止する。これらの検証項目を証明するのに、以下の補題を必要とする。

検証手続きに必要な補題

データリスト DL の ID2 の値を V に置き換えた後、ID1 で検索する。もし、ID1 と ID2 が等しいならば、検索値は V である。そうでなければ、置き換え前の DL の検索値と等しい。

```
search(ID1,replace(data(ID2,V),DL)) ==
  if_v(eq_id(ID1,ID2),V,search(ID1,DL));
```

ただし、if_v は value を返す条件分岐である。

この補題及び上述の二つの定理が記述した仕様上で満たされることが、Metis 環境で証明可能である。

7 仕様と検証項目の関係

先に示したデータベース更新の定義は、関数間の関係を記述したもので、その実行アルゴリズムを記述したものではない。しかし、データベース更新関数 renew と置き換え関数 replace の関係を記述することで、暗黙の内にデータベースの更新は置き換えによって実現されることになる。この意味からすれば、記述した renew の代数仕様は、形式化された要求仕様というより形式化されたアルゴリズム仕様といえる。一方、検証項目はデータベース更新関数の持つべき性質を直接表現しており、記述者の要求により近い。ただし、例示した様な検証項目から実行アルゴリズムは導けない。

そこで、仕様と検証項目の関係は以下の様にまとめられる。従来、記述者は自分の持つ要求からアルゴリズムを考え出しそれを直接プログラミングで実現した。代数的仕様記述は、そのアルゴリズムと一意に対応する形式的表現であり、その対応付けを行なう手続きが完備化である。仕様によって terminology が決定すれば、それを使って要求を検証項目に翻訳する。検証機能により仕様と検証項目の整合性が確かめられれば、アルゴリズムの正しさが保証される。要求に対し実現方法が一つでないのと同様に、検証項目を満たす仕様は複数存在し得る。

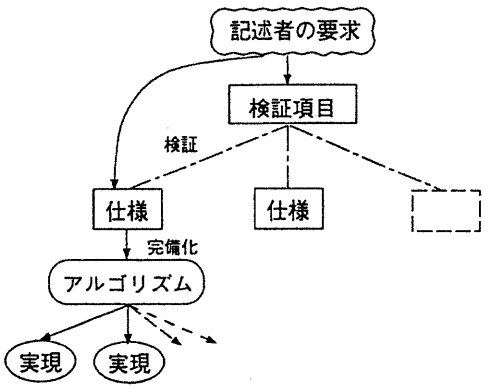

図 5: 仕様と検証項目の関係

8 仕様からプログラムへ

検証によって仕様の正しさが確認されたら、各仕様はメソッド、すなわち並列プログラミング言語 KL1 で記述されたプログラム部品に変換される。この変換は、仕様から生成された項書換え規則の KL1 への翻訳である。例として、データベース更新関数の KL1 プログラム部品を示す。プログラム中に現れる他の述語に関しても、同様に変換される。

---

replace と renew の KL1 プログラム部品

```
replace(A,nil,B):-true|B=cons(A,nil).
replace(data(A,B),cons(data(C,D),E),F):-true|
    eq_id(A,C,G),
    replace(data(A,B),E,H),
    if_dlist(G,cons(data(A,B),E),
                cons(data(C,D),H),F).
```

```
renew(nil,A,B):-true|B=A.
renew(cons(A,B),C,D):-true|
   renew(B,C,E),
   replace(A,E,D).
```

この時点では、メソッド間のデータフローやオブジェクト内の記憶領域(アクション部ではブラントデータベース)へのアクセスなどが、まだ記述されていない。そこで次に、MENDELS ZONEの持つペトリネットエディタで、その内部構造をペトリネットとして表現する。

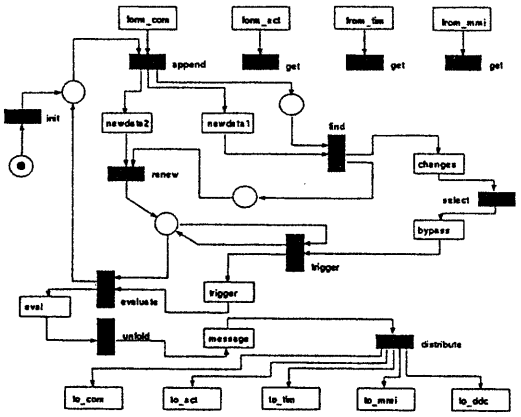

図 6: アクション部のペトリネット表現

同様に、オブジェクトの結合もペトリネットエディタ上でマウス操作のみで行なえる。各オブジェクト及びオブジェクト内の各メソッドは並行に動作するが、場合によってはその実行順序に制約を必要とする。この時、必要なタイミング制約を時相論理で記述するとMENDELS ZONEは、制約を満たす様にペトリネットの調整を行なう。最後に、ペトリネットで表現されたプログラム構造をKL1プログラムにコンパイルし、プログラムが生成される。完成したプログラムは、MENDELS ZONE環境でモニタリングしながら実行できる。

9 評価

先に述べた様に、この制御システムは、他の研究グループによってKL1で直接開発されている。それを我々は、代数的仕様記述によるオブジェクトのメソッド仕様、ペトリネットによるオブジェクトの内部構造表現及びオブジェクトの結合、時相論理によるペトリネットの調整を基に再実現した。

9.1 記述量

KL1によって直接開発されたプログラムのサイズは、約4600行であった。一方、我々は約4000行の代数仕様とプログラム構造を表すペトリネットから、6200行のKL1プログラムを生成した。生成されるプログラムは多少大きくなったが、記述者が直接入力する仕様は、むしろコンパクトになっている。代数的仕様記述の内訳を以下の表にまとめる。

表1：代数仕様の記述量

オブジェクト	構成子	関数	公理	行数
共通データ	348	29	78	454
アクション部	34	35	125	504
DDC部	15	28	51	332
タイマー部	9	11	19	148
伝送部	6	17	184	494
制御知識	0	3	387	2100
合計	412	123	844	4032

9.2 開発工数

KL1による直接開発では設計から完成まで約10人月要したが、我々の開発ではその半分、約5人月でシステムを完成させた。これは、前開発で6人月要していたデバッグ時間を0.5人月に短縮できたことに依っている。関数単位で検証を行ないその正しさを確認したことが、結合してからのデバッグ時間を大幅に短縮させた。

9.3 プログラムの品質

プログラムの信頼性 関数の正しさは代数仕様の検証機能が保証し、構造の正しさは時制論理によるペトリネット調整機能が保証する。そのため、生成されたプログラムには高い信頼性が期待できる。

実行スピード この制御システムの場合には、実行スピードに関して問題なく動作した。仕様からプログラムの変換規則は、並列項書換えをシミュレートしているので並列実行に適合している[8]。加えて、現在は

変換時の最適化等を行なっていないが、それを実現することでいっそうの高速化が望まれる。

プログラムの保守性 このアプローチの利点の一つは、仕様とプログラムの完全な一致である。プログラム構造、オブジェクト構造のペトリネット表現、そして、各メソッドの明確な代数仕様とメソッドの満たすべき検証項目が、改めてプログラムを理解しようとする場合に非常に有用である。

まとめると、我々の用いた形式的手法による開発アプローチは、信頼性を重視するプログラム開発に対し有用である。さらに、開発工数の短縮と保守容易性が期待できる。

10 課題

今回の開発事例で、前節の様な利点が確認された一方、解決すべき問題点も明らかになった。

第一に、記述能力の不足である。今回の記述では検証機能に主眼をおき、記述文法を最低限のものに限定した。問題は、検証機能を失わずに記述能力をいかに高めるかという点にある。もう一つの問題点は、検証手続きの発散である。先に例示したデータベース更新関数の検証例でも補題を必要とした。しかし、補題の発見はかなりの労力を要する。補題の自動発見と言わないまでも、記述者の補題発見を支援する機能が必要である。

11 終わりに

我々は、形式的手法の実システム開発への適用可能性を評価するために、実際にプラント制御エキスパートシステムの開発を行なった。本稿は、特に代数的仕様記述に焦点を当てた開発結果の報告である。この開発を通して形式的手法の有用性を確認する一方、解決すべき問題点も明らかになった。今後は、記述能力を拡張しその上での検証方法を確立したい。

謝辞

本研究は、第五世代コンピュータプロジェクト(FGCS)の一環として行なわれたものである。研究の機会をいただいた、(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] Bachmair, L. Proof by Consistency. In *Proc. 3rd IEEE Symposium on Logic in Computer Science*, pp. 228–233, 1988.

[2] Fribourg, L. A strong restriction of the inductive completion procedure. *J. Symbolic Computation*, Vol. 8, No. 3&4, pp. 253–276, 1989.

[3] 本位田真一, 大須賀昭彦, 内平直志. 代数的仕様と時制論理によるリアルタイムSAとオブジェクト指向設計の融合手法. 情報処理論文誌, Vol. 33, No. 2, pp. 173–182, 1992.

[4] 大須賀昭彦, 坂井公. 項書換えシステムに基づく帰納的定理証明. Tech. Memorandum TM-0424, ICOT, 1987.

[5] Ohsuga, A. and Sakai, K. Metis: A Term Rewriting System Generator – An inference engine for Equations and Inequations –. In Nakata, I. and Hagiya, M., editors, *Software Science and Engineering*, pp. 1–15. World Scientific, 1991.

[6] Togashi, A. and Noguchi, S. A program transformation from equational programs. *J. Logic Programming*, Vol. 4, pp. 85–103, 1987.

[7] 山本純一, 大須賀昭彦, 本位田真一. 代数的仕様による制御装置仕様の記述と検証. In ソフトウェア工学研究会. 情報処理学会, 1992.

[8] 吉田和樹, 大須賀昭彦, 本位田真一. 項書き換えシステムから並列論理型言語へのプログラム変換. In プログラム合成・変換研究会. 日本ソフトウェア科学会, 1992.