設 計 自 動 化 60－9
（1991．12．12）

宣言的仕様記述からの制御回路の自動合成

手嶋茂晴, 佐野範佳
(株) 豊田中央研究所　LSI 設計・評価研究室

ハードウェアの制御部分の宣言的仕様から状態遷移を得る手法を提案する. 対象のハードウェアは, いくつかの機能ブロックとそれを制御する制御ブロックから構成されるものとする. 機能ブロックの起動条件をデータの依存関係を用いて宣言することによって制御ブロックの動作を記述する. 記述の各節の公平な (fair) 実行, つまり, 節が実行可能になった場合, 有限ステップの間に実行が開始されることが保証されるとき, 記述から一意に状態遷移に変換可能である. 本報告では, fair な記述から状態遷移に変換するアルゴリズムと 記述の fairness 判定のアルゴリズムを示す. 2つのアルゴリズムはともに機能ブロックの実行による履歴の変化に着目し, 履歴の有限部分を解析するものである.

Synthesis Algorithm of Logic Specification to Finite-State Automaton

Shigeharu TESHIMA and Noriyoshi SANO
Toyota Cent. Res. & Develop. Labs. Inc.

An algorithm to synthesize a control circuit of a hardware module from a logic specification is discussed. Logic description let be a design spefication that denotes I/O dependency. It is prooved that a fair specification can be realized by a finite state machine. Here, "fair" means that a specification ensures fair excution of each clause.

1 はじめに

論理プログラムなどの宣言的な記述は内部構造や処理手順と独立に入出力関係を定義することができるので，ハードウェア合成の仕様の記述体系としての利用が可能である．我々は，宣言的記述によってハードウェアの制御部分の仕様を与え，制御部分の状態遷移を合成する手法を提案する．

宣言的記述では物理的な制約に拘らない機能的な記述が可能である一方，有限の資源では動作が保証されないような記述も可能である．そのために宣言的記述からの合成では，手続き記述からの合成のように素朴なマッピングさえ実現されていない．

我々は，回路の制御を宣言的に記述する言語を定義する．次に記述がハードウェアで合成可能であるため条件である fainess について議論する．最後に，記述が fair であることを判定するアルゴリズムと fair な記述から制御回路の状態遷移に変換する手法について述べる．

2 入出力依存関係による仕様記述

ハードウェアは，関数 (多値関数) で表現される機能ブロックとそれを制御する制御回路から構成されると考える．機能ブロックは決められた共有領域を介してデータの受け渡しを行なう．共有領域は信号線やレジスタに相当するものである．制御回路は機能ブロックの起動を制御する．概念図を図 1に示す．

各ブロックの関数としての定義は別に与えられるものとして，機能ブロックが起動される条件を宣言することによってハードウェアの動作の仕様を記述する．つまり，制御回路の仕様を記述する．

起動の条件は起動対象ブロックの入力となるデータについて，そのデータがどのブロックによって生成されたかというデータの依存関係に関するものである．だたし，ここではデータを時間幅を持つ事象と考える．

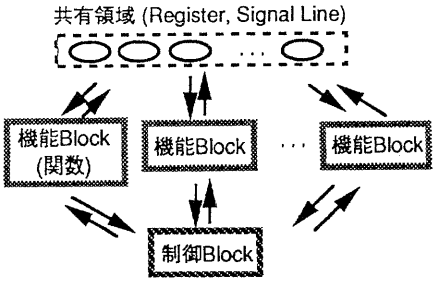

図 1: 対象回路の構成

構文 記述は，論理型言語と同様の構文を用いる．定数，変数はデータを表す．機能ブロック名を関数記号とする．また，項も同様に定義される．ただし，機能ブロックは多値関数としてモデル化されているので，データを一意に区別するために項の関数記号の部分に領域名を含める．また，項，述語は S 式で表記する．つまり，項 $(f[c]\ \ a\ \ b)$ は a, b を入力データとして，機能ブロック f が領域 c 上に生成したデータを表す．

定義 1 記述の構文は次の通りである．

$$G_1 \rightarrow (A_1\ (\boldsymbol{x}_1)(c_{11}c_{12}...))$$
$$G_2 \rightarrow (A_2\ (\boldsymbol{x}_2)(c_{21}c_{22}...))$$
$$...$$

ここで，G_k は表 1に示す述語の論理積である．A_k は機能ブロック名である．$\boldsymbol{x}_k$ は “ $x_{k1}\ x_{k2}...$ ” のような変数の列である．$c_{k1}c_{k2}...$ は領域名である．記述の各行を節 *(Clause)* と呼ぶ．

□

直観的意味 1 つの節で機能ブロックが起動される 1 つの条件を記述する．G_k が機能ブロック A_k の起動に必要な条件を表す．G_k を true にする変数の割り当てのもとで，$\boldsymbol{x}_k$ に割り当てられたデータを引数として，機能ブロック A_k を起動する．実行結果が領域 $c_{k1}c_{k2}...$ に返される．

記述の 1 つの節は図 2に示すような 2 つのグラフ (正確には，Hyper Graph) の組みに相当する．グ

表 1: 述語

構文	意味
$(== x\ t)$	変数 x と項 t とが unifiable
$(=\mathrm{v}\ t\ v_1\ v_2..)$	項 t で表現されるデータが $\{v_1, v_2, ...\}$ のいずれかの値をとる

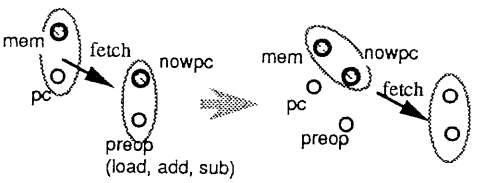

図 2: グラフによる節の表現

```
((== preop (fetch[op] mem pc))
 (=v preop 'load 'add 'sub)
 (== nowpc (fetch[pc] mem pc)))
      → (fetch (mem nowpc) (op op2 pc)))
```

図 3: 図 2 の表現に対応するテキスト記述

ラフでは, データを node で表し, 機能ブロックの 1 回の実行を hyper edge で表す. 左側のグラフが起動条件を表現する. ハードウェアが左側のグラフが照合する動作をした場合, ブロックが起動され, その結果が右側のグラフのようになることを表す.

図 2は CPU で memory fetch を行なうブロックの起動条件の一部を表している. 「1 つ前の命令語が命令語の stream を変更しないものならば, 命令語の実行終了を待たずに次の命令語の fetch を開始する」ことを表現している.

図 2に対応する記述を図 3に示す. グラフ中の node を変数に対応づけ, 枝を 1 つの述語に対応させる.

形式的意味 論理型プログラムと同様の手法で意味を与えることができる.

論理型プログラムの解釈 (interpretation) に相当する概念として, 履歴を定義する. 履歴はハードウェアが生成したデータとその因果関係である. 履歴はデータを要素として, 因果関係を表す 2 項関係の定義された集合である. 本報告の以下では, 履歴をを前章で例示したグラフを用いて, 視覚的に表現することとする. ただし, 記述の節を表すグラフでは変数を含み項を node のラベルとしたが, 履歴では変数を含まない項 (grand term) を node のラベルとする.

節の実行に対応する関数 τ によって意味を定義する. 関数 τ は履歴を引数として, 1 つの節の実行の結果を引数 (履歴) に追加した履歴を返す関数である.

関数 τ は前章で説明した記述の操作的な意味を実現する関数である. つまり, 節を

$$G_k \ \rightarrow (A_k(\boldsymbol{x}_k)(c_{k1}c_{k2}...))$$

とするとする時, 関数 τ は, 機能ブロック A_k の起動条件 G_k を満たすデータを履歴の中から非決定的に選択し, それらを引数として機能ブロック A_k を実行した結果を履歴に追加する.

定義 2 関数 τ を用いて, 記述の意味は次のように定義される.

$$\bigsqcup_{n=0}^{\infty} \tau^n(I) \qquad (1)$$

ここで, I は履歴の初期値 (初期設定で領域上に与えられるデータ) である.

□

式 (1) で定義される意味は最小不動点意味あり, 各節を公平 (fair) に何回も適用して得られる結果の上限値である.

3 仕様記述の fairness

式(1)の意味定義では,具体的にどのように節を選択するかについては言及していない.しかし,現実の処理系では,定められた戦略で節を選択し,実行する訳であるので,戦略によっては,可算無限回の計算によっても最小不動点意味を得られないことがある.

現実の処理系の計算結果が意味の正しい近似となるための条件として,fairness という概念がある.通常,fairness は処理系に対して定義されるが,本報告では記述そのものの性質として,fairness を定義する.

定義 3 記述が *fair* であるとは,実行可能な節,つまり,起動条件を *true* にする変数の割当が存在する節に対しては,有限回の関数 τ の適用の間に必ず実行されることが保証されていることである.*fair* な記述とは,実行系がどんな戦略で節を選択,実行しても,*fair* な実行が保証されている記述のことである.

□

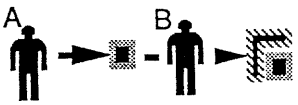

図 4: データフローの例

fair な記述と fair でない記述の例を示す.図5と図6は図4に示すデータフローを表す記述である.2つの記述はともに同じ最小不動点意味を持つが,図5は fair な記述ではなく,図6は fair な記述である.図5の記述では,ブロック **B** が実行可能にもかかわらず,ブロック **A** を優先的に実行することが許される.つまり,可算無限回の実行によって,図7に示す履歴をとり得る.

一方,図6の記述では,ブロック **A** と **B** は互いに同期しながら実行を進める.ブロック **A**, **B** と

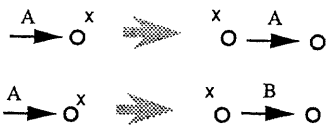

図 5: fair でない記述の例

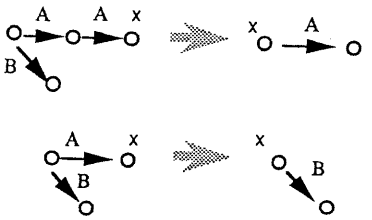

図 6: fair な記述の例

も,実行可能になれば,他のブロックを有限回,実行した後に必ず実行されるので,可算無限回の実行によって図8に示す履歴に至ることが保証される.図8は最小不動点意味である.

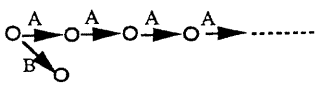

図 7: 図5の記述から得られる可能性のある履歴

4 制御ブロックの合成

4.1 記述の実現可能性

式(1)の関数 τ は Church-Rossa 性を持つ.したがって,関数 τ は,並列に適応されることに適している.関数 τ を並列に適応することは,機能ブロックの並列実行に対応する.

我々は,記述の実行過程つまり,履歴を求める過程のうち,次の操作を順序機械である制御ブロックで行なうとして,制御ブロックの状態遷移を求める.

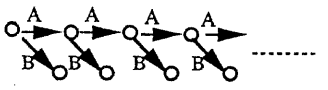

図 8: 図 6 の記述から得られるの履歴

- 機能ブロックの起動条件の判定
- 起動条件を満たす機能ブロックの起動

また, 機能ブロックは以下の操作を行なう能力を持つものとする.

- その機能ブロックの起動条件を記述する節が指定されると, その節で明記された必要なデータを読み込む.
- 同様に, 節で明記された領域にデータを生成する.
- 実行の完了を制御ブロックに伝える.

上記の仮定のもとで, 記述が制御可能であるとは, 履歴中で起動条件を満しながらまだ機能ブロックが起動されていない部分 (未実行部分) が有限状態で記憶可能なことである. つまり, 未実行部分を記憶し, 未実行部分を実行する機能ブロックを順に起動することによって, 制御ブロックはハードウェア全体を制御することができる.

また, 制御ブロックと各機能ブロックとの Interface は, 図 9 に示すように記述の節ごとに次の 2 つの信号線を設けることにする.

- ena(enable): 節が実行可能であることを制御ブロックから機能ブロックへ伝える.
- fin (finish): 節の実行が完了したことを機能ブロックから制御ブロックへ伝える.

一方, 記述が fair であるとは, 実行可能な節は必ず計算の有限過程で実行されることである. したがって, 記述が fair であることと, 任意の計算過程での履歴中の未実行部分は有限箇所であることとは等

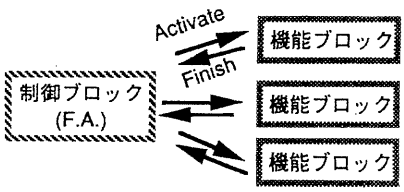

図 9: 制御, 機能ブロック間の制御信号線

価である. 全ての節が有限過程で実行されるので, 未実行部分が無限箇所になることはない. また, 逆に, 未実行部分が必ず有限箇所で押えられるということは, 各節は関数 τ の有限回の適用の間に実行されていることに他ならない.

したがって, fair な記述は, 上記の仮定を満たす機能ブロックとの間に図 9 に示す制御線を設けることによって, 順序機械の制御ブロックによって実現可能である.

4.2 状態遷移への変換

記述から制御ブロックの状態遷移を得るアルゴリズムを示す.

ただし, 回路構成 / 状態遷移を簡単にするため, 以下を仮定する.

- 機能ブロックの実行は同時に完了しない. もしくは, 同時の場合は何らかの形で完了が順序づけられるものとする. 実行している期間が時間的に重なることは構わない.
- 節の起動条件部分に書かれた値の条件を判定するのに必要なデータが提供される.

制御ブロックへの入力は, 各機能ブロックの担当する節の実行完了を伝える fin 信号線と節の起動条件の判定に必要なデータの値である. 出力は機能ブロックの節ごとの ena 信号線である. 今回は議論を簡単にするために入力を fin 信号線のみとし, データの値は取り扱わない. 提示する変換アルゴリズムは, データの値がない場合のものであるが, データの値を入力に含めるとするアルゴリズムへの拡張は可能である.

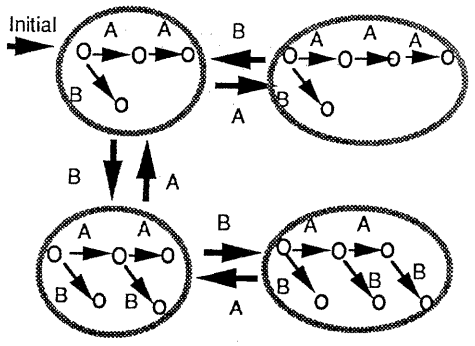

図 10: 図 6 の記述から得られる状態遷移

節の実行による履歴の変化に注目する．節の実行による履歴に変化を状態遷移で表現する．履歴において“履歴の先端から，節の起動条件に照合するが未実行の部分まで”の範囲を状態とする．“節の実行完了”が遷移となる． 遷移には機能ブロックを起動した節の名前 (機能ブロックを起動する節が 1 つの時は，機能ブロック名で代用) のラベルを付ける．

次のように状態遷移を構成する．

- ハードウェアに与えられる初期の履歴を初期状態とする．
- 状態で，未実行の節を実行が完了した結果得られる状態への遷移を加える．その遷移のラベルは実行した節とする．

図 6の記述から得られる遷移を図 10に示す．

5 記述の fairness 判定

値の領域が有限の場合に限定して，記述が fairness であるかを判定するアルゴリズムを示す．前節の合成手法を変更を加える． 変更点は，条件部分が表すグラフの最大の深さの 2 倍分の履歴部分を状態とすることである．

グラフの段数が有限で押えられかつ，値の領域が有限であるので，状態数は有限である．

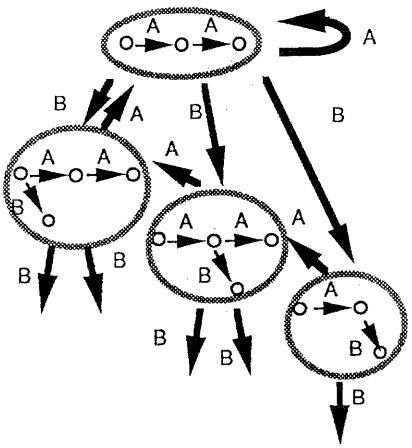

図 11: 図 5 の記述から得られる遷移

閉路が存在する時，閉路を構成する状態において未実行部分を切り捨てる遷移が存在する場合，その記述は unfair である． それ以外の場合，記述は fair である．“未実行部分を切り捨てる”とは，現状態で可能な遷移が，別の遷移をした結果，その遷移が実行が不可能になることである．

図 11に図 5から得られる遷移を示す．図 11では，経路を構成する状態からの遷移に未実行部分を切り捨てるものが存在する． したがって，fair でない．

6 まとめ

本報告で提案する記述形式はデータフローモデルを拡張したものである．従来のデータフローモデルでは，データの処理の依存関係を 1 つのグラフで表現する．我々は制御構造を追加することによってデータフローを複数のグラフで記述するように拡張した． それによって，これまでデータフローでは記述が困難であった一般の制御系の並列記述を可能にしている．

一方，提案する記述形式は，時間的な情報を一切使わず，データの依存関係だけを用いる． そのため，起動条件を記述する時，データを一意に識別することが困難である．つまり，複数のデータを引数

にとる機能ブロックでは, 引数として組み合わせられるべきデータを指定するために, 依存関係を深く遡る必要が生じる場合がある. これまでの記述言語では時刻というキーでデータを一意に識別できたため, このように記述が複雑になることはないが, その反面, 時間的はスケージューリングがあらかじめ必要とする.

また, 本報告でのデータ依存関係は 機能ブロック間での直接のデータの受け渡しを意味する. つまり, 履歴をグラフで表現した時, データ依存関係は隣接 node を結ぶ link に相当する. 記述では, グラフの path に相当する遷移的な関係を含めていない. 遷移的な関係を追加した場合, 記述形式の持つ計算能力が真に上がると予想される. その場合, 記述と順序機械との意味的な隔たりが広がり, 記述からの順序機械合成をさらに困難にする.

記述から状態遷移への変換は, 一種の記号シミュレーションである. 処理系がとり得る状態を全て列挙し 状態遷移に変換する. 記述が fair であることは, とり得る状態が有限で押えられるための必要十分条件である. また, fainess 判定はとり得る状態を順に列挙し, 状態数が有限となるかを調べることに他ならない.

また, 今回の議論では, 制御ブロックを合成する時, いくつかの条件を機能ブロックに課しているが, 機能ブロックの合成については, 言及していない. 機能ブロックを中心とするデータパス系の合成は今後の課題である.

参考文献

[1] 手嶋, 長瀬, 瀧川: “動作履歴をグラフで表現するハードウェアの機能動作モデルについて”, 情処第 34 回全国大会, pp. 1349-1350 (1989).

[2] 手嶋, 佐野: “ハードウェアの宣言的仕様記述の Fairness 判定について”, 情報処理学会 DA シンポジウム '91, pp. (1991).

[3] 手嶋, 平石, 矢島: “イベントの 2 項関係に基づく並列システムの代数的仕様記述”, 信学論 (D) Vol.J70-d No.1, pp. 19-29.

[4] 村上昌己: “並行論理型言語の形式的意味”, 情報処理, Vol.32, No.7, pp. 819-838. (July 1991).

[5] 古川, 溝口 編: “並列論理型言語 GHC とその応用”, 共立出版 (1987).

[6] C.D.Kloos: “Semacstics of Digital Circuits”, Lecture Note in Computer Science 289, Springer-Verlag (1987).