ソフトウェア工学 134−16
（２００１．１０．１９）

カラーペトリネットスライシングによるリアルタイム開発支援

渡辺 晴美 立命館大学 理工学部
E-mail: harumi@selab.cs.ritsumei.ac.jp

本稿では、カラーペトリネットのスタティックスライシングを提案する。カラーペトリネットは、リアルタイムシステムの性質である並行性や時間的性質に関する誤りを検出する能力を持つ。しかし、記述したモデルは複雑で理解が困難であり、変更が容易でないという問題がある。プログラミング言語では、理解と変更を効率良く行うための技術としてスライシングが知られている。本稿では、上記の方法をカラーペトリネットに適用することで、理解と変更の容易性を向上させる。

A Slicing Technique of Coloured Petri Nets for Real-Time Systems

Harumi Watanabe
Department of Computer Science, Ritsumeikan University

In this article, we propose a static slicing technique for the Coloured Petri Nets(CPN). The analisys power of the CPN contributes to detect errors in concurrent and timed propeties. However, in most cases, CPN models are difficult to understand and modify. To cope with this problem, we apply the static slicing techinique to the CPN. The techinique is wellknown to solve the problem in the programing language field.

1 はじめに

昨今、携帯電話や情報家電などリアルタイムシステムの多機能化、複雑化が進みつつあり、複雑なシステムを効率的にテスト検証できる仕組みが求められている。リアルタイムシステムは、リアルタイム性、リアクティブ性に関する時間的性質と並行性に関する性質を持つ。これらの性質に関する検証と性能評価が可能な形式記述言語の一つとして K. Jensen が提案したカラーペトリネット (Coloured Petri Nets) [4] が知られている。しかし、記述したモデルは複雑で理解が困難であり、変更が容易でないという問題がある。

本稿では、カラーペトリネットのスタティックスライシング方法を提案することで上記の問題を解決する。スタティックスライシングは、プログラミング言語の理解と変更を効率良く行う技術として知られている [5]。

プログラムスライシングはプログラムスライスを求める技法のことであり、プログラムスライスとは、ある命令の実行に影響を与える可能

性のあるすべての命令の集合のことである。プログラムスライスを得ること、すなわち、ある命令の実行に影響を与える部分プログラム得ることは、プログラムの中からある機能を実現している部分のみを取り出すことが可能であると言える。この技術により、プログラムを機能別に理解することが可能になる。また、プログラムスライスを求めることにより、変更する命令に対する影響範囲を捕らえることが可能になるため、変更が容易になる。以上により、プログラムスライシングは、テスト、デバッグ、保守、そしてリファクタリングの際に利用されている。

本稿で提案する方法により、プログラムスライシングの場合と同様、カラーペトリネットからある機能に着目した部分ネットを抽出することを可能にする。この方法により、カラーペトリネットを機能毎に理解することが可能になり、変更箇所を容易に発見することが可能になる。さらに、性能解析の結果、時間制約を満たさない命令に関するサブグラフを得ることが可能であるため、リアルタイム設計を効率的にする。

以下、まず 2でカラーペトリネットについて簡単に説明する。3では、まずカラーペトリネットのセマンティクスとスライシングに関する用語について定義し、提案する方法について論じる。4では提案した方法を例題に適用し評価を行う。その結果を 5で関連研究とともに考察する。最後に 6で、本稿のまとめと、今後の展望について述べる。

2　カラーペトリネット

図 1にカラーペトリネットの例を示す。カラーペトリネットは、図 1中の長方形内に記述したカラー宣言 (color declaration) 部分、とそれ以外のグラフィック部分からなる。カラー宣言部分では、カラーと呼ばれる型と変数の宣言を行う。図 1では、int 型のカラー Int が宣言され、Int 型の変数 arg1、arg2 の宣言を行っている。

グラフィック部分は図 1に示すとおり、プレース、カラー、トランジション、ガード、アーク、アーク式、時間からなる。カラーペトリネットでは、システムの挙動を発火と呼ばれる仕組みにより表す。トランジションが発火することで、トークンはアークの方向に従いプレースからプレースへ移動する。図 1のトランジション Operation が発火すると、プレース arg1 と arg2 にあるトークンはプレース Result に移動する (図 2参照)。

発火は、発火可能なトランジションのうち一つが任意に起きる。発火可能とは、トランジションに入力しているアーク式を満たすようにアークの始点となるプレースにトークンがあり、さらにアークを通ってきたトークンの値がガードを満たす場合のことである。

カラーペトリネットの解析は可達グラフを用いて行う。図 1の可達グラフを図 3に示す。図 3のノードは、マーキングの状態を示し、上のノードは、トランジション Operation が発火する前の状態を表し、下のノード発火後を示している。アークは発火したトランジションを表し、遷移に 202 タイム経過したことを示している。遷移時間は、図 1のトランジションとアーク式に添付された @マーク後の数値が計算される。

ところで、カラーペトリネットには融合プレースという概念がある。このプレースを利用することにより、複雑な図を整理して記述したり、大域変数を表したりすることが可能である。融合プレースの例を図 4に示す。4-(a) と 4-(b) は同じ意味である。FG が添付された同じ名前のプレースは融合プレースであり、表現上は二個のプレースであるが、実際は一つのプレースである (K. Jensen のカラーペトリネットでは融合プレース名とスコープを指定が一致したプレースが一つのプレースとして認識される)。

3　カラーペトリネットのスタティックスライシング

本節では、まずスライシング対象となるカラーペトリネットの意味について述べ、次にスライシングに関連した用語について定義し、カラーペトリネットをスタティックスライシングする方法について提案する。

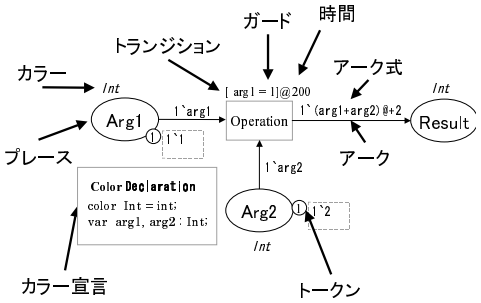


図 1: カラーペトリネットの例

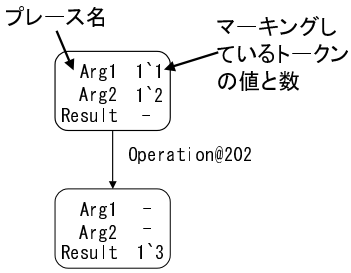


図 3: 可達グラフの例

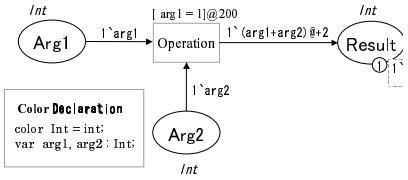


図 2: カラーペトリネットの発火

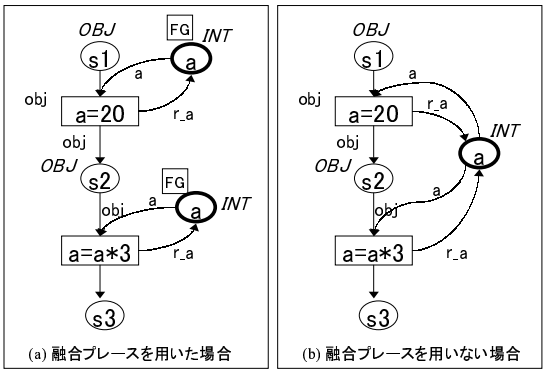


図 4: 融合プレースの例

3.1 カラーペトリネットの意味

カラーペトリネットは状態遷移図と異なり、グラフの構造と挙動を定義したにすぎず、その構成単位に様々な意味を持たせることができる。例えば、プレースを状態として扱う場合もあれば、変数として扱う記述者もいる ([4] の Volume3 参照)。表 1に本稿での意味を示す。

オブジェクト指向開発の普及にともない、我々は、状態図とクラス図をカラーペトリネットへ対応付ける方法を提案してきた [2][1]。本稿でも、これまでの取組みと同様の意味を与えることで、[2][1] との統合を目指す。統合により、複雑なカラーペトリネットを直接記述することなく、カラーペトリネットの検証能力を利用することが可能であり、状態図の部分テストも可能となる。表 1は [2] でスライシングと関連する箇所を単純化している。

表 1でオブジェクトを表すトークンをオブジェクトトークン、値を表すトークンを値トークンと呼ぶ。変数を表すプレースを変数プレース、オブジェクトのステップを表すプレースをオブジェクトプレースと呼ぶ。オブジェクトのステップとは、オブジェクトトークンを受取るプレースであり、オブジェクトトークンがオブジェクトプレース上を移動する順番に命令トランジション (命令を表すトランジション) が実行される。

3.2 スライシング

以下、本稿で提案するスライシング方法に関連した用語について述べる。

意味	カラーペトリネット要素	カラー
	名称	
オブジェクト	トークン	OBJ
	オブジェクトトークン	
ステップ	プレース	OBJ
	オブジェクトプレース	
変数	プレース	変数の型
	変数プレース	
変数の型	カラー	変数の型
値	トークン	値の型
	値トークン	
演算	トランジション	
	演算トランジション	
命令	トランジション	
	命令トランジション	

表 1: カラーペトリネットの意味

スライス： プログラム言語でのスライスとは、ある命令の実行に影響を与える可能性のあるすべての命令の集合である。カラーペトリネットのスライスもプログラム言語の場合と同様に、ある命令トランジションの実行に影響を与える可能性のあるすべての命令トランジションにより構成された部分ネットである。

スタティックスライシング： プログラミング言語でのスタティックスライシングとは、プログラムを実行せずに、スタティックスライスを静的に求める方法でる。カラーペトリネットのスタティックスライシングもプログラミング言語の場合と同様に、カラーペトリネットを実行せずに、命令の依存関係を静的に求める方法とする。

データ依存関係： プログラミング言語のスタティックスライシングを求める際、まずデータ依存関係を求める。プログラミング言語のデータ依存とは次の場合である。命令 s から命令 t に対してデータ依存関係があるとは、変数 w が命令 s と t に存在し、命令 t に到達可能である場合。

カラーペトリネットでのデータ依存関係も上記のの場合と同様である。相違は、命令をトランジションで表し、変数を変数プレースで表すことにある。従って命令 s は命令 s を表すトランジションになる。命令 t を基準トランジション、命令 s をデータ依存トランジション、変数 w を表すプレースをデータ依存プレースと呼ぶことにする。

カラーペトリネットのデータ依存関係は次の場合である。トランジション s からトランジション t に対してデータ依存関係があるとは、s と t に入力している変数プレース w が存在し、s から t に到達可能である場合。

制御依存関係： プログラミング言語でスタティックスライシングを求める際、データ依存関係を求めた後、制御依存関係を求める。プログラミング言語の制御依存関係とは次の場合である。命令 s から命令 t に対して制御依存関係があるとは、命令 s が分岐命令であり、命令 t がその分岐内に直接含まれている場合、あるいは、命令 s がループ命令であり、命令 t がそのループ文内に直接含まれている場合をいう。カラーペトリネットの場合も同様であるが、カラーペトリネットでの分岐命令とは、複数のアークを出力しているオブジェクトプレース (制御プレース) へ入力しているトランジション (制御トランジション) である。図 6 ではプレース $S2$ が制御プレースであり、トランジション $s : x < 5$ が制御トランジションである。またループ命令とは、複数の入力アークを持つプレース (制御プレース) が出力しているトランジション (制御トランジション) である。

さらに、本稿のカラーペトリネットの場合、オブジェクトに関する入力アークまたは出力アークを持つトランジションがある。複数入力アークを持つ場合、オブジェクトの同期またはオブジェクトの削除を表す。一方、複数の出力アークを持つトランジションはオブジェクトの生成を表す。これらのトランジションも制御依存トランジションである。

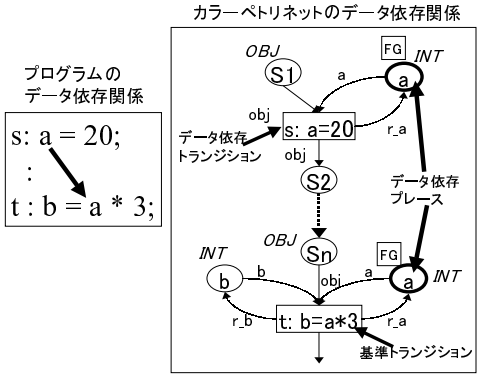


図 5: データ依存関係の例

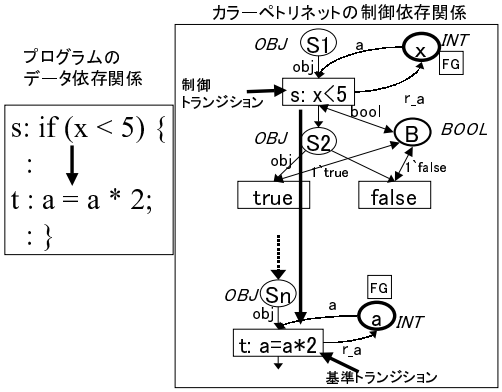


図 6: 制御依存関係の例

3.3 スライシング方法

カラーペトリネットのスタティックスライシングを求める方法を以下に示す。

1. スライシング基準と開始を選択する。
   スライシングを開始する命令トランジション (開始トランジション) と、スライシング基準となる命令トランジション (基準トランジション) を選択する。

2. データ依存関係を求める。

   (a) 基準トランジションに入力している変数プレース (データ依存プレース) を抽出する。図 5で変数プレースはプレース a である。

   (b) データ依存変数プレースに連結しているトランジションを抽出する。そのうち、開始トランジションから基準トランジションの間にあるトランジション (データ依存トランジション) を選択する。

   (c) スライスの構成要素は、データ依存プレース (図 5の a プレース)、データ依存トランジション ($s : a = 20$ トランジション)、各データ依存トランジションへ入力するオブジェクトプレース ($S1$ プレース) と出力するオブジェクトプレース ($S2$ プレース) である。これらを連結するアークとアーク式は元のカラーペトリネットの関係を維持する。また、プレースのカラー、トランジションのガード式についても元のカラーペトリネットと等しい。

3. 制御依存関係を求める。

   (a) 制御依存プレース、トランジションを発見する。
   基準トランジションから開始トランジションへアークを逆にたどり、制御プレース、すなわち、複数の入力アークまたは出力アークを持つオブジェクトプレースを抽出し、制御トランジションを得る。また同様に、同期トランジション、すなわち、複数の入力アークまたは出力アークを持つ制御トランジションを抽出する。

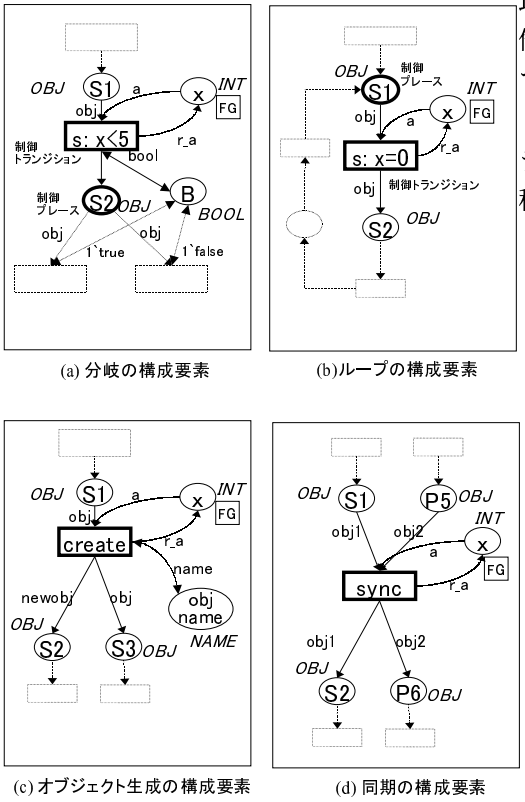
   (b) スライス構成要素は、分岐、ループ、同期、オブジェクト生成に応じて図 7の実線部分になる。

   (c) 制御依存トランジションに関するデータ依存関係を求める。

4. スライスを得る。

   (a) スライスの構成要素をデータ依存関係、制御依存関係から求める。

   (b) 切り離されたスライス構成要素を連結する。
   各構成要素で、同じオブジェクトプレースを連結する。独立した構成要



(a) 分岐の構成要素　(b)ループの構成要素

(c) オブジェクト生成の構成要素　(d) 同期の構成要素

図 7: 制御依存の構成要素

素は、構成要素の入り口にあたるプレースと出口にあたるプレースを連結していく。

4　適用例

図 8に現在位置に応じてルートを表示を行うプログラムの例を示す。それに対応したカラーペトリネットを図 9に示し、そのスタティックスライスを図 10に示す。本来は、プログラムからカラーペトリネットへ変換するのではなく、カラーペトリネットは仕様として記述する。図のプログラムはカラーペトリネットを説明するためのものである。

本例題では、現在位置、目的地、交通情報を獲得しルートを計算し、画面表示を行う。カラーペトリネットで解析を行った結果、ルートを獲得する処理 (GetBestRoute トランジション) で非常に時間がかかるため、ルートを獲得する処理の機能を抽出、すなわちスライシングを行い、他のオブジェクト (スライス) を得ること想定している。

3で示したスライシング方法に従い、図 9のカラーペトリネットから図 10のスライスを得る過程を以下に示す。

1. スライシング基準と開始トランジションを選択する。開始トランジションを $InputObjectPoint()$ トランジション、基準トランジションを $GetBestRoute()$ トランジションとする。スライスの構成要素はこれらと以下のオブジェクトプレース。($S1$、$S2$、$S9$)

2. データ依存関係を求める。

   (a) 基準トランジションに入力しているデータ依存プレースは、$ObjectPoint$、$CurrentPoint$、$Traffic$ である。

   (b) データ依存プレースから得られるデータ依存トランジションは以下である。$InputObjectPoint$、$GetCurrentPoint$、$CompPoint$、$GetTraffic$

   (c) スライスの構成要素として以下を得る。データ依存トランジション ($InputObjectPoint$、$GetCurrentPoint$、$CompPoint$、$GetTraffic$)、これらのトランジションの入出力となるオブジェクトプレース ($S1$、$S2$、$S3$、$S4$、$S7$、$S8$)、データ依存プレース ($ObjectPoint$、$CurrentPoint$、$Traffic$)、これらの要素を連結するアーク。

3. 制御依存関係を求める。

   (a) 制御依存プレース ($S2$、$S5$)、これらのプレースから制御トランジション ($while(TRUE)$、$CompPoint$) を得る。

   (b) 制御依存関係で得られるスライス構成要素は、制御プレース ($S2$、$S5$)、制御トランジション ($while(TRUE)$、

$CompPoint$)、オブジェクトプレース ($S3$、$S4$)、分岐プレース ($Bool$) である。

(c) 制御依存トランジションに関するデータ依存関係を求める。データ依存プレース ($ObjectPoint$、$CurrentPoint$) を得る。これらはすでにデータ依存関係を求めた際に得たプレースである。

4. スライスを得る。

(a) スライスの構成要素をデータ依存関係、制御依存関係から求める。

(b) 切り離されたスライス構成要素を連結する。オブジェクトプレース $S5$ と $S7$ を連結し $S57$ にし、$CompPoint()$ トランジションの出力を $S57$ プレースにし、$GetTraffic()$ トランジションの入力を $S57$ プレースにする。オブジェクトプレース $S8$ と $S9$ を連結し $S89$ にし、$GetTraffic()$ トランジションの出力を $S89$ プレースにし、$GetBestRoute()$ トランジションの入力を $S89$ プレースにする。

```
/* 目的地を得る。*/
ObjectPoint = InputObjectPoint();
/*  現在位置と目的地が同じになるまで
以下の処理を繰り返す。*/
while(TRUE){
   /* 現在位置を獲得する。*/
   CurrentPoint = GetCurrentPoint();
   /* 現在位置と目的地が同じであれば
   終了する。*/
   if(CompPoint(CurrentPoint, ObjectPoint)
   == TRUE){
      ExitTask();
    }
    /* 現在位置から表示地域を得る。*/
    CurrentArea = CalcArea(CurrentPoint);
    SetArea(CurrentArea);
    /* 渋滞情報を得る。*/
    traffic = GetTraffic();
    SetTraffic(traffic);
    /* ベストルートを計算する。*/
    BestRoute = GetBestRoute(CurrentPoint,
    ObjectPoint,traffic);
    SetViewRoute(BestRoute);
    /* エリアが更新されたら、地図画面
    を更新する */
    FlushView();
}
```

図 8: プログラムの例

5 議論

従来のカラーペトリネットでは、階層化の概念のみのため、同じ階層内に複数の機能を含んでいる場合、複数の階層により一つの機能を実現している場合、機能毎にネットを分割することはできない。書き換え方法もシンタックスに着目した方法しかない。従って、従来の方法では、カラーペトリネットを機能毎に理解し、変更による影響範囲を容易に抽出することはできなかった。

本稿で提案する方法により、プログラムスライシングの場合と同様、カラーペトリネットからある機能に着目した部分ネットを抽出することを可能であることを 4の例題で、ルートを獲得する処理機能を抽出することで示すことができた。

また例題で示したように、カラーペトリネットの従来の機能である性能解析とあわせ、スライシング基準を可達グラフで時間制約を満たさないトランジションにすることで、その機能を効率よく抽出できることも示した。

従来のカラーペトリネットの解析は、可達グラフによる方法のみであるため、データ依存関係を調べる方法も動的な方法のみであった。カラーペトリネットは同値仕様という方法で、状態空間を小さくすることが可能であるが、もとのネットが十分に大きい場合、この方法を用いても状態空間を小さくすることは難しい。従って、本方法により、機能別にスライスを獲得することで、検証テストを従来より効率的に行うことが可能である。

6 おわりに

本稿では、カラーペトリネットのスタティックスライシングを提案した。本稿で提案した方法により、カラーペトリネットからある機能に着目した部分ネットを抽出することを可能にした。

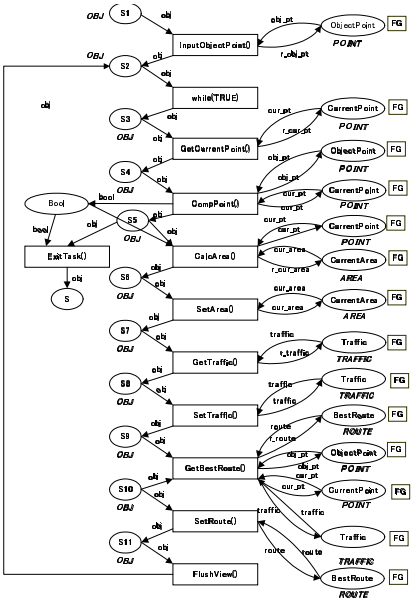


図 9: カラーペトリネットの例

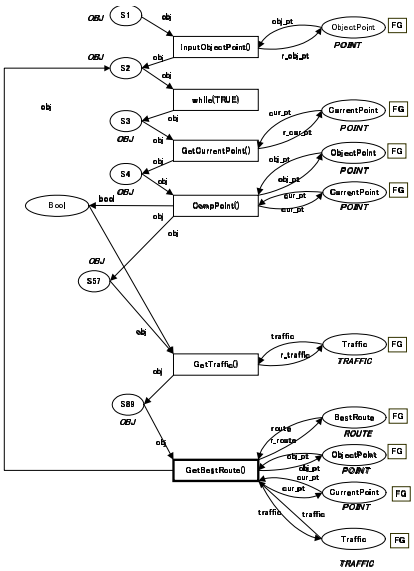


図 10: スライシングを行ったカラーペトリネット

この方法により、カラーペトリネットを機能毎に理解することが可能になり、変更箇所を容易に発見することが可能になった。さらに、性能解析の結果、時間制約を満たさない命令に関するサブグラフを得ることが可能であるため、リアルタイム設計を効率的にできることを適用例で示した。今後は、これまでの成果と統合できるように、オブジェクト指向ネットに関するスライシングに関して取り組んでいく予定である。

参考文献

[1] 伊藤恵，渡辺晴美，西田雅彦，片山卓也，“オブジェクト指向リアルタイムシステムのテスト支援環境”, 情報処理学会オブジェクト指向 2000 シンポジウム論文集, pp.49-56,2000, 9

[2] 渡辺 晴美, 徳岡 宏樹, Wu Wenxin, 佐伯 元司, “カラーペトリネットによるオブジェクト指向ソフトウエアのテストと解析方法”, 電子情報通信学会和文誌 D-I, Vol.J82-D-I, No.3, 1999, 3

[3] 渡辺 晴美, 工藤 知宏, “カラーペトリネットを用いた仕様記述からのテストケース生成方法”, 電子情報通信学会和文誌 A(基礎・境界), Vol.J80-A,pp.1073-1086, 1997,7

[4] K. Jensen, “COLOURED PETRI NETS – Basic Concepts, Analysis Methods and Practical Use – Volume 1-3”, Springer-Verlag, 1992, 1994, 1997

[5] 下村隆夫, “プログラムスライシング”, 共立出版, 1995