(19)日本国特許庁(JP)　(12)特　許　公　報(B2)　(11)特許番号
特許第6180739号
(P6180739)
(45)発行日　平成29年8月16日(2017.8.16)　(24)登録日　平成29年7月28日(2017.7.28)

(51)Int.Cl.　　F I
G06F 17/10 (2006.01)　　G06F 17/10 Z
G06F 17/50 (2006.01)　　G06F 17/50 604D

請求項の数 14　外国語出願　(全 37 頁)

(21)出願番号　特願2013-2191(P2013-2191)
(22)出願日　平成25年1月10日(2013.1.10)
(65)公開番号　特開2013-210997(P2013-210997A)
(43)公開日　平成25年10月10日(2013.10.10)
審査請求日　平成27年12月8日(2015.12.8)
(31)優先権主張番号　13/422,335
(32)優先日　平成24年3月16日(2012.3.16)
(33)優先権主張国　米国(US)

(73)特許権者　500520743
ザ・ボーイング・カンパニー
Ｔｈｅ　Ｂｏｅｉｎｇ　Ｃｏｍｐａｎｙ
アメリカ合衆国、６０６０６－２０１６
イリノイ州、シカゴ、ノース・リバーサイド・プラザ、１００
(74)代理人　100109726
弁理士　園田　吉隆
(74)代理人　100101199
弁理士　小林　義教
(72)発明者　ファーティグ，　ケネス　ダブリュ．
アメリカ合衆国　カリフォルニア　９４３０６，　パロ　アルト，　パラダイス　ウェイ　１０４２

最終頁に続く

(54)【発明の名称】論理式の迅速な管理のためのシステムおよび方法

(57)【特許請求の範囲】
【請求項１】
複雑な完全論理式（ＷＦＦ）を単純化するコンピュータ実行方法であって、
式プロセッサ（１８）内で入力ＷＦＦ（１００）を受け取るステップであって、前記入力ＷＦＦ（１００）は、
ｉ）真の原子、
ｉｉ）偽の原子、
ｉｉｉ）ブール述語、および／または
ｉｖ）等価述語
のうちの１つ以上、および
ｉ）否定演算子、
ｉｉ）連言演算子、および／または
ｉｉｉ）選言演算子
のうちの０個以上により成っている、ステップと、
式コンバータ（２０）を用いて、前記入力ＷＦＦ（１００）を、前記入力ＷＦＦ（１００）の全体を表現する初期ビット配列（２００）に変換すること、
ビット配列シンプリファイヤ（２２）を用いて、前記初期ビット配列（２００）において、前記入力ＷＦＦ（１００）を表現するために不要である冗長な１以上の述語に対応する次元を除去することによって、前記初期ビット配列（２００）を単純化されたビット配列（２１２）に単純化すること、および

ビット配列コンバータ（２４）を用いて、前記単純化されたビット配列（２１２）を連言標準形（１４２）または選言標準形（１４０）の戻りＷＦＦ（１３８）に変換することを実行するステップと、

前記式プロセッサ（１８）を用いて、前記戻りＷＦＦ（１３８）を返すステップと、を含み、

前記方法は、トレード研究において、データ依存制約ネットワークの一部である入力変数のユーザ指定の集合から出力変数のユーザ指定の集合の値を計算するための条件計算プランにおいて実行される、方法。

【請求項２】

前記入力ＷＦＦ（１００）を前記初期ビット配列（２００）に変換する前記ステップは、

前記入力ＷＦＦ（１００）内の述語（１１４）を特定すること、

前記述語（１１４）の各１つに関連する次元の要素（１２０）を特定すること、

前記述語（１１４）および前記次元の要素（１２０）に対応する前記初期ビット配列（２００）のビット配列次元（２１８）を特定すること、および

前記ビット配列次元（２１８）ならびにそれに関連する前記述語（１１４）および前記次元の要素（１２０）を有する前記初期ビット配列（２００）を返すために前記入力ＷＦＦ（１００）を処理することを含み、

前記入力ＷＦＦ（１００）は、好ましくは、真の原子ＷＦＦ（１４８）および偽の原子ＷＦＦ（１５０）のうちの１つであり、

前記初期ビット配列（２００）を返すために前記入力ＷＦＦ（１００）を処理する前記ステップは、好ましくは、

前記真の原子ＷＦＦ（１４８）または偽の原子ＷＦＦ（１５０）に対して、真のみを表す０次元の配列（２１４）または偽のみを表す０次元の配列（２１６）をそれぞれ返す、請求項１に記載の方法。

【請求項３】

前記入力ＷＦＦ（１００）は、ブール述語（１２４）または等価述語（１２６）を含む原子ＷＦＦ（１０６）であり、

前記初期ビット配列（２００）を返すために前記入力ＷＦＦ（１００）を処理する前記ステップは、

前記原子ＷＦＦがブール述語（１２４）である場合に、前記ブール述語（１２４）を等価述語（１２６）に変換すること、

前記等価述語（１２６）について、前記ビット配列次元（２１８）ならびにそれに関連する前記述語（１１４）および前記次元の要素（１２０）を特定すること、

配列キャッシュ（３０）から、前記等価述語（１２６）に関連する前記ビット配列次元（２１８）、前記述語（１１４）、および前記次元の要素（１２０）を有する初期ビット配列（２００）を取り出すこと、および

前記配列キャッシュ（３０）からの前記初期ビット配列（２００）の取出がない場合に前記初期ビット配列（２００）を構成することを含む、請求項２に記載の方法。

【請求項４】

前記入力ＷＦＦ（１００）は、複合ＷＦＦ（１３６）であり、

前記初期ビット配列（２００）を返すために前記入力ＷＦＦ（１００）を処理する前記ステップは、

前記処理の結果として原子ＷＦＦ（１０６）が得られるまで、前記複合ＷＦＦ（１３６）を再帰的に処理すること、

前記再帰的な処理の結果として得られた前記原子ＷＦＦ（１４８、１５０）の各１つについて初期ビット配列（２００）を返すこと、および

否定演算子（１３０）、連言演算子（１３２）、および選言演算子（１３４）のうちの少なくとも１つに基づいて前記初期ビット配列（２００）を組み合わせることを含む、請求項３に記載の方法。

【請求項５】
前記ビット配列次元（２１８）に、ビット要素を有する部分配列（２２０）が存在し、
前記初期ビット配列（２００）を単純化させる前記ステップは、
前記ビット配列次元（２１８）の各１つについて前記部分配列（２２０）の前記ビット要素（２２２）を比較すること、
等しいビット要素（２２２）を有する部分配列（２２０）を有する各ビット配列次元（２１８）を前記初期ビット配列（２００）から除去すること、
前記初期ビット配列（２００）から除去されないビット配列次元（２１８）を含む保存された次元インデックス（２８）を生成すること、および
単純化されたビット配列（２１２）および前記保存された次元インデックス（２８）を返すことを含む、請求項２から４のいずれか一項に記載の方法。
【請求項６】
前記単純化されたビット配列（２１２）は、ビット要素からなり、
前記単純化されたビット配列（２１２）を前記戻りＷＦＦ（１３８）に変換する前記ステップは、
前記単純化されたビット配列（２１２）内の前記ビット要素（２２２）の総量を判定すること、
１の値を有する前記単純化されたビット配列（２１２）内の前記ビット要素（２２２）の量を判定すること、
前記ビット要素（２２２）の前記総量の半分未満が１の値を有する場合に、前記単純化されたビット配列（２１２）を前記選言標準形（１４０）の前記ＷＦＦに変換すること、および
前記ビット要素（２２２）の前記総量の少なくとも半分が１の値を有する場合に、前記単純化されたビット配列（２１２）を前記連言標準形（１４２）の前記戻りＷＦＦ（１３８）に変換することを含む、請求項１から５のいずれか一項に記載の方法。
【請求項７】
前記式プロセッサ（１８）内で前記入力ＷＦＦ（１００）を受け取る前に、古典的プロセッサ（１４）内で、１つまたは複数の古典的アルゴリズムを使用して前記入力ＷＦＦ（１００）を処理することによって所定のタイムアウト期間内に結果のＷＦＦ（１０４）を処理するステップと、
前記結果のＷＦＦ（１０４）またはそれがないことを前記式プロセッサ（１８）に通信するステップと、
前記結果のＷＦＦ（１０４）がない場合に前記式プロセッサ（１８）内で前記入力ＷＦＦ（１００）を受け取るステップと、をさらに含み、
前記古典的アルゴリズムは前記入力ＷＦＦ（１００）の前記連言標準形または前記選言標準形を判定するためのアルゴリズムである、請求項１から６のいずれか一項に記載の方法。
【請求項８】
設計制約式計算をサポートするプロセッサベースのシステムであって、
完全論理式（ＷＦＦ）である入力ＷＦＦ（１００）を、前記入力ＷＦＦ（１００）の全体を表現する初期ビット配列（２００）に変換するように構成された式コンバータ（２０）と、
前記初期ビット配列（２００）において、前記入力ＷＦＦ（１００）を表現するために不要である冗長な１以上の述語に対応する次元を除去することによって、前記初期ビット配列（２００）を単純化されたビット配列（２１２）に単純化するように構成されたビット配列シンプリファイヤ（２２）と、
前記単純化されたビット配列（２１２）を連言標準形（１４２）または選言標準形（１４０）の戻りＷＦＦ（１３８）に変換するように構成されたビット配列コンバータ（２４）と、を含み、
前記入力ＷＦＦ（１００）は、

i）真の原子、
ii）偽の原子、
iii）ブール述語、および／または
iv）等価述語
のうちの１つ以上、および
i）否定演算子、
ii）連言演算子、および／または
iii）選言演算子
のうちの０個以上により成っており、
前記プロセッサベースのシステムは、トレード研究において、データ依存制約ネットワークの一部である入力変数のユーザ指定の集合から出力変数のユーザ指定の集合の値を計算するための条件計算プランの入力ＷＦＦを単純化するように構成されている、システム。

【請求項９】
前記式コンバータ（２０）は、前記入力ＷＦＦ（１００）内の前記述語（１１４）を特定し、前記述語（１１４）の各１つに関連する次元の要素（１２０）を特定し、前記述語（１１４）および前記次元の要素（１２０）に対応する前記初期ビット配列（２００）のビット配列次元（２１８）を特定するように構成され、
前記入力ＷＦＦ（１００）は、好ましくは、真の原子ＷＦＦ（１４８）および偽の原子ＷＦＦ（１５０）のうちの１つであり、
前記式コンバータ（２０）は、好ましくは、前記真の原子ＷＦＦ（１４８）または前記偽の原子ＷＦＦ（１５０）を、真のみを表す０次元の配列（２１４）および偽のみを表す０次元の配列（２１６）のそれぞれの１つを含む初期ビット配列（２００）に変換するように構成される、請求項８に記載のシステム。

【請求項１０】
１つまたは複数の初期ビット配列（２００）を含む配列キャッシュ（３０）と、
ブール述語（１２４）または等価述語（１２６）を含む前記入力ＷＦＦ（１００）と、
前記ブール述語（１２４）を等価述語（１２６）に変換し、ビット配列次元（２１８）ならびに前記等価述語（１２６）に関連する前記述語（１１４）および前記次元の要素（１２０）を特定し、前記配列キャッシュ（３０）から、前記等価述語（１２６）を表す初期ビット配列（２００）を取り出すように構成された前記式コンバータ（２０）と、をさらに含み、任意に、
前記配列キャッシュ（３０）内に前記初期ビット配列（２００）がない場合に前記初期ビット配列（２００）を構成するように構成されたビット配列コンストラクタ（２６）と、をさらに含む、請求項８または９に記載のシステム。

【請求項１１】
前記入力ＷＦＦは、少なくとも１つの演算子に関連する２つ以上の原子ＷＦＦを含む複合ＷＦＦであり、
前記式コンバータは、前記原子ＷＦＦのうちの各１つに対して初期ビット配列を返すように構成され、否定演算子、連言演算子、および選言演算子のうちの少なくとも１つに基づいて前記初期ビット配列を組み合わせるように構成される、請求項１０に記載のシステム。

【請求項１２】
前記初期ビット配列（２００）は、前記述語（１１４）に関連する複数のビット配列次元（２１８）を有し、
前記ビット配列シンプリファイヤ（２２）は、前記ビット配列次元（２１８）のうちの少なくとも１つを除去することによって前記初期ビット配列（２００）を単純化させるように構成され、
前記ビット配列次元に、ビット要素を有する部分配列（２２０）が存在し、
前記ビット配列シンプリファイヤ（２２）は、前記ビット配列次元（２１８）の各１つ

について前記部分配列（２２０）の前記ビット要素（２２２）を比較し、等しいビット要素（２２２）を有する部分配列を有する各ビット配列次元を前記初期ビット配列（２００）から除去するように構成される、請求項９から１１のいずれか一項に記載のシステム。

【請求項１３】

ビット配列コンバータ（２４）であって、

前記単純化されたビット配列（２１２）内のビット要素（２２２）の総量に対する１の値を有する前記単純化されたビット配列（２１２）内の前記ビット要素（２２２）の量を判定し、

前記ビット要素（２２２）の前記総量の半分未満が１の値を有する場合に、前記単純化されたビット配列（２１２）を前記選言標準形（１４０）の前記ＷＦＦに変換し、

前記ビット要素（２２２）の前記総量の少なくとも半分が１の値を有する場合に、前記単純化されたビット配列（２１２）を前記連言標準形（１４２）の前記戻りＷＦＦ（１３８）に変換するように構成された、ビット配列コンバータ（２４）をさらに含む、請求項８から１２のいずれか一項に記載のシステム。

【請求項１４】

前記入力ＷＦＦ（１００）を受け取るように構成され、１つ以上の古典的アルゴリズムを用いて所定のタイムアウト期間内に結果のＷＦＦ（１０４）を判定するように構成された古典的プロセッサ（１４）をさらに含み、

前記古典的プロセッサ（１４）は、前記結果のＷＦＦおよびその欠如を式プロセッサ（１８）に伝えるように構成され、

前記式プロセッサ（１８）は、前記古典的プロセッサから結果のＷＦＦ（１０４）を受け取らなかったときに、前記入力ＷＦＦ（１００）を受け取るように構成されており、

前記古典的アルゴリズムは前記入力ＷＦＦ（１００）の前記連言標準形または前記選言標準形を判定するためのアルゴリズムである、請求項８から１３に記載のシステム。

【発明の詳細な説明】

【技術分野】

【０００１】

本開示は、全般的には、命題論理式または完全論理式の操作に関する。

【背景技術】

【０００２】

航空機または宇宙打上用ロケットなどの乗物の概念設計は、通常、多数のシステム構成および基準が考慮され得る設計トレードオフ研究またはトレード研究の組を伴う。最適設計に達するためには、乗物の性能、コスト、信頼性、および複数の分野にまたがるさまざまな他の要因の観点からさまざまな候補設計概念を評価することが望ましい。候補設計概念の評価を、制約ネットワークなどの計算手順で実施することができ、この制約ネットワークでは、当技術分野で既知のように、制約ネットワーク内の複数の変数の間の数学的関係が、関連する独立変数の値を与えられて、制約ネットワークによって決定される、変数のうちの任意の１つの値の計算を可能にする。設計トレード研究では、候補設計概念の評価が、設計変数、性能変数、およびコスト変数の間の制約を表す大量の代数方程式を伴う場合がある。

【０００３】

トレード研究応用のための制約ネットワークの古典的実施態様では、代数方程式の組は静的であり、すべての方程式が常に満足される。さらに、代替の計算方法が、航空機の空力抵抗を判定する次の表現のように、選択された方程式に組み込まれる。

ｄｒａｇＰｌａｎｅ＝Ｉｆ（ＣａｎａｒｄＩｓＰｒｅｓｅｎｔ，ｄｒａｇＢｏｄｙ＿ＣａｎａｒｄＡｔｔａｃｈｅｄ（ＦｕｓｅｌａｇｅＳｉｚｅ）＋ｄｒａｇＣａｎａｒｄ（ＣａｎａｒｄＳｉｚｅ），ｄｒａｇＢｏｄｙ＿ＮｏＣａｎａｒｄ（ＦｕｓｅｌａｇｅＳｉｚｅ））

【０００４】

残念ながら、上に示した表現のように方程式に計算方法を組み込むことは、多数の異な

る構成を組み込んだ複雑系のモデラについて扱いにくいものになる可能性がある。さらに、方程式に計算方法を組み込むことは、計算フローの逆転を必要とする、あるタイプのトレード研究の実行を妨げる可能性がある。

【０００５】

方程式に計算方法を組み込むことに対する代替案は、任意の所与の方程式の適用可能性を、同一の制約ネットワークによって決定される計算状態に依存するものにすることである。制約ネットワークモデリングの重要なプロパティは、計算パス内の制約集合の数値解からの計算プラニング（すなわち、所与のトレード研究の実行中の、計算ステップの順序付けられたシーケンスまたはネットワークを介する計算パスの決定）の分離である。計算パスのプラニングのそのような分離は、システム設計者にトレード研究中の比較的迅速なフィードバックを与えるために必須である。これは、システム設計者が、トレード研究中にさまざまな設計空間を探査することを可能にする。

【０００６】

各方程式の適用可能性が、静的ではなく、データ依存である場合には、そのようなデータ依存性をモデリングする有効な技法は、命題形式または完全論理式（ＷＦＦ）を各方程式に結び付けることであり、このＷＦＦは、ネットワーク内のデータに依存し、ＷＦＦが真と評価される場合に方程式が所与の状況で適用可能であることを意味する。これに関して、各ＷＦＦは、そのＷＦＦが真である世界の集合を定義する真値を有する。

【０００７】

データ依存ネットワーク内の計算プランを、計算ステップの順序付けられたシーケンスとしてモデリングすることもできる。そのような配置では、各計算ステップは、命題形式またはＷＦＦを関連付けられ、このＷＦＦは、ネットワーク内のデータおよび前の計算ステップで計算された結果に依存し、ＷＦＦが真と評価される場合には、計算ステップが所与の状況で評価されることを意味する。各計算ステップに関連付けられたＷＦＦは、解かれる必要がある方程式に関連するＷＦＦに対する和集合演算子、共通部分演算子、および差分演算子の異なる組合せによって入手することができる。そのようなＷＦＦは、制約ネットワーク内のどの変数が独立であり、そのような命題ＷＦＦの迅速な操作および組合せを必要とするのかに依存して非常に複雑になる可能性がある。他のＷＦＦの組合せを介して入手されるＷＦＦは、トレード研究中の効率的な計算のために単純化を必要とする。これに関して、ＷＦＦの組合せを単純化されない状態のまま残すことは、数千個の方程式を伴う比較的大きいネットワーク内でＷＦＦがさらに組み合わされる時に、メモリサイズの爆発をもたらす可能性がある。さらに、ＷＦＦが、例外なく偽のＷＦＦに単純化される時には、計算プラン生成手順は、制約ネットワークの不必要な分岐を枝刈りすることができ、これによって、コンパクトで効率的な計算プランを作ることができる。

【０００８】

そのようなＷＦＦ単純化プロセスは、有限だが大きい領域にわたる大量の述語を有する論理式に適用される時に、極端に計算集中型になる可能性がある。ＷＦＦの連言標準形またはＷＦＦの選言標準形を判定する古典的アルゴリズムは、計算結果を比較的短い時間期間（たとえば、数分）でシステム設計者に与えるのに不適当である。ＷＦＦの単純化は、好ましくは、計算時間を減らし、異なる設計トレードを検討し、調査するためにシステム設計者が使用できる時間の長さを増やすために、できる限りすばやく実行される。完全論理式を単純化する時間の長さの削減は、さらに、より大きくより複雑な設計空間を調査する能力をシステム設計者に与えることができる。

【０００９】

たとえば、超音速飛翔体の概念設計では、制約管理プラニングアルゴリズムが、多数の所望のトレード研究のうちの１つのプラニング中に大量の述語への多数の参照を含む多数のＷＦＦを単純化するために必要である。例のＷＦＦは、１０個から１５個までの述語のみを有することができ、各述語は、２個から２０個までの可能な値を有する。そのようなＷＦＦは、構文的に、類似するスケールである深さを伴って同一の述語を５回から１０回参照することができる（たとえば、Ａｎｄ（Ｏｒ（Ａｎｄ　Ｏｒ（Ｐ１＝－ｐ１１，Ｐ２

＝ｐ２１…）…Ｏｒ（Ａｎｄ（Ｐ１＝ｐ１３，Ｏｒ（Ｎｏｔ（Ｐ１＝ｐ１３）…））））など）。残念ながら、古典的アルゴリズムを使用する連言標準形または選言標準形へのそのようなＷＦＦの単純化は、一実施態様で１０分から３０分までの計算時間を必要とする。古典的アルゴリズムを使用するＷＦＦの単純化の計算時間の比較的長い期間は、異なる設計トレードを検討し、調査するために設計者が使用可能な時間を直接に減らす。

【発明の概要】

【発明が解決しようとする課題】

【００１０】

上からわかるように、当技術分野には、制約ネットワークなどの論理依存システムで完全論理式を単純化するのに必要な時間の長さを減らすシステムおよび方法の必要がある。

【課題を解決するための手段】

【００１１】

論理依存システムでの論理式の単純化に関連する上で示した必要は、本開示によって対処され、軽減され、本開示は、一実施形態で、制約ネットワークでの計算プラニングのプロセスをサポートできる、完全論理式（ＷＦＦ）を単純化する方法を提供する。一実施形態では、この方法は、式プロセッサ内で入力完全論理式（ＷＦＦ）を受け取るステップを含むことができる。この方法は、式コンバータを用いて、入力ＷＦＦを初期ビット配列に変換することをさらに含むことができる。さらに、この方法は、ビット配列シンプリファイヤを用いて、入力ＷＦＦを表すのに必要ではない述語を初期ビット配列から除去することによって、初期ビット配列を単純化されたビット配列に単純化することを含むことができる。単純化されたビット配列を、ビット配列コンバータによって戻りＷＦＦに変換することができる。この方法は、式プロセッサを用いて、戻りＷＦＦを返すことを含むことができる。

【００１２】

さらなる実施形態では、所定の時間期間内に結果のＷＦＦを生成するために１つまたは複数の古典的アルゴリズムを使用してデータ依存制約管理システムの少なくとも１つの入力ＷＦＦを古典的プロセッサを用いて処理するステップを含む、迅速な設計制約式計算をサポートする方法を開示する。この方法は、結果のＷＦＦまたはその不在を式プロセッサに通信することをさらに含むことができる。さらに、この方法は、式コンバータを用いて入力ＷＦＦを初期ビット配列に変換することを含むことができる。さらに、この方法は、ビット配列シンプリファイヤを用いて、入力ＷＦＦを表すのに必要ではない述語を初期ビット配列から除去することによって、初期ビット配列を単純化されたビット配列に単純化することを含むことができる。単純化されたビット配列を、ビット配列コンバータによって戻りＷＦＦに変換することができる。この方法は、式プロセッサを用いて、戻りＷＦＦを返すことを含むことができる。

【００１３】

迅速な設計制約式計算をサポートするシステムをも開示する。このシステムは、所定の時間期間内に結果のＷＦＦを生成するために１つまたは複数の古典的アルゴリズムを使用して入力ＷＦＦを処理するように構成され得る古典的プロセッサを含むことができる。古典的プロセッサを、結果のＷＦＦまたは結果のＷＦＦの不在をプロセッサの外部に通信するように構成することができる。このシステムは、古典的プロセッサに結合され、入力ＷＦＦを初期ビット配列に変換し、入力ＷＦＦを表すのに必要ではない述語を初期ビット配列から除去することによって、初期ビット配列を単純化されたビット配列に単純化し、単純化されたビット配列を戻りＷＦＦに変換するように構成された式プロセッサを含むことができる。

【００１４】

上で議論した特徴、機能、および利益を、本開示のさまざまな実施形態で独立に達成することができ、あるいは、他の実施形態で組み合わせることができ、そのさらなる詳細を、次の説明および図面を参照して知ることができる。

【００１５】

本開示の上記および他の特徴は、図面を参照する時により明白になり、図面では、同様の符号が、終始同様の部分を指す。

【図面の簡単な説明】

【００１６】

【図１】まずタイムアウト期間内に古典的アルゴリズムを使用して入力完全論理式（ＷＦＦ）を単純化することを試みること、タイムアウト期間を過ぎた場合に、入力ＷＦＦを初期ビット配列に変換すること、初期ビット配列を単純化すること、および単純化されたビット配列を戻りＷＦＦに変換することによって入力ＷＦＦを単純化する方法を示す流れ図である。

【図２Ａ】入力ＷＦＦを初期ビット配列に変換する方法に含めることができる１つまたは複数の動作を示す流れ図である。

【図２Ｂ】入力ＷＦＦを初期ビット配列に変換する方法に含めることができる１つまたは複数の動作を示す流れ図である。

【図３】配列キャッシュから初期ビット配列を取り出す方法に含めることができる１つまたは複数の動作を示す流れ図である。

【図４】初期ビット配列を構成する方法に含めることができる１つまたは複数の動作を示す流れ図である。

【図５】初期ビット配列を単純化する方法に含めることができる１つまたは複数の動作を示す流れ図である。

【図６】単純化されたビット配列を戻りＷＦＦに変換する方法に含めることができる１つまたは複数の動作を示す流れ図である。

【図７】単純化されたビット配列を選言標準形（ＤＮＦ）ＷＦＦに変換する方法に含めることができる１つまたは複数の動作を示す流れ図である。

【図８】単純化されたビット配列を連言標準形（ＣＮＦ）ＷＦＦに変換する方法に含めることができる１つまたは複数の動作を示す流れ図である。

【図９】原子ＷＦＦを作成する方法に含めることができる１つまたは複数の動作を示す流れ図である。

【図１０】否定の原子ＷＦＦを作成する方法に含めることができる１つまたは複数の動作を示す流れ図である。

【図１１】本明細書で開示されるシステムおよび方法を使用して生成できる初期ビット配列の実施形態を示す図であり、初期ビット配列は、ブール述語および等価述語を含み、それに関連する６つの可能世界のうちの２つを表す、入力ＷＦＦに対応することができる。

【図１２】ブール述語に関連する６つの可能世界のうちの４つおよび等価述語に関連する領域要素を表す入力ＷＦＦに対応する初期ビット配列のさらなる実施形態を示す図である。

【図１３】図２Ａ～２Ｂに示された方法に従って入力ＷＦＦを初期ビット配列に変換するトップレベルルーチンを示す擬似コードリスティングである。

【図１４】入力ＷＦＦを初期ビット配列に変換する図２Ａ～２Ｂに示された方法を実施するルーチンを示す擬似コードリスティングである。

【図１５】配列キャッシュから初期ビット配列を取り出す図３に示された方法を実施するルーチンを示す擬似コードリスティングである。

【図１６】図４に示された方法を使用して構成できる初期ビット配列の実施形態を示す図であり、この方法は、図１５および１７に示された擬似コードリスティングによって表されるルーチンを使用して実行することができる。

【図１７】初期ビット配列を構成する図４に示された方法で実施できるルーチンを示す擬似コードリスティングである。

【図１８】図１５の初期ビット配列の単純化された版を示す図であり、この単純化されたビット配列を、図５に示された方法を使用して単純化することができる。

【図１９】初期ビット配列を単純化する図５に示された方法で実施できるルーチンを示す擬似コードリスティングである。

【図２０】単純化されたビット配列を戻りＷＦＦに変換する図６に示された方法を実施するルーチンを示す擬似コードリスティングである。
【図２１】単純化されたビット配列を選言標準形の戻りＷＦＦに変換する図７に示された方法を実施するルーチンを示す擬似コードリスティングである。
【図２２】単純化されたビット配列を連言標準形の戻りＷＦＦに変換する図８に示された方法を実施するルーチンを示す擬似コードリスティングである。
【図２３】入力ＷＦＦを単純化する方法の１つまたは複数の動作を実施するプロセッサベースのシステムの実施形態を示すブロック図である。
【発明を実施するための形態】
【００１７】
図示が本開示のさまざまな実施形態を示すためのものである図面を参照すると、図１に示されているのは、制約管理システム１２（図２３）内で設計制約式計算をサポートするためなど、複雑な命題論理式または複雑な完全論理式（ＷＦＦ）１００を単純化する方法３００の図である。本開示では、入力ＷＦＦ　１００は、否定演算子１３０、連言演算子１３２、および選言演算子１３４（図２Ａ～２Ｂ）を用いてブール変数すなわちブール述語１２４（図１１）および等価述語１２６（図１１）を組み合わせる。システム（図２３）および方法３００は、入力ＷＦＦ　１００を、ステップ３０８（図１）で、入力ＷＦＦ　１００の初期ビット配列２００（図１）表現に変換する。ステップ３１０（図１）では、初期ビット配列２００を、単純化されたビット配列２１２に変換して、入力ＷＦＦ　１００を表すために必要ではない述語１１４（図１１）を除去する。ステップ３１２では、単純化されたビット配列２１２を、選言標準形（ＤＮＦ）１４０（図６）および／または連言標準形（ＣＮＦ）１４２（図６）を含む最小限の標準形の戻りＷＦＦ　１３８（図６）に変換する。戻りＷＦＦ　１３８を、ステップ３１４で式プロセッサ１８（図２３）によって返すことができる。
【００１８】
データ依存制約管理システム１２（図２３）内などでシステム１０（図２３）および方法３００（図１）を実施することの技術的影響は、完全論理式の和集合演算、共通部分演算、および差分演算の結果を単純化するのに必要な時間の長さの大幅な短縮である。さらに、データ依存制約管理システムでの方法の実施は、モデルローディングおよびトレード実行に必要な時間の長さを大幅に短縮することができる。たとえば、本発明人は、高度な超音速飛翔体設計の特定のトレード研究を計算するのに必要な時間の長さが数時間から数分に減らされた、本開示で説明される方法の実施形態を観察した。処理時間の短縮は、設計者がより大きくより複雑な設計空間を統合された形で探索することを可能にするという技術的影響を提供する。上で示したように、入力ＷＦＦの単純化は、トレード研究中にデータ依存制約ネットワークのノードをトラバースする時の分岐を減らすことができる。分岐を減らすことおよび複雑なＷＦＦの単純化に必要な時間の長さを減らすことによって、このシステムおよび方法は、それぞれが性能および／またはコストを評価するための異なる分析モデルを含む可能性がある広い範囲の代替システム構成を設計者がすばやくトレードオフすることを可能にするという技術的影響を提供する。
【００１９】
制約ネットワーク内の任意の個々の方程式または変数ノードに関連するＷＦＦが単純である場合があるが、そのようなＷＦＦが、所与のトレード研究またはネットワーク内の仮定される独立変数の所与の集合に関する計算パスに沿って、和集合演算および共通部分演算を使用して組み合わされることに留意されたい。たとえば、ＷＦＦ１ａを有するＮｏｄｅ１ａおよびＷＦＦ１ｂを有するＮｏｄｅ１ｂによって表される変数が、計算パス内のＷＦＦ２を有するＮｏｄｅ２によって表される関係の相補的先行値である場合には、Ｎｏｄｅ２を、選言標準形Ｏｒ（Ａｎｄ（ＷＦＦ１ａ，ＷＦＦ２），Ａｎｄ（ＷＦＦ１ｂ，ＷＦＦ２））またはより単純に連言標準形Ａｎｄ（Ｏｒ（ＷＦＦ１ａ，ＷＦＦ１ｂ），ＷＦＦ２）によって表される世界の集合内で実行することができる。計算パスは、数百個から数千個のステップを含む場合があり、複雑さの組合せ増加が発生する可能性がある。

【００２０】
データ依存制約管理システム１２の文脈で説明されるが、本明細書で開示されるシステムおよび方法を、単純なＷＦＦの長々しい組合せの操作を必要とするアプリケーションを含む命題論理式の操作を必要とするすべてのアプリケーションで有利に実施することができる。本開示が、米国カリフォルニア州オークランドのＦｒａｎｚ，Ｉｎｃ．から市販されているＡｌｌｅｇｒｏ　Ｃｏｍｍｏｎ　Ｌｉｓｐなどの言語でのリスト構造としてのＷＦＦの操作の文脈でシステムおよび方法を説明するが、このシステムおよび方法を、代替の形でのＷＦＦを操作するように動作可能とすることができることにも留意されたい。たとえば、このシステムおよび方法を、そのようなＷＦＦの単純化での改善された効率をもたらすオブジェクトとして表されたＷＦＦを操作するように動作可能とすることができる。

【００２１】
本開示では、入力ＷＦＦ　１００（図１）を、リストとして表すことができ、このリストでは、リスト内の最初の要素を、入力ＷＦＦ　１００のトップレベル演算子とすることができる。本開示の文脈での入力ＷＦＦ　１００は、次のうちの１つを含むことができる。

１．記号ＴまたはＦ（それぞれ真および偽を表す）、
２．次のいずれかである述語、
ａ．ブール述語（Ｂ１，Ｂ２，…）
ｂ．それぞれそれ自体の有限領域（Ｄ（Ｐ１），Ｄ（Ｐ２），…）にわたる等価述語（Ｐ１，Ｐ２，…）
３．Ｎｏｔ（ＷＦＦ）のように、ＷＦＦに適用される否定演算子Ｎｏｔ、
４．Ａｎｄ（ＷＦＦ１，ＷＦＦ２，…）のように、０個以上のＷＦＦに適用される連言演算子Ａｎｄ、
５．Ｏｒ（ＷＦＦ１，ＷＦＦ２，…）のように、０個以上のＷＦＦに適用される選言演算子Ｏｒ。

【００２２】
定義により、Ａｎｄ（）＝ＴおよびＯｒ（）＝Ｆであることに留意されたい。論理演算子「Ａｎｄ」、「Ｏｒ」、または「Ｎｏｔ」のうちの１つまたは複数を含むＷＦＦを、本明細書では「複合」ＷＦＦと称する。他のすべてのＷＦＦを、「原子」ＷＦＦと称する。原子ＷＦＦは、記号ＴおよびＦを含み、これらを、本明細書では「真の原子」ＷＦＦおよび「偽の原子」ＷＦＦと称する。原子ＷＦＦは、単一のブール述語ならびに「ＥｎｇｉｎｅＴｙｐｅ　＝　Ｒｏｃｋｅｔ」など、述語をその（有限の）領域内の要素と等価にすることによって形成される「等価」ＷＦＦを含むＷＦＦをも含み、ここで、ＥｎｇｉｎｅＴｙｐｅは、述語であり、「Ｒｏｃｋｅｔ」、「Ｒａｍｊｅｔ」、「Ｔｕｒｂｏｊｅｔ」その他などの値を有することができる。本開示は、世界が等価述語のそれぞれについて領域値の指定を含む可能世界のセマンティックスの文脈でＷＦＦを説明する。さらに、世界は、各ブール述語（たとえば、$Ｂ_{ｉ}$）またはその否定（たとえば、Ｎｏｔ（$Ｂ_{ｉ}$））が真であるかどうかに関する指定を含む。たとえば、

Ｐ１　＝　ＥｎｇｉｎｅＴｙｐｅ　ｗｉｔｈ　Ｄｏｍａｉｎ｛Ｒｏｃｋｅｔ，Ｒａｍｊｅｔ，Ｔｕｒｂｏｊｅｔ｝
かつ
Ｂ１　＝　ＨａｓＬｉｆｔｉｎｇＢｏｄｙ．

の場合には、複合ＷＦＦ

Ａｎｄ（Ｏｒ（（＝　ＥｎｇｉｎｅＴｙｐｅ　Ｒａｍｊｅｔ），（＝　ＥｎｇｉｎｅＴｙｐｅ　Ｔｕｒｂｏｊｅｔ）），Ｎｏｔ（ＨａｓＬｉｆｔｉｎｇＢｏｄｙ））

は、検討される乗物が揚力物体（ＬｉｆｔｉｎｇＢｏｄｙ）を有しておらず、ラムジェット（Ｒａｍｊｅｔ）またはターボジェット（Ｔｕｒｂｏｊｅｔ）のいずれかである機関を有する世界のすべてを表す。ＷＦＦによって指定される世界のもう１つの例では、次のＷＦＦ

Ｏｒ（（＝ ＥｎｇｉｎｅＴｙｐｅ Ｒａｍｊｅｔ），（＝ ＥｎｇｉｎｅＴｙｐｅ Ｔｕｒｂｏｊｅｔ））

は、乗物が揚力物体を有するか否かに関わりなく、検討される乗物がラムジェットまたはターボジェットのいずれかである機関を有する世界のすべてを表す。上で議論したように、原子ＷＦＦおよび／または複合ＷＦＦを、関係を条件付けるために制約管理システムで使用することができる。たとえば、検討される乗物に適用できる抗力方程式の特定の形を、乗物が揚力物体を有するか否か、乗物の機関のタイプ（たとえば、ロケット、ターボジェット、ラムジェットなど）、および／またはＷＦＦを表すことができる他の要因によって規定することができる。

【００２３】

図１を参照すると、方法３００のステップ３０２では、入力ＷＦＦ １００を単純化することを試みるために古典的プロセッサ１４（図２３）内で、入力ＷＦＦ １００を最初に受け取ることができる。ステップ３０４は、所定のタイムアウト期間３０６内に連言標準形（ＣＮＦ）または選言標準形（ＤＮＦ）の結果のＷＦＦ １０４を生成するために古典的プロセッサ１４内で１つまたは複数の古典的アルゴリズム１６（図２３）を使用することを含むことができる。当技術分野で既知のように、古典的アルゴリズムを使用する結果のＷＦＦへの入力ＷＦＦ １００の単純化または変換の焦点を、ブール文の充足可能性に合わせることができる。そのような充足可能性を、ＷＦＦの変数の割当が真に対するＷＦＦ評価をもたらすかどうかの判定に向けることができ、そのような充足可能性は、ＷＦＦがトートロジである（すなわち、必ず真である）、ＷＦＦが矛盾である（すなわち、絶対に真ではない）、またはＷＦＦが真であることをもたらす変数の割当とＷＦＦが偽であることをもたらす変数の他の割当とがあるという３つのケースを含む可能性がある。

【００２４】

上記で示したように、古典的アルゴリズム１６（図１）を使用するＷＦＦの単純化は、計算集中型である可能性がある。これに関して、古典的アルゴリズム１６を使用するＤＮＦ形１４０（図６）またはＣＮＦ形１４２（図６）での結果のＷＦＦ １０４（図１）への入力ＷＦＦ １００の変換は、残念ながら、変換を実行するためにかなりの時間およびメモリを要求する可能性がある。有利なことに、本明細書で開示される方法３００（図１）では、古典的プロセッサ１４（図１）が所定のタイムアウト期間３０６内に１つまたは複数の古典的アルゴリズム１６を使用して結果のＷＦＦ １０４を生成できない場合に、入力ＷＦＦ １００が、式プロセッサ１８（図２３）に通信される。タイムアウト期間３０６の満了は、オペレーティングシステムが、古典的計算プロセスに割り込み、古典的プロセッサ１４から式プロセッサ１８（図２３）に制御を渡すことができる。その後、式プロセッサ１８は、下で説明するように、入力ＷＦＦ １００を初期ビット配列２００に変換し、初期ビット配列２００を単純化し、単純化されたビット配列２１２を戻りＷＦＦ １３８に変換することができる。

【００２５】

より具体的には、図１で、方法３００のステップ３０２は、式プロセッサ１８（図２３）で入力ＷＦＦ １００を受け取ることを含むことができる。図２３に示されているように、式プロセッサ１８は、式コンバータ２０、ビット配列シンプリファイヤ２２、および／またはビット配列コンバータ２４を含むことができる。図１の方法３００のステップ３０８は、式コンバータ２０を使用して入力ＷＦＦ １００を初期ビット配列２００に変換することを含むことができる。下でより詳細に説明するように、初期ビット配列２００への入力ＷＦＦ １００の変換は、それぞれ入力ＷＦＦ １００が真の原子ＷＦＦ １４８

（図２Ａ）または偽の原子ＷＦＦ　１５０（図２Ａ）である場合の世界の真の配列２１４（図２Ａ）または世界の偽の配列２１６（図２Ａ）の生成を含むことができる。他のタイプのＷＦＦについて、初期ビット配列２００への入力ＷＦＦ　１００の変換は、初期ビット配列２００に必要な次元、述語、および領域の判定と、その後の、配列キャッシュ３０からの適当な初期ビット配列２００の取出、または配列キャッシュ３０の検索が正しい構成の初期ビット配列２００の回復に失敗する場合の初期ビット配列２００の構成とを含むことができる。

【００２６】

本明細書で開示されるビット配列を説明するために図１１を参照すると、初期ビット配列２００の実施形態の例が示されている。本開示では、ビット配列を、「１」または「０」のいずれかの値を有するビット要素２２２を有する配列として定義することができる。さらに、ビット配列は、任意の個数の次元を含むことができる。各次元は、異なるサイズを有することができる。ブール述語１２４について、対応するビット配列次元のサイズは、２である。等価述語１２６について、ビット配列次元のサイズは、領域の長さと等しい。本開示では、論理ビット配列を、ビット配列に含まれる述語（たとえば、ブールおよび／または等価の）のリストへのビット配列の各次元の写像を含むビット配列として定義することができる。前に示したように、入力ＷＦＦ　１００（図１）を、初期ビット配列２００によって表すことができる。本開示では、初期ビット配列２００を、Ａ［ｉ，ｊ，ｋ　…］＝１（たとえば、ビット要素値）の場合に、ブール述語および等価述語の値が配列の（ｉ，ｊ，ｋ，…）によって表される世界が真であるというセマンティクスを使用して説明することができる。

【００２７】

初期ビット配列２００の例を、図１１に示し、図１１は、ブール述語Ｂ１（ＨａｓＬｉｆｔｉｎｇＢｏｄｙ）（符号１２４）と、領域｛Ｒｏｃｋｅｔ，Ｒａｍｊｅｔ，Ｔｕｒｂｏｊｅｔ｝（符号１１８）にわたる等価述語Ｐ１（ＥｎｇｉｎｅＴｙｐｅ）（符号１２６）とを含む論理ビット配列系列を表し、Ｒｏｃｋｅｔ、Ｒａｍｊｅｔ、およびＴｕｒｂｏｊｅｔは、等価述語Ｐ１の領域１１８の領域要素１２０である。上で述べたセマンティクスに基づいて、初期ビット配列２００を、次によって表すことができる。

Ａ［1，1］：ＨａｓＬｉｆｔｉｎｇＢｏｄｙ Ａｎｄ ＥｎｇｉｎｅＴｙｐｅ＝Ｒｏｃｋｅｔ
Ａ［2，1］：Ｎｏｔ（ＨａｓＬｉｆｔｉｎｇＢｏｄｙ） Ａｎｄ ＥｎｇｉｎｅＴｙｐｅ＝Ｒｏｃｋｅｔ
Ａ［1，2］：ＨａｓＬｉｆｔｉｎｇＢｏｄｙ Ａｎｄ ＥｎｇｉｎｅＴｙｐｅ＝Ｒａｍｊｅｔ
Ａ［2，2］：Ｎｏｔ（ＨａｓＬｉｆｔｉｎｇＢｏｄｙ） Ａｎｄ ＥｎｇｉｎｅＴｙｐｅ＝Ｒａｍｊｅｔ
Ａ［1，3］：ＨａｓＬｉｆｔｉｎｇＢｏｄｙ Ａｎｄ ＥｎｇｉｎｅＴｙｐｅ＝Ｔｕｒｂｏｊｅｔ
Ａ［2，3］：Ｎｏｔ（ＨａｓＬｉｆｔｉｎｇＢｏｄｙ） Ａｎｄ ＥｎｇｉｎｅＴｙｐｅ＝Ｔｕｒｂｏｊｅｔ

ここで、初期ビット配列２００内の各ビット要素（たとえば、１または０）（符号２２２）は、領域｛Ｒｏｃｋｅｔ，Ｒａｍｊｅｔ，Ｔｕｒｂｏｊｅｔ｝にわたるブール述語（ＨａｓＬｉｆｔｉｎｇＢｏｄｙ）および等価述語（ＥｎｇｉｎｅＴｙｐｅ）の６つの可能世界の指定に対応する。

【００２８】

図１１では、次の入力ＷＦＦ　１００が、初期ビット配列２００に対応する６つの可能世界のうちの２つを指定する。

And(Or((= EngineType Ramjet), (= EngineType Turbojet)), Not(HasLiftingBody))

【００２９】

図１２では、次の入力ＷＦＦ １００が、初期ビット配列２００に対応する６つの可能世界のうちの４つを指定する。

Or((= EngineType Ramjet), (= EngineType Turbojet))

【００３０】

了解されるように、図１１および１２に示された初期ビット配列２００は、入力ＷＦＦ １００を表す初期ビット配列２００の非限定的な実施形態である。本開示では、入力ＷＦＦ １００を、任意の個数の次元を有する初期ビット配列２００によって表すことができ、領域１１８は、有限の長さを有することができる。

【００３１】

本明細書で開示される方法３００の図２Ａ～２Ｂを参照して、入力ＷＦＦ １００を初期ビット配列２００に変換する３０８のステップに含めることができる１つまたは複数の動作を示す。上で示した動作を、入力ＷＦＦ １００に対して再帰的に実行することができ、初期ビット配列２００を構成することができ、述語１１４（図１１～１２）のリストを、述語１１４が初期ビット配列２００内に現れる順序で生成することができる。たとえば、図１１および１２に示された初期ビット配列２００について、述語リスト１１６を、次によって表すことができる。

1. EngineType
2. HasLiftingBody

【００３２】

述語リスト１１６内の各述語１１４は、それに関連する領域１１８を有する。たとえば、等価述語１２６（EngineType）は、領域要素１２０{Rocket, Ramjet, Turbojet}を含む。ブール述語１２４（HasLiftingBody）は、領域要素１２０{True, False}を含む。本開示は、初期ビット配列２００の構成およびそれに関連する述語リスト１１６の生成を容易にする１つまたは複数のルーチンまたは関数を含むことができる。そのようなルーチンを、方法３００（図１）のプログラミング命令内に記述することができ、かつ／またはルーチンを、基礎になるプログラミング言語に含めることができる。次は、初期ビット配列２００への入力ＷＦＦ １００の変換を容易にするためにシステム１０（図２３）および方法３００（図１）に含めることができるルーチンまたは機能のうちの１つまたは複数の短い説明を含む。

【００３３】

関数「Operator（ＷＦＦ）」は、任意の入力ＷＦＦのトップレベル演算子を返すように構成される。そのような演算子は、ヌル、空集合、True（Ｔ）述語およびFalse（nil）述語のnil（または同等物）、ブール述語のブール述語自体、等価述語の＝（等価演算子）、ならびに複合ＷＦＦの連言演算子、選言演算子、または否定演算子（それぞれ、And、Or、またはNot）のいずれかである。

【００３４】

関数「Arg1（ＷＦＦ）」は、入力ＷＦＦの引数またはオペランドのリストの最初の要素を返すように構成される。そのような引数またはオペランドは、True述語、False述語、およびブール述語のnil、等価述語で使用される述語、And演算子、Or演算子、またはNot演算子に続く最初のＷＦＦ（たとえば、第１オペランド）、または何もない場合のnilである。

【００３５】

関数「Arg2（ＷＦＦ）」は、入力ＷＦＦの引数またはオペランドのリストの２番目の要素を返すように構成される。そのような引数またはオペランドは、True述語、F

ａｌｓｅ述語、およびブール述語のｎｉｌ、等価述語の領域値、Ａｎｄ演算子、Ｏｒ演算子、またはＮｏｔ演算子に続く２番目のＷＦＦ（たとえば、第２オペランド）、または何もない場合のｎｉｌである。

【００３６】

関数「Ｐｏｓｉｔｉｏｎ（ｉｔｅｍ，ｌｉｓｔ）」は、任意のリストｌｉｓｔについて、項目ｉｔｅｍの最初の出現のリストへのインデックスを返すように構成される。これに関して、「Ｐｏｓｉｔｉｏｎ（ｉｔｅｍ，ｌｉｓｔ）」は、リスト内の項目の位置を返す。たとえば、Ｐｏｓｉｔｉｏｎ（ｄ，｛ａ，ｂ，ｃ，ｄ，ｅ，ｆ｝）＝４である。

【００３７】

関数「Ｅｌｅｍｅｎｔ（ｌｉｓｔ，ｉｎｄｅｘ）」は、任意のリストｌｉｓｔについて、リスト内のインデクシングされた項目ｉｔｅｍを返すように構成される。これに関して、Ｅｌｅｍｅｎｔ（ｌｉｓｔ，ｉｎｄｅｘ）は、「Ｐｏｓｉｔｉｏｎ（ｉｔｅｍ，ｌｉｓｔ）」によって判定されるリスト位置を有する項目の値またはアイデンティティを返す。たとえば、Ｉｎｄｅｘ（｛ａ，ｂ，ｃ，ｄ，ｅ，ｆ｝，４）＝ｄである。

【００３８】

追加のルーチンまたは関数を、本システムおよび本方法を実行するためにプログラミング言語の実施態様に含めることができる。

【００３９】

たとえば、関数「ＢｉｔＮｏｔ（ａｒｒａｙ）」を、各ビット要素が１から０へおよび０から１へ反転された初期ビット配列を返すために提供することができる。

【００４０】

関数「ＢｉｔＡｎｄ（ａｒｒａｙ１，ａｒｒａｙ２）」では、入力配列すなわちａｒｒａｙ１およびａｒｒａｙ２が、同一の次元を有することを要求される。「ＢｉｔＡｎｄ（ａｒｒａｙ１，ａｒｒａｙ２）」関数は、個々の初期ビット配列のビット要素の「ＡＮＤ」（すなわち、連言）と等しいビット要素を有する、同一次元の配列を返す。たとえば、ＡＮＤ（１，１）＝１、ＡＮＤ（１，０）＝ＡＮＤ（０，１）＝ＡＮＤ（０，０）＝０である。

【００４１】

関数「ＢｉｔＯｒ（ａｒｒａｙ１，ａｒｒａｙ２）」では、入力配列すなわちａｒｒａｙ１およびａｒｒａｙ２が、関数「ＢｉｔＡｎｄ（ａｒｒａｙ１，ａｒｒａｙ２）」について要求されるものに似て、同一の次元を有することを要求される。「ＢｉｔＯｒ（ａｒｒａｙ１，ａｒｒａｙ２）」関数は、個々の初期ビット配列のビット要素の「ＯＲ」と等しいビット要素を有する、同一次元の配列を返す。たとえば、ＯＲ（１，１）＝ＯＲ（１，０）＝ＯＲ（０，１）＝１、ＯＲ（０，０）＝０である。

【００４２】

関数「ＳｅｔＢｉｔ（ａｒｒａｙ，ｉｎｄｉｃｅｓ，ｖａｌｕｅ）」は、インデックス「ｉｎｄｉｃｅｓ」の項目のリストによってインデクシングされる配列「ａｒｒａｙ」のビット要素に、０または１のいずれかである値「ｖａｌｕｅ」をセットする。たとえば、２×３初期ビット配列について、「ＳｅｔＢｉｔ（ａｒｒａｙ，（２，１），１）」は、「ａｒｒａｙ」の第２行第１列のビット要素（たとえば、図１１～１２の符号２２２）の値に１をセットする。

【００４３】

図２Ａ～２Ｂに示されているのは、入力ＷＦＦ　１００を初期ビット配列２００に変換するステップ３０８である。ステップ３０８は、入力ＷＦＦ　１００のタイプに応じて入力ＷＦＦ　１００の処理をディスパッチする。入力ＷＦＦ　１００は、真または偽の原子ＷＦＦ　１０８、原子ブール述語ＷＦＦ　１２４、原子等価述語ＷＦＦ　１２６、否定演算子（ＮＯＴ）を含む否定ＷＦＦ　１３０、ならびに連言演算子ＡＮＤ　１３２および選言演算子ＯＲ　１３４（図２Ｂ）を含む複合ＷＦＦ　１３６を含むことができる。この方法のステップ３０８は、入力ＷＦＦ　１００内の述語１１４を判定するステップ３１６、述語１１４の各１つに関連する領域要素１２０を判定するステップ３１８、および初期ビ

ット配列２００のビット配列次元２１８（図１１）を判定するステップ３２０を含む。この方法のステップ３２２は、内部プログラム（たとえば、「ＷＦＦ＿ｔｏ＿ａｒｒａｙ＿ｉｎｔｅｒｎａｌ」）を呼び出すことによって入力ＷＦＦ　１００を処理すること（たとえば、再帰的に）および入力ＷＦＦ　１００に関連するビット配列次元２１８、述語１１４、および領域要素１１８、１２０を有する初期ビット配列２００を返すこと３０９を含む。図１３は、入力ＷＦＦ　１００を初期ビット配列２００に変換する、ステップ３０８（図２Ａ～２Ｂ）に関連する動作に対応するトップレベルメソッドの擬似コードリスティング２４８である。

【００４４】

簡単に図１４を参照すると、図２Ａ～２Ｂに示された形で入力ＷＦＦ　１００を初期ビット配列２００に再帰的に処理する上で述べた内部プログラム「ＷＦＦ＿ｔｏ＿ａｒｒａｙ＿ｉｎｔｅｒｎａｌ」の擬似コードリスティング２５０が示されている。一般に、内部プログラム「ＷＦＦ＿ｔｏ＿ａｒｒａｙ＿ｉｎｔｅｒｎａｌ」は、入力ＷＦＦ　１００が原子ＷＦＦ　１０６であるケースを最初に検討するが、図２Ａ～２Ｂに示されているように、真または偽の原子ＷＦＦ　１０８、ブール述語ＷＦＦ　１２４、等価述語ＷＦＦ　１２６、または複合ＷＦＦ　１３６を含むことができる。真１４８または偽１５０の原子ＷＦＦ　１０８について、この方法のステップ３２４は、それぞれ世界の真の配列２１４または世界の偽の配列２１６を返す。図１４の擬似コードリスティング２５０は、それぞれ項目１ａおよび１ｂに示されているように、真１４８または偽１５０の原子ＷＦＦを世界の真の配列２１４および世界の偽の配列２１６へ変換することを示す。

【００４５】

図２Ａ～２Ｂでは、入力ＷＦＦ　１００が、単一のブール述語１２４を含む原子ＷＦＦ　１０６である場合について、ステップ３２６は、図１４の擬似コードリスティング２５０の項目１ｃに示されているように、単一のブール述語１２４を等価述語１２６に変換する。その後、この方法は、ステップ３２２で述語を再帰的に処理して、入力ＷＦＦ　１００を初期ビット配列２００に変換する。入力ＷＦＦ　１００が原子等価述語１２６である場合には、入力ＷＦＦ　１００は、上で説明したステップ３１６、３１８、３２０で処理されて、ビット配列次元２１８ならびにそれに関連する述語１１４および領域要素１２０が判定される。ステップ３２８は、さらに、配列キャッシュ３０（図３）から、等価述語１２６に関連するビット配列次元２１８（図１１）、述語１１４、および領域要素１２０を有する初期ビット配列２００を取り出すことを含む。ステップ３２８は、下でより詳細に説明するように、配列キャッシュ３０内の初期ビット配列２００の取出の不在（たとえば、突き止めることの不可能性）の場合に、初期ビット配列２００を構成するステップ３３０を含むことができる。図１４の擬似コードリスティング２５０の項目２ａに示されているように、内部プログラム「ＷＦＦ＿ｔｏ＿ａｒｒａｙ＿ｉｎｔｅｒｎａｌ」は、所与の述語、領域要素、およびビット配列次元を有する初期ビット配列２００を配列キャッシュ３０内で検索し、配列キャッシュ３０（図３）内で初期ビット配列を突き止めることができない場合には、下で説明する図４に示された方法によって初期ビット配列２００を構成する。

【００４６】

図２Ａ～２Ｂでは、入力ＷＦＦ　１００が、０個以上の原子ＷＦＦ　１０６を含む１つの複合ＷＦＦ　１３６、選言演算子（ＯＲ）もしくは連言演算子（ＡＮＤ）のいずれかに関連する複数の複合ＷＦＦ　１３６、または否定演算子に関連する正確に１つの原子ＷＦＦ　１０６もしくは複合ＷＦＦ　１３６である場合に、この方法のステップ３２２は、まず、複合ＷＦＦ　１３６の各オペランドを再帰的に処理する。これに関して、ステップ３２２は、原子ＷＦＦ　１０６に出会うまで、演算子に関連する複合ＷＦＦを再帰的に処理する。この方法は、複合ＷＦＦ　１３６の演算子が否定演算子（ＮＯＴ）１３０（図２Ａ）、連言演算子（ＡＮＤ）１３２（図２Ｂ）、または選言演算子（ＯＲ）１３４（図２Ｂ）のどれであるのかに従って、再帰的に処理されたＷＦＦを組み合わせる。この方法は、再帰的に処理された原子ＷＦＦ　１０６の各１つについて１つの初期ビット配列２００を

返す。組み合わされた初期ビット配列２００を、そのビット配列次元２１８が個々の初期ビット配列２００のビット配列次元２１８（図１１）と等しくなるように処理することができる。

【００４７】

たとえば、図２Ａでは、ステップ３３４は、演算子タイプを判定することを含む。演算子が否定演算子１３０である場合には、この方法は、ＷＦＦがブール述語１２４であるかどうかを判定する。ＷＦＦがブール述語１２４である場合には、この方法は、それぞれのステップ３１６、３１８、および３２０で、初期ビット配列２００の述語１１４、領域要素１２０、およびビット配列次元２１８を判定するステップ３１６を実行することを含む。その後、この方法は、配列キャッシュ３０（図３）から入手可能な場合に適当に構成された初期ビット配列２００を取り出し、あるいは、配列キャッシュ３０から入手不能な場合に上で説明したように初期ビット配列２００を構成するステップ３３０を実行するステップ３２８を実行する。図２Ａでは、ＷＦＦがブール述語１２４ではない場合に、この方法のステップ３２２は、複合ＷＦＦ　１３６を再帰的に処理し、ステップ３３６では、１から０へおよび０から１へ各ビット要素の値を反転する（図示せず）ことによって結果の初期ビット配列２００を否定する。図１４の擬似コードリスティング２５０の項目２ｂは、否定される複合ＷＦＦ　１３６のケースの処理を示す。

【００４８】

図２Ｂを参照すると、否定されない複合ＷＦＦ　１３６のケースについて、演算子が連言演算子１３２（たとえば、ＡＮＤ）または選言演算子１３４（たとえば、ＯＲ）の場合に、この方法は、組み合わされた初期ビット配列２００内のオペランド１０２の量を判定するステップ３３８を含む。連言演算子１３２について、ステップ３４０は、ビット要素２２２が個々の初期ビット配列２００の連言（「ＡＮＤ」）と等しくなる形で個々の初期ビット配列２００のビット要素２２２を組み合わせることを含む。選言演算子１３４について、ステップ３４２は、ビット要素２２２が個々の初期ビット配列２００の選言（「ＯＲ」）と等しくなる形で個々の初期ビット配列２００のビット要素２２２を組み合わせることを含む。ステップ３４４は、図２Ａ～２Ｂで上記のケースのいずれもが有効でない場合にエラーを返すことを含む。図１４に、演算子が連言演算子１３２（たとえば、ＡＮＤ）または選言演算子１３４（たとえば、ＯＲ）である場合の否定されない複合ＷＦＦ　１３６の処理の擬似コードを、項目２ｃに示す。否定される複合ＷＦＦおよび否定されない複合ＷＦＦについて、内部プログラム「ＷＦＦ＿ｔｏ＿ａｒｒａｙ＿ｉｎｔｅｒｎａｌ」は、複合ＷＦＦ　１３６のトップレベル句に対してそれ自体を再帰的に呼び出し、適当な組合せ関数（「ＢｉｔＮｏｔ」、「ＢｉｔＯｒ」、または「ＢｉｔＡｎｄ」）を使用して結果（すなわち、初期ビット配列２００）を組み合わせる。

【００４９】

図３を参照すると、上で説明した、等価述語１２６（図２Ａ）に関連する所与のビット配列次元２１８、述語１１４、および領域要素１２０を有する初期ビット配列２００を配列キャッシュ３０から取り出すステップ３２８に含めることができる動作が示されている。ステップ３４６は、配列キャッシュ３０内で初期ビット配列２００を検索することを含むことができる。図１５に示されているのは、所与の述語１１４、領域要素、およびビット配列次元を有する初期ビット配列２００を配列キャッシュ３０内で検索し、図３の流れ図に示されているように配列キャッシュ３０から初期ビット配列２００を取り出すか、図４の流れ図に示されているように初期ビット配列２００を構成するかのいずれかを行うルーチン「ＧｅｔＬｅａｆＷｆｆＡｒｒａｙ」の擬似コードリスティング２５２である。

【００５０】

図４に、初期ビット配列２００を構成するステップ３３０に含めることができる動作を示す。ステップ３４８は、対応するビット配列次元２１８（図１１）を有する入力ＷＦＦ　１００（図１）のトップレベル述語領域を使用して初期ビット配列２００を作成することを含むことができる。ステップ３５０は、初期ビット配列２００のすべてのビット要素２２２に「０」をセットすることを含むことができる。ステップ３５２は、述語インデッ

クス（たとえば、ビット配列次元）および値インデックス（たとえば、述語領域内の値のオフセット）に沿ってすべてのビット要素２２２に「１」をセットすることを含むことができる。図４の方法は、適当に構成された初期ビット配列２００を返す。この方法は、所与のビット配列次元２１８を有し、図１５の擬似コードリスティング２５２内に示されているように「ｐｒｅｄ＿ｉｎｄｅｘ」および「ｖａｌｕｅ＿ｉｎｄｅｘ」によって指定される世界に対応するビット要素２２２値（たとえば、１および０）を有する初期ビット配列２００を構成する。一実施形態では、構成された初期ビット配列２００（図４）を、ステップ３３２（図２Ｂ）によって実行されるように、配列キャッシュ３０（図４）に追加することができる。有利なことに、この方法のメモリ要件を、新たに構成された初期ビット配列２００を配列キャッシュ３０に格納された以前に構成された初期ビット配列２００のコレクションに追加することによって、減らすことができる。

【００５１】

図４のステップ３５２を、初期ビット配列２００の構成中にそのビット要素２２２に「１」をセットするルーチンを示す図１７に示された擬似コードリスティング２５４によって表すことができる。上で説明したように、「ｂｉｔ＿ａｒｒａｙ」のビット要素２２２の「１」の値は、「ｓｅｔ＿ｄｉｍｅｎｓｉｏｎ」に関係する述語およびインデックスに関係するその述語の領域要素に関連する世界のすべてに対応する。たとえば、図１６を参照すると、領域、Ｄｏｍａｉｎ（ＨａｓＬｉｆｔｉｎｇＢｏｄｙ）＝｛”ＴＲＵＥ”，”ＦＡＬＳＥ”）、Ｄｏｍａｉｎ（ＥｎｇｉｎｅＴｙｐｅ）＝｛Ｒｏｃｋｅｔ，Ｒａｍｊｅｔ，Ｔｕｒｂｏｊｅｔ｝を有する述語＝｛ＨａｓＬｉｆｔｉｎｇＢｏｄｙ，ＥｎｇｉｎｅＴｙｐｅ｝について、原子ＷＦＦ、ＥｎｇｉｎｅＴｙｐｅ＝Ｔｕｒｂｏｊｅｔに対応するビット配列の要素をセットすることが望ましい場合がある。代表的な初期ビット配列２００のビット要素２２２の値に「１」をセットするプロセスは、引数すなわち、２と等しい「ｓｅｔ＿ｄｉｍｅｎｓｉｏｎ」（すなわち、述語リスト内の第２の述語に対応する）、３と等しいｉｎｄｅｘ（すなわち、述語の領域内の第３の領域要素に対応する）、｛２，３｝と等しいｄｉｍｅｎｓｉｏｎｓ、１と等しい「ｃｕｒｒｅｎｔ＿ｄｉｍｅｎｓｉｏｎ」、および空リストと等しいｉｎｄｉｃｅｓを用いて、ビット要素２２２の初期の「０」の値を有する「ｂｉｔ＿ａｒｒａｙ」に対して関数「ＳｅｔＡｒｒａｙＤｉｍｅｎｓｉｏｎＩｎｄｅｘ」を呼び出すことを含む。図１７では、初期ビット配列２００に「１」の値を割り当てる上で説明したプロセスが、「ｃｕｒｒｅｎｔ＿ｄｉｍｅｎｓｉｏｎ」の値を用いて開始すること、および入力ビット配列の適当な値をセットするためのインデックスと一緒に「ｃｕｒｒｅｎｔ＿ｄｉｍｅｎｓｉｏｎ」の値を再帰的に増分することによって示されている。この形で、初期ビット配列２００内の値は、変数ｉｎｄｉｃｅｓによって指定される世界に対応し、この変数ｉｎｄｉｃｅｓは、各再帰呼出で、呼出し側プログラム内で入力ＷＦＦによって表される世界を表すｉｎｄｉｃｅｓの増加するリストに追加される。

【００５２】

図５を参照すると、初期ビット配列２００を単純化するステップ３１０に含めることができる動作が示されている。初期ビット配列２００は、述語１１４に関連する複数のビット配列次元２１８を含むことができる。上で示したように、ステップ３１０は、モデリングされている入力ＷＦＦ　１００（図１）を表すのに必要ではない述語１１４（図１１～１２）を除去し、単純化されたビット配列２１２を返す。図５に示されているように、ステップ３５４は、全般的に、セマンティックス的に冗長なビット配列次元２１８を除去することによって初期ビット配列２００を縮小することを含む。この方法は、ビット配列次元２１８が縮小可能であるかどうかを判定するために、ビット配列次元２１８の各１つについて部分配列２２０のビット要素２２２（図２Ｂ）を比較するステップ３５６を含むことができる。部分配列２２０のビット要素２２２が等しい場合には、部分配列２２０に関連する次元を除去することができる。ステップ３５８は、入力ＷＦＦ　１００を表すのに必要ではない述語１１４が単純化されたビット配列２１２から除去されるように、等しいビット要素２２２を有する部分配列２２０を有する各ビット配列次元２１８を初期ビット

配列２００から除去することを含むことができる。ステップ３６０は、初期ビット配列２００から除去されないビット配列次元２１８を含む保存された次元インデックス２８（図３）を生成することを含むことができる。ステップ３６２は、単純化されたビット配列２１２および保存された次元インデックス２８を返すことを含むことができる。

【００５３】

ここでビット配列単純化を示すために図１６を参照すると、次の複合ＷＦＦに対応する２×３ビット配列が示されている。

Ｏｒ（Ａｎｄ（ＥｎｇｉｎｅＴｙｐｅ＝Ｔｕｒｂｏｊｅｔ，ＨａｓＬｉｆｔｉｎｇＢｏｄｙ），Ａｎｄ（ＥｎｇｉｎｅＴｙｐｅ＝Ｔｕｒｂｏｊｅｔ，Ｎｏｔ（ＨａｓＬｉｆｔｉｎｇＢｏｄｙ））

【００５４】

図１６からわかるように、ブール述語（ＨａｓＬｉｆｔｉｎｇＢｏｄｙ）の部分配列２２０は、（Ｎｏｔ（ＨａｓＬｉｆｔｉｎｇＢｏｄｙ））の部分配列２２０と同一のビット要素２２２値を有する。同等のビット要素２２２に起因して、ブール述語の次元を除去することができ、図１６に示されたビット配列は、次のＷＦＦと論理的に同等になる。

ＥｎｇｉｎｅＴｙｐｅ＝Ｔｕｒｂｏｊｅｔ

これは、図１８に示された１次元の単純化されたビット配列２１２に対応する。

【００５５】

図１９には、初期ビット配列２００（図５）から次元を除去し、単純化されたビット配列２１２（図５）を返すルーチン「ＳｉｍｐｌｉｆｙＡｒｒａｙ」の擬似コードリスティング２５６が示されている。「ＳｉｍｐｌｉｆｙＡｒｒａｙ」ルーチンは、所与の次元内の各インデックスの部分配列２２０（図５）がお互いと等しい次元を検索して初期ビット配列２００をトラバースする。「ＳｉｍｐｌｉｆｙＡｒｒａｙ」ルーチンは、そのような次元を除去し、除去されない次元を記録し、呼出し側プログラムが、初期ビット配列２００の述語リスト１１６を使用して、単純化されたビット配列２１２の述語リスト１１６（図１１）を構成できるようにする。図１９に示されたルーチンは、１つまたは複数の次元を除去することによって初期ビット配列２００を縮小するために実施することのできる多数の再帰ルーチンのうちの１つである。図１９では、「ＳｉｍｐｌｉｆｙＡｒｒａｙ」の角括弧内で提示される引数は、オプションであり、このルーチンへの再帰呼出中に使用され得る。「ＳｉｍｐｌｉｆｙＡｒｒａｙ」ルーチンは、ある軸に沿った初期ビット配列の縮小可能性をチェックする関数「ｄｉｍｅｎｓｉｏｎ－ｓｌｉｃｅ」を含むことができる。初期ビット配列が縮小可能である場合には、このルーチンは、複数の値として縮小された配列およびＴｒｕｅを返すことができる。初期ビット配列が縮小可能ではない場合には、このルーチンは、複数の値として入力ビット配列およびＦａｌｓｅを返すことができる。上で示したように、初期ビット配列の縮小は、初期ビット配列によって表される入力ＷＦＦ　１００の単純化を容易にする。

【００５６】

初期ビット配列２００の縮小可能性に関するテストを、複数の形のうちの１つで実行することができ、ＭＡＴＡＬＢプログラミング言語に組み込まれた配列に関する次の表記を使用して記述することができる。ＭＡＴＡＬＢ構文では、特定の軸次元の「：」項目は、その軸のすべての値を指す。たとえば、Ａが２×３行列である場合には、Ａ［２，：］は、行列の第２行を指し、Ａ［２，：］＝（Ａ［２，１］，Ａ［２，２］，Ａ［２，３］）である。配列の第ｋ軸が縮小可能であるかどうかをテストするために、本開示のルーチン「ＣｏｌｌａｐｓｅＩｆＰｏｓｓｉｂｌｅ」は、ｊ＝２，…，Ｊすなわち第ｋ軸の長さについて、

Ａ［：，…，：，ｊ，：，…，：］　＝　Ａ［：，…，：，１，：，…，：］

をテストし、ここで、添字ｊは、上の位置ｋにある。上が真である場合には、このルーチンは、縮小された配列Ｂ［：，…，：］＝Ａ［：，…，：，１，：，…，：］を返し、ここで、配列Ｂは、第ｋ軸を除去することによって、Ａより１つ少ない次元を有する。「ＣｏｌｌａｐｓｅＩｆＰｏｓｓｉｂｌｅ」ルーチンの実際の実施態様は、必要な場合に限って作成される仮説のＢ配列への行優先添字付けアクセスを使用し、さらに、そのような添字からＡ内の実際の「１」への写像を使用する。行優先添字は、多次元配列を単一次元であるかのように扱うことを可能にする。たとえば、次元ｄ［１］×ｄ［２］×…×ｄ［ｎ］を有するｎランク配列Ａを考慮すると、ｋがＡへの行優先添字である、すなわち、Ａ［ｋ］＝Ａ［ｉ［１］，…，ｉ［ｎ］］である場合には、

ｋ　＝　（ｉ［１］－１）＊ｆ［１］＋…＋（ｉ［ｎ］－１）＊ｆ［ｎ］，
ｆ［１］　＝　ｄ［２］＊…＊ｄ［ｎ］
ｆ［２］　＝　ｄ［３］＊…＊ｄ［ｎ］
ｆ［ｎ－１］　＝　ｄ［ｎ］
ｆ［ｎ］　＝　１

である。

【００５７】

図６を参照すると、単純化されたビット配列２１２を選言標準形（ＤＮＦ）１４０または連言標準形（ＣＮＦ）１４２の戻りＷＦＦ　１３８に変換するステップ３１２に含めることができる１つまたは複数の動作が示されている。この方法は、述語１１４（図１１）の集合およびそのそれぞれの領域要素１２０（図１１）を与えられて単純化されたビット配列２１２を体系的に処理すること、および戻りＷＦＦ　１３８を構成することを含むことができる。この方法は、さらに、単純化されたビット配列２１２内のビット要素２２２の総量を判定することを含むことができるステップ３６４を含むことができる。ステップ３６６は、１の値を有する単純化されたビット配列２１２内のビット要素２２２の量を判定することを含むことができる。ステップ３６８は、ビット要素２２２の総量の半分未満が１の値を有する場合に、単純化されたビット配列２１２を選言標準形（ＤＮＦ）１４０の戻りＷＦＦ　１３８に変換することを含むことができる。ステップ３７２は、ＤＮＦ　ＷＦＦ　１４０を返すことを含むことができる。ステップ３７０は、ビット要素２２２の総量の少なくとも半分が１の値を有する場合に、単純化されたビット配列２１２を連言標準形（ＣＮＦ）１４２の戻りＷＦＦ　１３８に変換することを含むことができる。ステップ３７４は、ＣＮＦ　ＷＦＦ　１４２を返すことを含むことができる。

【００５８】

図２０に、述語の集合「ｐｒｅｄｉｃａｔｅｓ＿ｉｎ」および述語の領域要素「ｄｏｍａｉｎｓ＿ｉｎ」を与えられて、単純化されたビット配列２１２（図６）を戻りＷＦＦ　１３８（図６）に変換するルーチン「ＡｒｒａｙＴｏＷｆｆ」の擬似コードリスティング２５８を示す。図２０の「ＡｒｒａｙＴｏＷｆｆ」ルーチンは、当初に、選言標準形（ＤＮＦ）１４０（図５）のまたはＤＮＦの否定としての（「ｎｏｔ　ＤＮＦ」）戻りＷＦＦ　１３８を判定する。前述のステップ３６６で示したように、「ＡｒｒａｙＴｏＷｆｆ」ルーチンは、入力配列内の「１」の値を有するビット要素２２２（図６）の量をカウントすることによって、戻りＷＦＦ　１３８を生成すべき形を判定することができる。前述のステップ３６８で示したように、ＤＮＦまたはｎｏｔ　ＤＮＦの選択は、「１」ビット要素２２２の量が単純化されたビット配列２１２内のビット要素２２２の総量の半分未満であるかどうかに依存する。

【００５９】

図７は、単純化されたビット配列２１２を選言標準形（ＤＮＦ）ＷＦＦ　１４０に変換するステップ３６８の流れ図の実施形態である。ステップ３６８は、単純化されたビット配列２１２に関連する述語１１４の量を判定するステップ３７６を含むことができる。述

語１１４の量が１より多い場合には、この方法は、１の値を有するビット要素２２２ごとに原子ＷＦＦ　１０８（図３）を作成するステップ３７８を含むことができる。原子ＷＦＦ　１０８（図２Ａ）の作成は、図９に示され、下でより詳細に説明されるステップ３７８で実行される。図７では、ステップ３８２は、１の値を有するビット要素２２２ごとに作成された原子ＷＦＦ　１０８の各１つを含む連言ＷＦＦを作成すること（たとえば、各原子ＷＦＦを一緒に「ＡＮＤ」すること）を含むことができる。ステップ３８４は、原子ＷＦＦ　１０８の連言ＷＦＦを含む選言ＷＦＦすなわちＤＮＦ　ＷＦＦ　１４０を作成すること（たとえば、各連言ＷＦＦを一緒に「ＯＲ」すること）を含むことができる。

【００６０】

図７では、単純化されたビット配列２１２に関連する述語１１４の量が正確に１である場合に、単純化されたビット配列２１２をＤＮＦ　ＷＦＦ　１４０に変換する方法は、１の値を有するビット要素２２２ごとに原子ＷＦＦ　１０８を作成することを含むステップ３７８と、１の値を有するビット要素２２２ごとに作成された原子ＷＦＦ　１０８の各１つを含む選言ＷＦＦすなわちＤＮＦ　ＷＦＦ　１４０を作成すること（たとえば、各原子ＷＦＦを一緒に「ＯＲ」すること）を含むステップ３８４とを含むことができる。図７では、述語１１４がない場合に、ステップ３８６は、ビット要素２２２の値（たとえば、「１」または「０」）を判定すること、およびステップ３８８で、ビット要素２２２が１の値を有する場合に真１４８の世界ＷＦＦを返すことを含む。ステップ３９０は、ビット要素２２２が０の値を有する場合に偽１５０の世界ＷＦＦを返すことを含む。

【００６１】

図８は、単純化されたビット配列２１２を連言標準形（ＣＮＦ）ＷＦＦ　１４２に変換する流れ図の実施形態である。この方法のステップ３７６は、上で説明したように、単純化されたビット配列２１２に関連する述語１１４の量を判定することを含むことができる。述語１１４の量が１より多い場合には、この方法は、０の値を有するビット要素２２２ごとに否定原子ＷＦＦ　１１０を作成するステップ３８０を含むことができる。否定原子ＷＦＦ　１１０の作成は、図１０の流れ図に示され、下でより詳細に説明される。図８では、ステップ３８４は、否定原子ＷＦＦ　１１０の選言ＷＦＦを作成すること（たとえば、各否定原子ＷＦＦを一緒に「ＯＲ」すること）を含むことができる。ステップ３８２は、否定原子ＷＦＦ　１１０の選言ＷＦＦを含む連言ＷＦＦすなわちＣＮＦ　ＷＦＦ　１４２を作成すること（たとえば、各選言ＷＦＦを一緒に「ＡＮＤ」すること）を含むことができる。

【００６２】

図８では、単純化されたビット配列２１２に関連する述語１１４の量が正確に１である場合に、単純化されたビット配列２１２をＣＮＦ　ＷＦＦ　１４２に変換する方法は、０の値を有するビット要素２２２ごとに否定原子ＷＦＦ　１１０を作成することを含むステップ３８０と、０の値を有するビット要素２２２ごとに作成された否定原子ＷＦＦ　１１０の各１つを含む連言ＷＦＦすなわちＣＮＦ　ＷＦＦ　１４２を作成すること（たとえば、各否定原子ＷＦＦを一緒に「ＡＮＤ」すること）を含むステップ３８２とを含むことができる。図８では、述語１１４がない場合に、ステップ３８６は、ビット要素２２２の値（たとえば、「１」または「０」）を判定することを含む。ステップ３８８は、ビット要素２２２が１の値を有する場合に真１４８の世界ＷＦＦを返すことを含む。ステップ３９０は、ビット要素２２２が０の値を有する場合に偽１５０の世界ＷＦＦを返すことを含む。

【００６３】

図９は、原子ＷＦＦ　１０８（図７）を作成するステップ３７８の流れ図の実施形態である。この方法は、述語がブール述語１２４または等価述語１２６のどちらであるのかを判定するステップ３９２を含むことができる。述語が等価述語１２６である場合には、ステップ３９４は、「（＝述語　値）」として原子ＷＦＦを返す。述語がブール述語１２４である場合には、ステップ３９６は、述語　値＝Ｔｒｕｅであるかどうかを判定することを含むことができる。述語　値＝Ｔｒｕｅである場合には、ステップ３９８は、「（Ｎｏ

ｔ　述語）」として原子ＷＦＦを返すことを含む。述語　値がＴｒｕｅではない場合には、ステップ４００は、述語１１４（図８）として原子ＷＦＦ　１０８を返すことを含む。

【００６４】

図１０は、否定原子ＷＦＦ　１１０を作成するステップ３８０の流れ図の実施形態である。ステップ３９２は、述語がブール述語１２４または等価述語１２６のどちらであるのかを判定することを含むことができる。述語が等価述語１２６である場合には、ステップ４０２は、述語１１４の領域長が２であるかどうかを判定することを含む。述語１１４（図８）の領域長が２ではない場合には、ステップ４０４は、「Ｎｏｔ（＝述語　値）」として否定原子ＷＦＦ　１１０を返す。述語１１４の領域長が２である場合には、ステップ４０６は、（＝述語　［他の値］）として否定原子ＷＦＦ　１１０を返す。述語１１４がブール述語である場合には、ステップ３９６は、述語　値＝Ｔｒｕｅであるかどうかを判定することを含む。述語　値＝Ｔｒｕｅである場合には、ステップ３９８は、（Ｎｏｔ　述語）として否定原子ＷＦＦ　１１０を返すことを含む。述語１１４値がＴｒｕｅではない場合には、ステップ４００は、述語１１４（図８）として否定原子ＷＦＦ　１１０を返すことを含む。

【００６５】

図２１は、適用可能な述語および領域を与えられ、Ｔｒｕｅ状態を表すのにおよび原子ＷＦＦ　１０８（図７）を生成するのに単純化されたビット配列２１２（図７）内の「１」ビット要素２２２（図７）を使用する、単純化されたビット配列（「ｂｉｔ＿ａｒｒａｙ＿ｉｎ」）を選言標準形（ＤＮＦ）１４０（図７）の戻りＷＦＦ　１３８（図７）に変換するルーチン「ＡｒｒａｙＴｏＤＮＦ」の擬似コードリスティング２６０である。「ＡｒｒａｙＴｏＤＮＦ」ルーチンは、当初に、単純化されたビット配列２１２が多次元配列であるかどうかを判定するために、述語１１４（図７）の総量を判定することができる。そうである場合には、単純化されたビット配列２１２を、それぞれが原子ＷＦＦ　１０８の連言であるより小さいＷＦＦのコレクションの選言としてのＤＮＦ　ＷＦＦ　１４０に変換する。原子ＷＦＦ　１０８の生成は、単純化されたビット配列２１２内の０のビット要素２２２値に関する領域値に基づく。原子ＷＦＦ　１０８の各１つは、各述語１１４の値および単純化されたビット配列２１２内のそのビット要素２２２位置での値を表す。

【００６６】

短く図１１を参照すると、２次元ビット配列内のインデックス［２，２］に「１」のビット要素２２２を与えられて、ビット要素２２２は、ＥｎｇｉｎｅＴｙｐｅの領域値でのインデックス２に対応する値すなわち「Ｒａｍｊｅｔ」と、ＨａｓＬｉｆｔｉｎｇＢｏｄｙの領域値でのインデックス２に対応する値すなわち「Ｆａｌｓｅ」とを表す。そのような値に対応する原子ＷＦＦ　１０８は、

Ｏｒ（Ｎｏｔ（＝　ＥｎｇｉｎｅＴｙｐｅ　Ｒｏｃｋｅｔ），Ｎｏｔ（ＨａｓＬｉｆｔｉｎｇＢｏｄｙ））

である。

【００６７】

図１１のビット配列の完全なＷＦＦは、

Ｏｒ（Ａｎｄ（（＝　ＥｎｇｉｎｅＴｙｐｅ　Ｒａｍｊｅｔ），Ｎｏｔ（ＨａｓＬｉｆｔｉｎｇＢｏｄｙ）），Ａｎｄ（（＝　ＥｎｇｉｎｅＴｙｐｅ　Ｔｕｒｂｏｊｅｔ），Ｎｏｔ（ＨａｓＬｉｆｔｉｎｇＢｏｄｙ））

である。

【００６８】

単一の述語を与えられて、原子ＷＦＦ　１０８が、「１」値のビット要素ごとに１つの原子ＷＦＦ　１０８の選言として作成される。各原子ＷＦＦ　１０８は、上で図９の記述

で説明した形で構成される。述語１１４が与えられない時には、結果の０次元ビット配列のコンピュータ言語固有表現を調べて、真の世界または偽の世界のいずれかを返すことができる。ＬＩＳＰプログラミング言語では、ビット配列が＃０Ａ１と等しい時に、真の世界が返される。ＬＩＳＰプログラミング言語では、ビット配列が＃０Ａ０と等しい時に、偽の世界が返される。

【００６９】

図２２は、適用可能な述語および領域を与えられ、真の状態を表すのにおよび否定原子ＷＦＦ　１１０（図８）を生成するのに単純化されたビット配列２１２（図８）内の「０」ビット要素２２２（図８）を使用する、単純化されたビット配列（「ｂｉｔ＿ａｒｒａｙ＿ｉｎ」）を連言標準形（ＣＮＦ）１４２（図８）の戻りＷＦＦ　１３８（図８）に変換するルーチン「ＡｒｒａｙＴｏＮｏｔＤＮＦ」の擬似コードリスティング２６２である。上で示したように、「ＡｒｒａｙＴｏＮｏｔＤＮＦ」ルーチンは、当初に、単純化されたビット配列２１２が多次元配列であるかどうかを判定するために、述語１１４（図８）の総量を判定することができる。そうである場合には、単純化されたビット配列２１２は、それぞれが否定原子ＷＦＦ　１１０の選言であるより小さいＷＦＦのコレクションの連言としてＣＮＦ　ＷＦＦ　１４２（図８）に変換される。否定原子ＷＦＦ　１１０の生成は、単純化されたビット配列２１２内の０のビット要素２２２値に関する領域値に基づく。

【００７０】

図１２を参照すると、インデックス［１，１］で「０」のビット要素２２２を与えられると、これは、ＥｎｇｉｎｅＴｙｐｅの領域値でのインデックス１に対応する値すなわち「Ｒｏｃｋｅｔ」と、ＨａｓＬｉｆｔｉｎｇＢｏｄｙの領域値でのインデックス１に対応する値すなわち「Ｔｒｕｅ」とを表す。そのような値に対応する原子ＷＦＦ　１０８は、

Ｏｒ（Ｎｏｔ（＝　ＥｎｇｉｎｅＴｙｐｅ　Ｒｏｃｋｅｔ），Ｎｏｔ（ＨａｓＬｉｆｔｉｎｇＢｏｄｙ））

である。

【００７１】

図１２のビット配列の完全なＷＦＦは、

Ａｎｄ（Ｏｒ（Ｎｏｔ（＝　ＥｎｇｉｎｅＴｙｐｅ　Ｒｏｃｋｅｔ），Ｎｏｔ（ＨａｓＬｉｆｔｉｎｇＢｏｄｙ）），　Ｏｒ（Ｎｏｔ（＝　ＥｎｇｉｎｅＴｙｐｅ　Ｒｏｃｋｅｔ），ＨａｓＬｉｆｔｉｎｇＢｏｄｙ））

である。

【００７２】

単一の述語について、ＷＦＦは、「０」ビット要素ごとに１つの否定原子ＷＦＦ　１１０を含む否定原子ＷＦＦ　１１０の連言として構築される。述語１１４が与えられない場合には、結果の０次元ビット配列のコンピュータ言語固有表現を調べて、上で図１１に関して述べたものと同一の真世界または偽世界のいずれかを返すことができる。

【００７３】

図２３を参照すると、プロセッサベースのシステム１０または他の適切なコンピュータシステム上などのコンピュータ実施されるプロセス内で、開示される方法の上で説明したステップまたはステップの任意の組合せを全体的にまたは部分的に実施するシステムのブロック図が示されている。プロセッサベースのシステム１０（図２３）は、迅速な設計制約式計算または命題論理式の操作を必要とする任意のアプリケーションをサポートするためにＷＦＦの単純化中に上で説明したステップのうちの１つまたは複数を実施することができる。プロセッサベースのシステム１０は、上で説明した動作またはステップのうちの１つまたは複数を実施するためにプロセッサベースのシステム１０に供給しまたはこれに

ロードすることのできるコンピュータ可読プログラム命令７２を実行することができる。非限定的な例で、プロセッサベースのシステム１０および／またはコンピュータ可読プログラム命令７２は、制約管理システム１２（図２３）内での使用のためなどの完全論理式（ＷＦＦ）の単純化を容易にすることができる。

【００７４】

図２３のブロック図は、入力ＷＦＦ　１００（図１）を入力ＷＦＦ　１００の初期ビット配列２００（図１）表現に変換し、入力ＷＦＦ　１００を表現するのに必要ではない述語１１４（図１）を除去するために初期ビット配列２００を単純化し、単純化されたビット配列２１２を最小の標準形（すなわち、ＤＮＦまたはＣＮＦ）の戻りＷＦＦ　１３８（図１）に変換することのできる、有利な実施形態でのプロセッサベースのシステム１０を示す。図２３に示された実施形態では、プロセッサベースのシステム１０は、１つまたは複数のコンポーネントの間のデータの転送を容易にするためにそのようなコンポーネントに通信的に結合されたデータ通信パス５０（たとえば、データリンク）を含むことができる。通信パス５０は、１つもしく複数のデータバスまたはプロセッサベースのシステム１０のコンポーネントおよびデバイスの間のデータの転送を容易にする任意の他の適切な通信パスを含むことができる。

【００７５】

非限定的な実施形態では、コンポーネントは、プロセッサ５２、メモリデバイス５４、不揮発性ストレージデバイス５６、通信デバイス６０、入出力デバイス５８、およびディスプレイデバイス６２（図２３）のうちの１つまたは複数を含むことができる。このシステムは、さらに、古典的プロセッサ１４および式プロセッサ１８を含むことができる。式プロセッサ１８は、式コンバータ２０、ビット配列シンプリファイヤ２２、ビット配列コンバータ２４、ビット配列コンストラクタ２６、配列キャッシュ３０、および保存された次元インデックス２８を含むことができる。上で示したように、古典的プロセッサ１４を、所定のタイムアウト期間３０６（図１）以内に１つまたは複数の古典的アルゴリズム１６（図１）を使用して入力ＷＦＦ　１００の単純化を試みるように構成することができる。古典的プロセッサ１４を、結果のＷＦＦ　１０４またはその不在を式プロセッサ１８に通信するように構成することができる。

【００７６】

式プロセッサ１８を、入力ＷＦＦ　１００（図１）を初期ビット配列２００（図１）に変換するように構成することができる。式コンバータ２０を、入力ＷＦＦ　１００内の述語１１４（図２Ａ）を判定し、述語１１４の各１つに関連する領域要素１２０（図２Ａ）を判定し、述語１１４および領域要素１２０に対応する初期ビット配列２００のビット配列次元２１８（図２Ａ）を判定するように構成することができる。これに関して、原子ＷＦＦ　１０８を、世界の真の配列２１４（図２Ａ）および世界の偽の配列２１６（図２Ａ）のうちの１つを含む初期ビット配列２００に変換するように式コンバータ２０を構成することができる。式コンバータ２０を、さらに、ブール述語１２４に関連する初期ビット配列２００のビット配列次元２１８と述語１１４と領域要素１２０とを判定し、配列キャッシュ３０（図３）から、ブール述語１２４を表す初期ビット配列２００を取り出すように構成することができる。ビット配列コンストラクタ２６は、配列キャッシュ３０の検索が適当な構成を有する初期ビット配列２００を発見できない場合に、初期ビット配列２００を構成することができる。式コンバータ２０を、さらに、原子ＷＦＦ　１０６（図２Ａ）の各１つの初期ビット配列２００を返し、否定演算子１３０（図２Ａ）、連言演算子１３２（図２Ｂ）、および選言演算子１３４（図２Ｂ）のうちの少なくとも１つに基づいて初期ビット配列２００を組み合わせるように構成することができる。

【００７７】

ビット配列シンプリファイヤ２２は、モデリングされつつある入力ＷＦＦ　１００（図１）を表すのに必要ではない述語１１４（図５）を除去することによって、初期ビット配列２００を単純化することができる。これに関して、ビット配列シンプリファイヤ２２を、ビット配列次元２１８（図５）のうちの少なくとも１つを除去することによって初期ビ

ット配列２００を縮小するように構成することができる。ビット配列シンプリファイヤ２２を、さらに、ビット配列次元２１８の各１つの部分配列２２０のビット要素２２２を比較し、等しいビット要素２２２（図５）を有する部分配列２２０（図５）を有する各ビット配列次元２１８を初期ビット配列２００から除去するように構成することができる。さらに、ビット配列シンプリファイヤ２２は、初期ビット配列２００から除去されないビット配列次元２１８を含む保存された次元インデックス２８（図５）を生成することができる。

【００７８】

ビット配列コンバータ２４は、単純化されたビット配列２１２（図６）を、選言標準形（ＤＮＦ）１４０（図６）または連言標準形（ＣＮＦ）１４２（図６）の戻りＷＦＦ　１３８（図６）に変換することができる。一実施形態では、ビット配列コンバータ２４は、単純化されたビット配列２１２内のビット要素２２２（図６）の総量に対する、１の値を有する単純化されたビット配列２１２内のビット要素２２２の量を判定することができる。ビット配列コンバータ２４は、ビット要素２２２の総量の半分未満が１の値を有する場合に、単純化されたビット配列２１２をＤＮＦ形１４０（図６）の戻りＷＦＦ　１３８（図６）に変換することができる。その代わりに、ビット配列コンバータ２４は、ビット要素２２２の総量の少なくとも半分が１の値を有する場合に、単純化されたビット配列２１２をＣＮＦ形１４２（図６）の戻りＷＦＦ　１３８（図６）に変換することができる。ビット配列コンバータ２４を、図６～１０および／または図２１～２２に示され、上で説明された動作のうちの１つまたは複数を実行するように構成することができる。

【００７９】

引き続き図２３を参照すると、入力ＷＦＦ　１００（図１）の初期ビット配列２００（図１）表現に入力ＷＦＦ　１００を変換し、述語１１４（図１）を除去するために初期ビット配列２００を単純化し、単純化されたビット配列２１２を戻りＷＦＦ　１３８（図１）に変換する、上で説明されたステップのうちの任意の１つの結果を、入出力デバイス５８に送ることができる。入出力デバイス５８を、ディスプレイデバイス６２に通信的に結合することができ、このディスプレイデバイス６２を、ＷＦＦ変換および単純化プロセスの結果を表示するように構成することができる。ディスプレイデバイス６２を、ＷＦＦ変換および単純化プロセスの実施の進行状況および／または結果を表示するように構成することができる。さらに、ディスプレイデバイス６２を、データ依存制約管理システム１２（図２３）内で実施されるトレード研究の結果を表示するように構成することができる。

【００８０】

一実施形態では、プロセッサベースのシステム１０は、メモリデバイス５４にインストールできるコンピュータ可読プログラム命令７２の命令を実行する１つまたは複数のプロセッサ５２を含むことができる。代替案では、プロセッサ５２は、複数の一体化されたプロセッサコアを有するマルチプロセッサコアを含むことができる。さらに、プロセッサ５２は、チップ上に一体化されたメインプロセッサおよび１つまたは複数の副プロセッサを含むことができる。プロセッサ５２は、複数の類似して構成されたプロセッサを有するメニープロセッサシステムを含むこともできる。

【００８１】

引き続き図２３を参照すると、プロセッサベースのシステム１０は、さらに、揮発性または不揮発性のストレージデバイス５６のうちの１つまたは複数を含むことができる１つまたは複数のメモリデバイス５４を含むことができる。しかし、メモリデバイス５４は、図４に示されたプロセス中に構成される初期ビット配列２００などのデータを格納する配列キャッシュ３０などの任意のハードウェアデバイスを含むことができる。たとえば、メモリデバイス５４は、通信パス５０内に含めることができるインターフェースおよび／または一体化されたメモリコントローラハブのランダムアクセスメモリまたはキャッシュを含むことができる。メモリデバイス５４を、さまざまな異なるタイプのデータ、コンピュータ可読コードもしくはコンピュータ可読プログラム命令７２、または任意の他のタイプの情報のうちの任意の１つを永久的におよび／または一時的に格納するように構成するこ

とができる。不揮発性ストレージデバイス５６を、フラッシュメモリデバイス、ハードドライブ、光ディスク、ハードディスク、磁気テープ、または長期ストレージに関する任意の他の適切な実施形態を含むがこれに限定されないさまざまな構成で提供することができる。さらに、不揮発性ストレージデバイス５６は、リムーバブルハードドライブなどのリムーバブルデバイスを含むことができる。

【００８２】

プロセッサベースのシステム１０は、さらに、プロセッサベースのシステム１０に接続できるコンポーネントの間のデータの転送を容易にするために、１つまたは複数の入出力デバイス５８を含むことができる。入出力デバイス５８を、プロセッサベースのシステム１０に直接におよび／または間接に結合することができる。入出力デバイス５８は、キーボード、マウス、ジョイスティック、タッチスクリーン、およびプロセッサベースのシステム１０へのデータの入力のための任意の他の適切なデバイスなどの周辺デバイスによってユーザ入力を容易にすることができる。入出力デバイス５８は、さらに、プロセッサベースのシステム１０の出力を表すデータを転送する出力デバイスを含むことができる。たとえば、入出力デバイス５８は、プロセッサベースのシステム１０によって処理されたデータの結果を表示するコンピュータモニタまたはコンピュータスクリーンなどのディスプレイデバイス６２を含むことができる。入出力デバイス５８は、オプションで、プロセッサベースのシステム１０によって処理された情報のハードコピーを印刷するプリンタまたはファックス機を含むことができる。

【００８３】

引き続き図２３を参照すると、プロセッサベースのシステム１０は、コンピュータネットワーク内のおよび／または他のプロセッサベースのシステムとのプロセッサベースのシステム１０の通信を容易にする１つまたは複数の通信デバイス６０を含むことができる。コンピュータネットワークまたは他のプロセッサベースのシステムとのプロセッサベースのシステム１０の通信を、無線手段によるものおよび／またはハードワイヤ接続によるものとすることができる。たとえば、通信デバイス６０は、プロセッサベースのシステム１０とコンピュータネットワークとの間の無線通信またはケーブル通信を可能にするネットワークインターフェースコントローラを含むことができる。通信デバイス６０は、モデムおよび／もしくはネットワークアダプタまたはデータを送信し受信するさまざまな代替デバイスのうちの任意の１つを含むこともできる。

【００８４】

入力ＷＦＦ　１００（図１）の最小の標準形の戻りＷＦＦ　１３８（図１）への変換および単純化について上で説明した方法論の動作のうちの１つまたは複数を、コンピュータ可読プログラム命令７２を使用して、プロセッサ５２によって、ならびに／または式プロセッサ１８、式コンバータ２０、ビット配列シンプリファイヤ２２、ビット配列コンバータ２４、およびビット配列コンストラクタ２６のうちの１つもしくは複数によって、実行することができる。コンピュータ可読プログラム命令７２は、コンピュータ使用可能プログラムコードおよびコンピュータ可読プログラムコードを含むことができるプログラムコードを含むことができる。コンピュータ可読プログラム命令７２を、プロセッサ５２によって読み取り、実行することができる。コンピュータ可読プログラム命令７２は、プロセッサ５２が、入力ＷＦＦ　１００を戻りＷＦＦ　１３８に単純化することに関連する上で説明した実施形態の１つまたは複数の動作を実行することを可能にすることができる。

【００８５】

引き続き図２３を参照すると、コンピュータ可読プログラム命令７２は、プロセッサベースのシステム１０の動作させる命令を含むことができ、さらに、アプリケーションおよびプログラムを含むことができる。コンピュータ可読プログラム命令７２を、式プロセッサ１８、式コンバータ２０、ビット配列シンプリファイヤ２２、ビット配列コンバータ２４、および／またはビット配列コンストラクタ２６による実行のためにメモリデバイス５４および／または不揮発性ストレージデバイス５６のうちの１つまたは複数に含め、かつ／またはロードすることができる。上で示したように、メモリデバイス５４および／また

は不揮発性ストレージデバイス５６のうちの１つまたは複数を、通信パス５０を介して図２３に示された残りのコンポーネントのうちの１つまたは複数に通信的に結合することができる。

【００８６】

コンピュータ可読プログラム命令７２を、有形のまたは非有形の、過渡的なまたは非過渡的なコンピュータ可読媒体６６に含めることができ、プロセッサ５２による実行のためにプロセッサベースのシステム１０にロードしまたは転送することができる。コンピュータ可読プログラム命令７２およびコンピュータ可読媒体６６は、コンピュータプログラム製品６４を構成する。一実施形態では、コンピュータ可読媒体６６は、コンピュータ可読記憶媒体６８および／またはコンピュータ可読信号媒体７０を含むことができる。

【００８７】

コンピュータ可読記憶媒体６８は、ドライブにロードできる光ディスクおよび磁気ディスク、フラッシュメモリデバイス、またはハードドライブなどのストレージデバイスへのデータの転送のための他のストレージデバイスもしくは他のストレージハードウェアを含むがこれに限定されない、さまざまな異なる実施形態を含むことができる。コンピュータ可読記憶媒体６８を、プロセッサベースのシステム１０にノンリムーバブルにインストールすることができる。コンピュータ可読記憶媒体６８は、任意の適切な記憶媒体を含むことができ、限定なしに、半導体システムまたは伝搬媒体を含むことができる。これに関して、コンピュータ可読記憶媒体６８は、電子媒体、磁気媒体、光媒体、電磁媒体、および赤外線媒体を含むことができる。たとえば、コンピュータ可読記憶媒体６８は、磁気テープ、コンピュータディスケット、ランダムアクセスメモリ、および読取り専用メモリを含むことができる。光ディスクの非限定的な実施形態の例は、コンパクトディスク読取り専用メモリ、書換可能コンパクトディスク、およびディジタルビデオディスクを含む。

【００８８】

コンピュータ可読信号媒体７０は、コンピュータ可読プログラム命令７２を含むことができ、電磁信号および光信号を含むがこれに限定されないさまざまなデータ信号構成で実施され得る。そのようなデータ信号を、無線手段またはハードワイヤ手段によるものを含む任意の適切な通信リンクによって送信することができる。たとえば、ハードワイヤ手段は、光ファイバケーブル、同軸ケーブル、信号線、および無線手段または物理手段によってデータを送信する任意の他の適切な手段を含むことができる。

【００８９】

引き続きさらに図２３を参照すると、コンピュータ可読信号媒体７０は、プロセッサベースのシステム１０内での使用のための不揮発性ストレージまたは他の適切なストレージデバイスもしくはメモリデバイスへのコンピュータ可読プログラム命令７２のダウンロードを容易にすることができる。たとえば、コンピュータ可読記憶媒体６８内に含まれるコンピュータ可読プログラム命令７２を、コンピュータネットワークを介して別のシステムのサーバコンピュータまたはクライアントコンピュータからプロセッサベースのシステム１０にダウンロードすることができる。

【００９０】

プロセッサベースのシステム１０のさまざまな異なる実施形態の任意の１つを、コンピュータ可読プログラム命令７２を実行できる任意のハードウェアデバイスまたはシステムを使用して実施することができる。たとえば、プロセッサ５２は、１つまたは複数の特定の機能を実行するように構成されたハードウェアユニットを含むことができ、ここで、その機能を実行するコンピュータ可読プログラム命令７２を、メモリデバイス５４に事前にロードすることができる。

【００９１】

一実施形態では、プロセッサ５２は、特定用途向け集積回路（ＡＳＩＣ）、プログラマブル論理デバイス、または１つもしくは複数の特定の機能もしくは動作を実行するように構成された任意の他のハードウェアデバイスを含むことができる。たとえば、プログラマブル論理デバイスを、入力ＷＦＦ　１００（図１）を戻りＷＦＦ　１３８（図１）に単純

化する方法論に関する動作のうちの１つまたは複数を実行するように一時的にまたは永久的にプログラムすることができる。プログラマブル論理デバイスは、限定なしに、プログラマブル論理アレイ、プログラマブルアレイ論理、フィールドプログラマブル論理アレイ、およびフィールドプログラマブルゲートアレイ、ならびに任意の他の適切な論理デバイスを含むことができる。一実施形態では、コンピュータ可読プログラム命令７２を、１つもしくは複数のプロセッサ５２によって、および／またはプロセッサ５２と通信する１つもしくは複数のハードウェアユニットを含む他のデバイスによって、動作させることができる。コンピュータ可読プログラム命令７２のある部分を、プロセッサ５２によって走行させ、コンピュータ可読プログラム命令７２の他の部分を、ハードウェアユニットによって走行させることができる。

【００９２】

下記は、上のテキストおよび図面で開示された例示の態様、変形、実例、および例である。一態様では、複雑な完全論理式（ＷＦＦ）を単純化する方法であって、式プロセッサ１８内で入力ＷＦＦ　１００を受け取り、式コンバータ２０を用いて、入力ＷＦＦ　１００を初期ビット配列２００に変換すること、ビット配列シンプリファイヤ２２を用いて、入力ＷＦＦ　１００を表すのに必要ではない述語１１４を初期ビット配列２００から除去することによって、初期ビット配列２００を単純化されたビット配列２１２に単純化すること、およびビット配列コンバータ２４を用いて、単純化されたビット配列２１２を連言標準形１４２または選言標準形１４０の戻りＷＦＦ　１３８に変換することを実行するステップと、式プロセッサ１８を用いて、戻りＷＦＦ　１３８を返すステップとを含む方法を開示する。１つの変形形態では、入力ＷＦＦ　１００を初期ビット配列２００に変換するステップは、入力ＷＦＦ　１００内の述語１１４を判定すること、述語１１４の各１つに関連する領域要素１２０を判定すること、述語１１４および領域要素１２０に対応する初期ビット配列２００のビット配列次元２１８を判定すること、およびビット配列次元２１８ならびにそれに関連する述語１１４および領域要素１２０を有する初期ビット配列２００を返すために入力ＷＦＦ　１００を処理することを含む。別の変形形態では、入力ＷＦＦ　１００は、真１４８または偽１５０の原子ＷＦＦであり、初期ビット配列２００を返すために入力ＷＦＦ　１００を処理するステップは、真１４８または偽１５０の原子ＷＦＦについて、それぞれの世界の真の配列２１４または世界の偽の配列２１６を返すことを含む。

【００９３】

１つの実例では、入力ＷＦＦ　１００は、ブール述語１２４または等価述語１２６を含む原子ＷＦＦ　１０６であり、初期ビット配列２００を返すために入力ＷＦＦ　１００を処理するステップは、原子ＷＦＦがブール述語１２４である場合に、ブール述語１２４を等価述語１２６に変換すること、等価述語１２６について、ビット配列次元２１８ならびにそれに関連する述語１１４および領域要素１２０を判定すること、配列キャッシュ３０から、等価述語１２６に関連するビット配列次元２１８、述語１１４、および領域要素１２０を有する初期ビット配列２００を取り出すこと、および配列キャッシュ３０からの初期ビット配列２００の取出の不在時に初期ビット配列２００を構成することを含む。もう１つの実例では、入力ＷＦＦ　１００は、複合ＷＦＦ　１３６であり、初期ビット配列２００を生成するために入力ＷＦＦ　１００を処理するステップは、原子ＷＦＦ　１０６に出会うまで、複合ＷＦＦ　１３６を再帰的に処理すること、再帰的に処理された原子ＷＦＦ　１４８、１５０の各１つについて初期ビット配列２００を返すこと、および否定演算子１３０、連言演算子１３２、および選言演算子１３４のうちの少なくとも１つに基づいて初期ビット配列２００を組み合わせることを含む。もう１つの実例では、初期ビット配列２００は、述語１１４に関連する複数のビット配列次元２１８を有し、初期ビット配列２００を単純化するステップは、セマンティックス的に冗長なビット配列次元２１８を除去することによって初期ビット配列２００を縮小させることを含む。さらにもう１つの実例では、ビット配列次元２１８は、ビット要素を有する部分配列２２０を含み、初期ビット配列２００を縮小させるステップは、ビット配列次元２１８の各１つについて部分配列

２２０のビット要素２２２を比較すること、等しいビット要素２２２を有する部分配列２２０を有する各ビット配列次元２１８を初期ビット配列２００から除去すること、初期ビット配列２００から除去されないビット配列次元２１８を含む保存された次元インデックス２８を生成すること、および単純化されたビット配列２１２および保存された次元インデックス２８を返すことを含む。

【００９４】

一例では、単純化されたビット配列２１２は、ビット要素からなり、単純化されたビット配列２１２を戻りＷＦＦ　１３８に変換するステップは、単純化されたビット配列２１２内のビット要素２２２の総量を判定すること、１の値を有する単純化されたビット配列２１２内のビット要素２２２の量を判定すること、ビット要素２２２の総量の半分未満が１の値を有する場合に、単純化されたビット配列２１２を選言標準形１４０のＷＦＦに変換すること、およびビット要素２２２の総量の少なくとも半分が１の値を有する場合に、単純化されたビット配列２１２を連言標準形１４２の戻りＷＦＦ　１３８に変換することを含む。もう１つの例では、この方法は、式プロセッサ１８内で入力ＷＦＦ　１００を受け取る前に、古典的プロセッサ１４内で、１つまたは複数の古典的アルゴリズムを使用して入力ＷＦＦ　１００を処理することによって所定のタイムアウト期間内に結果のＷＦＦ　１０４を処理するステップと、結果のＷＦＦ　１０４またはその不在を式プロセッサ１８に通信するステップと、結果のＷＦＦ　１０４の不在の場合に式プロセッサ１８内で入力ＷＦＦ　１００を受け取るステップとをさらに含む。

【００９５】

一態様では、設計制約式計算をサポートする方法であって、所定の時間期間内に結果の完全論理式（ＷＦＦ）１０４を生成するために１つまたは複数の古典的アルゴリズムを使用して少なくとも１つの入力ＷＦＦを、古典的プロセッサ１４を用いて処理するステップと、結果のＷＦＦ　１０４またはその不在を式プロセッサ１８に通信するステップと、式プロセッサ１８内で古典的プロセッサ１４から入力ＷＦＦ　１００を受け取り、式コンバータ２０を用いて入力ＷＦＦ　１００を初期ビット配列２００に変換すること、ビット配列シンプリファイヤ２２を用いて、入力ＷＦＦ　１００を表すのに必要ではない述語１１４を除去し、単純化されたビット配列２１２をもたらすことによって初期ビット配列２００を単純化すること、ビット配列コンバータ２４を用いて、単純化されたビット配列２１２を連言標準形１４２または選言標準形１４０の戻りＷＦＦ　１３８に変換すること、および戻りＷＦＦ　１３８を返すことを実行するステップとを含む方法を開示する。

【００９６】

もう１つの態様では、設計制約式計算をサポートするプロセッサベースのシステムであって、入力完全論理式ＷＦＦを初期ビット配列２００に変換するように構成された式コンバータ２０と、入力ＷＦＦ　１００を表すのに必要ではない述語１１４を初期ビット配列２００から除去することによって、初期ビット配列２００を単純化されたビット配列２１２に単純化するように構成されたビット配列シンプリファイヤ２２と、単純化されたビット配列２１２を連言標準形１４２または選言標準形１４０の戻りＷＦＦ　１３８に変換するように構成されたビット配列コンバータ２４とを含むシステムを開示する。１つの変形形態では、式コンバータ２０は、入力ＷＦＦ　１００内の述語１１４を判定し、述語１１４の各１つに関連する領域要素１２０を判定し、述語１１４および領域要素１２０に対応する初期ビット配列２００のビット配列次元２１８を判定するように構成される。別の変形形態では、入力ＷＦＦ　１００は、真１４８または偽１５０の原子ＷＦＦであり、式コンバータ２０は、真１４８または偽１５０の原子ＷＦＦを、０次元で世界の真の配列２１４または世界の偽の配列２１６のそれぞれの１つを含む初期ビット配列２００に変換するように構成される。もう１つの変形形態では、このシステムは、１つまたは複数の初期ビット配列２００を含む配列キャッシュ３０と、ブール述語１２４または等価述語１２６を含む入力ＷＦＦ　１００と、ブール述語１２４を等価述語１２６に変換し、ビット配列次元２１８ならびに等価述語１２６に関連する述語１１４および領域要素１２０を判定し、配列キャッシュ３０から、等価述語１２６を表す初期ビット配列２００を取り出すように

構成された式コンバータ２０とをさらに含む。さらにもう１つの変形形態では、このシステムは、配列キャッシュ３０内での初期ビット配列２００の不在の場合に初期ビット配列２００を構成するように構成されたビット配列コンストラクタ２６をさらに含む。

【００９７】

１つの実例では、入力ＷＦＦ　１００は、少なくとも１つの演算子によって関連付けられる複数の原子ＷＦＦ　１４８、１５０を含む複合ＷＦＦ　１３６であり、式コンバータ２０は、原子ＷＦＦ　１４８、１５０のうちの各１つの初期ビット配列２００を返し、否定演算子１３０、連言演算子１３２、および選言演算子１３４のうちの少なくとも１つに基づいて初期ビット配列２００を組み合わせるように構成される。もう１つの実例では、初期ビット配列２００は、述語１１４に関連する複数のビット配列次元２１８を有し、ビット配列シンプリファイヤ２２は、ビット配列次元２１８のうちの少なくとも１つを除去することによって初期ビット配列２００を縮小させるように構成される。さらにもう１つの実例では、ビット配列次元２１８は、ビット要素を有する部分配列２２０を含み、ビット配列シンプリファイヤ２２は、ビット配列次元２１８の各１つについて部分配列２２０のビット要素２２２を比較し、等しいビット要素２２２を有する部分配列２２０を有する各ビット配列次元を初期ビット配列２００から除去するように構成される。一例では、このシステムは、単純化されたビット配列内のビット要素２２２の総量に対する１の値を有する単純化されたビット配列２１２内のビット要素２２２の量を判定し、ビット要素２２２の総量の半分未満が１の値を有する場合に、単純化されたビット配列２１２を選言標準形１４０のＷＦＦに変換し、ビット要素２２２の総量の少なくとも半分が１の値を有する場合に、単純化されたビット配列２１２を連言標準形１４２の戻りＷＦＦ　１３８に変換するように構成されたビット配列コンバータ２４をさらに含む。もう１つの例では、このシステムは、入力ＷＦＦ　１００を受け取り、１つまたは複数の古典的アルゴリズムを使用して所定のタイムアウト期間内に結果のＷＦＦ　１０４を判定するように構成された古典的プロセッサ１４をさらに含み、古典的プロセッサ１４は、結果のＷＦＦ　１０４またはその不在を式プロセッサ１８に通信するように構成され、式プロセッサ１８は、古典的プロセッサ１４からの結果のＷＦＦ　１０４の不在の場合に入力ＷＦＦ　１００を受け取るように構成される。

【００９８】

有利なことに、入力ＷＦＦを単純化する、本明細書で開示されるシステムおよび方法を、上で示したようにデータ依存制約ネットワーク内の条件プラニングで実施することができる。これに関して、入力ＷＦＦを単純化する、本明細書で開示されるシステムおよび方法を、データ依存制約ネットワークの一部である入力変数のユーザ指定の集合から出力変数のユーザ指定の集合の値を計算するための条件計算プランを作成するプロセスで実施することができる。有利なことに、条件計算プランを作成するプロセスは、伝統的な条件プラニングアルゴリズムによって要求されるプラニングと計算との混合を回避する。そのようなプロセスを使用する計算プランの判定は、識別できる待ち時間をほとんどまたは全く伴わずに、比較的迅速である。時間の節約を、設計者が有利に使用して、トレード研究中により大きくより複雑な設計空間を探査することができる。

【００９９】

前述の説明および関連する図面で提示される教示の利益を有する、本開示が関連する技術分野における通常の技量を有するものは、本開示の多数の修正形態および他の実施形態を思い浮かべるであろう。本明細書で説明される実施形態は、例示的であることを意図され、限定的または網羅的であることは意図されていない。本明細書で特定の用語が使用されるが、それらの用語は、包括的で説明的な意味でのみ使用され、限定のためのものではない。

【符号の説明】

【０１００】

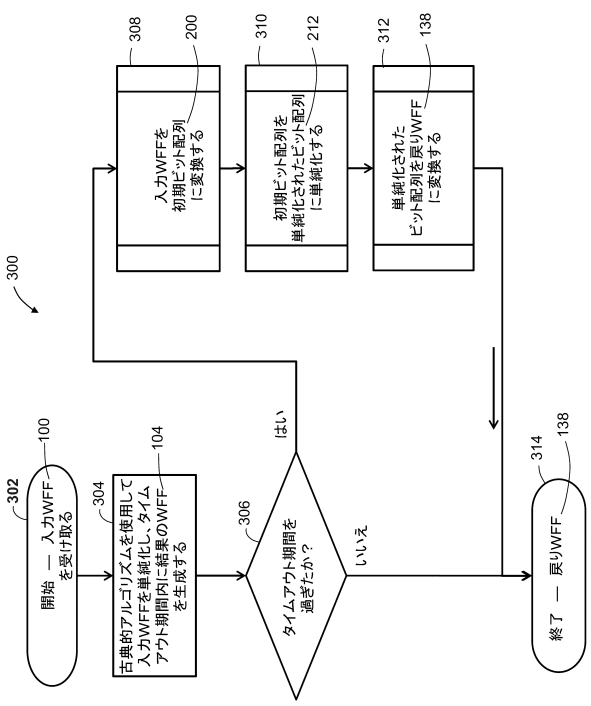
１０　　システム

１２　　データ依存制約管理システム

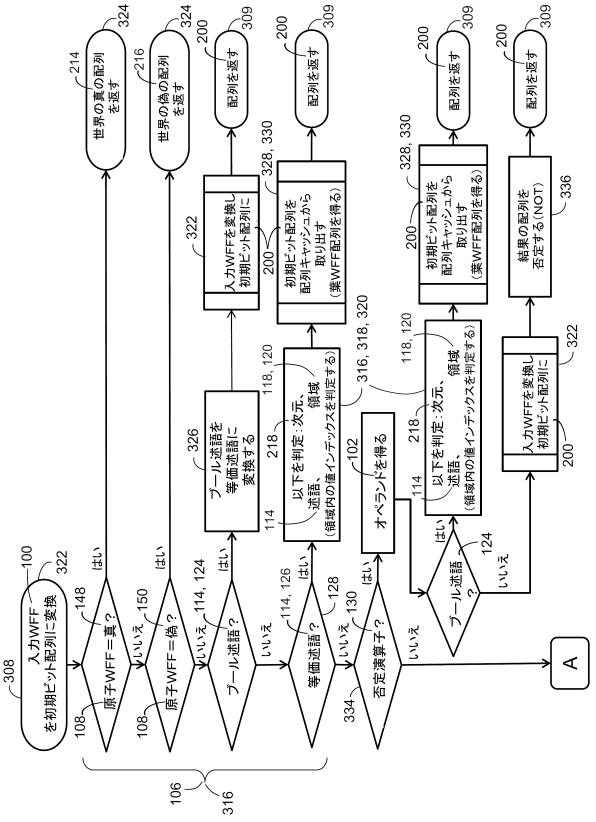
１４　古典的プロセッサ
１６　古典的アルゴリズム
１８　式プロセッサ
２０　式コンバータ
２２　ビット配列シンプリファイヤ
２４　ビット配列コンバータ
２６　ビット配列コンストラクタ
２８　保存された次元インデックス
３０　配列キャッシュ
５０　データ通信パス
５２　プロセッサ
５４　メモリデバイス
５６　不揮発性ストレージデバイス
５８　入出力デバイス
６０　通信デバイス
６２　ディスプレイデバイス
６４　コンピュータプログラム製品
６６　コンピュータ可読媒体
６８　コンピュータ可読記憶媒体
７０　コンピュータ可読信号媒体
７２　コンピュータ可読プログラム命令
１００　入力ＷＦＦ
１０２　オペランド
１０４　結果のＷＦＦ
１０６　原子ＷＦＦ
１０８　真または偽の原子ＷＦＦ
１１０　否定原子ＷＦＦ
１１４　述語
１１６　述語リスト
１１８　領域
１２０　領域要素
１２４　ブール述語
１２６　等価述語
１３０　否定演算子
１３２　連言演算子
１３４　選言演算子
１３６　複合ＷＦＦ
１３８　戻りＷＦＦ
１４０　選言標準形（ＤＮＦ）
１４２　連言標準形（ＣＮＦ）
１４８　真の原子ＷＦＦ
１５０　偽の原子ＷＦＦ
２００　初期ビット配列
２１２　単純化されたビット配列
２１４　世界の真の配列
２１６　世界の偽の配列
２１８　ビット配列次元
２２０　部分配列
２２２　ビット要素
２４８　擬似コードリスティング

２５０　　擬似コードリスティング
２５２　　擬似コードリスティング
２５４　　擬似コードリスティング
２５６　　擬似コードリスティング
２５８　　擬似コードリスティング
２６０　　擬似コードリスティング
２６２　　擬似コードリスティング
３００　　方法
３０６　　所定のタイムアウト期間

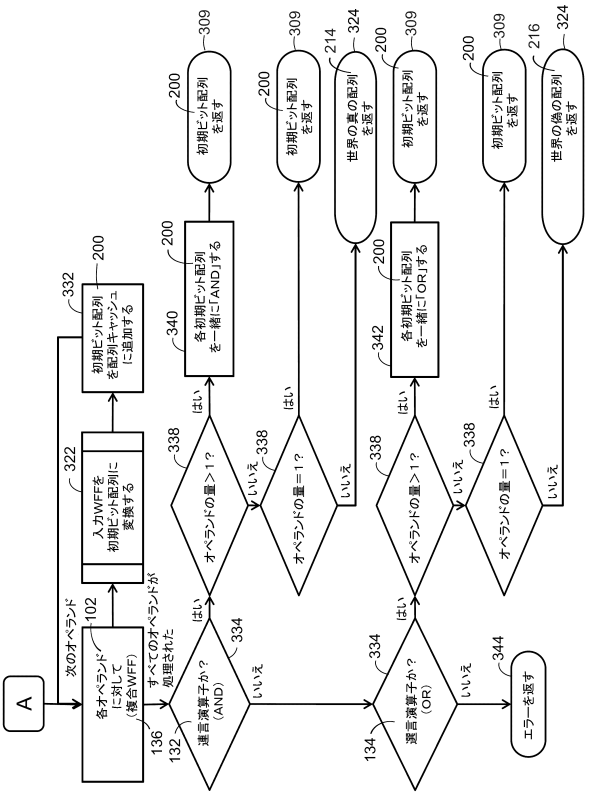
【図１】



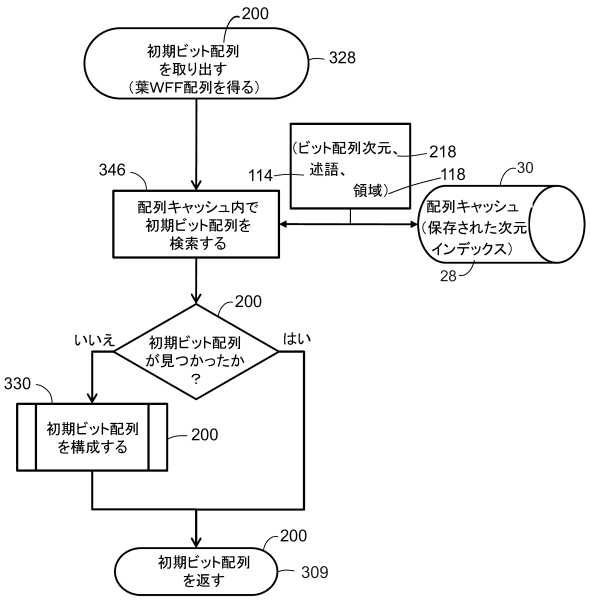
【図２Ａ】



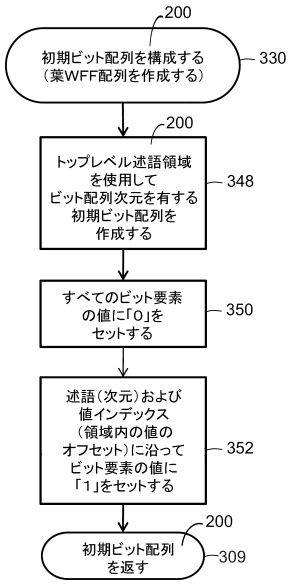
【図２Ｂ】


A
次のオペランド
各オペランドに対して（複合WFF）
102
すべてのオペランドが処理された
136
述語演算子か？（AND）
132
334
はい
いいえ
オペランドの量＞1？
338
入力WFFを初期ビット配列に変換する
322
初期ビット配列を配列キャッシュに追加する
332
200
各初期ビット配列を一緒に「AND」する
340
初期ビット配列を返す
309
オペランドの量＝1？
214
世界の真の配列を返す
324
述語演算子か？（OR）
134
342
各初期ビット配列を一緒に「OR」する
エラーを返す
344
世界の偽の配列を返す
216

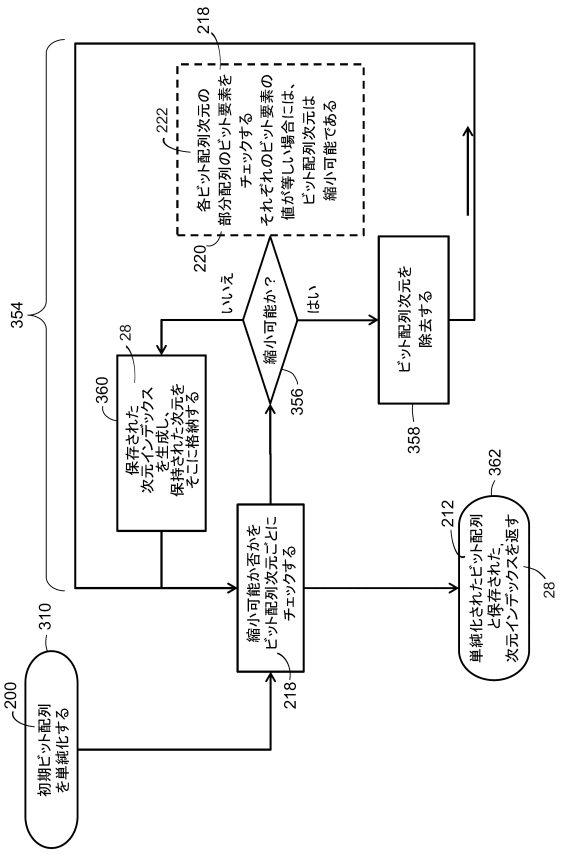
【図３】


初期ビット配列を取り出す（葉WFF配列を得る）
200
328
（ビット配列次元、述語、領域）
218
114
118
30
配列キャッシュ（保存された次元インデックス）
28
346
配列キャッシュ内で初期ビット配列を検索する
初期ビット配列が見つかったか？
いいえ
はい
330
初期ビット配列を構成する
初期ビット配列を返す
309

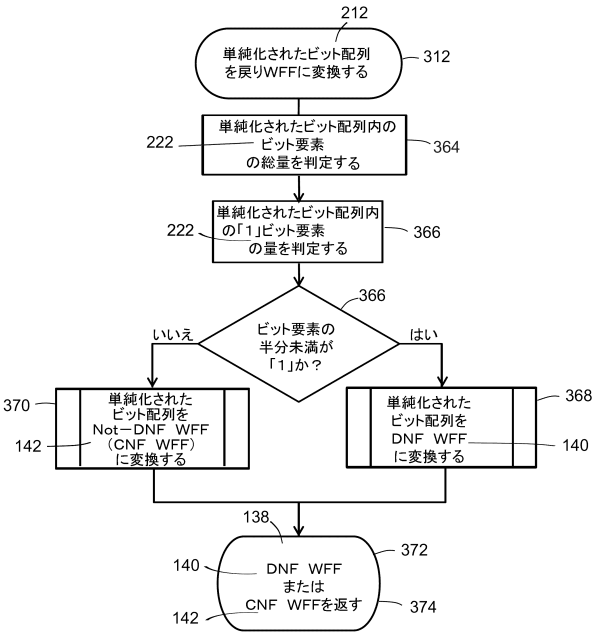
【図４】


初期ビット配列を構成する（葉WFF配列を作成する）
200
330
トップレベル述語領域を使用してビット配列次元を有する初期ビット配列を作成する
348
すべてのビット要素の値に「0」をセットする
350
述語（次元）および値インデックス（領域内の値のオフセット）に沿ってビット要素の値に「1」をセットする
352
初期ビット配列を返す
309

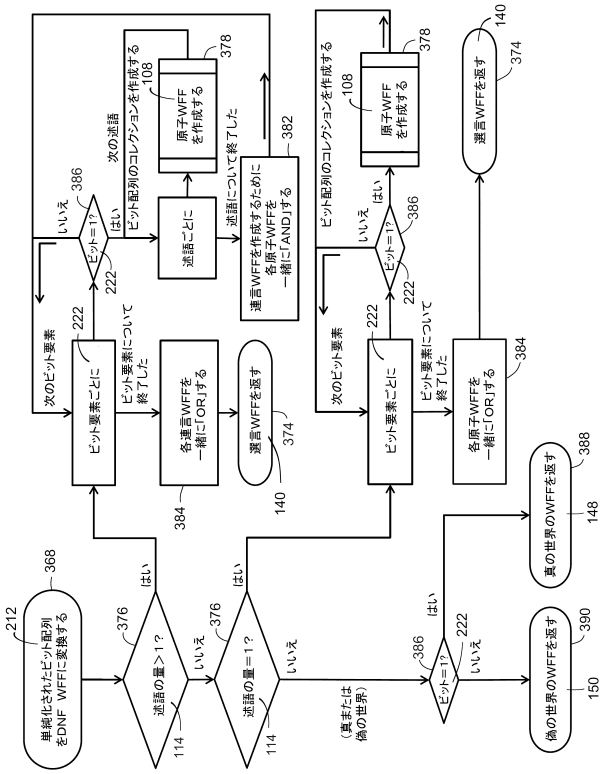
【図５】


218
各ビット配列次元の部分配列のビット要素をチェックする
それぞれのビット要素の値が等しい場合には、ビット配列次元は縮小可能である
222
縮小可能か？
220
いいえ
はい
356
ビット配列次元を除去する
358
保存された次元インデックスを生成し、保持された次元をそこに格納する
28
360
354
縮小可能か否かをビット配列次元ごとにチェックする
218
単純化されたビット配列と保存された次元インデックスを返す
212
362
28
初期ビット配列を単純化する
200
310

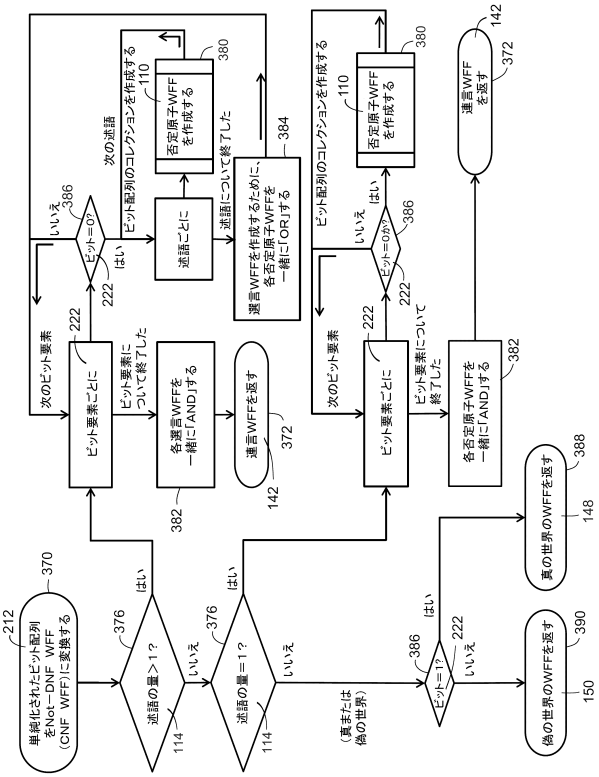
【図６】


212
単純化されたビット配列を戻りWFFに変換する
312
222
単純化されたビット配列内のビット要素の総量を判定する
364
222
単純化されたビット配列内の「1」ビット要素の量を判定する
366
366
ビット要素の半分未満が「1」か？
いいえ
はい
370
142
単純化されたビット配列をNot－DNF WFF（CNF WFF）に変換する
368
140
単純化されたビット配列をDNF WFFに変換する
138
140
142
DNF WFF または CNF WFFを返す
372
374

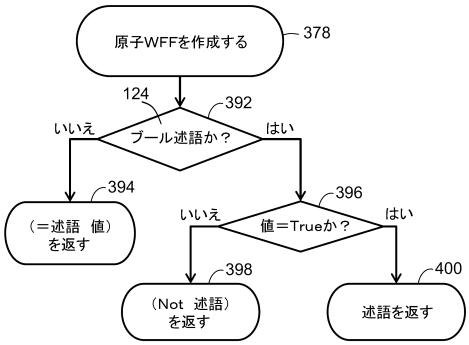
【図７】


212
単純化されたビット配列をDNF WFFに変換する
368
376
述語の量＞1？
114
376
述語の量＝1？
114
（真または偽の世界）
386
ビット＝1？
222
偽の世界のWFFを返す
390
150
真の世界のWFFを返す
388
148
次のビット要素
ビット要素ごとに
222
ビット要素について終了した
各連言WFFを一緒に「OR」する
384
連言WFFを返す
374
140
次の述語
ビット配列のコレクションを作成する
108
原子WFFを作成する
378
述語ごとに
連言について終了した
連言WFFを作成するために各原子WFFを一緒に「AND」する
382
各原子WFFを一緒に「OR」する

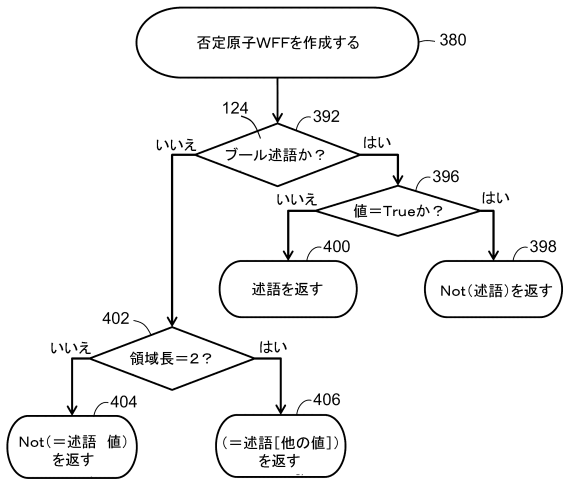
【図８】


212
単純化されたビット配列をNot－DNF WFF（CNF WFF）に変換する
370
376
述語の量＞1？
114
376
述語の量＝1？
114
（真または偽の世界）
386
ビット＝1？
222
偽の世界のWFFを返す
390
150
真の世界のWFFを返す
388
148
次のビット要素
ビット要素ごとに
222
ビット要素について終了した
各連言WFFを一緒に「AND」する
382
連言WFFを返す
372
142
次の述語
ビット配列のコレクションを作成する
110
否定原子WFFを作成する
380
述語ごとに
連言について終了した
連言WFFを作成するために、各否定原子WFFを一緒に「OR」する
384
ビット＝0？
386
各否定原子WFFを一緒に「AND」する

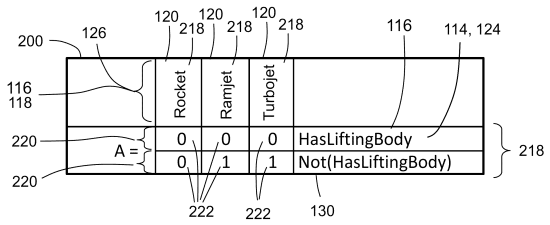
【図９】


原子WFFを作成する
378
124
392
ブール述語か？
いいえ
はい
394
（＝述語 値）を返す
396
値＝Trueか？
いいえ
はい
398
（Not 述語）を返す
400
述語を返す

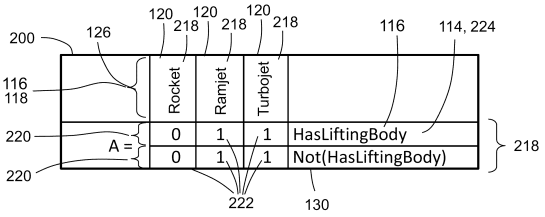
【図１０】



【図１１】



【図１２】



【図１３】

Method: wff_to_array (wff_in)

1. **sorted_predicates** = Canonical sort of predicates in **wff_in**
2. **domains** = List of domains of each predicate in the order of sorted_predicates. For Boolean predicates, this is the special domain {“TRUE”, “FALSE”}
3. **dimensions** = List of integers, the j’th one being the length of the j’th domain.
4. Initialize *array-cache* to empty for special caching of arrays for reuse to control extensive memory use.
5. Set **bit-array** = wff_to_array_internal(**wff_in, sorted_predicates, domains, dimensions**)
6. Return **bit-array** and **predicates** as multiple values

248

【図１４】

Method: wff_to_array_internal (wff_in, predicates, domains, dimensions)

1. Atomic True or False WFF or Boolean Predicate WFF:
   a. If **wff_in** = True, return zero dimensional array with specified initial bit element of 1. In LISP, this is #0A1
   b. If **wff_in** = False, return zero dimensional bit array with specified initial bit element of 0. In LISP, this is #0A0
   c. If **wff_in** is a single Boolean predicate,
      i. Convert single Boolean WFF to look like an equality predicate for processing by the wff_1_to_array method. Set **boolean_wff** to (= wff_in “TRUE”)
      ii. Return **wff_to_array_internal** (**boolean_wff, predicates, domains, dimension**)
   d. Else, go to 2.
2. Equality predicate or a compound WFF. Treat the following cases in order depending upon the value of **operator**, which is set to return value of **Operator(wff_in)**
   a. If operator is “=”, this is an equality predicate; do the following, else go to b:
      i. Set **pred_index** to return value of **Position (Predicate(wff_in), predicates)**
      ii. Set **domain** to return value of **Element (domains, pred_index)**
      iii. Set **value_index** to return value of **Position (PredicateValue(wff_in),domain)**
      iv. Return **GetLeafWffArray(dimensions, pred_index, value_index)**
   b. If **operator** is “Not”, do the following, else go to c.
      i. Set **wff_2** to **Arg1(wff_in)**
      ii. If **wff_2** is Boolean predicate, do the following, else go to iii.
         1. Set **pred_index** to return value of **Position(wff_2, predicates)**
         2. Set **domain** to return value of **Element(domains, pred_index)**
         3. **value_index** to return value of **Position (“FALSE”, domain)**
         4. Return **GetLeafWffArray(dimensions, pred_index, value_index)**
      iii. At this point we have a negated compound WFF. Process the arguments and negate the result.
         Set **array** to return value of **wff_to_array_internal (arg1, predicates, domains, dimensions)**
      iv. Return **BitNot(array)**
   c. Non-negated compound statement headed by And or Or. Process the results recursively and combine the answers using the appropriate bit array combiner.
      i. For each **wff[i]** in the list of WFF’s returned by **Args(wff_in)**, collect the return value of wff_to_array_internal(wff[i], predicates, domains, dimensions) into the list **results**
      ii. If **operator** = AND, set **combined_array** to the combination of the bit arrays in **results** using **BitAnd**, else **operator** must be OR, so set **combined_array** to the combination the bit arrays in **results** using **BitOr**. Return the **combined_array**.

250

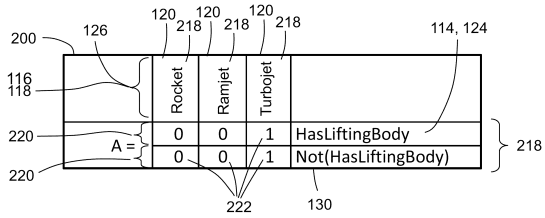
【図１５】

Method: GetLeafWffArray(dimensions, pred_index, value_index)

1. Return the **array** in ***array-cache*** with key **(pred_index, value_index)** if it exists. Otherwise, go to 2
2. Set **array** to **MakeBitArray**(**dimensions**, InitialElements=0)
3. Set elements of **array** by calling the recursive function **SetArrayDimensionIndex(array, pred_index, value_index, dimensions, current_dimension, starting_indices),** where
   set_dimension is set to **pred_index**
   index is set to **value_index**
   current_dimension is initialized to 1
   starting_indices is initialized to the empty list
4. Add **array** to ***array-cache*** with key **(pred_index, value_index)**
5. Return **array**.

252

【図１６】



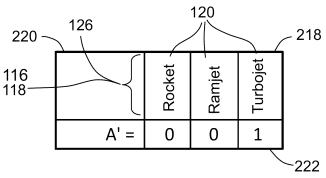
【図１７】

Method: SetArrayDimensionIndex (bit_array, set_dimension, index, dimensions, current_dimension, indices)

1. Set **this_dimension_size** to **Elements(dimensions, current_dimension)**
2. If **this_dimension** does not equal **Length(dimensions),** then go to 3, else go to 2.a.
   a. We are now processing the last dimension of the array. If **current_dimension** does not equal **set_dimension**, then go to 2.b. Otherwise, the last dimension is the same as **set_dimension** and we are at the leaf of recursive calls that increment **indices**.
   Set the appropriate element of bit_array with the call **SetBit(array, indices**, 1) and return from **SetArrayDimensionIndex**.
   b. We are at the last dimension after already passing the **set_dimension**. In this case, set all worlds associated with **current_dimension**. That is, for k from 1 to **this_dimension_size**, call **SetBit**(array, Append(**indices**, {k}), 1) and then return from **SetArrayDimensionIndex.**
3. If **current_dimension** does not equal **set_dimension,** go to 4; otherwise, we are at the **set_dimension** and no looping over all the domain values for **current_dimension** is appropriate. We need to set the elements associated with **index** only. Thus, at this point we just append **index** to the growing **indices** list, increment **current_dimension** to the next level, and modify **bit_array** with the recursive call:
SetArrayDimensionIndex (bit_array, set_dimension, index, dimensions, (1+ current_dimension), Append(indices, {index}), after which we return form this call to **SetArrayDimensionIndex.**
4. At this point, **current_dimension** does not equal **set_dimension** and we are not at the last dimension. Here we recursively call **SetArrayDimensionIndex** with **indices** augmented with each value for the **current_dimension** and with a new **current_dimension** incremented to the next level. Thus, for each k from 1 to **this_dimension_size**, we recursively call as follows:
SetArrayDimensionIndex (bit_array, set_dimension, index, dimensions, (1+ current_dimension), Append(indices, {k})
We then return from **SetArrayDimensionIndex** with the modified **bit_array.**

254

【図１８】



【図１９】

Method: SimplifyArray(bit_array, [(dimension-slice = 1), (axis = 1), (dimensions-remaining = {})])

1. If **dimension-slice** is less than (array-rank **bit-array**), go to 2. Else, we are processing the last axis of the input **bit-array**. Collapse along this axis if possible and return the result along with the list of axes that were not collapsed by executing the following:
   a. Set **updated-array** = CollapseIfPossible(**bit-array, dimension-slice**)
   b. Unless **updated-array** was collapsed from **bit-array** in this last call, store **axis** in **dimensions-remaining**.
   c. Return from SimplifyArray: **updated-array** and **dimensions-remaining**.
2. We are processing an axis of bit-array less than the last axis. See if we can collapse along this axis, update internal variables, and recur to the next axis by executing the following:
   a. Set **updated-array** = CollapseIfPossible(**bit-array, dimension-slice**).
   b. Unless **updated-array** was collapsed from **bit-array** in this last call, do the following:
      i. store **axis** in **dimensions-remaining**
      ii. increment **dimension-slice**
   c. Increment axis.
   d. Recur to next **dimension-slice** by calling
      SimplifyArray(**updated-array, dimension-slice, axis, dimensions-remaining**)

256

【図２０】

Method: ArrayToWff(bit_array_in, predicates, domains)

1. **ones** = count the “one” bits in the array
2. If ones < half the total number of bits in the array, then **wff** = ArrayToDNF(**bit_array_in**, **predicates_in**, **domains_in**)
3. Else, **wff** = ArrayToNotDNF(**bit_array_in**, **predicates_in**, **domains_in**)
4. Return **wff**

258

【図２１】

Method: ArrayToDNF(bit_array_in, predicates, domains)

1. **npred** = number of items in **predicates**
2. If **npred** > 1, then
   a. For each “1” bit in **bit_array_in**, collect the following:
      i. For each predicate/domain value pair associated with this bit array position
         1. Create an atomic wff from the predicate and domain value
      ii. Create a conjunction, **wff1**, containing each wff created in 2.a.i.
   b. Create a disjunction, **wff2**, containing each **wff1** created in 2.a.
   c. Return **wff2**
3. If **npred** = 1, then
   a. For each “1” bit in **bit_array_in**, collect the following:
      i. Create an atomic wff from the predicate and domain value associated with this bit array position
   b. Create a disjunction, **wff1**, containing each wff created in 2.a.
   c. Return **wff1**
4. If **npred** = 0, then
   a. If **bit_array_in** = 1, then return **T**
   b. Else return **Nil**

260

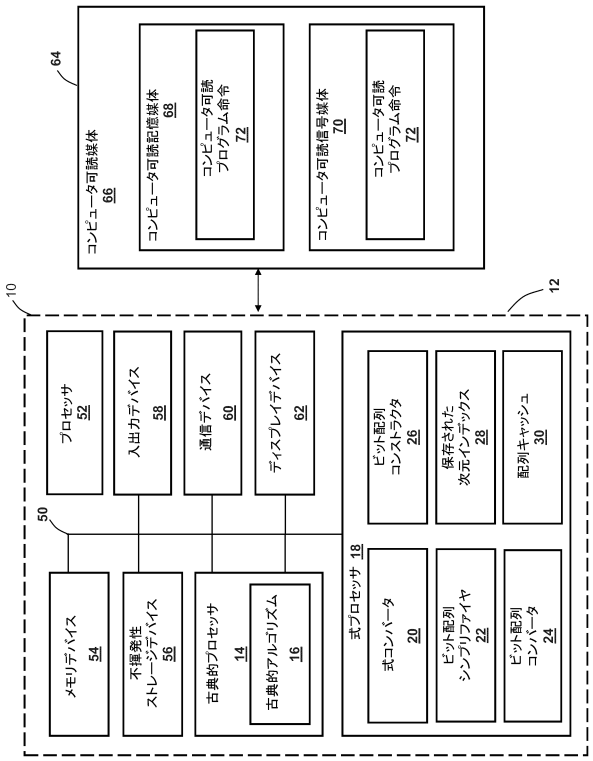
【図２２】

Method: ArrayToNotDNF(bit_array_in, predicates, domains) (Not DNF = CNF)

1. **npred** = number of items in **predicates**
2. If **npred** > 1, then
   a. For each “0” bit in **bit_array_in**, collect the following:
      i. For each predicate/domain value pair associated with this bit array position
         1. Create a negative atomic wff (eg. Not(= predicate value)) from the predicate and domain value
      ii. Create a disjunction, **wff1**, containing each wff created in 2.a.i.
   b. Create a conjunction, **wff2**, containing each **wff1** created in 2.a.
   c. Return **wff2**
3. If **npred** = 1, then
   a. For each “0” bit in **bit_array_in**, collect the following:
      i. Create a negative atomic wff (eg. Not(= predicate value)) from the predicate and domain value associated with this bit array position
   b. Create a conjunction, **wff1**, containing each wff created in 2.a.
   c. Return **wff1**
4. If **npred** = 0, then
   a. If **bit_array_in** = 1, then return **T**
   b. Else return **Nil**

262

【図２３】


64
コンピュータ可読媒体
66
コンピュータ可読記憶媒体
68
コンピュータ可読
プログラム命令
72
コンピュータ可読信号媒体
70
コンピュータ可読
プログラム命令
72
10
12
プロセッサ
52
入出力デバイス
58
通信デバイス
60
ディスプレイデバイス
62
ビット配列
コンストラクタ
26
保存された
次元インデックス
28
配列キャッシュ
30
50
メモリデバイス
54
不揮発性
ストレージデバイス
56
古典的プロセッサ
14
古典的アルゴリズム
16
式プロセッサ
18
式コンパータ
20
ビット配列
シングリファイヤ
22
ビット配列
コンパータ
24

フロントページの続き

(72)発明者 レディ, スダカー ワイ.
アメリカ合衆国 カリフォルニア ９５０５４, サンタ クララ, ラス ドライブ ２３６３
(72)発明者 リギンス, ブルース
アメリカ合衆国 カリフォルニア ９０６０３, ウィッティア, ラリーリン ドライブ １０８００

審査官 田中 幸雄

(56)参考文献 特開２０１２－１９４８３３（ＪＰ，Ａ）
特開２０１０－１４８１１８（ＪＰ，Ａ）
米国特許出願公開第２０１０／００８８２５７（ＵＳ，Ａ１）

(58)調査した分野(Int.Cl., ＤＢ名)
Ｇ０６Ｆ １７／１０
Ｇ０６Ｆ １７／５０