Vol.2018-AL-170 No.11
2018/11/13

誤りの効果的な説明のための反例空間解析

戸田 貴久[1,a)]　井上 武[2]

概要：モデル検査はハードウェアやソフトウェアの検証のための自動推論技術である．システムのモデルに不具合がある場合，モデル検査器は，誤りの説明として違反状態に陥るシステムの動作例 (反例) を返す．反例はモデルにおける不具合を特定するための有用な手がかりであるが，反例計算と不具合特定の間には大きな隔たりがある．本研究では，効果的な誤りの説明のために，反例の集合を対象にして，その構造的な特徴を解析する一般的な計算の枠組みを提案する．反例の集合は，変数ドメインから継承された構造をもつ一種の幾何学空間とみなされる．これを反例空間と呼ぶ．本研究では，反例空間の上の区間構造に焦点を当て，最長区間問題を定式化する．最長区間は，空間の局所領域における顕著な特徴を表すものと考えられ，誤りをより良く理解するために役立つことが期待される．主結果として，最長区間問題を解くアルゴリズムを与え，その計算量を解析する．

1. はじめに

ハードウェアの設計やソフトウェアのコードは人手によって作られているので，誤りが含まれることは避けられない．今日では社会の至るところが情報化され，日常生活はハードウェアやソフトウェアの上に動作する情報システムによって支えられている．したがって，ハードウェアやソフトウェアの安全性を保つことは大切である．

モデル検査は，対象のシステムが期待された通りに正しく動作することを判定したり，誤りがあるとわかった場合には，誤りに至るシステムの動作例 (**反例**) を求めたりする．反例はシステムのどこに不具合があるのかを突き止めるための手がかりになり，大変有用な情報である．実際，モデル検査は産業に広く応用され，実用的な手法として認められている．

しかし，モデル検査だけでは誤りのあるシステムを修正するのに十分ではない．反例はモデル検査器によって自動的に計算されるが，欠陥を特定することはすぐにはできない．実際，システムの記述における欠陥とそれによって引き起こされる動作との間には大きな隔たりがある．一般に，欠陥の特定は反例の人手による検査によってなされる．これは個人の経験や直感に大きく依存する作業であり，反例は長く複雑であるから，人手でこれをすることは煩雑で気が滅入る作業である．このタスクを自動化あるいは半自動化することは困難であるため，我々は**エラーの説明**に集中することがより現実的であると考える．ここで，エラーの説明とは，誤りのトレースから誤りの本質の理解 (可能ならばさらに問題の訂正) に円滑に移れるようにユーザを支援することとして定義される [7].

我々と同じ動機に基づくいくつかの先行研究がある．それらは以下のように 3 つのアプローチに分類される．第一のアプローチでは，特定ドメインに焦点を絞り，ドメイン知識をおおいに活用する．典型的な手法はアニメーションである．しかしながら，アニメーションはモデルの抽象化が過度になされているときにうまく適用できない．このことは，例えば，鉄道車両の連動制御の検証において議論され，その代替として自然言語の解釈に基づく方法が提案されている [18].

第二のアプローチでは，単一の反例をより有用な情報が得られやすいように適した形に変換する．典型的な手法は反例の長さを最小化したり，無関係な情報を除外したりする．他にも変数の値を最小化する手法も提案されている [8].

第三のアプローチでは，複数の反例を生成し，それらを比較して分析する．我々が知る限りにおいてほとんどの先行研究はソフトウェア検証に関するものである．それらでは，ソースコードの行の概念を仮定している．ただ，それ以上のドメイン知識をほとんど用いないので，第一のアプローチとは区別している．第三のアプローチの 1 つの先行研究では，行の概念を活用して誤りの動作と正しい動作の概念を導入し，これら 2 つの種類の動作において共通する特徴や異なる特徴を観察するために集合演算を適用す

[1] 電気通信大学
〒 182-8585 東京都 調布市調布ケ丘 1 丁目 5-1
[2] 日本電信電話株式会社
[a)] todat@acm.org

る [9]．別の先行研究では，ルイスの因果関係の理論をプログラムの実行に応用して，誤りの動作に最も類似すると考えられる正しい動作例を生成する手法が提案されている [7]．

ここで，これら 3 つのアプローチについて議論したい．特定ドメインのアプローチはドメイン知識をフル活用することで大きな効果が期待されるが，応用の範囲は当然狭まり，他のドメインに容易に適用できない．単一の反例に基づくアプローチはそれ程大きな変更をしないでも既存のモデル検査器に後処理として付け加えることができる．しかし，ドメイン知識を活用しないので，反例において何が有用な情報であるかすら不明であることが多い．したがって，反例の複雑さを大幅に減らすことは困難であると考えられる．さらに，モデル検査器によって返された反例は，仕様に反する動作の 1 例に過ぎない．また，そのような反例は反例探索アルゴリズムに完全に依存して決定される．したがって，そのようなたった 1 つの特定の場合を用いて誤りに関する見通しを得るのは困難である．この点において，多数の反例に基づくアプローチは大量の情報を利用できるので有望である．主要な課題は情報の量の利点をいかに活用するかである．1 つの反例でさえ理解するのは困難であるのに，複数の反例ならなおさら困難である．我々は複数の反例をただ計算するだけでなく，反例集合から何らかの方法で誤りの本質を抽出することが重要であると考える．

本稿では，エラーの説明のための汎用的な計算の枠組み (**反例空間解析**) を確立する．反例の集合は一種の幾何学空間と考えられる．この空間では，各反例はベクトル $\vec{c} = (c_1, \ldots, c_n)$ として表される．各値 c_i は対応する変数のドメインから取られる：例えば，整数の集合，グラフの頂点の集合，文字列の集合などである．そのようなドメインにはその上に関係が定義されていることが多い．例えば，整数ドメインには順序関係がある．これらの関係は反例集合の上の関係に拡張されうる．本稿では，そのような関係が導入された反例集合を**反例空間**と呼ぶ．

本稿では，反例空間の区間構造に注目する．着目する整数変数 v が与えられるとき，反例の列 $\vec{c}_1, \ldots, \vec{c}_m$ が v に関する**区間**であるとは，すべての $i \in \{1, \ldots, m-1\}$ に対して $\vec{c}_{i+1}$ は v に相対的に定まる順序に関して $\vec{c}_i$ の直後の元であるときをいう．区間構造は空間の局所領域における顕著な特徴を表すものとして考えられ，誤りをより良く理解するために有用であると期待される．最長区間問題は以下のように定義される：整数変数 v と反例 $\vec{c}$ が与えられて，v に関する反例のなす区間 I であり，I に属す反例は v に対応する値を除いてすべての値が $\vec{c}$ と一致し，そのような区間 I のうち最長のものを計算する問題である．本稿では反例空間解析のための汎用的な計算の枠組みを与えるだけでなく，最長区間問題を計算するための効率的なアルゴリズムもまた与える．さらに，最長区間問題は，ネットワーク設定の検証において，目的の場所に転送されなかった連続するアドレスからなる最長のパケット範囲の計算に対応することを示す．

本稿の構成は以下の通りである．3 節で，反例空間解析の基本的な考え方をネットワーク設定の検証の文脈で説明する．4 節で，反例空間の概念，その上の構造，そして最長区間問題を定式化する．5 節で，最長区間問題に対する近似アルゴリズムと厳密アルゴリズムを与え，厳密アルゴリズムの計算量を解析する．6 節で，反例空間解析に対する計算の枠組みを与える．7 節で，本研究をまとめる．

2. 準備

本節では，モデル検査と BDD について後の節で必要になる概念を簡単にまとめる．

モデル検査は，内部状態が時間とともに非決定的に変化するシステムに対して，その振る舞いを検査する手法である [4] [3]．例えば，n ビットシフトレジスタを考えよう．この内部状態は長さ n のビット列で表される．そのような状態はシフト操作により別の状態に変化する．状態は時間変化するので，システムの状態に関して言及する原始命題の真理値もまた同様である．

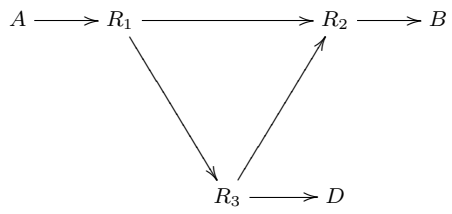
クリプキ構造はそのようなシステムを組 (S, I, T, l) でモデル化したものである．ここで，S は状態の集合である．I は S の部分集合であり，初期状態の集合を表す．T は状態集合 S の上の二項関係であり，状態の遷移を表す．l は状態集合から原始命題の集合のべき集合への写像であり，状態 s に対して $l(s)$ は s において真となる原始命題の集合を意味する．

モデル検査は，シフトレジスタの例を用いるとき，例えば，「すべてのビットがいづれ 0 になる」というようなシステムの振る舞いに関する性質 (**仕様**) をシステムのモデルが満たすか否かを決定する．

仕様は時相論理を用いて記述される．典型的な時相演算子は X, F, G であり，それぞれ「次の時刻で」,「将来のある時刻で」,「将来のすべての時刻で」ということを意味する．さまざまな種類の時相論理のうち，本稿は**線形時相論理** (*LTL*) を用いる．なぜなら本研究の計算の枠組みは SAT ソルバーに基づく有界モデル検査を拡張するからである．

パスは，状態の無限列であり，すべての隣り合う 2 状態はモデルの遷移関係を満たすものを意味する．パスはシステムの可能な動作の 1 例を表している．LTL 式の真偽値はパスに沿って決定される．これは与えられた仕様はある動作によってたまたま満たされるかもしれないが，別の動作によって満たされないことがあるということを意味している．LTL 式 f とパス π が与えられるとき，$\pi \models f$ によって f が π に沿って成立することを表す．

LTL **モデル検査問題**とは，LTL 式 f とクリプキ構造 M が与えられるとき，M の初期状態から始まるすべてのパス



in	dst	src	out
R_1	$10\star$	$01\star$	R_2
R_1	$1\star\star$	$\star\star\star$	R_3
R_2	$10\star$	$\star\star\star$	B
R_3	$\star\star\star$	$1\star\star$	D
R_3	$1\star\star$	$\star\star\star$	R_2

図 1 ネットワークとルーティングテーブル

π に対して $\pi \models f$ が成立するか決定する問題である．

LTL モデル検査は PSPACE 困難であり計算困難であるから，その近似として**有界モデル検査**が提案されている [1]．有界モデル検査では，探索の範囲 (初期状態からのステップ数：**バウンド**) を限定し，その範囲の中だけで反例を見つけようとする．したがって，検証の完全性は保証されない．範囲を逐次広げながら有界モデル検査を繰り返すことで検証は可能だが，有界モデル検査は検証の手段というよりもむしろ，バグ発見のための実用的手法として広く認められている．有界の設定における LTL 式の意味論や**有界モデル検査問題**は同様の方法で定義される．特に，バウンドを k とするとき，LTL 式 f がパス π に沿って成立することを $\pi \models_k f$ と表す．f が成立しないときパス π のこと (すなわち，$\pi \not\models_k f$ を満たす π) を**反例**という．

二分決定グラフ（*BDD*）は論理関数のためのデータ表現である．BDD は閉路のない有向グラフであり，根節点はちょうど 1 つ存在する．各非終端節点は変数インデックスと 2 つの子節点（*LO* **子節点**と *HI* **子節点**と呼ばれる）へのポインタを保持する．*LO* 子節点への枝を *LO* **枝**と呼び，*HI* 子節点への枝を *HI* **枝**と呼ぶ．節点 f の *LO* 枝は f に割り当てられた変数に 0 を代入することを意味する．同様に，*HI* 枝の場合は 1 を代入することを意味する．2 つの終端節点 $\top, \bot$ が存在する．根から $\top$ へのパスは論理関数を真にする変数への値割当を表現する．BDD の変数は根から終端節点に至るパスに沿ってあらかじめ決められた変数順序に従って現れる．BDD は**既約**である：(1) 等価な部分グラフは共有されている;(2)*HI* 枝と *LO* 枝の行き先が同一となる節点は削除され，明示的にグラフに表現されない．

3. 基本的な考え方

本節では，近年注目を集めているネットワーク検証 [16] の文脈で反例空間解析の基本的な考え方について述べる．

図 1 はルータ R_1, R_2, R_3 と末端節点 A, B, D からなるネットワークである．これは [13] で用いられたネットワークを単純化したものである．ネットワークの下の表はルーティングテーブルを表している．単純のため，パケットは *dst* と *src* の 2 つのフィールドだけからなる．各フィールドは長さ 3 のビット列で表される．表の各行はパケットの転送規則である．列 *in* はパケットの現在の場所を表し，列 *dst* と *src* はアドレスパターンを表す．ここで $\star$ はワイルドカードであり，0 でも 1 でも照合する．列 *out* は次に転送されるパケットの場所を表す．各パケットに対して，表の規則は上から順に走査され，in, dst, src の条件にすべて該当する最初の規則が適用され，その規則により，パケットは *out* によって示される場所に転送される．例えば，R_1 は *dst* が $10\star$ で *src* が $01\star$ のパケットを R_2 に転送し，その他のパケットのうち，*dst* が $1\star\star$ のパケットを R_3 に転送し，残りのすべてのパケットを破棄する．

このように与えられたネットワークの設定はクリプキ構造を用いてモデル化される．上のネットワークに対する状態は 2 つの変数 *packet* と *location* によって表現される．ここで、*packet* は *dst* と *src* を表す 2 つのビット列を保持する．それぞれ *packet.dst* と *packet.src* で参照される．変数 *location* はネットワーク節点を保持する．パケットの最初の場所を A とせよ．その後の場所は各ルーターのパケット転送規則に応じて変化し，それにより遷移関係が定まる．

それでは，以下の LTL 式を考えてみよう．

$$packet.dst\&110 = 100 \rightarrow F(location = B), \qquad (1)$$

ここで, 記号 & はビット毎の積演算を表す．この式は，もしパケットの *dst* が $10\star$ に照合するならば，いづれ B に到達することを表している．有界モデル検査を行うことで，例えば以下の反例を得る．ここで，記号 – は値が変化しなかったことを表し，時刻 3 の状態はその後無限に繰り返される．

時刻	0	1	2	3
packet.dst	101	–	–	–
packet.src	101	–	–	–
location	A	R_1	R_3	D

この反例から *location* の値の時間変化を観察して，LTL 式 1 が実際満たされないことが分かる．この例は単純だが，実際の問題はずっと大きく複雑である．そのような場合に，たった 1 つの反例だけで満足できないことがある．例えば，*packet.dst* や *packet.src* のどんな値が LTL 式が満たされないことに関係しているのかを知りたい．何ら特徴のない値を任意に 1 つ選ぶことはモデル検査器のユーザに有用な情報をもたらさない．ネットワークの場合には，ひとかたまりのパケットが同一の経路で転送されて，最終的に反例になるということが起こりうる．これは連続するアドレス範囲に対する設定誤りに起因するかもしれない．

このような誤りに気づくために，本研究では単一の反例

から複数の反例に注意を振り向けたい．我々は以下の方法で連続するアドレスからなる最長範囲を計算する．

(1) 通常のモデル検査を実行し，1 つの反例を求める．これを**基準反例**と呼ぶ．
(2) 連続するアドレスを計算したい変数 (例えば，時刻 0 の $packet.src$) を選ぶ．これを**目標変数**と呼ぶ．
(3) 指定時刻かそれ以降の目標変数の値を除いて[*1]，基準反例と同一の値を持つすべての反例を生成する．
(4) 生成された反例の中で，指定時刻の目標変数 (すなわち，時刻 0 の $packet.src$) の値が区間を形成し，そのようなものの中で最長となる反例集合を計算する．

計算結果は数多くの反例からなるかもしれないが，それらは区間を形成するので，その区間の両端だけによって表現することができる．この解析により，1 つの反例よりもずっと抽象度の高い情報が簡潔に得られるのである．本稿では最長区間の計算に焦点を絞るが，本稿で与えられる計算の枠組みは最長区間計算だけに限定されない．容易に追加の解析処理を組み込める汎用的な枠組みとなっている．

4. 定式化

本節では，反例空間の概念，その上の構造，そして最長区間問題を定式化する．

制約充足問題 (CSP) は，三組 $(X, \mathcal{D}, \mathcal{C})$ であり，以下のように定義される．X は変数の有限集合である．$\mathcal{D}$ は変数の**ドメイン**の集まりであり，変数 x のドメイン ($D(x)$ と書く) は，x が値を取る有限集合である．$\mathcal{C}$ は X の上の制約 (すなわち，関係) の集合である．

有界モデル検査において与えられた仕様に対する反例を探索する問題 (すなわち，$\pi \not\models_k f$ を満たす π の探索) は，CSP として定式化されうる．等価な定式化は多く存在するが，本論文はそれらの違いを考えない．等価な定式化のうち任意に 1 つを選んで固定し，それを以降 B_{CSP} と表記する．論理式としての定式化の詳細については，[1] を参照されたい．文脈から明らかであるか，明示的に述べる必要がない場合は，単に B_{CSP} とだけ書き，それが具体的にどの問題から符号化されたものであるかを明白に述べない．

前節のネットワークの例を用いて，B_{CSP} を手短に説明する．k をモデル検査のバウンド (最大の遷移回数) とせよ．各時刻 $i \in \{0, \ldots, k\}$ に対して，B_{CSP} の変数 dst_i, src_i, loc_i がある．ここで，loc_i は時刻 i の変数 $location$ を表す．各変数のドメインはすでに明らかだろう．ルーティングテーブルの最初の行の規則は，以下のように記述される．

$$loc_i = R_1 \wedge dst_i = 10\star \wedge src_i = 01\star \rightarrow loc_{i+1} = R_2$$

このように記述される論理式を用いて，遷移関係を得ることができるので，その関係を k 回展開する．さらに，LTL 式 1 も符号化される必要があるが，詳細は省く．

B_{CSP} の変数は，上の例において変数記号の添数で示されるように，時刻の属性を持つことに注意されたい．本稿では, 時刻の属性を持たない変数を**時刻なし変数**と呼び，B_{CSP} の変数は時刻なし変数と区別したいとき**時刻あり変数**と呼ぶことがある．例えば，loc_i と loc_j は $i \neq j$ のとき時刻あり変数としては異なるが，時刻なし変数としては同一と考える．

B_{CSP} の**解**は，B_{CSP} のすべての変数に対する値割当であり，B_{CSP} のすべての制約が充足されるものを意味する．$\pi(w)$ によって，解 π において変数 w に割り当てられた値を表す．この記法は，必ずしも B_{CSP} の解ではない値割当に対しても用いられる．解は変数ドメインからとられた値のなすベクトルとみなされる．あるいは，幾何学の観点からは，$\prod D(v)$ における点ともみなされる．ここで，v は B_{CSP} のすべての変数の上を動き，変数は任意の順番で順序付けられているとする．構成により，反例は B_{CSP} の解と 1 対 1 に対応する．したがって，反例の集合は，$\prod D(v)$ の部分集合と同一視される．いくつかの変数ドメインは順序関係や隣接関係のような変数に固有の関係を持つので，反例の集合には，それらのドメインに相対的な関係が誘導され，一種の幾何学空間をなすと見ることができる．

定義 1. w を B_{CSP} の変数とする．$D(w)$ を w のドメインとし，$D(w)$ 上には二項関係 R が定義されているとする．このとき w **に相対的な** $\prod D(v)$ **における二項関係** R_w は，以下のように定義される．任意の $c, d \in \prod D(v)$ に対して $(c, d) \in R_w$ が成立するのは，以下の 2 条件が成り立つときである．

- $(c(w), d(w)) \in R$ が成立する．
- B_{CSP} の変数 u で，u は時刻なし変数として w と異なるか，u の時刻は w の時刻より小さいものに対して，$c(u) = d(u)$ が成立する．

この定義により，反例の集合には $\prod D(v)$ の部分集合として構造が定まるので，これを**反例空間**と呼ぶ．

本稿は以降，整数ドメインの上の順序関係 $\leq$ に焦点を絞るが，上の定義は他の構造に関する様々な計算を考えることができるように，順序関係の場合だけに限定されない一般的な形で定式化を与えている．

整数変数 w に対して，$c \in \prod D(v)$ は w に関して $d \in \prod D(v)$ の**直後の元**とは，$d \leq_w c$ かつ $c(w)$ は $D(w)$ において $d(w)$ の直後の元となるときをいう．反例の列 $c_1, \ldots, c_n$ が w **に関して区間**であるとは，すべての $i \in \{1, \ldots, n-1\}$ に対して c_{i+1} が c_i の (w に関する) 直後の元のときをいう．区間の**長さ**とはそれに含まれる反例の個数である．

一般に，$\leq_w$ は半順序であり，最長区間は一意に定まらない．しかしながら，本研究では w に割り当てられた値に

[*1] 指定された時刻 (ここでは 0) だけでなくそれ以降の時刻も除外している．指定された時刻の値が変化することで，多くの場合，それ以降の時刻の値も変化すると考えられるからである．

実質的に関心があるだけなので，最長区間がひとつ求まれば十分であり，一意に定まらないことは重要でない．

定義 2. 最長区間問題とは，整数変数 w と B_{CSP} の解 c が与えられるとき，B_{CSP} において w に関する (反例のなす) 区間 I であり，すべての $d \in I$ に対して $d \leq_w c$ または $c \leq_w d$ を満たす長さが最大の区間 I を求める問題である．

$d \leq_w c$ または $c \leq_w d$ という条件は定義 1 の 2 番目の条件が成立することを単に意味しているにすぎない．

5. 主要アルゴリズム

本節では，最長区間問題に対して単純な近似アルゴリズムおよび手の込んだ厳密アルゴリズムを与える．

単純な方のアルゴリズムは，アルゴリズム 1 において疑似コードが与えられているように，与えられた解を含む最長区間を線形探索で求める．これは近似アルゴリズムである．なぜなら $c \leq_w d$ または $d \leq_w c$ という条件は c が最長区間に含まれることを必要としないからである．アルゴリズム 1 は，現在の候補 r の隣の解 d の存在を確認しながら，区間の右端を探索する．解の存在確認は，B_{CSP} の制約と d が r の直後の元であるための制約の積により得られる SAT 問題を解くことによってなされうる．前述の通り，d は r に対して一意には定まらないので，得られた SAT 問題は自明に解かれ得ないが，現代の高速 SAT ソルバーを用いることで実用上素早く計算できることが期待される．ただ，SAT ソルバーの呼び出し回数は最悪時において w のビット幅に関して指数的に増加する．

アルゴリズム 1 変数 w に関して解 c を始点とする右端探索

```
r ← c;
while B_CSP の解 d が存在して w に関して r の直後の元となる.
do
  r ← d;
end while
return r;
```

左端探索も同様の手続きで計算できる．これら 2 つの探索を実行することで，与えられた解を含む最長区間の左端と右端を計算できる．

さて，これから最長区間問題に対する厳密アルゴリズムに移りたい．反例空間解析の枠組みでは反例生成機能が提供されていて，生成された反例は BDD として効率的に表現される．したがって，厳密アルゴリズムは入力として BDD を受け取る．この BDD は目標変数 w に関して基準反例 c と比較可能なすべての解を表現する．そして，アルゴリズムはこれらの解の中から最長区間を計算する．ここで，簡単のため w は符号なし整数を表現する変数と仮定する．

効率的な計算のため，B_{CSP} の変数の論理変数への符号化と変数順序を以下のように決める．まず B_{CSP} の変数は対数符号化で変換する．すなわち，v の二進表現において各ビットに 1 つの論理変数を対応付ける．議論を明確にするために，以降では $bits(v)$ により，B_{CSP} の変数 v から符号化されて得られる論理変数の集合を表す．そのような論理変数は BDD の変数ともみなされることに注意されたい．そして，論理変数 x に対して，$var(x)$ により，$x \in bits(v)$ を満たす B_{CSP} の変数 v を表す．さらに，$time(x)$ により，$var(x)$ の時刻を表す．BDD の変数は以下の条件を満たす順番で順序付けられている：BDD 変数 x, y に対して $x < y$ が成立するのは，(1) $time(x) < time(y)$ が成立するか、または (2) $var(x) = var(y) = w$ が成立して，x の対応するビットは y の対応するビットよりも重要である (二進表現で位がより高い) ときである．

アルゴリズム 2 の前に，$bits(w)$ 上に射影することで BDD を単純化する．$bits(w)$ 以外の変数への値割当は最長区間計算に無関係なので，そのような変数に対応する BDD 節点をすべて削除しても問題ない．上で決めた変数順序のおかげで，これを線形時間で行うことができるが，紙面の制約のためその詳細は省略する．一般性を失うことなく，単純化された BDD における変数インデックスは 1 から $|bits(w)|$ の範囲の値をとると仮定できる．

アルゴリズム 2 において用いられる記法をここでまとめる．各非終端 BDD 節点 f に対して，その変数インデックスは $f.idx$ で表され，LO 子節点と HI 子節点はそれぞれ $f.lo$, $f.hi$ で表される．計算の効率性のため，f はさらに 2 つの補助フィールド $f.done, f.dir$ を持つ．ここで，$f.done$ は真偽値を保持し，最初は偽で初期化される．このフィールドは f に対する処理が終わったか否かを表している．一方，$f.dir$ は方向ラベル L, R, M のいづれか 1 つを保持する．このフィールドは f のどちらの子節点に，あるいは両方の子節点をまたいで，現在の最長区間が属しているかを表している．BDD f と方向ラベル dir が与えられるとき，関数 FindForkedPath は f から $\top$ へ向けて BDD を下りながら $bits(w)$ への値割当の対を計算する．この計算結果は区間を表している．

アルゴリズム 2 B_{CSP} の解を表す単純化された BDD f が与えられたときの最長区間計算

```
function LongestInterval(f)
if f は終端節点であるか f.done = true が成立する. then
  return;
end if
LongestInterval(f.lo);
LongestInterval(f.hi);
I_L ← FindForkedPath(f, L);
I_R ← FindForkedPath(f, R);
I_M ← FindForkedPath(f, M);
I ← I_L, I_R, I_M のうち最長のもの;
f.dir ← I が属する方向を表すラベル;
f.done ← true;
end function
```

アルゴリズム 2 の基本的なアイディアを説明する．まず，2 つの符号なし整数 a, b $(a < b)$ に対して区間 $[a, b]$ を表現する BDD のグラフ構造について観察してみたい．この BDD では，a と b の二進表現において異なるビットのうちもっとも重要なビット (もっとも位が高い桁) において，根節点から始まるパスは 2 つに分岐し，その後，同一の接尾辞の最初のビット (あるいは $\top$) において合流する．区間 $[a, b]$ には一般に多くの整数が含まれるにもかかわらず，$bits(w)$ の変数順序と BDD の節点削除規則により，この BDD 上にはこのパス上に現れる節点以外に節点は存在しない．本稿では，この BDD のようなグラフ構造を**分岐パス**と呼び，分岐パスから 2 つに分岐するパスのうち，最小の整数に対応する分岐を**左分岐**，最大の整数に対応する分岐を**右分岐**と呼ぶ．明らかに，分岐パスが連続する整数 (区間) を表現するための必要十分条件は，左分岐のすべての HI 枝は $\top$ か非終端節点を指し，右分岐のすべての LO 枝は $\top$ か非終端節点を指すことである．以降，単に分岐パスというときこの条件を満たすとする．

アルゴリズム 2 の再帰の各ステップにおいて計算される I_L, I_R は分岐パスであり，I_L と I_R はそれぞれ $f.lo$ と $f.hi$ を根節点とする BDD において最大の幅を持つ分岐パスである．ここで，分岐パスの幅は，分岐パスによって表現される区間の長さを意味する．I_M もまた分岐パスであり，左分岐が $f.lo$ を根節点とする BDD に属し，右分岐が $f.hi$ を根節点とする BDD に属する (これを f の両方の子供にまたがると呼ぶ).

I_L の計算は自明な場合か $f.lo$ のある孫節点に対する I_M の計算に帰着されうる．I_R の計算も同じようになされうるので，I_R の場合の説明は省く．I_L の自明な場合は以下の通りである：関数 FindForkedPath(f, L) は，もし $f.lo$ が $\perp$ ならば Nil を返す (空の区間を意味する)；もし $f.lo$ が $\top$ ならば，区間を表す値割当の対であり，左端の値割当は，$i = f.idx$ とするとき，i 番およびそれ以降のすべての変数に 0 が割り当てられたものであり，右端の値割当は，i 番の変数に 0，それ以降のすべての変数に 1 が割り当てられたものである．それでは，I_M に帰着される場合は以下の通りである：関数 FindForkedPath は，最初，引数として与えられた方向ラベル L によって指示された通り，$f.lo$ に移動する；その後，現在の節点の dir フィールドによって指示された方の f の子節点に移動することを，dir が M となるか自明な場合が生じるまで繰り返す．

アルゴリズム 3 は I_M の左分岐を計算する．簡単のため，自明な場合 (開始節点の LO 枝が終端節点を指す場合) を除外している．アルゴリズムは，左分岐が前述の必要十分条件を満たすように，開始節点 $curr$ から BDD を下りながら，配列 $assign$ によって表現される変数への値割当を埋めている．その各ステップにおいて，関数 $Fill$ を呼び出すことで値割当の一部を埋めている．具体的には，

アルゴリズム 3 $curr$ を開始節点とした場合の I_M の左分岐の計算．ここで，$curr.lo, curr.hi$ は非終端節点とする．

```
配列 assign をすべての要素を −1 で初期化する;
Fill(assign, curr, L, 1);
prev ← curr;
curr ← curr.lo;
while (1) curr.hi は非終端節点，または (2) curr.hi = ⊤ が成立し，curr.lo は非終端節点． do
  if (1) が成立する． then
    Fill(assign, curr, R, 1);
    curr ← curr.hi;
  else if (2) が成立する． then
    Fill(assign, curr, L, 1);
    prev ← curr;
    curr ← curr.lo;
  end if
end while
if (3) curr.hi = ⊤ かつ curr.lo = ⊥ then
  Fill(assign, curr, R, 0);
else if (4) curr.hi = ⊥ then
  if prev.hi = ⊤ then
    Fill(assign, prev, R, 0);
  else
    return Nil;
  end if
end if
return assign;
```

配列 $assign$，BDD g，方向ラベル $dir \in \{L, R\}$，真偽値 $value \in \{0, 1\}$ が与えられるとき，関数 $Fill$ はもし $dir = L$ ならば $assign[f.idx]$ に 0 を代入し，$dir = R$ ならば 1 を代入する；さらに，f と dir によって指示される子節点との間で削除された各節点 g に対して，関数 $Fill$ は $assign[g.idx]$ に $value$ を代入する．

アルゴリズム 3 の while 文では，開始節点の LO 枝によって指された BDD に含まれる左分岐を探索しようとしている．このループの各ステップにおいて，疑似コードに示されているように，$curr$ の子節点に関して (1) から (4) の 4 通りの場合だけが存在する．(1) と (2) は非終端節点に下る必要があることを意味する．(3) は左分岐を発見したことを意味する．(4) は $prev$ にバックトラックする必要があることを意味する．なぜならもしバックトラックしなかったら，左分岐の満たすべき条件が破綻するからである．バックトラックした後，$prev$ の HI 枝に沿って進む．もし (2) がこのループにおいて一度でも成立するならば，$prev.hi$ は $\top$ にならなければならない．そのときは左分岐が発見されたことを意味する．もし一度も (2) が成立しなかったならば，$prev$ はこのアルゴリズムの開始節点でなければならない．この場合は開始節点の LO 枝側の BDD に左分岐が存在しないことを意味し，そのことを表す Nil を返す．

I_M の右分岐はアルゴリズム 3 と同様の方法で計算され，左分岐と右分岐を組み合わえることで I_M を得ることがで

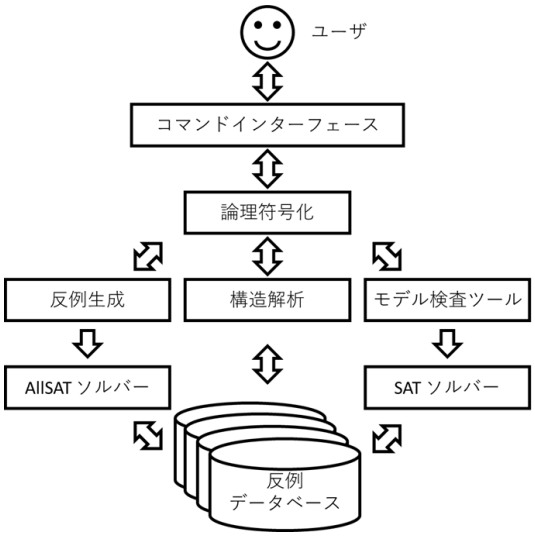


図 2 システム構成

きる．ここで，開始節点の一方の分岐が存在しないとき，たとえもう一方の分岐が存在したとしても，I_M は空の区間であることに注意されたい．また I_M が空であったとしても，開始節点が非終端節点ならば $\top$ に至るパスが存在するので I_L または I_R は非空となるので，アルゴリズム 2 の I は非空である．

定理 1. w を符号なし整数のための B_{CSP} の変数とする．f を B_{CSP} の解集合を表す BDD とする．このとき，f から w に関する最長区間は，$O(nm)$ 時間で計算できる．この計算は各 BDD 節点につき 2 つの追加のフィールドと $O(n)$ サイズの作業領域で可能である．

Proof. f の単純化は $O(m)$ 時間である．単純化された BDD を f' と表す．もし f' が $\top$ ならば最長区間は $[0, 2^{|bits(w)|} - 1]$ である．もし f' が $\bot$ ならば空である．もし f' が非終端節点ならば，関数 LongestInterval を入力 f' で計算する．これは $O(nm)$ 時間でできる．なぜなら，関数 FindForkedPath の計算と区間の比較は $O(n)$ 時間でできるからである．その後，関数 FindForkedPath を入力 $f', f'.dir$ で計算して，その結果として最長区間を得る．全体の計算時間は $O(nm)$ 時間である．必要な領域サイズはアルゴリズムから明らかであろう． □

6. 計算の枠組み

本節で，反例空間解析の計算の枠組みを与える．図 2 にこの枠組みの主要構成要素を示す．本研究では NuSMV version 2.5.4 [2] を拡張することで，この枠組みを実装した．

対話型コマンドインターフェース．ユーザはコマンドライン・インターフェースを通して反例空間解析システムと対話する．ユーザはシステムに対してコマンドを発行し，システムはその計算結果を返す．それを注意深く観察することで，ユーザは (モデルの誤りに関する) 仮説を立て，それを検証するために新たなコマンドを発行する．その結果に応じて，ユーザは仮説が誤りであると分かったり，仮説に対する確信をより強めたりするかもしれない．このサイクルを繰り返すことで，ユーザは現象をよりよく理解し，もしかしたら誤りの核心に近づけるかもしれない．

論理符号化．様々なソルバーやデータ構造 (例えば，SAT ソルバー，AllSAT ソルバー，BDD) を活用できるように，(何らかのモデル化言語で記述された) クリプキ構造や LTL 式は命題論理式の標準形 CNF に変換される．特に，B_{CSP} の変数は論理変数に符号化され，この符号化は一般に 1 対多対応の関係となる．

反例生成．非有界の設定ですべての反例を計算するのは 1 つの手であるが，計算するには困難すぎる．したがって，反例生成部は代替的な方法として**制約付き反例生成**および**有界反例生成**を提供する．ネットワークの設定におけるより効率的な反例生成として [12] を参照せよ．

制約付き反例生成では，追加の制約の下で特定のバウンドにおいてすべての反例を生成する．これは 3 節で述べたような状況において活用できる：目的変数の値を除いたすべての値が基準反例と一致する反例を生成する計算である．他方で，有界反例生成は，有界モデル検査のように，バウンドで探索の範囲を限定し，その範囲内だけですべての反例を生成する．すなわち，反例が存在する最小のレベル i を充足可能性判定を繰り返すことで求める．そして，そのレベルにおける論理式の充足解をすべて生成する．

これらの反例生成は AllSAT ソルバーを用いて実現されうる．AllSAT ソルバーとは，与えられた命題論理式の充足可能な変数値割当てをすべて計算するソフトウェアである．特に，最長区間問題の厳密アルゴリズムに対して，AllSAT ソルバーの 1 種である *BDD* **ソルバー**を用いる．BDD ソルバーは現代の SAT ソルバーが基礎とする探索アルゴリズム *CDCL* に対して BDD を統合したものである．最も顕著な特徴は以下の通りである．BDD ソルバーは BDD を，動的計画法において部分問題の解を表現する表のような役割として用いて，過去に解いた等価な部分問題をスキップすることで探索を高速化する [10] [11] [17]．BDD ソルバーは解の集合を BDD として出力する．一般に他の AllAT ソルバーは少なくとも解の個数に依存した計算時間が必要であるが，BDD ソルバーはこの課題を克服する潜在的な能力を有している．このことは過去の研究 [17] において実施された大規模な評価実験により確認されている．

反例データベース．複数の反例データベースからなり，個々のデータベースは特定の B_{CSP} 問題例に対する反例空間を表現する．各反例データベース上で利用可能ないくつかの基本操作も提供される．例えば，所属検査やランダムサンプリングである．

構造解析．5 節で詳細に述べられた最長区間計算やその他の解析処理機能が提供される．

7. まとめ

本稿では，モデル検査における誤りの効果的な説明を目的として，反例集合の構造的な特徴について研究した．数学的な扱いを与えるために，反例空間の概念やその上の構造，そして最長区間問題を定式化した．さらに，そのような構造解析処理を実現するための汎用的な計算の枠組みを提案した．本研究のとった方法は，前処理として反例生成を行い，そのような反例を BDD として表現することに基づいている．反例の完全な生成は一般に困難なので，代替的な手法として制約付き反例生成と有界反例生成を与えた．制約付き反例生成は，既存の有界モデル検査ツールを用いて反例を求め，それに対して適切な追加制約を課すものである．一方，有界反例生成は，有界モデル検査のように，バウンドで探索の範囲を限定し，その範囲内ですべての反例を生成する．最長区間問題に対して，厳密アルゴリズムを与えた．そのアルゴリズムは連続整数の特徴をとらえるために BDD のグラフ構造を最大限活用している．これにより，解集合が BDD として与えられるとき最長区間問題を $O(nm)$ 時間で計算できることを示した．ここで，n は論理変数の個数，m は BDD サイズ (節点の個数) である．本稿ではさらに，ネットワーク検証への応用として，最長区間計算をパケット範囲の計算に適用できることを示した．高度なデータ構造 (例えば，ZDD [14]，DNNF [5]，SDD [6]，ZSDD [15]) を活用して他の解析処理を考察することは今後の研究課題として興味深い．

参考文献

[1] Biere, A., Cimatti, A., Clarke, E. M. and Zhu, Y.: Symbolic Model Checking Without BDDs, *Proceedings of the 5th International Conference on Tools and Algorithms for Construction and Analysis of Systems*, TACAS '99, Berlin, Heidelberg, Springer-Verlag, pp. 193–207 (online), available from ⟨http://dl.acm.org/citation.cfm?id=646483.691738⟩ (1999).

[2] Cimatti, A., Clarke, E., Giunchiglia, E., Giunchiglia, F., Pistore, M., Roveri, M., Sebastiani, R. and Tacchella, A.: NuSMV 2: An OpenSource Tool for Symbolic Model Checking, *Computer Aided Verification* (Brinksma, E. and Larsen, K. G., eds.), Berlin, Heidelberg, Springer Berlin Heidelberg, pp. 359–364 (2002).

[3] Clarke, E. M., Emerson, E. A. and Sifakis, J.: Model Checking: Algorithmic Verification and Debugging, *Commun. ACM*, Vol. 52, No. 11, pp. 74–84 (online), DOI: 10.1145/1592761.1592781 (2009).

[4] Clarke, Jr., E. M., Grumberg, O. and Peled, D. A.: *Model Checking*, MIT Press, Cambridge, MA, USA (1999).

[5] Darwiche, A.: Decomposable Negation Normal Form, *J. ACM*, Vol. 48, No. 4, pp. 608–647 (online), DOI: 10.1145/502090.502091 (2001).

[6] Darwiche, A.: SDD: A New Canonical Representation of Propositional Knowledge Bases, *Proceedings of the Twenty-Second International Joint Conference on Artificial Intelligence - Volume Volume Two*, IJCAI'11, AAAI Press, pp. 819–826 (online), DOI: 10.5591/978-1-57735-516-8/IJCAI11-143 (2011).

[7] Groce, A.: Error Explanation with Distance Metrics, *Tools and Algorithms for the Construction and Analysis of Systems* (Jensen, K. and Podelski, A., eds.), Berlin, Heidelberg, Springer Berlin Heidelberg, pp. 108–122 (2004).

[8] Groce, A. and Kroening, D.: Making the Most of BMC Counterexamples, *Electronic Notes in Theoretical Computer Science*, Vol. 119, No. 2, pp. 67 – 81 (online), DOI: https://doi.org/10.1016/j.entcs.2004.12.023 (2005).

[9] Groce, A. and Visser, W.: What Went Wrong: Explaining Counterexamples, *Model Checking Software* (Ball, T. and Rajamani, S. K., eds.), Berlin, Heidelberg, Springer Berlin Heidelberg, pp. 121–136 (2003).

[10] Huang, J. and Darwiche, A.: Using DPLL for Efficient OBDD Construction, *Theory and Applications of Satisfiability Testing* (Hoos, H. H. and Mitchell, D. G., eds.), Berlin, Heidelberg, Springer Berlin Heidelberg, pp. 157–172 (2005).

[11] Huang, J. and Darwiche, A.: The Language of Search, *J. Artif. Int. Res.*, Vol. 29, No. 1, pp. 191–219 (online), available from ⟨http://dl.acm.org/citation.cfm?id=1622606.1622613⟩ (2007).

[12] Inoue, T., Chen, R., Mano, T., Mizutani, K., Nagata, H. and Akashi, O.: An Efficient Framework for Data-Plane Verification With Geometric Windowing Queries, *IEEE Transactions on Network and Service Management*, Vol. 14, No. 4, pp. 1113–1127 (online), DOI: 10.1109/TNSM.2017.2723725 (2017).

[13] Lopes, N. P., Bjørner, N., Godefroid, P., Jayaraman, K. and Varghese, G.: Checking Beliefs in Dynamic Networks, *Proceedings of the 12th USENIX Conference on Networked Systems Design and Implementation*, NSDI'15, Berkeley, CA, USA, USENIX Association, pp. 499–512 (online), available from ⟨http://dl.acm.org/citation.cfm?id=2789770.2789805⟩ (2015).

[14] Minato, S.: Zero-Suppressed BDDs for Set Manipulation in Combinatorial Problems, *30th ACM/IEEE Design Automation Conference*, pp. 272–277 (online), DOI: 10.1145/157485.164890 (1993).

[15] Nishino, M., Yasuda, N., Minato, S.-i. and Nagata, M.: Zero-suppressed Sentential Decision Diagrams, *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence*, AAAI'16, AAAI Press, pp. 1058–1066 (online), available from ⟨http://dl.acm.org/citation.cfm?id=3015812.3015969⟩ (2016).

[16] Qadir, J. and Hasan, O.: Applying Formal Methods to Networking: Theory, Techniques, and Applications, *IEEE Communications Surveys Tutorials*, Vol. 17, No. 1, pp. 256–291 (online), DOI: 10.1109/COMST.2014.2345792 (2015).

[17] Toda, T. and Soh, T.: Implementing Efficient All Solutions SAT Solvers, *J. Exp. Algorithmics*, Vol. 21, pp. 1.12:1–1.12:44 (online), DOI: 10.1145/2975585 (2016).

[18] van den Berg, L., Strooper, P. and Johnston, W.: An Automated Approach for the Interpretation of Counter-Examples, *Electronic Notes in Theoretical Computer Science*, Vol. 174, No. 4, pp. 19 – 35 (online), DOI: https://doi.org/10.1016/j.entcs.2006.12.027 (2007).