情報処理学会論文誌 Vol.54 No.3 1165–1184 (Mar. 2013)

照合操作の識別のためのOCL解析方法

井上 拓[1,a)] 本位田 真一[2,3]

受付日 2012年2月14日, 採録日 2012年12月7日

概要：情報システムのコンポーネントベース開発において，データの一貫性を保つために，照合操作が利用される．操作の仕様は，宣言型の記述言語である OCL（Object Constraint Language）を用いて記述されるが，照合の表現が暗黙的であるため，人間が誤って解釈する恐れがある．本論文では，OCL 仕様記述を静的に解析し，照合操作を識別する手法を提案する．この手法は，OCL の意味論に従う近似的な解釈（抽象解釈）を行う点が特徴である．本手法を用いて，照合操作の仕様の解釈の誤りを発見・除去することが可能になる．このことによって，データの一貫性が保たれた安全な情報システムを構築するための足がかりが得られる．

キーワード：コンポーネント，照合操作，OCL，抽象解釈

An OCL Analysis Method for Identifying Collation Operations

Taku Inoue[1,a)] Shinichi Honiden[2,3]

Received: February 14, 2012, Accepted: December 7, 2012

Abstract: In component-based IS (information systems), collation operation is used for managing data consistency, and collations are specified implicitly with OCL (Object Constraint Language) in a declarative manner, which may cause human error in understanding their definitions. This paper presents an OCL analysis method for identifying collations, which employs an abstract interpretation in order to gain the meaning of description statically. With the method we can detect the human errors and correct them, which gives us a foothold to construct the consistent IS.

Keywords: component, collation operation, OCL, abstract interpretation

1. はじめに

情報システムの開発において，Unified Modeling Language（UML）を用いたコンポーネントベース開発方法論である Catalysis [1] や UML Components [2] が数多く適用されている．コンポーネントベース開発において関心の分離は重要な概念であるが，これらの開発方法論では，情報システムの主要な関心事であるデータの管理とビジネスゴールの実現を，ビジネスコンポーネントとシステムコンポーネントの 2 種類のソフトウェア部品に集約して，システムを構築する．

ビジネスコンポーネントはシステムが扱うデータを永続化・管理する責務を持ち，データにアクセスするための操作を提供する．操作の振舞いの仕様は，宣言型の仕様記述言語である Object Constraint Language（OCL）を用いて厳密にモデル化される．一方システムコンポーネントは，ビジネスコンポーネントのデータにアクセスし，ビジネスゴールを実現するためのシステム機能をサービスとして提供する．サービスの振舞いの仕様は，UML の Activity 図を用いてモデル化される．

一般的な情報システムには，データオブジェクトの参照制約 [3] や，ライフサイクルの整合性 [4] 等の，データの整合性制約が定義される．参照制約はデータの一貫性を保証する重要な制約であり，ときにビジネスコンポーネント間

[1] キヤノン株式会社
Canon Inc., Ota, Tokyo 146–8501, Japan
[2] 東京大学
The University of Tokyo, Bunkyo, Tokyo 113–8656, Japan
[3] 国立情報学研究所
National Institute of Informatics, Chiyoda, Tokyo 101–8430, Japan
[a)] taku.inoue@gmail.com

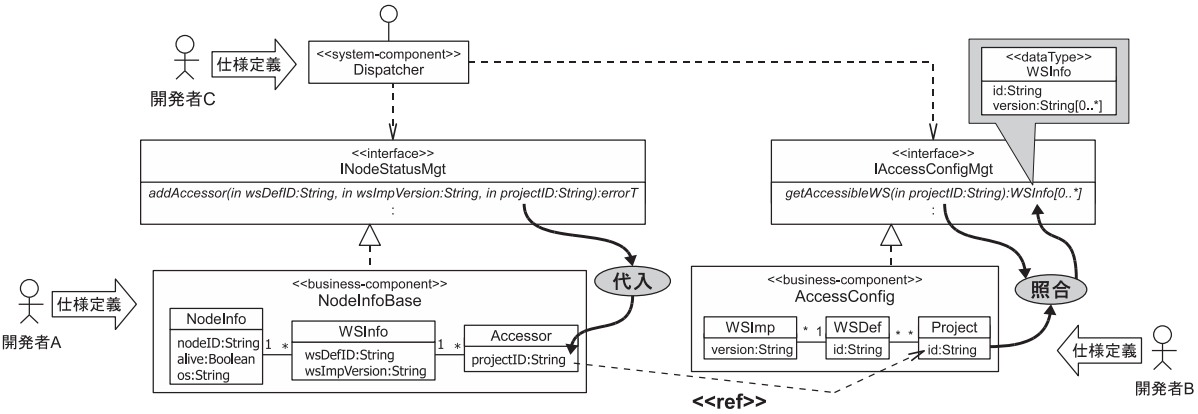

図 1 2 種類のコンポーネントとデータの依存関係
Fig. 1 Two types of component and data reference.

にまたがって定義される．多くの場合，コンポーネント間にまたがる参照制約は，ビジネスコンポーネントとシステムコンポーネントの振舞いの組合せによって実現される．その際に，ビジネスコンポーネントの次の 2 種類の操作を適切に利用することが必要となる．

代入操作：入力パラメータ値を，永続データの属性に代入する操作．

照合操作：入力パラメータ値を属性に持つデータの存在を照合し，出力パラメータの値によって照合結果を伝える操作．

図 1 の例を用いて説明する．図中の ⟨⟨ref⟩⟩ は，ビジネスコンポーネント `NodeInfoBase`，`AccessConfig` が管理するデータの属性 `Accessor.projectID` と `Project.id` の間の参照制約を示している．システムコンポーネント `Dispatcher` のあるサービス `X` が `NodeInfoBase` の代入操作 `addAccessor` を呼び出すケースを考える．代入操作 `addAccessor` は，引数 `projectID` の値を特定の `Accessor` オブジェクトの属性 `projectID` に代入する．そこで上記の参照制約を満たすために，サービス `X` の振舞い仕様には，`addAccessor` の呼び出しに先立ち，`AccessConfig` の照合操作 `getAccessibleWS` を呼び出して，引数 `projectID` と同じ値を属性 `id` に持つ `Project` オブジェクトの存在を照合する手順が定義される．

規定された参照制約を，ビジネスコンポーネントとシステムコンポーネントの振舞いによって実現するうえで，次の 2 つの課題が存在する．

- 開発者 A，B が，代入操作と照合操作の厳密な振舞い仕様（OCL）を記述し，開発者 C がそれらの記述内容を正確に理解する．
- 開発者 C が，代入操作と照合操作を適切な手順で利用する，サービスの振舞い仕様（Activity 図）を記述する．

後者の課題を扱うためには，まず前者のビジネスコンポーネントの操作仕様の厳密な取扱いを実現することが前提となる．そこで本論文では，参照制約を実現するための足がかりとして，前者の課題の解決に取り組む．

宣言的な仕様記述言語である OCL には，代入と照合を明示的に表現する演算子が存在しないため，それらを正確に識別することは容易ではない，という問題がある [5], [6]．筆者らは文献 [6] において，振舞い仕様（OCL）の抽象構文木を解析して代入が発生しうる OCL 表現ノードを識別する，データフロー解析手法を提案した．しかし文献 [6] の手法は代入の識別を目的としており，照合操作には適用することができない．そこで本論文では照合操作を対象として，

1. 照合が発生しうる OCL 表現ノード
2. 照合結果を出力する OCL 表現ノード

の組を識別する，OCL 解析手法を提案する．

振舞い仕様（OCL）は，オブジェクト状態に対する制約として記述されるが，制約を満たす状態は無限に存在しうる．そのため，とりうる状態を網羅的に扱うことができる静的解析手法が必要であり，その解析手法には，OCL の意味論が持つ文脈依存性・非決定性・不完全性 [6] の性質を正しく取り扱う能力が必要である（技術課題 1）．本手法では，文献 [6] で提案した抽象解釈技術を用いて，これらの性質を厳密に扱うことによって，この技術課題を解決する．また照合の出力と結果の対応を得るために，上記 1，2 の OCL 表現ノードの値の組を求める必要がある．この値の組は，抽象構文木中のノードの意味の解釈の組合せで決まるが，そのような組合せの数は，ノード数に対して指数関数的に増大する（技術課題 2）．本手法では，抽象解釈技術を応用して，1，2 のノード値の組を与える解釈の組合せの有無を求めることにより，計算を効率化し，この技術課題を解決する．

本論文では，提案手法の説明に続いて，手法の適用を支援するツールと，実システムを用いて行った評価実験について述べる．そして，提案する解析方法が，上記 2 つの技

術課題を解決するための厳密性と効率性を備えていることを示し，手法の有用性と適用範囲について議論する．

本手法は，ビジネスコンポーネントの振舞い仕様（OCL）に記述された照合操作の仕様が，記述者や利用者の意図や理解と合致しているか確認する目的で用いることを想定している．本手法の適用により，照合操作の仕様を正確に記述・理解することが可能になる．

本論文の構成は以下のとおりである．2 章で本手法が扱うモデルを説明し，3 章で静的解析の要件と技術課題を述べる．4 章で手法の概要を述べ，5 章で手法の解析方法を説明する．6 章で支援ツールについて述べ，7 章で評価実験を説明する．8 章で解析方法の妥当性，効率性と，手法の適用範囲，有用性を議論する．9 章で関連研究について述べ，10 章で本論文をまとめる．

2. ビジネスコンポーネントの仕様モデル

本章では手法が扱うビジネスコンポーネントの仕様モデルを説明する．モデルは，ビジネスコンポーネントの仕様記述に関する研究 [7] を基にしており，構造モデルと振舞いモデルから構成される．

2.1 構造モデル

構造モデルは，ビジネスコンポーネントの静的な側面を UML のクラス図で記述したモデルであり，データモデルとインタフェース記述からなる．図 1 は実問題の構造モデルの一例である．

データモデルはビジネスコンポーネントが扱うデータを *Class* として，データの関係を順序なしの関連として表現する．各 *Class* は属性定義を持つが，操作を持たない．ビジネスコンポーネント内のデータ間の制約は OCL の不変条件として，コンポーネント間にまたがる参照制約は ⟨⟨ref⟩⟩ を付加した依存として記述される．

インタフェース記述は，クラス図の *Interface* として表現され，ビジネスコンポーネントが提供する操作のシグニチャとパラメータの型が定義される．操作パラメータの型は *DataType*，*PrimitiveType*，*Enumeration* のいずれか，またはその順序なしの *Collection* であり，in/out はパラメータの方向を示す．利用者は操作の呼び出しを通じてデータにアクセスすることができるが，データは値渡しの操作パラメータにシリアライズされ，ビジネスコンポーネント内のオブジェクトを直接参照することはできない．これによりコンポーネント間の結合が疎に保たれる．

2.2 振舞いモデル

ビジネスコンポーネントの操作の振舞いを，OCL を用いて宣言的に記述したモデルであり，操作を呼び出すための前提条件（事前条件）と，操作を実行することによる効果（事後条件）からなる．事後条件には操作の内容が，オブジェクト，属性，リンク，操作パラメータの間に成り立つ論理式で記述される．本論文では，事後条件に記述されないモデル要素の状態は操作を通じて変化しない，というフレーム仮定 [5] の下に事後条件を解釈する．

```
1  context AccessConfig::getAccessibleWS
2   (in projectID:String):Set(WSInfo)
3  post: let wsDefs:Set(WSDef)=Project.allInstances()
4   ->select(i|i.id@pre=projectID).WSDef in
5   (wsDefs->isEmpty() implies result->isEmpty()) and
6   wsDefs->forAll(j|result->exists(k|
7   k.id=j.id@pre and k.version=j.WSImp.version@pre))
```

図 2 振舞いモデルの例
Fig. 2 An example of behavioral model.

照合操作 `AccessConfig::getAccessibleWS` の振舞いモデルの例を**図 2** に示す．図中の事後条件は，操作パラメータ `projectID` から特定される `Project` オブジェクトに紐付く各種のデータが，戻り値 `result` に含まれることを表現している．すなわち，集合 `result` が非空である場合に，`projectID` から特定される `Project` オブジェクトが存在することが分かる．

3. OCL 記述の静的解析

本章では，OCL 記述を静的に解析するうえでの要件と，解決すべき技術課題を説明する．

3.1 要件

本論文では，実問題において主流である，単一の出力パラメータの値によって照合結果を伝える照合操作を扱う．そのような照合操作の仕様は以下の情報からなる．

- 照合値情報：照合する値を指定する，入力パラメータもしくはその属性[*1]
- 照合先情報：照合先を特定する *Class* とその属性
- 照合出力情報：照合結果を指定する，出力パラメータもしくはその属性

静的解析の要件は，振舞いモデルからこれらの情報の組（*CbPA*: collation between an operation parameter and a class attribute）を求めることである．

CbPA は，操作内容を規定する情報として事後条件に記述されるはずである．またフレーム仮定より，事後条件には操作の実行によるすべての効果が記述されるため，照合操作の事後条件には CbPA が含まれるはずである．したがって，操作の事後条件を対象に静的解析を行えばよい．

構造モデルの不変条件によって，事後条件中の OCL 表現が恒真あるいは充足不能となる場合には，事後条件のみを対象とした解析から誤った CbPA が抽出される恐れがある．たとえば，事後条件 “`o implies (p and q)`” の解析

[*1] パラメータが構造を持つ場合に，その属性の値によって照合値あるいは照合結果が指定される可能性がある．

から，p と o をそれぞれ，照合とその結果を表す OCL 表現ノードとして識別したと仮定する．もし不変条件において q が充足不能ならば，事後条件は “o=false” となるため，上記の識別結果は不適切である．本論文では，有効な仕様モデルが備えるべき性質として，恒真あるいは充足不能な表現が事後条件に含まれないことを前提とする．

3.2 解決すべき技術課題

OCL の論理式は次の性質を有する．

文脈依存性：論理式を構成する個々の OCL 表現の内容の解釈は，その表現の文脈に依存する．表現 X が CbPA を含む場合に，“not X” は CbPA の不在を表す．

非決定性・不完全性：OCL は三値論理に基づいており，一般に論理式は非決定性と不完全性を有する．表現 “X or Y” の値が true の場合に，X と Y のいずれが true の値をとるか決定することはできない（非決定性）．また X の値が true である場合に Y の値を特定することができない（不完全性）．

したがって，OCL の論理式である操作の事後条件から CbPA を誤りなく抽出するためには，OCL 記述の形式だけではなく，これらの性質に沿って記述の意味を厳密に解釈する必要がある（技術課題 1）．

また，照合の出力と結果の対応を得るために，これらの値を保持する OCL 表現ノードの値の組を求める必要がある．この値の組は，抽象構文木の各ノードの解釈の組合せで決まるが，その組合せの数はノード数に対して指数関数的に増大する．実問題の典型的な事後条件のノード数は数十から数百であり，組合せを総当たりで計算することは困難であるため，効率的な計算方法が必要である（技術課題 2）．

4. 提案手法の概要

本章では提案手法の概要を示す．提案手法は構造モデルと振舞いモデルの事後条件を入力として，これらのモデルを解析して CbPA を抽出し，出力する．手法のプロセスは，照合値情報と照合先情報を求める Step 1 と，照合出力情報を求める Step 2 の 2 つのステップからなる（**図 3**）．

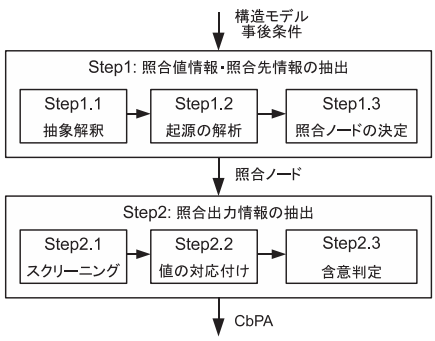

図 3 提案手法のプロセス
Fig. 3 Process of our method.

Step 1 で，OCL の意味論を厳密に扱う抽象解釈の方法を導入し，技術課題 1 を解決する．さらに Step 2 では，抽象解釈の技術を応用して，出力値と照合結果の値の組を与える各ノードの解釈の組合せの存在の有無を求めることにより，計算を効率化し，技術課題 2 を解決する．

4.1 Step 1：照合値情報・照合先情報の抽出

本手法では，入力パラメータと属性の間の照合を表す OCL 表現ノードを “照合ノード” と呼ぶ．照合ノードは以下の形式を持つ．

> **照合ノードの形式：** 演算子 “=” を用いた等価関係，または演算子 “includes” 或いは “includeAll” を用いた *Collection* の包含関係．

上記の形式で表現される照合ノードの，等価演算子あるいは包含演算子の左右のオペランドが，CbPA の照合値情報と照合先情報を表し，照合ノードの値が照合結果を表す．ただし両オペランドの順序は不同であり，それらの形式に関する仮定を設けない．Step 1 では，照合ノードを求めることにより，照合値情報と照合先情報を抽出する．

照合ノードの形式は，OCL 表現ノード E が照合を表すための形式的な制約である．実際にノード E が照合を表すための意味的な制約として，次の 3 つの条件を課する．

条件 1： E の値が true または false となる，事後条件を満たすオブジェクト状態が存在する．

条件 2： E の一方のオペランドに操作パラメータまたはその属性が，他方のオペランドにデータモデル中の *Class* の属性値が含まれる．

条件 3： 条件 2 の操作パラメータは入力パラメータであり，*Class* の属性値は操作実行前の値である．

条件 1 は照合が実際に行われる状態が存在するための条件であり，条件 2 は CbPA の定義に由来する．条件 3 は，フレーム仮定の下で，副作用をともなわない照合の記述が満たすべき条件である[*2]．

本手法は，これらの条件を満たす照合ノードを，次の 3 つのサブステップによって求める．各サブステップは，それぞれ条件 1–3 に対応している．

Step 1.1： 抽象解釈を行い，照合ノードの形式を持ち，条件 1 を満たす OCL 表現ノードを求める．

Step 1.2： Step 1.1 で求めた各 OCL 表現ノードの起源の解析を行い，条件 2 を満たすものを照合ノードの候補として選択する．各候補の左右のオペランドの起源に含まれる操作パラメータあるいはその属性と，*Class* の属性の組合せを，照合値情報と照合先情報として記録する．

Step 1.3： Step 1.2 で求めた候補から，条件 3 を満た

[*2] 本手法では照合の記述として，操作の実行前と実行後で *Class* の属性値が変化しないこと，実行後の属性値と入力パラメータを照合すること，の 2 段階に分割する冗長な記述を許容しない．

す組合せを選び，照合ノードを決定する．

次章で，各サブステップの内容を詳しく説明する．

なお，筆者らは文献 [6] において，操作パラメータと *Class* の属性の間の代入（*AbPA*: assignment between an operation parameter and a class attribute）[*3]を表す代入ノードの形式と，代入ノードが意味的に代入を表すための 3 つの条件を定義した．照合ノードと代入ノードの形式は同一であるが，照合ノードの条件 1–3 と，代入ノードの 3 つの条件には差異がある．本手法の Step 1.1–1.3 は，文献 [6] で提案した解析手法に基づいているが，照合ノードと代入ノードの条件の差異に対応するために，文献 [6] の解析手法に変更を加えている．次章の Step 1.1–1.3 の説明の中で，解析手法の変更内容について言及する．

4.2 Step 2：照合出力情報の抽出

Step 1 で求めた照合ノードに対応する照合出力情報を表す OCL 表現ノードを “出力ノード” と呼ぶ．3.1 節で述べた CbPA の定義から，照合出力情報として，次の条件を満たす出力ノードを求める．

条件 4： つねに確定した値を持つ[*4]．

条件 5： 出力ノードと照合ノードのとりうる値の組は事後条件を満たす．

条件 6： 出力ノードの値から，照合結果が真となることが演繹的に決定される．

条件 6 の代わりに，出力ノードの値と照合結果の間に，他の条件[*5]を課することも考えられる．しかし，照合操作は照合データの存在を確認するために用いる操作であるため，条件 6 が妥当である．

本手法では，これらの条件を満たす出力ノードを，次の 3 つのサブステップによって求める．各サブステップは，それぞれ条件 4–6 に対応している．

Step 2.1： 出力パラメータまたはその属性を表す OCL 表現ノードの中で，つねに確定値を持つノードを，出力ノードの候補としてスクリーニングする．

Step 2.2： Step 2.1 で求めた出力ノード候補と，Step 1 で求めた照合ノードに対して，事後条件の制約を充足する値の対応を求める．

Step 2.3： Step 2.2 で求めた値の対応から，出力ノード値と，照合結果が真であることの含意判定を行い，条件 6 を満たす出力ノードを決定する．

次章で，抽象解釈（Step 1.1），起源の解析（Step 1.2），照合ノードの決定（Step 1.3），スクリーニング（Step 2.1），値の対応付け（Step 2.2），含意判定（Step 2.3）について説明する．スクリーニングと値の対応付けは，抽象解釈の解析技術に基づいており，抽象解釈の解析手続を呼び出し，その結果を利用する．

[*3] 文献 [6] の代入は，操作パラメータと *Class* の属性の間の 2 方向の値の代入を指す．一方は，本論文の 1 章の代入操作の説明で言及した，入力パラメータ値の，*Class* の属性への代入であり，他方は，*Class* の属性の値の，出力パラメータへの代入である．

[*4] 不定値から，照合結果を特定することはできないため．

[*5] たとえば，出力ノードの値から，照合結果が偽となることが演繹的に決定される等の条件．

5. 解析方法

本章では提案手法の 6 つのサブステップの解析（抽象解釈，起源の解析，照合ノードの決定，スクリーニング，値の対応付け，含意判定）を説明する．

5.1 抽象解釈

本節では，Step 1.1 で条件 1 の判定を行うための抽象解釈の解析方法を説明する．

OCL の事後条件は，構文解析を経て抽象構文木として表現される．抽象構文木のノード（OCL 表現ノード）は，OCL メタモデルで定められている OCL 表現のインスタンスである．各 OCL 表現ノードは単一の型を持ち，オブジェクト状態に対して構文木を評価することにより，ノードの値が決定される．本論文で扱うコンポーネントモデルにおける OCL 表現ノードの型と値の対応を**表 1** に示す．*OclVoidValue* は OCL の不完全性により生じる未定義の値を表す[*6]．*Collection* の中では要素の順序を持たない *Set*, *Bag* を解析の対象とする[*7]．

表 1 に示した OCL 表現の値を具体値，その集合を具体領域と呼ぶ．具体領域には *Integer* や *Class* のインスタンス等無限個の要素が含まれる．したがって，具体領域上で各 OCL 表現ノードがとりうる具体値を求めるためには，無限個の要素の組合せを扱わなければならない．そこで具体領域の部分集合に対応する抽象値を定義し，有限の抽象値を要素とする抽象領域上で近似的な意味の解釈（抽象解釈）を行う．

5.1.1 抽象領域

具体領域 D に対して，抽象領域 $\underline{D}$ ={T,F,U} を定義する．具体領域の要素 d は，抽象化写像 $\alpha : D \to \underline{D}$ によって抽象領域の要素に対応付けられる．

表 1 OCL 表現の値と型
Table 1 Value and type of OCL expressions.

値	型
PrimitiveValue	*PrimitiveType, Enumeration*
TupleValue	*DataType*
ObjectValue	*Class*
CollectionValue (SetTypeValue,BagTypeValue)	*Collection(Set,Bag)*
OclVoidValue	All types

[*6] OCL 言語仕様では，0 除算等により発生する例外も *OclVoidValue* に含まれるが，本論文では事後条件中で例外を扱わない．

[*7] 要素の順序を持つ *Sequence* と *OrderedSet* の解析方法は今後の研究課題である．

表 2 *Boolean* 型の論理演算ノードの解釈
Table 2 Interpretation of *Boolean* operations.

	and		or		xor		not	implies	
T	T	T	T/U	U/T	T/F/T	T/T/F	F	T/F	T/U
F	F/U	U/F	F	F	F	F	T	T	F

$$\alpha(d) = \begin{cases} \mathrm{T} & \text{if } d \text{ が } \texttt{true}(\textit{Boolean}),\ \text{非 } 0(\textit{Integer/Real}),\\ & \textit{OclVoidValue} \text{ 以外の任意の値 } (\textit{DataType/Class/String/Enumeration}),\\ & \text{非空 } (\textit{Collection}) \text{ のいずれか} \\ \mathrm{F} & \text{if } d \text{ が } \texttt{false}(\textit{Boolean}),\ 0(\textit{Integer/Real}),\ \text{空 } (\textit{Collection}) \text{ のいずれか} \\ \mathrm{U} & \text{if } d \text{ が } \textit{OclVoidValue} \end{cases}$$

一方，抽象領域の要素は具体化対応 $\gamma = \alpha^{-1}$ によって具体領域の部分集合に対応付けられる．

上記の定義から，CbPA の照合ノード形式を持つ *Boolean* 型の OCL 表現は，具体値 `true` を持つ場合かつその場合に限って抽象値 T を持つため，抽象値によって条件 1 の充足を判断することができる．

5.1.2 抽象領域における解釈規則

抽象領域上では，OCL 表現の種類ごとに定めた規則に従って，ノードの意味を解釈する．一例として**表 2** に *Boolean* 型の論理演算の解釈規則を示す．表の行と列はノードのとりうる抽象値と演算の種類を示し，交点のセルは抽象構文木の子ノードに割り当てるべき抽象値を示す．本手法では 5.1.3 項で述べる方法に従って，抽象構文木の根ノードから順に各ノードの抽象値を下向きに計算するため，抽象値の割当て規則は親ノードから子ノードへの 1 方向である．子ノードが複数存在する場合は，構文木と同一の順序で子ノードごとにセルを設けている．OCL の非決定性により，子ノードの抽象値が一意に定まらない場合は，とりうる値を “/” で区切って記載している．たとえば子ノード 1 と子ノード 2 の抽象値が各々 “T/U” と “U/T” である場合，子ノード 1 と子ノード 2 の抽象値の組合せは (T,U) または (U,T) である．

表 2 の解釈規則は，OCL 記述の不完全性を再現する．たとえば `and` 演算では，親ノードの抽象値が F のケースにおいて，子ノード 1 に F を割り当てる場合は，子ノード 2 に U を割り当てる．子ノード 2 に T または F を指定する過度な解釈（Overspecification）は行わない．

本手法では，OCL メタモデルで定義されている全 12 種類の OCL 表現のうち，OCL の基本サブセットである BasicOCL パッケージに対応する 9 種類の OCL 表現を扱う．抽象構文木の OCL 表現ノードには，OCL 表現の種類とノード名の組合せによって，89 のバリエーションが存在する．本手法では，その中の 57 のバリエーションについて解釈規則を定義し，全体の約 64%を網羅する[*8]．付録 A.1 に 57 のバリエーションと解釈規則の定義方法を記載する．

```
CALCULATE-ABSTRACT-VALUES(T:AST)
 1 solution_set := empty_set
 2 foreach e in C
 3   b_set := T.allblocks(e)
 4   b_set.remove(e.inconsistentblocks())
 5   solution_set.add(OBTAIN-POSSIBLE-VALUES(b_set))
 6 return solution_set

OBTAIN-POSSIBLE-VALUES(S:SET(BLOCK))
S1: Disable belows from terminal nodes upward:
      blocks having disabled child blocks, and
      blocks not included in S.
S2: Perform all possible interpretations downward.
S3: Return block-set of all enabled ones.
```

図 4 抽象値の計算アルゴリズム
Fig. 4 An algorithm for obtaining abstract values.

5.1.3 静的解析

事後条件を満たす抽象構文木上の各 OCL 表現ノードの解釈の組合せを求めるアルゴリズム `CALCULATE-ABSTRACT-VALUES` を**図 4** に示す．OCL の抽象構文木は DAG（directed acyclic graph）の構造を持ち，複数の親ノードからリンクされる合流点ノードは，事後条件中の複数の箇所で参照される変数を表す．あるオブジェクト状態において，各 OCL 表現ノードは同時に複数の値をとりえないため，合流点ノードに至る複数の経路上で，合流点ノードに同一の値を割り当てる必要がある[*9]．そこで合流点ノードの値の一意性を保証するために，アルゴリズムは DAG の閉路の分岐点と合流点のノードの抽象値と解釈の一意な組合せの集合 C を作成し（2 行目），C の各要素 e について，サブルーチン `OBTAIN-POSSIBLE-VALUES` を用いて抽象値の可能な割当てを計算する（5 行目）．その際に，分岐点と合流点のノードの，e に含まれない抽象値と解釈を計算の対象から除外する（3，4 行目）．最後に，各ノードの抽象値と解釈の可能な組合せの集合を返す（6 行目）．この実行結果から，条件 1 の充足を確認することができる．

図 2 の事後条件の，C のある要素 e_1 についての計算結果を**図 5** に示す．各ノード[*10]には上下 2 層のブロック群を設けてあり，下層の各ブロック中の文字（T または F）はノードの抽象値を，ブロック間のリンクは子ノードの抽象値の割当てを表す．上層のブロックは，同一の値を持つ下層のブロック群を代表して，親ノードからのリンクを受

[*8] ただし，21 のバリエーションの解釈規則は，適用対象の型を *Set* または *Bag* に限定する．
[*9] ただし，1 経路上での合流点ノードへの U の割当ては，その経路上で合流点ノードの値が不定であることを意味し，他の経路上で同ノードに T または F を割り当てることが可能である．
[*10] ノードを識別するため，具象構文の字句名と番号を付与してある．具象構文で無名の字句は括弧内に抽象構文の名前を示す．

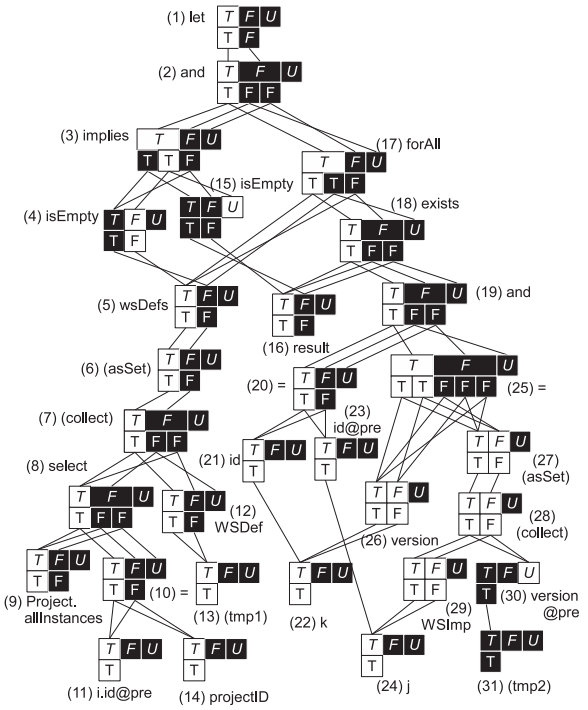

図 5 抽象値の計算結果
Fig. 5 A calculation result of abstract values.

ける．上層のブロック中の文字が U の場合は，そのノードの抽象値が U であり，子ノードにも U が割り当てられる．白黒が反転表示されたブロックは，事後条件を満たす解釈の組合せに含まれない，無効な割当てであることを示す．

図 4 のサブルーチン `OBTAIN-POSSIBLE-VALUES` を説明する．まず `S1` において，末端ノードから順に抽象構文木を上向きに走査し，ノードの各下層ブロックについて，ブロック集合 `S` に含まれない，もしくはリンク先の子ノードのブロックが無効であるようなブロックを無効化する．次に `S2` において，値 T を持つ根ノードの下層ブロックから順に下向きに抽象値の解釈を行い[*11]，可能な解釈の経路を持たないブロックを無効化する．ノードの走査順序を保証するために，`S1`，`S2` ではそれぞれ，各ノードですべての子ノード，親ノードからの走査を待ち合わせてから処理を行う．最後に `S3` において，すべての有効なブロックの集合を返す．

本論文では以降，ノード (i) の左端から 0 起点で j 番目の，値 v を持つ下層ブロックを $(i)jv$ のように記述する．図 5 は，分岐点ノード (2)，(19) と合流点ノード (5)，(16)，(22)，(24) のブロックの組合せ $e_1 = \{(2)0\mathrm{T}, (19)0\mathrm{T}, (5)0\mathrm{T}, (16)0\mathrm{T}, (22)0\mathrm{T}, (24)0\mathrm{T}\}$ についての計算結果を示している．この計算結果から，e_1 において，照合ノードの形式を持つノード (10)，(20)，(25) がいずれも抽象値 T，すなわち具体領域で具体値 `true` をとり，条件 1 を満たすことが分かる．後の Step 1.2 の起源の解析と，Step 1.3 の照合ノードの決定の手続きによって，

[*11] 事後条件の定義から根ノードは具象値 `true`（抽象値 T）を持つ．

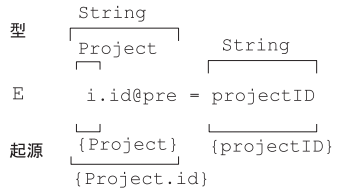

図 6 OCL 表現ノードの型と起源
Fig. 6 Type and origins of an OCL expression.

これら 3 つの照合ノード候補からノード (10) が照合ノードとして決定される．

本論文の抽象解釈の方法は，基本的には文献 [6] で述べた方法と同じである．ただし，文献 [6] では *Collection* 全般を解析対象に含み，OCL 表現ノードのバリエーションごとに解釈規則を個別に定義していたが，本論文では，*Collection* の中で解析対象を *Set* と *Bag* に限定するとともに，付録 A.1 に示す方法によって解釈規則を系統的に定義するように改善を行った．また文献 [6] では，サブルーチン `OBTAIN-POSSIBLE-VALUES` で抽象構文木を上向きに走査する際に，複数の子ノードからの走査の待ち合わせを行っていなかったが，ノードの走査順序を保証するために待ち合わせが必要なことが分かったため，本論文で改善を行った．さらに，抽象解釈の結果の利用方法に差異が存在する．本論文では条件 1 に対応して，`true`，`false` のいずれかの具体値をとりうるノードを照合ノードの候補とするのに対して，文献 [6] で述べた代入ノードの場合は，具体値 `true` をとりうるノードを，代入ノードの候補として選択する．

5.2 起源の解析

照合ノードの形式を持つ OCL 表現ノード `E` について，条件 2 を判定するためには，`E` の子ノードの具体的な値が由来するモデル要素を特定する必要がある．子ノードが *Class*/*DataType* 型であれば，由来するモデル要素を型情報から容易に特定することができるが，`E` が照合ノードを表す場合は子ノードは *PrimitiveType* 型もしくはその *Collection* であり，子ノードの型情報からはモデル要素を特定することができない．そこで以下の OCL 表現ノードの起源の概念を導入する．

> **OCL 表現ノード `E` の起源：** `E` の値が由来する構造モデル上の名前付き要素の集合

図 6 に，照合ノードの形式を持つノード `E` の具象構文と，その子ノードの型と起源を示す．この例におけるノード `E` は，図 5 のノード (10) である．`E` の左右の子ノードの型は *String* であり，CbPA と無関係な `'a'`，`'b'` のようなリテラルと型の上では区別がつかない．一方，左右の子ノードの起源はそれぞれ，`Project` クラスの属性 `id` と，操作パラメータ `projectID` を要素に持つ．これらの起源

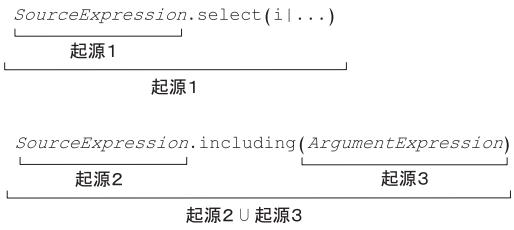

図 7 起源の導出
Fig. 7 Derivation of node origin.

の情報から，E について条件 2 の充足を判定することができる．

図 5 の照合ノードの形式を持つノード (20)，(25) の左の子ノードの起源はそれぞれ，操作の戻り値 return の要素の属性 id，version を要素に持ち，(20)，(25) の右の子ノードの起源はそれぞれ，WSDef クラスの属性 id，WSImp クラスの属性 version を要素に持つ．したがって，図 5 のノード (10)，(20)，(25) は，いずれも条件 2 を充足する．

本論文では，筆者らが文献 [6] で提案したアルゴリズム OBTAIN-ORIGINS を用いて OCL 表現ノードの起源を求める．OBTAIN-ORIGINS は，対象ノードの子ノードを再帰的にたどり，各ノードの起源を導出する．起源の導出方法は，ノードの種類に依存する．図 6 中のノード‘i.id@pre’のような，属性の参照を表す *PropertyCallExp* ノードの起源は，自身の左の子ノードの起源の要素の，右の子ノードが表す属性からなる．*Collection* 要素の選択を表す演算 select の起源は，子ノード *SourceExpression* の起源と等しい．また *Collection* へのオブジェクトの追加を表す演算 including の起源は，子ノード *SourceExpression*，*ArgumentExpression* の起源の和集合である（**図 7**）．他の種類のノードについても同様に，起源の導出方法が定義される．

照合ノードの条件 2 は，文献 [6] に記載されている，代入ノードに課する条件の 1 つと同一である．そのため，本節で説明した起源の解析は，代入ノードの形式を持つノードに対して実施する，文献 [6] の起源の解析と同一の内容である．

5.3 照合ノードの決定

照合ノードの形式を持つ OCL 表現ノード E について，条件 3 の判定を行う方法を説明する．Step 1.2 の起源の解析で求めた，E の起源に含まれる操作パラメータとクラスの属性を，それぞれ param，att とする．param が入力パラメータであるか否かは，UML モデルに記載される param の方向（in/out）から特定される．また，att が示す値が操作実行前の値か否かは，操作実行前の値を参照するための OCL の@pre キーワードの有無によって特定される．そこで Step 1.3 では，param の方向が in で，かつ att に@pre が付加されている場合に限り，条件 3 が充足されると判定し，E を照合ノードと決定する．

表 3 照合・代入の決定条件
Table 3 Decision conditions for collation and assignment.

操作パラメータの方向	*Class* の属性の@pre	照合	代入
in	有	○	×
	無	×	○（入力パラメータ → クラスの属性）
out/return	有	×	○（クラスの属性 → 出力パラメータ）
	無	×	○（クラスの属性 → 出力パラメータ）

本論文の例題の場合には，Step 1.2 で求めたノード (10)，(20)，(25) の中で，ノード (10) のみが条件 3 を満たし，照合ノードと決定される．

文献 [6] では，代入ノードの決定方法が述べられている．この方法は，起源の解析で求めた操作パラメータの方向と，クラスの属性の@pre の有無の情報を用いる点が，照合ノードの決定方法と類似しているが，決定の条件が異なる．代入ノードの決定のポイントは以下の 2 点である．

- 操作実行前の値を意味する@pre 付きの属性と，方向が in の操作パラメータは，被代入オペランドにはなりえない．
- 方向が out/return の操作パラメータには，操作内で値が代入される．

表 3 に，本論文で定める照合ノードの決定条件と，文献 [6] の代入ノードの決定条件の差異を示す．表中の矢印（→）は代入の方向を表す．

本手法の Step 1 の解析と，文献 [6] の解析手法の違いについてまとめる．OCL では，照合と代入はともに，等価関係・包含関係の形式で宣言的に記述される．照合ノード，代入ノードを特定するためには，いずれの場合も，等価関係・包含関係がとりうる値の特定と，等価演算・包含演算のオペランドの値が由来するモデル要素の特定が必要であり，そのための方法として，本論文と文献 [6] ではともに抽象解釈と起源の解析を行う．一方，照合と代入は，演算のオペランドが表すパラメータの方向と，クラスの属性の@pre の有無の組合せによって，区別される．本論文と文献 [6] では，照合と代入の決定の際に，表 3 に示す異なる組合せを用いる．

5.4 スクリーニング

本節では，抽象構文木中の，出力パラメータまたはその属性を表す OCL 表現ノード集合 S_{out} から，つねに確定値 T または F をとるノードを，出力ノードの候補としてスクリーニングする方法を述べる．

まず S_{out} の構成方法を説明する．操作パラメータとその属性の参照は，各々 *VariableExp* と *PropertyCallExp* によって表されるため，これらの種類の OCL 表現ノードを抽出し，さらにその中で出力パラメータまたはその属性を起源とするノードを選んで S_{out} を構成する．各ノードの起源は，Step 1.2 の解析結果を利用する．図 5 の例では

```
DECIDE-CERTAINTY(E:NODE,SS:SET(SET(BLOCK)))
 1 foreach S in SS
 2   b_set := S
 3   b_set.removeAll(E.lowerblocks)
 4   solution := OBTAIN-POSSIBLE-VALUES(b_set)
 5   if solution.notEmpty() then return false
 6 return true
```

図 8 ノード値の確定性の判定アルゴリズム
Fig. 8 An algorithm for deciding certainty of node-value.

$S_{out} = \{(16), (21), (22), (26)\}$ である．

次に S_{out} の要素 E の値の確定性を判定する方法を説明する．5.1 節で述べた抽象解釈の内容から，E の下層ブロック（値が T または F）を含まない有効な割当が存在する場合，かつその場合に限り，ノード E は不定値 U をとる．そこで，ノード E が不定値 U をとりうるか否かを調べることにより，確定性を判定する．

確定性判定アルゴリズム DECIDE-CERTAINTY を図 8 に示す．アルゴリズムは，ノード E と，Step 1.1 で計算した C の各要素の有効ブロックの組合せ集合 SS をパラメータとして動作する．集合 SS の要素である組合せ S について，E の下層ブロックを除いたブロック集合 b_set を求め（2，3 行目），Step 1.1 の抽象解釈のサブルーチン OBTAIN-POSSIBLE-VALUES を用いて，有効な解釈の組合せを求める（4 行目）．そのような組合せが存在する場合，E が不定値 U をとることを意味するので，E の値が確定的ではないと判定する（5 行目）．

S_{out} の要素であるノード (16) について，組合せ S' ∈ SS に関する確定性の判定を行った結果を図 9 に示す．S' は，Step 1.1 で集合 C の要素 $e_2 = \{(2)0T, (19)2F, (5)1F, (16)1F, (22)0T, (24)0T\}$ について計算した，有効な解釈の組合せであり，斜線のハッチングで示している．アルゴリズム DECIDE-CERTAINTY の 3 行目で，S' からノード (16) の下層ブロックである (16)1F を除いて計算した結果，S' のブロックは無効となる．したがって，白黒反転で示したブロックを含めて，すべてのブロックが無効となり，S' に関してノード (16) は不定値 U をとらない．SS の他の要素についても同様に，ノード (16) が不定値 U をとる割当てが存在しないため，ノード (16) の値が確定的であると判定し，出力ノード候補として抽出する．一方，ノード (16) 以外の S_{out} の要素 (21)，(22)，(26) は，図 9 から分かるように，S' において不定値 U をとるため，出力ノード候補から除外する．

5.5 値の対応付け

本節では，Step 2.1 で求めた出力ノード候補 E_0 と，Step 1 で求めた照合ノード E_C の組合せについて，E_0 と E_C の抽象値の間の対応 $f_{E_0,E_C} : \{T,F\} \to \underline{D}$ を求める解析方法を述べる．なお出力ノード候補 E_0 はつねに確定値をとるため，

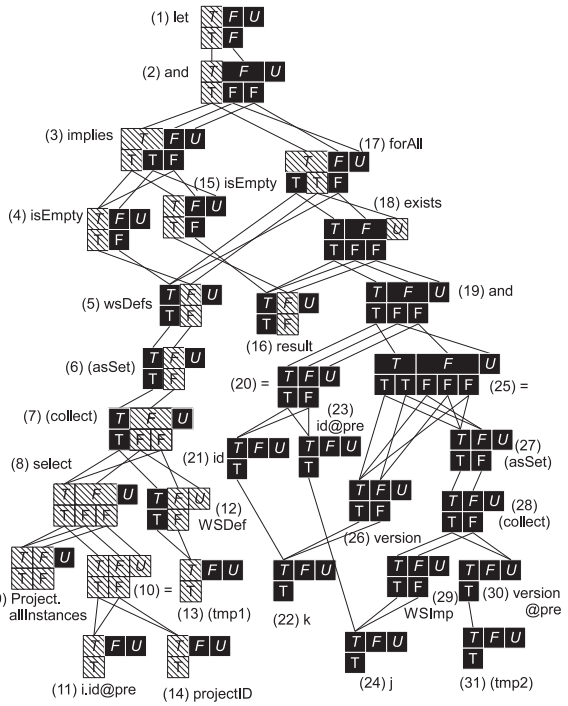

図 9 ノード値の確定性の判定の例
Fig. 9 An example of certainty decision.

$U \notin dom(f_{E_0,E_C})$ である．

本解析方法の工夫は，抽象構文木中のノードの解釈の個々の有効な組合せから f_{E_0,E_C} を求める代わりに，E_0 と E_C のある抽象値の組を与える解釈の組合せの存在の有無を解析して，f_{E_0,E_C} を求める点にある．この工夫により，8 章で議論するように，実用的なオーダの計算量で解析を行うことが可能になる．

f_{E_0,E_C} の計算アルゴリズム OBTAIN_VALUE_MAPPING を図 10 に示す．アルゴリズムは，ノード E_0，E_C と，Step 1.1 で求めた C の各要素の有効ブロックの組合せ集合 SS をパラメータとして動作する．まずノード E_0，E_C の，値 T，F を持つブロック集合 oT，cT，oF，cF を求める（3–7 行目）．そして集合 SS の各要素 S について，Step 1.1 の抽象解釈のサブルーチン OBTAIN-POSSIBLE-VALUES を用いて，次の 4 つの有効な解釈の部分集合を求める（9–12 行目）．

sol_oT：E_0 の値が T の組合せ

sol_oT_cU：E_0 と E_C の値がそれぞれ T，U の組合せ

sol_oF：E_0 の値が F の組合せ

sol_oF_cU：E_0 と E_C の値がそれぞれ F，U の組合せ

sol_oT ∩ cT ≠ φ の場合，E_0 と E_C の値がともに T であるケースが存在することを意味するので，f_{E_0,E_C} を表す v_map に対 (T,T) を追加する（13，14 行目）．同様に sol_oT ∩ cF ≠ φ の場合は対 (T,F) を追加し（15，16 行目），sol_oF ∩ cT ≠ φ の場合は対 (F,T) を追加し（18，19 行目），sol_oF ∩ cF ≠ φ の場合は対 (F,F) を追加する（20，

```
OBTAIN-VALUE-MAPPING(Eo,Ec:NODE,SS:SET(SET(BLOCK)))
 1 v_map := empty_map
 2 v_map.put(T,{}), v_map.put(F,{})
 3 oT, oF, cT, cF := empty_set
 4 foreach b in Eo.lowerblocks
 5   if b.value = T then oT.add(b) else oF.add(b)
 6 foreach b in Ec.lowerblocks
 7   if b.value = T then cT.add(b) else cF.add(b)
 8 foreach S in SS
 9   sol_oT := OBTAIN-POSSIBLE-VALUES(S-oF)
10   sol_oT_cU := OBTAIN-POSSIBLE-VALUES(S-oF-cT-cF)
11   sol_oF := OBTAIN-POSSIBLE-VALUES(S-oT)
12   sol_oF_cU := OBTAIN-POSSIBLE-VALUES(S-oT-cT-cF)
13   if sol_oT.intersection(cT).notEmpty()
14   then v_map.get(T).add(T)
15   if sol_oT.intersection(cF).notEmpty()
16   then v_map.get(T).add(F)
17   if sol_oT_cU.notEmpty() then v_map.get(T).add(U)
18   if sol_oF.intersection(cT).notEmpty()
19   then v_map.get(F).add(T)
20   if sol_oF.intersection(cF).notEmpty()
21   then v_map.get(F).add(F)
22   if sol_oF_cU.notEmpty() then v_map.get(F).add(U)
23 return v_map
```

図 10 値の対応付けアルゴリズム
Fig. 10 An algorithm for mapping the node-values.

21 行目）．また sol_oT_cU $\neq \phi$ の場合には，E_0 と E_C の値がそれぞれ T，U であるケースが存在することを意味するので，v_map に対 (T,U) を追加する（17 行目）．同様に sol_oF_cU $\neq \phi$ の場合は対 (F,U) を追加する（22 行目）．最後に v_map を f_{E_0,E_C} として返す（23 行目）．

出力ノード候補 (16) と照合ノード (10) について，有効な解釈の組合せ S' $\in$ SS に関するブロック集合 sol_oF_cU の計算例を**図 11** に示す．図中の白抜きの有効なブロック集合が sol_oF_cU であり，斜線のハッチングは，アルゴリズム OBTAIN-VALUE-MAPPING の 12 行目で，S' からノード (10) の下層ブロックである (10)0T と (10)1F を除いて計算した結果，無効となったブロック集合 S'-sol_oF_cU を示している．図 11 の計算結果から，sol_oF_cU $\neq \phi$ であり，E_0 と E_C の値がそれぞれ F，U である組合せが存在するため，$f_{(16),(10)}(\mathrm{F}) \ni \mathrm{U}$ となる．本アルゴリズムを実行することにより，最終的に $f_{(16),(10)}(\mathrm{T}) = \{\mathrm{T}\}$，$f_{(16),(10)}(\mathrm{F}) = \{\mathrm{T,F,U}\}$ が得られる．

5.6 含意判定

本節では，Step 2.2 で求めた対応 f_{E_0,E_C} から，出力ノード候補 E_0 の値と，照合結果が真であることの間の含意判定を行う方法を述べる．

4.2 節の条件 6 から，出力ノードのある出力値に対して，照合結果が一意に T に定まる．本手法では，そのような出力値の抽象値が，T，F のいずれか一方に定まるという制限を設け，次の条件式の成立を判定する（V_0 を出力ノード候補 E_0 の抽象値の集合とする）．

$$\exists v \in V_0(\forall v' \in V_0(v = v' \Leftrightarrow f_{E_0,E_C}(v') = \{\mathrm{T}\}))$$

上記の式を満たす 2 つのパターンを**表 4** に示す．第 1 のパターンでは $f_{E_0,E_C}(\mathrm{T}) = \{\mathrm{T}\}$ であり，第 2 のパターンでは $f_{E_0,E_C}(\mathrm{F}) = \{\mathrm{T}\}$ である．記号 $\mathfrak{P}$ は冪集合を表す．本手法では，f_{E_0,E_C} が表 4 のパターンのいずれかに合致する場合に，E_0 を出力ノードと判定する．

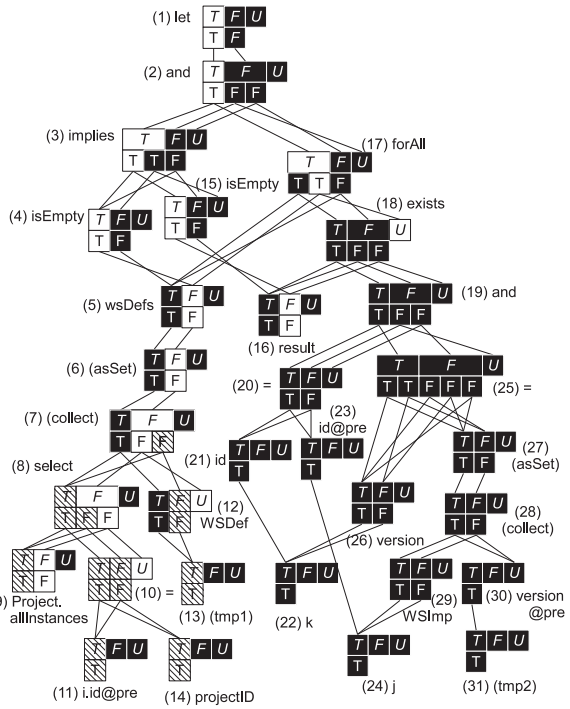

図 11 値の対応付けの例
Fig. 11 An example of node-value mapping.

表 4 含意の成立パターン
Table 4 Patterns of implication.

出力値 $v_0 \in V_0$	$f_{E_0,E_C}(v_0)$	
T	{T}	S$\in \mathfrak{P}$({T,F,U})\{T}
F	S$\in \mathfrak{P}$({T,F,U})\{T}	{T}

6. 支援ツール

我々は，本手法の適用を支援するためのツールを作成した．ツールは，解析部と表示部の 2 つの部分から構成される（**図 12**）．解析部は，ビジネスコンポーネントの仕様モデルが記述された XMI 形式のファイル[*12]を入力として，4 章と 5 章で説明した解析を自動で実行し，ビジネスコンポーネントの操作の事後条件ごとに，解析結果，抽象構文木の解釈規則，抽象値の計算結果を出力する．一方，表示部は，解釈規則と抽象値の計算結果を可視化する．

[*12] XMI は，UML モデルの交換に用いられる標準的なファイル形式である．

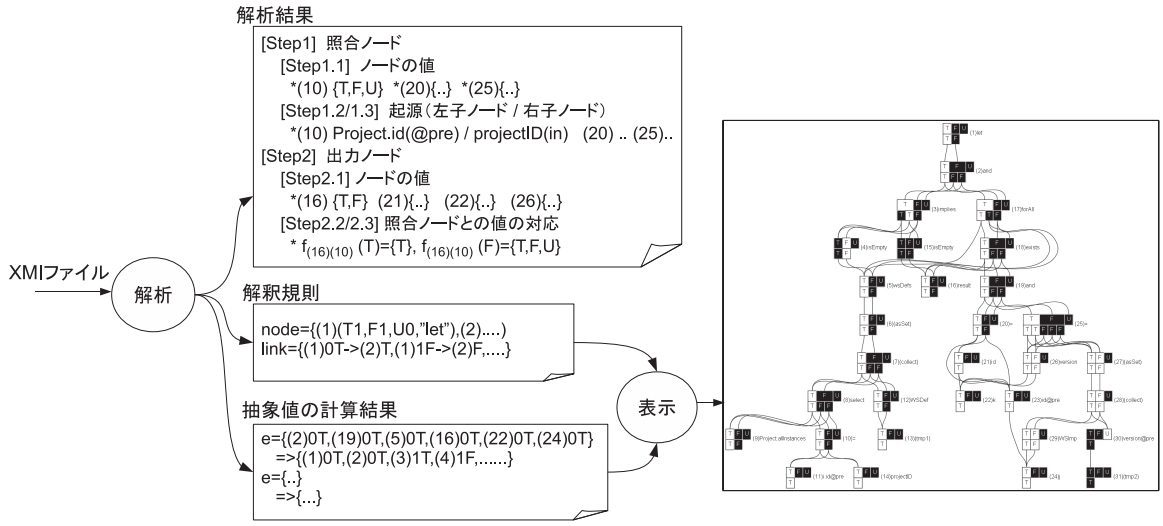

図 12 支援ツールの概要
Fig. 12 Tool overview.

解析結果には，各サブステップごとの計算結果と，条件 1–6 の判定結果がテキスト形式で記載される．記号 * が付加された部分は，該サブステップの条件を充足することを表す情報である．ツールの利用者は，解析結果の内容をサブステップごとに順を追って参照することによって，解析の途中経過と最終結果を知ることができる．

解釈規則は，5.1.2 項で説明した，事後条件の親子ノードの抽象値の割当て規則の情報であり，抽象値の計算結果は，5.1.3 項で説明した，抽象値の可能な割当て情報である．後者の割当て情報は，構文木の分岐点ノードと合流点ノードの解釈の組合せ e の各々について，各ノードがとりうる抽象値の組合せが，テキストファイルの 1 行に記載される．ツールの利用者は，grep 等のプログラムを用いて，抽象値の計算結果のパターンマッチングを行い，関心のあるノード値の組合せを抽出することができる．たとえば ‘(16)[0-9]T’ と ‘(10)[0-9]T’ を含む行を抽出し，照合が成功する場合のノード値の組合せを得ることができる．ただし [0-9] は数字 1 文字を表すものとする．

ある e のノード値の組合せと解釈規則を入力として，ツールの表示部を実行すると，e における有効な解釈の組合せが図 5 の形式のグラフとして表示される．解釈規則のみを入力すると，Step 1.3 の静的解析を実行する前の，すべてのノードが有効なグラフが表示される．これらのグラフ表示は，解析結果をより深く理解するために役立つ．たとえば，利用者が手動で抽出した CbPA の内容が，ツールの解析結果と異なる場合に，グラフを詳細に調べることにより，利用者とツールの間で解釈の違いが生じた OCL 表現ノードを探すことができる．

我々は，Java 言語の OCL 処理系である MDT/OCL [8] を利用して，ツールの解析部の実装を行った．また表示部の実装では，グラフ描画プログラムである Graphviz [9] を利用した．

7. 評価実験

提案手法を評価するために，筆者の勤務先の社内システムに手法を適用する実験を行った．対象システムは 5 つのビジネスコンポーネントと 7 つのシステムコンポーネントから構成され，Web ブラウザを介してシミュレーションを投入する機能を提供する．5 つのビジネスコンポーネントには，合計 58 の操作と 61 の事後条件が定義される．

実験には，2 名の仕様記述者が被験者として参加した．被験者 1 は OCL の詳細な知識を有し，実開発で OCL を使用した経験を持つ．被験者 2 は OCL の基本知識を有し，実開発で OCL を使用するのは初めてである．

実験の手順を説明する．まず，2 名の被験者が，実開発の一環として操作の振舞い仕様を OCL で記述し，仕様レビューを通じて OCL 記述の洗練を行った．次に，61 の事後条件に含まれる照合を手動で抽出し，それらを特定する CbPA の集合を明示的に指定した．最初の振舞い仕様の記述作業では，各被験者が異なる操作の記述を担当したが，それ以降の，OCL 記述の洗練と照合の抽出の作業は，2 名が共同で実施した．その後，61 の事後条件に対して，6 章で述べた支援ツールを実行して，CbPA を自動抽出した．本論文では以降，被験者が手動で求めた CbPA 集合を M，ツールで自動抽出した CbPA 集合を A と呼ぶ．M と A の関係を**表 5** に示す．

2 名の被験者が分担して記述した 61 の事後条件の質は，共同の洗練作業を通じて均一化されたと見なせる．また集合 M は共同で抽出を行っており，2 名の属人性は排除されている．したがって，8.3 節で本実験の結果をもとに手法の有用性を議論する際に，2 名の被験者の作業分担と，OCL の知識・経験の差異による影響を考慮しない．

表 5 CbPA 集合 M，A の関係
Table 5 The relation between M and A.

集合	M	A	$M \cap A$	$M \setminus A$	$A \setminus M$
要素数	44	35	31	13	4

8. 議論

本章では最初に，技術課題 1 に対応して，本手法の解析の妥当性を議論する．次に技術課題 2 に対応して，解析の効率性を評価する．さらに，実開発における手法の有用性について論じる．最後に手法の適用範囲を述べる．

8.1 解析の妥当性

抽象解釈の正当性と CbPA の識別性能の 2 つの観点から，解析の妥当性を検討する．前者の観点では，本論文で提案した抽象解釈について，近似の安全性（本来の解釈結果を漏れなく含んでいること）と，技術課題 1 に対応する厳密性（OCL の文脈依存性，非決定性，不完全性を備えていること）を考察する．後者の観点では，手法全体の網羅性と正確性を把握する．

8.1.1 抽象解釈の正当性

まず，抽象解釈の安全性について考察する．付録 A.1 に記載した 57 の解釈規則の定義方法は，以下の 2 つに大別される．

方法 1：OCL 表現の親子ノードがとりうる具体値の組を考えた後に，抽象化を行い，親子ノードがとりうる抽象値の組に変換する．

方法 2：OCL 表現の親子ノードがとりうる具体値の組を定義できない場合に，抽象領域上でのみ，親子ノードがとりうる抽象値の組を定める．

Collection に対する演算 size，count，collect と，それらを用いて定義される演算の *IteratorExp*/*OperationCallExp* ノードのバリエーションは，方法 2 によって解釈規則が定義され，その他のすべてのバリエーションは，方法 1 によって解釈規則が定義される．

抽象構文木が，方法 1 のバリエーションのみで構成されている場合を考える．OCL 表現ノード $\mathtt{E}$ と $n_{\mathtt{E}}$ 個の子ノードがとりうる，具体値の組合せ集合と，抽象値の組合せ集合を，それぞれ $I^{\mathtt{E}} \subseteq (D - \{OclVoidValue\}) \times D^{n_{\mathtt{E}}}$，$\underline{I}^{\mathtt{E}} \subseteq (\underline{D} - U) \times \underline{D}^{n_{\mathtt{E}}}$ とする．$I^{\mathtt{E}}$ と $\underline{I}^{\mathtt{E}}$ は，ノード $\mathtt{E}$ の具体領域，抽象領域上の解釈規則を表す．方法 1 では，$I^{\mathtt{E}}$ の要素に抽象化を行うことによって，$\underline{I}^{\mathtt{E}}$ を求める．すなわち $\forall(d, d_1, \ldots, d_{n_{\mathtt{E}}}) \in I^{\mathtt{E}}((\alpha(d), \alpha(d_1), \ldots, \alpha(d_{n_{\mathtt{E}}})) \in \underline{I}^{\mathtt{E}})$ が成り立つ．$I^{\mathtt{E}}$ と $\underline{I}^{\mathtt{E}}$ の関係を**図 13** に示す．

5.1.3 項で説明した静的解析は，他ノードの値との組合せによって事後条件が成立しうるような，各ノードの解釈規則の部分集合を求める．本手法では，静的解析を抽象領域で実行して $\underline{I_s}^{\mathtt{E}} \subseteq \underline{I}^{\mathtt{E}}$ を求めるが，同様の解析を具体領域で実行して，具体値の組合せ集合 $I_s^{\mathtt{E}} \subseteq I^{\mathtt{E}}$ を求めることが可能である．もしノード $\mathtt{E}, \mathtt{E}', \ldots$ の具体値の組 $(d, d_1, \ldots, d_{n_{\mathtt{E}}}) \in I_s^{\mathtt{E}}$，$(d', d'_1, \ldots, d'_{n_{\mathtt{E}'}}) \in I_s^{\mathtt{E}'}, \ldots$ の組合せが事後条件を満たすならば，対応する抽象値の組 $(\alpha(d), \alpha(d_1), \ldots, \alpha(d_{n_{\mathtt{E}}})) \in \underline{I}^{\mathtt{E}})$，$(\alpha(d'), \alpha(d'_1), \ldots, \alpha(d'_{n_{\mathtt{E}'}})) \in \underline{I}^{\mathtt{E}'}), \ldots$ の組合せも事後条件を満たす．したがって $\forall(d, d_1, \ldots, d_n) \in I_s^{\mathtt{E}}((\alpha(d), \alpha(d_1), \ldots, \alpha(d_n)) \in \underline{I_s}^{\mathtt{E}})$ が成り立つ．

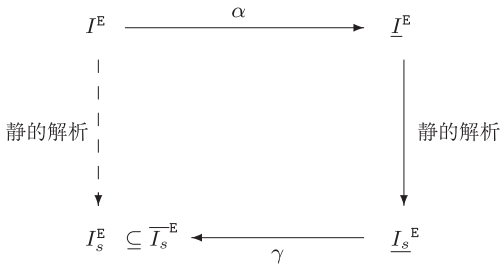

図 13 具体領域と抽象領域におけるノード $\mathtt{E}$ の解釈規則
Fig. 13 Interpretations of node $\mathtt{E}$ in concrete/abstract domain.

5.1.1 項で導入した抽象領域と抽象化写像は，$\forall d \in D(d \in \gamma(\alpha(d)))$ を満たす．したがって $\forall \overrightarrow{d} = (d, d_1, \ldots, d_n) \in I_s^{\mathtt{E}}(\overrightarrow{d} \in (\gamma(\alpha(d)), \gamma(\alpha(d_1)), \ldots, \gamma(\alpha(d_n))))$ が成り立つ．すなわち，抽象領域上の解析結果 $\underline{I_s}^{\mathtt{E}}$ を具体化した $\overline{I_s}^{\mathtt{E}}$ は，具体領域上の解析結果 $I_s^{\mathtt{E}}$ を含む（図 13）．

以上の考察から，方法 1 のノードのみで構成された抽象構文木に対しては，本手法の解析が，具体領域上での解釈結果を漏れなく抽出する，安全性を備えていることが分かる．しかし，方法 2 のノードについては，具体領域での解釈規則が定義できないため，方法 2 のノードを含む抽象構文木に対しては，近似の安全性が保証されない．

次に，OCL 記述の文脈依存性・非決定性・不完全性に関する抽象解釈の厳密性について考察する．

抽象領域での解釈規則 $\underline{I}^{\mathtt{E}}$ は，OCL 言語仕様 [10] で規定されている意味論に基づいて設計されており，OCL の非決定性，不完全性を反映している．また $\mathrm{U} \notin dom(\underline{I}^{\mathtt{E}})$ であり，親ノードの値が不定値 U の場合に，子ノードの値を特定する過度な解釈を避け，不完全性を確保している．さらに 5.1.3 項のアルゴリズム `CALCULATE-ABSTRACT-VALUES` は，抽象構文木の根ノードから下向きに有効な解釈の組合せを求めることにより，各ノードの文脈に則した解釈を行っている．以上のことから，本手法の抽象解釈は，技術課題 1 で述べた OCL の性質（文脈依存性・非決定性・不完全性）を厳密に取り扱っていると考えられる．

本項では，本手法の抽象解釈の安全性と厳密性について検討を行った．ただし本項の考察は，我々が定義した解釈規則が OCL の意味論に則したものであることを前提としている．もし解釈規則に誤りが含まれる場合には，検討結果にも誤りがある可能性がある．

8.1.2 CbPA の識別性能

まず，手法の網羅性について述べる．Step 1.1–2.1 において，照合ノード E_C と出力ノード候補 E_0 を特定する目的に対しては，抽象解釈を用いた近似的な計算が有効に機能する．しかし，Step 2.2 で求めた対応 f_{E_0,E_C} を用いて Step 2.3 で含意判定を行う際に，E_0 の値の近似の精度不足による偽陰性のエラーが生じうる．

例として，事後条件 “p='NG' or (p='OK' and exp)” を考える．p は出力パラメータ，exp は照合を表す記述である．具象領域では，p に該当する出力ノード候補 E_0 の値が'OK' の場合に限り，exp に該当する照合ノード E_C は値 true をとり，条件 6 を満たす．しかし本手法の抽象解釈では，具体値'OK' と'NG' はともに抽象値 T に対応付けられるため，$dom(f_{E_0,E_C}) = \{T\}$，$f_{E_0,E_C}(T)=\{T,U\}$ となり，Step 2.3 の含意判定で E_0 を出力ノードから除外するエラーが発生する．

上記の例のような，*String* または *Enumeration* 型の出力パラメータとリテラル値の等価関係は，実問題の OCL 記述でしばしば見られる．これらの型の具体値は，つねに抽象値 T に対応付けられるため，近似の精度不足によるエラーの恐れがある．そこで本手法では，そのような形式に合致する出力ノード候補を抽出した場合は，Step 2.3 の含意判定を行わずに，出力ノードの決定をユーザに委ねる．

次に，手法が識別する CbPA の正確性について述べる．本手法では，4.1 節の “照合ノードの形式” に合致する照合ノードごとに，CbPA を求める．この方法は，操作の実行を通じてオブジェクト状態が変化する場合に，偽陽性の CbPA を抽出する可能性がある．

例として，事後条件 “result=(x.att@pre=p and not x.att=p)” を考える．p と x は，照合値を指定する入力パラメータと照合先を表すオブジェクトであり，result は操作の戻りパラメータである．本手法は，等価関係 “x.att@pre=p” に該当する照合ノード E_C と，result に該当する出力ノード E_0 との間の CbPA を抽出する．しかし，事後条件中の表現 “not x.att=p” は，照合先の値の更新やオブジェクトの削除によって，操作の事後に照合データが存在しないことを表している．この種の誤抽出のエラーを防ぐためには手法の拡張が必要であり，今後の研究課題である．

8.2 解析の効率性

本手法の解析の効率性を次の 2 つの観点から評価する．

- 解析アルゴリズムの計算量
- 実問題の OCL 記述に対する実行性能

まず，アルゴリズムの計算量について述べる．抽象構文木を特徴づける属性を**表 6** に，各サブステップごとの 1 事後条件あたりの解析の時間計算量と空間計算量を**表 7** に，それぞれ示す[*13]．属性 L，N_b，L_b はいずれも $O(N)$ であり，手法全体の時間計算量，空間計算量はともに $O(NC)$，すなわちノード数 N の線形オーダである．ただし C は，分岐点と合流点の合計ノード数の指数オーダで増大する．したがって，複数箇所で参照される変数の数が少ない OCL 記述に関しては，本手法は技術課題 2 を解決しているといえる．

表 6 抽象構文木の属性
Table 6 The attributes of AST.

名前	意味
N	ノード数
L	ノード間のリンク数
N_b	ブロック数
L_b	ブロック間のリンク数
C	分岐・合流点ノードの下層ブロックの組合せ数

表 7 解析の計算量
Table 7 Complexity of analysis.

Step	解析	時間計算量	空間計算量
1.1	抽象解釈	$O((N_b+L_b)C)$	$O((N_b+L_b)C)$
1.2	起源の解析	$O(N+L)$	$O(N)$
1.3	照合ノードの決定	$O(1)$	$O(1)$
2.1	スクリーニング	$O((N_b+L_b)C)$	$O((N_b+L_b)C)$
2.2	値の対応付け	$O((N_b+L_b)C)$	$O((N_b+L_b)C)$
2.3	含意判定	$O(1)$	$O(1)$

表 8 実モデルに対する実行性能
Table 8 Analysis performance against the real model.

	平均値	最大値
N	38	88
C	2,748	87,480
解析時間 (s)	1.74	48.16

次に，実問題に対する手法の実行性能を述べる．7 章の評価実験において，対象モデルの属性 N，C と，支援ツールの実行時間を計測した．計測には，一般的な能力（Intel Core2 Duo 1.60 GHz，4 GB RAM）を持つ PC を使用した．計測結果を**表 8** に示す．ツールは，平均的なノード数 N と組合せ数 C を持つ事後条件の解析を 2 秒未満で実行する．この実行時間は，人間がモデルの記述や理解に要する時間に比べて明らかに短く，ストレスなくツールを使用することができる．したがって，本手法は実用上十分な効率を備えているといえる．

また本手法の各ステップは，集合 C の各要素の計算を独立に実行することができる．文献 [11] の事例のような，N が数百に及ぶ大規模な事後条件に対しても，計算の並列化によって実用的な計算時間で解析を行うことが可能である．

8.3 手法の有用性

7 章の評価実験の結果をもとに，照合操作の厳密な仕様

[*13] 付録 A.2 に計算量の見積もり方法を記載する．

を記述するうえでの，手法の有用性を検討した．$M \cap A$ の 31 要素の内容を詳細に調べたところ，いずれも正しく抽出された CbPA であった．また $M \setminus A$ の 13 要素は，実際には CbPA ではない誤答であった．そして $A \setminus M$ の 4 要素には，8.1.2 項で述べた，ユーザへの判断委譲と，オブジェクト状態の変化による誤抽出のエラーが，2 件ずつ含まれていた．これらのデータから，支援ツールは約 89%の精度と 100%の再現率で CbPA を自動抽出したことが分かる．

仕様記述者による 13 の誤答（$M \setminus A$）には，次の 2 通りの誤謬が見られた．ここで `X` は照合データの集合を表す OCL 表現とする．

全称記号の不完全性の誤謬：事後条件 “`X->forAll(i| result->includes(i))`” を，`X` が空ならば `result` も空と誤って解釈し，`result` を照合出力情報とするケース．正しくは，`X` が空のときに `result` は未定義値をとる．

前件否定の誤謬：事後条件 “`result implies not X->notEmpty()`” の裏の命題 “`not result implies X->notEmpty()`” が成り立つという誤った解釈から，`result` を照合出力情報とするケース．正しくは，前件が不成立の場合には，後件中の `X` は未定義値をとる．

今回の実験のように，OCL の使用経験を持つ仕様記述者であっても，人手で作業を行う場合には，上記のような非決定性・不完全性の誤謬が起こる可能性がある．仕様記述者が操作仕様の作成時に，支援ツールを用いて誤謬を取り除くことによって，正確な仕様を記述することができる．さらに，照合操作の利用者が操作仕様を正しく理解するうえでも同様に，支援ツールを活用することができる．

8.4 適用範囲

提案手法が扱う仕様モデル（2 章）は，Catalysis [1] や UML components [2] のコンポーネントモデルに適合する．これらの開発手法を採用する多数の実システムの開発で，本手法を適用することができる．大規模な構造モデルを扱うコンポーネントの照合操作では，仕様の記述と理解の誤りが起きやすい．そこで本論文では，コンポーネントベース開発を題材として手法の提案を行った．しかし，手法の解釈規則や解析方法そのものは，一般的な UML クラスの操作に対しても適用可能である．

3.1 節で述べたように，本手法では解析の対象を，単一の出力パラメータによって照合結果を伝える照合操作に限定している．また 5.6 節で述べたように，Step 2.3 の含意判定において，照合値が T になる場合に，出力ノードの抽象値が T，F のいずれか一方に定まるという制限を設けている．これらの限定と制限は，実問題の照合操作の性質を想定して設けており，7 章の実験対象モデルと，文献 [1], [2] の照合操作の事例に対して，例外なく適合することを確認した．さらに，他の多くの実システムの照合操作にも適合することが期待できる．本手法を拡張し，上記の限定と制限の範囲外のケースを扱うことは可能であるが，照合操作を識別する条件が複雑になり，解析精度が低下する恐れがある．

本論文では述べなかったが，“`result=(X.att@pre=p or Y.att@pre=p)`” のように，複数の照合ノードの値の組合せによって照合結果が決定されるケースがありうる．本手法の Step 2.2，2.3 を拡張することにより，このようなケースを取り扱うことができる．

9. 関連研究

宣言的な仕様記述の解釈において，事後条件に明示的に記述されないモデル要素の状態が確定しないフレーム問題 [12] が存在する．本論文では UML components と同様に，フレーム仮定 [5] を採用することによって，この問題を解消している．一般にフレーム仮定を用いた場合には，モデル要素間の継承関係の解釈に矛盾が生じうる [13] が，本手法が扱うコンポーネントモデルは継承関係を含まないため，矛盾の恐れはない．

Correa らは文献 [14] において，解釈の誤りを生じやすい OCL の記述形式をあげ，リファクタリングを行うための記述パターンを提案している．また Cabot は文献 [5] において，宣言的な OCL 記述を命令的なアクションに変換するための変換パターンを提案している．これらのパターンは有用であるが，OCL 記述の文脈依存性・非決定性・不完全性の性質が考慮されていない．一方，本手法の抽象解釈はこれらの性質を厳密に扱い，本論文の 8.3 節であげた，全称記号の不完全性の誤謬や，前件否定の誤謬を網羅的に抽出する．

抽象解釈は，プログラムの性質を近似的に解析するための理論的な枠組みであり，Cousot らによって文献 [15] で提案された．種々のプログラム言語に対する応用を中心に数多くの研究が行われてきたが，UML・OCL 記述の抽象解釈の研究は数少ない．Baruzzo らは文献 [16] において，クラス図と OCL 制約記述に対するシーケンス図の整合性検証の計算量を削減する目的で，抽象解釈の適用を提案している．しかし文献 [16] には，抽象化の具体的な方法が示されていない．

Cabot らは文献 [17] において，モデルの整合性制約を破壊するようなモデル要素の変更を特定する手法を提案している．Cabot らの手法は，OCL の抽象構文木の各ノードに 3 通りの記号（‘+’, ‘−’, ‘und’）を割り当てる点が，本論文の解釈規則と類似しているが，OCL 記述の文脈依存性・非決定性・不完全性を考慮していない．また Cabot らの手法が，抽象構文木を根ノードから下向きに 1 度だけ走査するのに対して，本手法では分岐点と合流点のノードの値の整合性を考慮して，上向きと下向きの走査を行う点が異なる．

本手法の，他の問題への応用について述べる．UML モデルに対する OCL 制約記述の整合性を検証するために，OCL 記述を高階論理式に変換して定理証明を行う方法 [18] や，制約充足問題の制約に変換して，制約ソルバを用いて検証を行う方法 [19] が提案されてきた．しかし，いずれの方法も，モデルのサイズに対して指数オーダの計算量を要するという問題がある．Shaikh らは文献 [20] において，スライシングを用いて計算量を減らす手法を提案した．我々が提案する抽象解釈は，Shaikh らの手法とは異なるアプローチの解法として，この問題に対して応用できる可能性がある．

10. おわりに

本論文では，宣言的な仕様記述言語である OCL を用いて記述されるビジネスコンポーネントの振舞いモデルを静的に解析して，照合操作を識別する手法を提案した．解析に用いる抽象解釈は，OCL の持つ文脈依存性・非決定性・不完全性の性質を厳密に解釈する．また本手法は，仕様記述のサイズの線形オーダの計算量で解析を行う．さらに本論文では，実問題の仕様モデルを用いて，提案手法が実用上十分に効率的であり，上記の OCL の性質に起因する仕様記述の解釈の誤りを除去するうえで有用であることを確認した．

本手法を用いて，ビジネスコンポーネントの照合操作の仕様を正確に記述・理解することが可能になる．このことによって，データの一貫性が保たれた安全な情報システムを構築するための足がかりが得られる．

今後は，多くの実開発に手法の導入を図るとともに，手法の拡張と応用について検討していきたい．

参考文献

[1] D'Souza, D. and Wills, A.: *Objects, Components and Frameworks With UML: The Catalysis Approach.*, Addison-Wesley (1998).

[2] Cheesman, J. and Daniels, J.: *UML Components – A Simple Process for Specifying Component-Based Software*, Addison-Wesley (2001).

[3] Trcua, N., van der Aalst, W. and Sidorova, N.: Data-Flow Anti-patterns: Discovering Data-Flow Errors in Workflows, *Advanced Information Systems Engineering*, van Eck, P., Gordijn, J. and Wieringa, R. (Eds.), LNCS, Vol.5565, pp.425–439, Springer Berlin/Heidelberg (2009).

[4] Ryndina, K., Kuster, J. and Gall, H.: Consistency of Business Process Models and Object Life Cycles, *Models in Software Engineering*, Kuhne, T. (Ed.), LNCS, Vol.4364, pp.80–90, Springer Berlin/Heidelberg (2007).

[5] Cabot, J.: From Declarative to Imperative UML/OCL Operation Specifications, *Conceptual Modeling – ER 2007*, Parent, C., Schewe, K.-D., Storey, V. and Thalheim, B. (Eds.), LNCS, Vol.4801, pp.198–213, Springer Berlin/Heidelberg (2007).

[6] Inoue, T. and Honiden, S.: A method for data-flow analysis of business components, *Proc. 14th International ACM Sigsoft Symposium on Component Based Software Engineering*, *CBSE '11*, pp.51–60, ACM (2011).

[7] Ackermann, J. and Turowski, K.: A Library of OCL Specification Patterns for Behavioral Specification of Software Components, *CAiSE 2006*, Dubois, E. and Pohl, K. (Eds.), LNCS, Vol.4001, pp.255–269, Springer (2006).

[8] MDT project: MDT/OCL, available from ⟨http://www.eclipse.org/modeling/mdt/?project=ocl⟩.

[9] AT&T Research: Graphviz, available from ⟨http://graphviz.org/⟩.

[10] OMG: Object Constraint Language OMG Available Specification Version 2.0 (2006), available from ⟨http://www.omg.org/spec/OCL/2.0/⟩.

[11] Frias, L., Queralt, A. and Olive, A.: EU-Rent Car Rentals Specification, Technical Report LSI-03-59-R, Universitat Politecnica de Catalunya (2003).

[12] Borgida, A., Mylopoulos, J. and Reiter, R.: On the frame problem in procedure specifications, *IEEE Trans. Softw. Eng.*, Vol.21, No.10, pp.785–798 (1995).

[13] Kosiuczenko, P.: Specification of Invariability in OCL, *Model Driven Engineering Languages and Systems*, Nierstrasz, O., Whittle, J., Harel, D. and Reggio, G. (Eds.), LNCS, Vol.4199, pp.676–691, Springer Berlin/Heidelberg (2006).

[14] Correa, A., Werner, C. and Barros, M.: Refactoring to improve the understandability of specifications written in object constraint language, *Software, IET*, Vol.3, No.2, pp.69–90 (2009).

[15] Cousot, P. and Cousot, R.: Abstract interpretation: A unified lattice model for static analysis of programs by construction or approximation of fixpoints, *Proc. 4th ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages*, *POPL '77*, New York, NY, USA, ACM, pp.238–252 (1977).

[16] Baruzzo, A. and Comini, M.: Static verification of UML model consistency, *3rd Workshop on Model Design and Validation*, Hearnden, D., Süß, J.G., Rapin, N. and Baudry, B. (Eds.), pp.111–126 (2006).

[17] Cabot, J. and Teniente, E.: Incremental integrity checking of UML/OCL conceptual schemas, *Journal of Systems and Software*, Vol.82, No.9, pp.1459–1478 (2009).

[18] Brucker, A. and Wolff, B.: HOL-OCL: A Formal Proof Environment for UML/OCL, *Fundamental Approaches to Software Engineering*, Fiadeiro, J. and Inverardi, P. (Eds.), Lecture Notes in Computer Science, Vol.4961, pp.97–100, Springer Berlin/Heidelberg (2008).

[19] Cabot, J., Clariso, R. and Riera, D.: Verifying UML/OCL Operation Contracts, *Integrated Formal Methods*, Leuschel, M. and Wehrheim, H. (Eds.), Lecture Notes in Computer Science, Vol.5423, pp.40–55, Springer Berlin/Heidelberg (2009).

[20] Shaikh, A., Clarisó, R., Wiil, U.K. and Memon, N.: Verification-driven slicing of UML/OCL models, *Proc. IEEE/ACM International Conference on Automated Software Engineering*, *ASE '10*, New York, NY, USA, ACM, pp.185–194 (2010).

付　　録

A.1 抽象領域における解釈

本手法が扱う OCL 表現ノードのバリエーションと，抽

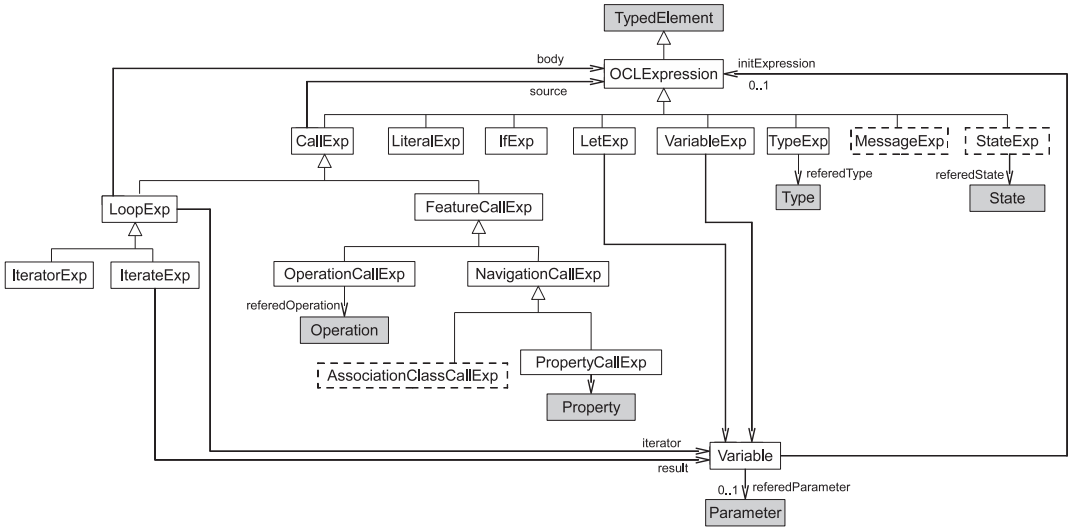

図 **A·1** OCL 表現の抽象構文メタモデル
Fig. A·1 Abstract syntax metamodel for OCL expressions.

象領域における解釈規則について説明する．

A.1.1 OCL 表現ノードのバリエーション

OCL 言語仕様で定義されている，OCL 表現の抽象構文メタモデルを**図 A·1** に示す．図中のグレー部分は UML メタモデルの要素である．OCL 標準ライブラリの基本サブセットである BasicOCL パッケージは，図 A·1 の OCL 表現の 12 種類の具象クラスのうち，点線部以外の 9 種類を扱う[*14]．本手法では，この 9 種類の OCL 表現について，OCL 表現ノードの解釈の規則を定義する．

OCL 表現ノードは，その種類とノード名の組合せ（バリエーション）によって意味が異なるため，バリエーションごとに解釈規則を定める必要がある．ノード名は，*IteratorExp*/*IterateExp*/*OperationCallExp* ノードでは演算の名前を，*LiteralExp* ノードでは参照するリテラルを，*TypeExp* ノードでは参照する *Type* の名前を，*VariableExp* では参照する変数の名前を，それぞれ表す．*IfExp*/*LetExp*/*PropertyCallExp* ノードは無名である．BasicOCL における 89 のバリエーションを**表 A·1** に示す．手法は 57 のバリエーションの解釈規則を定義し，その中の 21 は適用対象の型を限定する．解釈規則を定義する場合は “x”，解釈規則を定義しない場合は空欄，手法の対象外である場合は “-”（ハイフン）で表す．“x(T)” は，型 T についてのみ解釈規則を定義することを表す．

表 A·1 中の “x” 以外のバリエーションについて，手法で解釈規則を定義しない，あるいは対象外とする理由を以下に説明する．

- *TypeExp* ノードは，UML モデルで静的に与えられた *Type* の参照を表し，事後条件を満たすオブジェクト状態の有無に直接影響を与えないため，値の割当てが不要である．
- 演算 allInstances の *OperationCallExp* ノードは，子ノードがつねに *TypeExp* ノードであり，子ノードの値の解釈が不要である．
- 演算 oclIsUndefined の *OperationCallExp* ノードは未定義値の評価を表す．該ノードが複数の親ノードを持つ合流点ノードである場合には，すべての親ノードからの経路上で未定義値 U を割り当てるケースにおいてのみ，評価結果が真となる．しかし，5.1.3 項で述べた静的解析手法では，一部の経路で T/F の値を持つケースを許容しており，正しい計算結果が得られない可能性があるため，手法の対象外とする．このバリエーションを扱うためには，手法の拡張が必要であり，今後の課題である．
- 演算 oclIsInvalid の *OperationCallExp* ノードは例外の評価を表すが，本手法では事後条件で例外を扱わないため，このバリエーションは不要である．
- 演算 oclAsType，oclIsTypeOf，oclIsKindOf の *OperationCallExp* ノードは，型のキャストや適合を評価する表現であるが，本手法が扱うコンポーネントモデルは継承関係を含まないため，これらのバリエーションは不要である．
- 演算 oclInState，hasReturned，result，isSignalSent，isOperationCall の *OperationCallExp* ノードは，BasicOCL の対象外である状態とメッセージに関する演算を表すため，これらのバリエーションを扱わない．
- *IteratorExp*/*OperationCallExp* ノードは，*Collection* 全般に対して定義されるが，手法では頻度の高い *Set* の演算に限定し，*Bag* の演算は扱わない．ただし，演算 collect は適用結果が *Bag* であるが，実モデルで頻

[*14] 点線部の *AssociationClassCallExp*，*StateExp*，*MessageExp* は，それぞれ関連クラス，状態，メッセージを扱う OCL 表現である．

表 A·1 OCL 表現ノードのバリエーション
Table A·1 OCL expression node variations.

OCL 表現	ノード名	解釈規則	OCL 表現	ノード名	解釈規則	OCL 表現	ノード名	解釈規則
LiteralExp	リテラル	x		=	x	*OperationCallExp*	size	x(*Set*)
IterateExp	iterate	-		<>	x	(*Collection*)	includes	x(*Set*)
IteratorExp	exists	x(*Set*)	*OperationCallExp*	+	x		excludes	x(*Set*)
	forAll	x(*Set*)	(*PrimitiveType*)	-	x		count	x(*Set*)
	isUnique	-		*	x		includesAll	x(*Set*)
	any	-		-(negative)	x		excludesAll	x(*Set*)
	one	x(*Set*)		/	x		isEmpty	x(*Set*)
	collect	x(*Set*)		abs	x		notEmpty	x(*Set*)
	select	x(*Set*)		floor	x		sum	-
	reject	x(*Set*)		round	x		product	-
	collectNested	-		max	x		union	x(*Set*)
	sortedBy	-		min	x		intersection	x(*Set*)
IfExp		x		<	x		ñ	x(*Set*)
LetExp		x		>	x		including	x(*Set*)
VariableExp	変数名	x		>=	x		excluding	x(*Set*)
TypeExp	型名			<=	x		symmetricDifference	x(*Set*)
PropertyCallExp		x		div	x		flatten	-
OperationCallExp	oclIsNew	x		mod	x		asSet	x(*Bag*)
	oclIsUndefined	-		concat	x		asOrderedSet	-
	oclIsInvalid	-		size	x		asSequence	-
	allInstances			substring	x		asBag	-
	oclAsType	-		toInteger	x	*OperationCallExp*	subSequence	-
	oclIsTypeOf	-		toReal	x	(*Sequence*)	append	-
	oclIsKindOf	-		ToUpper	x		prepend	-
	oclInState	-		toLower	x		insertAt	-
	hasReturned	-		and	x		subOrderedSet	-
	result	-		or	x		at	-
	isSignalSent	-		xor	x		indexOf	-
	isOperationCall	-		not	x		first	-
				implies	x		last	-

用されている．また *Bag* の演算 asSet は，collect の演算結果を *Set* に変換するために併用される．そこで手法は，特例としてこれらの演算の解釈規則を定義する．

- 演算 sortedBy，any の *OperationCallExp* ノードは，Sequence の演算を用いて定義されるため，扱わない．

IterateExp ノードと一部の *IteratorExp*/*OperationCallExp* ノード（isUnique，collectNested，sum，product）を扱わない理由は，次の解釈規則の説明の中で述べる．

A.1.2 解釈規則

OCL 仕様書 [10] の 10 章に規定されているように，OCL 表現ノードの値は，オブジェクト状態，すなわち環境に対して動的に評価を行うことにより決定される．一方，本手法では，各ノードが表す意味に基づき，親子ノードがとりうる抽象値の組合せを解釈規則として静的に定める．以下に，提案手法が定める解釈規則を説明する．限られた誌面の中で，57 のバリエーションのすべての解釈規則を網羅的に列挙して説明することは難しい．そこで本論文では，OCL 言語仕様から解釈規則を導くための手順を定義して，いくつかのバリエーションの解釈規則を例示する．

OCL 表現ノードの解釈規則を求める手順を以下に示す．

1. OCL 仕様書 [10] の 11 章と Annex A で定められたノードの意味論に従って，ノードと，その子ノードがとりうる抽象値の組合せを求める．
2. 手順 1 で求めた，親子ノードの抽象値の組合せから，親ノードの値が U のケースを除外する．
3. 親ノードの値（T/F）に対して子ノードがとりうる値の組合せを解釈規則として求め，表 2 の形式にまとめる．

手順 1 は対象ノードのバリエーションと型に依存し，手順 2，3 は汎用の処理である．以下では，手順 1 についてバリエーションごとに説明する．

LiteralExp ノードでは，リテラルの値を固定的な具体値として，抽象化写像 α を用いてノードの抽象値を求める．演算 oclIsNew の *OperationCallExp* ノードでは，子ノード source が表すオブジェクトが，操作実行中に生成されたか否かによって，親ノードが抽象値 T，F の両方をとりうる．また以下のノード群では，親ノードが子ノードや関連先のクラスの値を引き継ぐ．その引き継ぎ方に従って，抽象値の組合せを求める．

- *IfExp* ノードは，子ノード condition の Boolean 値によって，thenExpression/elseExpression のいずれか片方の子ノードの値を引き継ぐ．
- *LetExp* ノードは子ノード in の値を引き継ぐ．

表 A·2 各種 OCL 表現ノードの解釈
Table A·2 Interpretation of various OCL expressions.

	LiteralExp	*OperationCallExp* (oclIsNew)	*PropertyCallExp*	*LetExp*	*IfExp*			*VariableExp*	*OperationCallExp* (+)	
T	T	T	T	T	T/F	T/U	U/T	T	T/T/F	F/T/T
F	F	T	T	F	T/F	F/U	U/F	F	T/F	T/F

- *PropertyCallExp* ノードは，子ノード source が表すオブジェクトの *Property* の値を引き継ぐ．
- *VariableExp* ノードは，参照する *Variable* の値を引き継ぐ．*Variable* の値は，関連先の initExpression ノードや referedParameter の値に対応する．

これらのバリエーションの解釈規則を**表 A·2** に記載する．なお *LiteralExp*，*PropertyCallExp*，*VariableExp* については，子ノードの代わりに，リテラル，*Property*，*Variable* の抽象値を，交点のセルに示している．

PrimitiveType 型の *OperationCallExp* ノードでは，5.1.1 項で定義した抽象化写像 α に基づいて，具体領域における子ノードの領域分割を行い，子ノードの領域の組合せごとに親ノードのとりうる値を求める．たとえば *Integer* 型の演算+の *OperationCallExp* ノードの場合は，具象領域を $P=(0,+\infty)$, $N=(-\infty,0)$, $Z=[0,0]$, *OclVoid* の 4 つの領域に分割する．演算+の意味から，ある $p,p' \in P$, $n,n' \in N$，$z,z' \in Z$，$v \in OclVoid$ に対して，$p+z \in P$, $p+p' \in P$，$p+n \in P$（$p > -n$ の場合），$p+n \in N$（$p < -n$ の場合），$p+n \in Z$（$p = -n$ の場合），$n+z \in N$, $n+p \in P$（$p > -n$ の場合），$n+p \in N$（$p < -n$ の場合），$n+p \in Z$（$p = -n$ の場合），$n+n' \in N$，$z+p \in P$, $z+n \in N$，$z+z' \in Z$，$v+? \in V$，$?+v \in V$ の演算結果が考えられる．ここで記号？は p，n，z，v のいずれかを表す．以上の具体値の組合せを抽象化して，さらに手順 2，3 を実施すると，表 A·2 の解釈規則が得られる．

次に，*Set* に対する *IteratorExp*/*OperationCallExp* ノードの手順 1 を説明する．そのためにまず *IterateExp* について説明する．OCL 表現 $e_1 \in Set(\mathrm{T})$ の iterate 演算の意味は，次の式で表される[*15]．

$$I[[e_1\text{->iterate}(v_1; v_2 = e_2 | e_3)]] =$$
$$\begin{cases} I[[e_2]] \\ \qquad \text{if } I[[e_1\{v_2/e_2\}]] = \phi, \\ I[[Set\{x_2,..,x_n\}\text{->iterate}(v_1; v_2 = e_3\{v_1/x_1, v_2/e_2\} | e_3)]] \\ \qquad \text{if } I[[e_1\{v_2/e_2\}]] = \{x_1,..,x_n\}. \end{cases}$$

iterate 演算は，e_1 の各要素 v_1 について e_3 の評価を繰り返し，繰返しのたびに評価結果を変数 v_2 に保持することを表す．v_2 の初期値は e_2 であり，繰返し完了時の v_2 の値が演算の結果を表す．一般的に子ノード e_3 の値は，e_1 の要素について評価を繰り返すたびに更新されるため，とりうる値を静的に一意に定めることができない．したがって *IterateExp* ノードは，本手法で扱うことができない．

Set に対するすべての OCL 表現ノードの演算は，iterate 演算を利用して定義されており，一般的には親子ノードの値を静的に定めることはできない．ただし中には，iterate 演算を利用する際に引数として与える OCL 表現が，次の 2 つの条件を満たすような演算が存在する．

条件 1：ある繰返しにおいて，v_2 の値が変化するならば，そのときの e_3 が一定値 c_3 をとり，以降の繰返しを通じて v_2 は一定値 c_3 をとる．

条件 2：演算を通じて v_2 の値が不変ならば，e_3 がつねに一定値 c'_3 をとる．

そのような演算では，iterate 演算ノード，e_1，e_3 の値の組として，条件 1 に対応して (c_3, S, c_3) を，条件 2 に対応して (e_2, S, c'_3) を，さらに e_1 が空の場合に対応して $(e_2, Set\{\}, OclVoidValue)$ を，静的に定めることができる．ただし S は条件 1，2 を満たす非空集合である．たとえば exists 演算

$$e_1\text{->exists}(v_1|body) =$$
$$e_1\text{->iterate}(v_1; v_2 : Boolean = \text{false} \mid v_2 \text{ or } body)$$

の場合は $c_3 = \text{true}$，$c'_3 = \text{false}$ である．これらの値から，exists 演算ノード，e_1，*body* の値の組を $(\text{true}, S, \text{true})$, $(\text{false}, S, \text{false})$，$(\text{false}, Set\{\}, OclVoidValue)$ と定義する．さらに，これらに対応する抽象値の組合せを求めて，手順 2，3 を実施することによって，解釈規則が得られる．同様に演算子 forAll の解釈規則を求めることができる．演算子 size，count では，具体領域では v_2 の変化のたびに値が 1 ずつ増加するため，条件 1 を満たさないが，抽象領域ではつねに T となり条件 1 を満たす．そこで，抽象領域上での値の組合せを直接求めて，解釈規則を定める．これらの演算ノードの解釈規則を**表 A·3** に示す．

演算 collect は，演算 collectNested/flatten を用いて定義される[*16]．

$$e_1\text{->collect}(v_1|body) = e_1\text{->collectNested}(v_1|body)\text{->flatten}()$$
$$e_1\text{->collectNested}(v_1|body) =$$
$$e_1\text{->iterate}(v_1; v_2 : Bag(body.type) = Bag\{\} \mid v_2\text{->including}(body))$$

[*15] $e\{v'/e'\}$ は，OCL 表現 e に出現する変数 v' を e' で置き換えて得られる OCL 表現を表し，$I[[e]]$ は，ある環境における e の値を表す．

[*16] flatten 演算は，演算を適用する $src \in Bag$ の要素が *Collection* 型の場合には，*src* のすべての要素のすべての要素を含む *Bag* を返し，*src* の要素が *PrimitiveType* 型の場合には *src* を返す．

表 A·3 *Collection* 型の演算ノードの解釈
Table A·3 Interpretation of *Collection* operations.

	IteratorExp (exists)		*IteratorExp* (forAll)		*OperationCallExp* (size,asSet)	*IteratorExp* (count)		*IteratorExp* (collect:*PrimitiveType*)		*IteratorExp* (collect:*Collection*)	
T	T	T	T/F	T/U	T	T/T	T/F	T	U	T	T
F	T/F	F/U	T	F	F	F/T/T	U/T/F	F	U	T/F	F/U

演算 collectNested では，演算 iterate の繰返しのたびに *body* ノードの値が *Bag* である v_2 に追加されるので，条件 1 を満たさない．しかし抽象領域上では，*body* が *PrimitiveType* 型の場合は，$e_1 \neq \phi$ ならば演算 collect の結果は *body* の値によらずつねに T である．そこで collect 演算ノード，e_1，*body* の値の組として，条件 1 に対応して (T,T,U) を，e_1 が空の場合に対応して (F,F,U) を定義する．また *body* が *Collection* 型の場合は，*body* $\neq \phi$ となる繰返しが存在すれば，演算 collect の結果は T となり，*body* がつねに ϕ ならば，演算結果は F である．したがって，collect 演算ノード，e_1，*body* の値の組として，条件 1 に対応して (T,T,T) を，条件 2 に対応して (F,T,F) を，e_1 が空の場合に対応して (F,F,U) を定める．演算 collect の解釈規則を表 A·3 に示す．

演算 asSet は，演算を適用する *src* $\in$ *Bag* が非空の場合には，*src* の要素のみを含む集合を返し，*src* が空の場合には空集合を返す．したがって表 A·3 に示す解釈規則が得られる．

演算 isUnique/collectNested/sum/product は，演算 iterate を用いて定義されているが，前述の条件 1，2 を満たさないため，本手法では扱うことができない．

その他の *IteratorExp*/*OperationCallExp* の演算は，すでに説明した演算を利用して定義されている．たとえば *src*->isEmpty() の演算結果は *src*->size() = 0 の値と等しい．そのような演算ノードの解釈規則は，既出の解釈規則を用いて構成することができる．

A.2 解析の計算量

本手法の各解析の計算量の見積もり方法を以下に説明する．8.2 節の表 6 に定義した抽象構文木の属性を用いて，計算量を表現する．

抽象解釈（Step 1.1）：5.1.3 項のサブルーチン `OBTAIN-POSSIBLE-VALUES` は，抽象構文木の下層ブロック間を，末端ノードから上向きに 1 度，根ノードから下向きに 1 度，走査を行い，下層ブロックの部分集合を返す．したがって，その時間計算量は $O(N_b + L_b)$，空間計算量は $O(N_b)$ である．そしてアルゴリズム `CALCULATE-ABSTRACT-VALUES` は，事後条件ごとに 1 回実行され，分岐点と合流点の下層ブロックの C 個の一意な組合せごとに，`OBTAIN-POSSIBLE-VALUES` を呼び出し，得られる結果の集合を返す．ゆえに時間計算量は $O((N_b+L_b)C)$，空間計算量は $O(N_bC)$ である．

起源の解析（Step 1.2）：この解析に用いるアルゴリズム `OBTAIN-ORIGINS` [6] は，ノード間のリンクをたどって各ノードを最大 1 回訪問するので，時間計算量は $O(N + L)$ である．また各ノードの起源には構造モデル上の複数の名前付き要素が含まれ得るが，現実のモデルではその要素数はたかだか数個なので，起源を保持するために必要な記憶容量は O(1) である．ゆえに空間計算量は $O(N)$ である．

照合ノードの決定（Step 1.3）：Step 1.2 で抽出した O(1) 個の照合ノードの候補の，照合値情報と照合先情報の組合せから，表 3 の決定表を用いて照合ノードを決定する．組合せ数は O(1) であり，決定は O(1) 回の演算で実行できるため，必要な計算ステップ数と記憶容量はともに O(1) である．

スクリーニング（Step 2.1）：この解析に用いるアルゴリズム `DECIDE-CERTAINTY` は，O(1) 個の出力ノード候補ごとに実行され，Step 1.1 の解析で得られる，最大 C 個の有効な下層ブロック集合の各々について，5.1.3 項のサブルーチン `OBTAIN-POSSIBLE-VALUES` を 1 回実行する．ゆえに時間計算量と空間計算量は Step 1.1 の計算量と等しい．

値の対応付け（Step 2.2）：この解析に用いるアルゴリズム `OBTAIN-VALUE-MAPPING` は，O(1) 個の出力ノード候補と照合ノードの組合せごとに実行され，Step 1.1 の解析で求めた最大 C 個の有効な下層ブロック集合の各々について，5.1.3 項のサブルーチン `OBTAIN-POSSIBLE-VALUES` を 4 回実行する．ゆえに時間計算量と空間計算量は Step 1.1 の計算量と等しい．

含意判定（Step 2.3）：Step 2.2 で抽出した O(1) 個の出力ノード候補と照合ノードの組合せの値の対応 $f_{\mathrm{E_0,E_c}}$ を，表 4 の成立パターンに対して，O(1) 回の演算でマッチングする．ゆえに時間計算量と空間計算量はともに O(1) である．

井上 拓 （正会員）

1974 年生．1997 年早稲田大学理工学部応用物理学科卒業．1999 年同大学大学院理工学研究科修士課程修了．同年キヤノン株式会社入社．デジタルカメラの開発，ソフトウェア自動テスト環境の構築に従事．現在，同社デジタルシステム開発本部にて医療機器の開発に従事．要求分析，形式手法，コンポーネントベース開発等に興味を持つ．

本位田 真一 （フェロー）

1953 年生．1978 年早稲田大学大学院理工学研究科修士課程修了．（株）東芝を経て 2000 年より国立情報学研究所教授，2012 年より同研究所副所長を併任，現在に至る．2001 年より東京大学大学院情報理工学系研究科教授を兼任，現在に至る．現在，電気通信大学，英国 UCL 等の客員教授を兼任．2005 年度パリ第 6 大学招聘教授．工学博士（早稲田大学）．1986 年度情報処理学会論文賞受賞．日本ソフトウェア科学会理事，情報処理学会理事，日本ソフトウェア科学会編集委員長を歴任．ACM 日本支部会計幹事，日本学術会議連携会員．