修 士 論 文

定理証明技法を用いた
ユースケースの追加支援システムの研究

北陸先端科学技術大学院大学
情報科学研究科情報システム学専攻

牛尾 遼平

2007 年 3 月

修 士 論 文

定理証明技法を用いた
ユースケースの追加支援システムの研究

指導教官　落水 浩一郎 教授

審査委員主査　落水 浩一郎 教授
審査委員　片山 卓也 教授
審査委員　鈴木 正人 助教授

北陸先端科学技術大学院大学
情報科学研究科情報システム学専攻

510014 牛尾 遼平

提出年月: 2007 年 2 月

概 要

ユースケースモデリングにおいて，システムに新たなユースケースの追加を行う際，既存のユースケースを再利用できる場合がある．しかし，実システムではユースケースの数は膨大になり，ユースケース記述も複雑かつ曖昧になることが多いので，新たに追加するユースケースが既存のどのユースケースに関連しているかを把握するのは困難である．本稿の手法では，ユースケース記述の事前条件，事後条件を一階述語論理の論理式で記述する．そして，新たなユースケースと既存のユースケースの間で成り立つ論理的関係を定理証明技法を用いて調べ，再利用できる可能性があるユースケースを抽出する．

目 次

第1章　はじめに

1.1　背景

近年，オブジェクト指向ソフトウェア開発において，分析や設計を行うための言語としてUMLが用いられることが多い．その要素の1つであるユースケース図とユースケース記述は，機能要求の定義に用いられる．UMLのユースケースとは，アクターとシステムの相互作用をイベント系列として表現したもので，ユースケース図の各ユースケースにおいて，相互作用の詳細をユースケース記述として補う．アクターとは，対象としているシステムと直接関係する人やシステムなどを役割として表現したものである．

システムに新たなユースケースの追加を行う際，既存のユースケースを再利用できる場合がある．例として，携帯電話アプリケーションを考える．既存の携帯電話アプリケーションに新たなユースケース「写真付きメールを送信する」を追加するとしよう．この際，ユースケース「メールを送信する」が，携帯電話アプリケーションに既に存在すれば，このユースケースを新たに追加するユースケース「写真付きメールを送信する」のシナリオの一部として再利用可能である．この例のような，あるユースケースの一部に他のユースケースが用いられるという関係をユースケースモデリングでは，include（包含）関係という．しかし，実システムにおいて，ユースケースの数は膨大になり，ユースケース記述も複雑かつ曖昧になりがちであるので，新たに追加するユースケースが既存のどのユースケースに関連しているかが把握しにくいという問題がある．

1.2　目的

本研究の目的は，システムに新たなユースケースを追加する際，関連する既存のユースケースを把握できる機構を構築することである．本研究では，include（包含）関係で包含できる可能性があるユースケースを抽出することにより新たなユースケースの追加を支援するようなシステムを開発する．新たなユースケースの追加といっても色々なものが考えられるが，本研究では既存のユースケースを変更せずに，新たなユースケースの一部として再利用できる場合を対象とする．

1.3 本論文の構成

第３章　本研究のアプローチ

UML では，ユースケース記述にユースケースの事前条件，事後条件を記述する．事前条件とはユースケースを実行する前にシステムが満たすべき条件で，事後条件とはユースケースを実行した後にシステムが満たすべき条件である．自然言語で記述されたユースケース記述において，事前条件，事後条件の表現は曖昧で，通常は一文で単純に記述されることが多い．しかし，実際にはハードウェアの条件などさまざまな暗黙の条件が含まれる．本研究ではこれらの暗黙の条件も含めて事前条件，事後条件を一階述語論理の論理式で形式的に表現する．本稿では，ユースケースの include（包含）関係において，包含するユースケースを基底ユースケース，包含されるユースケースをサブユースケースと呼ぶ．本研究では，論理式で記述した事前条件，事後条件において，論理的関係を定義する．そして，この関係を定理証明技法を用いて調べ，再利用できる可能性があるユースケースを抽出する．

第４章　ユースケースから一階述語論理への変換

ユースケースの事前条件，事後条件を一階述語論理の論理式へ変換する方法を提案する．述語を表現する際の引数は，クラス図とユースケース図より洗い出す．そして，引数の状態，役割を述語名で表現する．そして，満たすべき各条件を述語の組み合わせで表し，最後に各条件を論理積で結び事前条件，事後条件を表現する．

第５章　サブユースケース抽出アルゴリズム

論理式で記述した事前条件，事後条件における論理的な関係を定理証明技法で調べることにより，基底ユースケースに包含できる可能性があるサブユースケースを抽出するアルゴリズムを提案する．入力は新たに追加するユースケースの事前条件，事後条件を一階述語論理の論理式で表したものである．出力は新たに追加するユースケースが包含できる可能性がある既存のサブユースケースの組み合わせである．なお，可能な限り全ての候補を出力する．

第６章　ツールの開発

サブユースケース抽出アルゴリズムを実現するツールを論理型言語 prolog で開発した．prolog で開発した理由は，アルゴリズムの処理の中で，論理式を扱うため，論理型言語を用いるのが適切だと考えたからである．また，サブユースケースを抽出する際に，再帰処理が必要となるので，これに関しても論理型言語が適していると考えた．このツールを用いてサブユースケースの抽出を試みる際，既存のユースケースの情報（ユースケース名，

事前条件，事後条件）をあらかじめ規則としてソースプログラム内に記述しておく必要がある．

第７章　適用例

本研究で提案するサブユースケース抽出アルゴリズムをGommaのエレベータ制御システムの事例研究を題材に適用を試みた．

初めに，エレベータ制御システムの全てのユースケースに対して，事前条件，事後条件を一階述語論理の論理式で記述した．次に，エレベータ制御システムに新たなユースケースを追加し，それらの事前条件，事後条件を一階述語論理の論理式で記述した．そして，開発したツールを用いてサブユースケースの抽出を試みた．

本適用では，新たなユースケースが既存のユースケースとinclude（包含）関係にあるように，新たなユースケースのシナリオを設計した．つまり，あらかじめ包含可能なサブユースケースを把握していたので，ツールの出力結果について，サブユースケースの組み合わせが適切か不適切かを判断することができた．適用の結果，適切なサブユースケースの組み合わせを抽出できた．しかし，同時に不適切なサブユースケースの組み合わせも抽出されることがわかった．今回の適用では，あらかじめ適切な候補を予想できた．しかし，一般的には既存のシステムに新たなユースケースを追加する際，関連する既存のユースケースを予想することはできない．そこで，利用者が出力結果を基に，出力された候補が適切か不適切かを検討する必要がある．

第８章　おわりに

本研究では新たなユースケースの追加を行う方式を，利用者が既存のユースケースを再利用しつつ，定理証明システムを適用するものとして開発した．本研究で開発したツールは，新たなユースケースが既存のユースケースの組み合わせで実現できる場合，再利用可能なユースケースを抽出できるが，既存のユースケースの一部を変更して使用する場合には対応できない．よって，既存のユースケースの変更に伴う事前条件，事後条件の変更が他のユースケースにどのような影響を及ぼすかが調べられるようなシステムも実現したい．

第2章　本研究のアプローチ

本研究のアプローチでは，ユースケース記述の事前条件，事後条件を一階述語論理の論理式で記述する．そして，論理式の間の準含意（≫）関係を定義し，include（包含）関係によって包含できる可能性のあるユースケースを，準含意関係を調べることで抽出する．

2.1　include（包含）関係

あるユースケース A と別のユースケース B の一部に同様のシナリオがある場合に，その部分をくくり出して，独立にユースケース C とすることができる．この場合，ユースケース A とユースケース C，ユースケース B とユースケース C の間には包含の関係が成り立つ．包含は，包含するユースケースから，包含されるユースケースに対して，を矢印付きの破線とステレオタイプ≪include≫でその関係を表す [1]．本稿では，包含するユースケースを基底ユースケース，包含されるユースケースをサブユースケースと呼ぶ．以下の図 2.1 に，ユースケース図における包含の記法を示す．

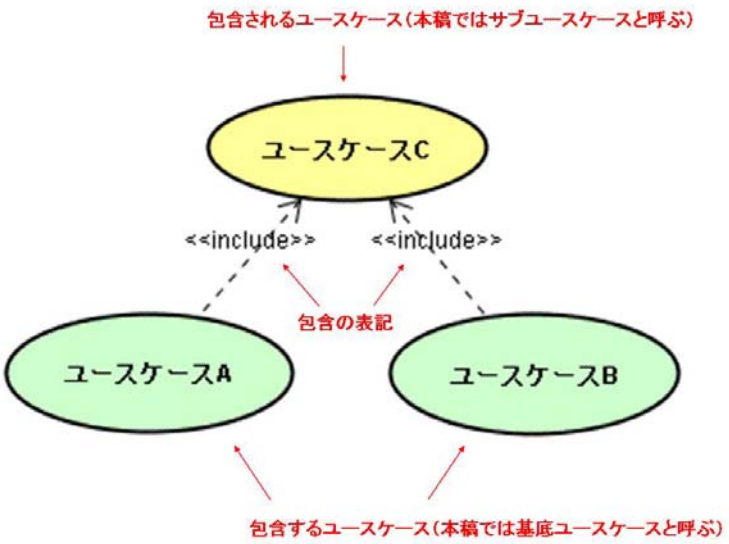

図 2.1: ユースケース図における包含の記法

エレベータ制御システム [2] を例に挙げると，ユースケース「エレベータを要求する」，

ユースケース「行き先を選ぶ」は両方とも「エレベータを派遣する」,「エレベータをフロアに止める」というシナリオを含む．すなわち，これらのユースケースから,「エレベータを派遣する」,「エレベータをフロアに止める」のシナリオをサブユースケースとしてくくり出すことができる．そして，基底ユースケース「エレベータを要求する」，基底ユースケース「行き先を選ぶ」からサブユースケース「エレベータを派遣する」，サブユースケース「エレベータをフロアに止める」に対して，包含の関係で表すことができる．以下の図 2.2 に，エレベータ制御システム [2] のユースケース図を示す．

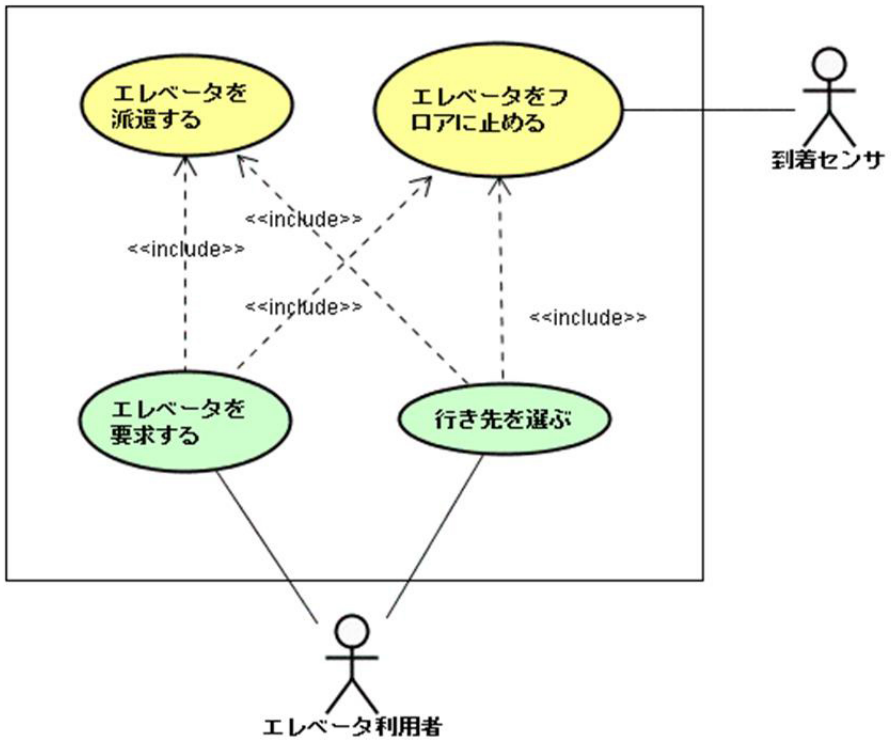

図 2.2: エレベータ制御システムのユースケース図

2.2 事前条件， 事後条件

UML では，ユースケース記述に事前条件 (precondition) と事後条件 (postcondition) を記述する．事前条件とはユースケースを実行する前にシステムが満たすべき条件で，事後条件とはユースケースを実行した後にシステムが満たすべき条件である．以下の図 2.3 にエレベータ制御システム [2] のユースケース「行き先を選択する」の例を示す．

ユースケース名：行き先を選ぶ
概要：エレベーターの中にいる利用者は，移動すべき目的階を選ぶためにエレベータボタンを押す．
アクター：エレベータ利用者，到着センサ
事前条件：利用者がエレベータ内にいる．
記述：
１．利用者はエレベータボタンを押す．エレベータボタンセンサは，利用者が行きたい目的階を識別しつつ，システムにエレベータボタン要求を送る．
２．新しい要求が行き先階リストに追加される．もし，エレベータがアイドル状態の場合，include ”エレベータを派遣する ”
３． include ”エレベータをフロアに停める ”
－ 他に要求があるときには，エレベータは利用者によって要求されたこれらの階を訪問する．最終的にはエレベータは利用者によって選択された目的階に到着する．
例外記述：
１．利用者が下降エレベータボタンを押す．システムの応答は主系列と同じである．
２．もし，エレベータが目的階にいて，移動すべき階がない場合には，システムはその階に，ドアを開けたまま留まる．
事後条件：エレベータは利用者によって選択された階に到着している．

図 2.3: ユースケース記述「行き先を選択する」

2.3 事前条件， 事後条件の一階述語論理表現

本研究では，ユースケースの事前条件， 事後条件を一階述語論理の論理式で表現する．一階述語論理とは，変数である引数とそれらに対する量化記号のみを許し，述語が定数である述語論理のことである [3]．この事前条件， 事後条件を一階述語論理の論理式で表現するというアプローチは文献 [4] を参考にした．[4] では，ユースケースの代替シナリオ，別シナリオを統合して状態遷移図を記述し，テストケースを生成するのに用いている．代替シナリオとは，シナリオの途中で条件分岐が生じた場合や，ユースケースが達成できなかった場合のシナリオのことである．

自然言語で記述されたユースケースにおいて，事前条件，事後条件の表現は曖昧で，通常は一文程度で単純に記述されることが多いが，実際には，様々な暗黙の条件が含まれる．エレベータ制御システムの文献 [2] を例に挙げると，ユースケース「行き先を選択する」において，エレベータのドアの開閉に関する条件や，モータのオン，オフに関する条件が暗黙の条件として含まれる．本研究では，それらの暗黙の条件も視野に入れて，事前

条件，事後条件を設定し論理式で表す（付録 A 参照）.

以下の図 2.4 に，ユースケース「行き先を選択する」（図 2.3）の事前条件，事後条件を一階述語論理式で表現した例を示す.

```
ユースケース名：行き先を選ぶ
事前条件：beInTheElevator(user, elevator) and
          needToGo(user, userDestinationNumber) and
          not pushedElevatorButton(elevator, userDestinationNumber)
--------------------------------------------------------------------------------

                                    シナリオ

--------------------------------------------------------------------------------
事後条件：arrived(elevator, userDestinationNumber) and
          doorOpened(elevator, userDestinationNumber) and
          arrivalSensorDetectedApproach(elevator,userDestinationNumber) and
          motorOff(elevator) and
          beOnTheFloor(user, userDestinationNumber) and
          not doorClosed(elevator) and
          not motorOn(elevator) and
          not beInTheElevator(user, elevator) and
          not needToGo(user, userDestinationNumber) and
          not move(elevator, userFloorNumber, userDestinationNumber) and
          not arrived(elevator, userFloorNumber) and
          not doorOpened(elevator, userFloorNumber) and
          not arrivalSensorDetectedApproach(elevator, userFloorNumber) and
          (haveDestination(elevator, otherUserFloorNumber) →
                                          not lightDirectionLamp(elevator))
```

図 2.4: ユースケース「行き先を選択する」の事前条件，事後条件を一階述語論理式で表現した例

自然言語で記述されたユースケースから一階述語論理式への変換に関しては 3 章で述べる.

事前条件，事後条件を一階述語論理式で表現したものより，数理論理学の含意記号を用いて「事前条件 → 事後条件」と表すことができる．なぜなら，ユースケースにおいて，事前条件が成立していると仮定すると，シナリオを達成した後，事後条件が成り立つからである．ここで，シナリオを介して条件が変化することから，ユースケースによって，システムのある状態がシステムの別の状態に推移すると見ることができる．本稿の図では，

「事前条件 → 事後条件」の関係を図 2.5 のように表す.

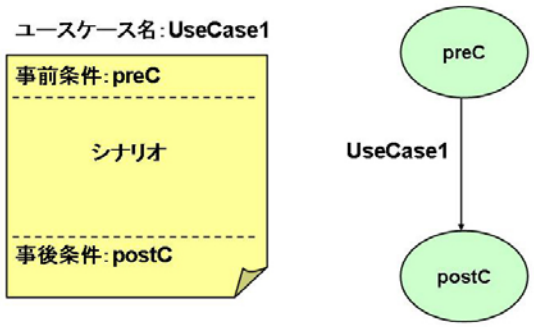

図 2.5: 本稿の図で用いる「事前条件 → 事後条件」の関係

2.3.1 準含意関係

本研究では，論理式の間の準含意（≫）関係を以下のとおり定義する.

$$X \gg Y$$

$X = A1 \vee A2 \vee \cdots \vee An$ のとき,
$(A1 \rightarrow Y) \vee (A2 \rightarrow Y) \vee \cdots \vee (An \rightarrow Y)$ が成り立つ.

準含意関係が成立してもサブユースケースを基底ユースケースの任意の箇所に包含できるわけではない．包含できるのは図 2.6 に示すように，シナリオの最初または最後のみである.

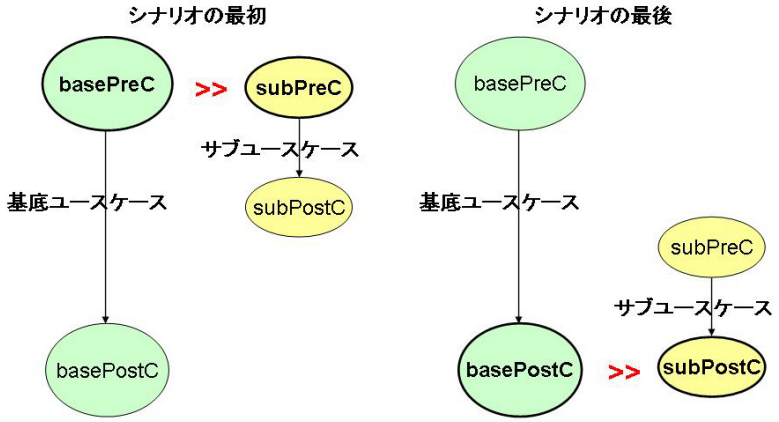

図 2.6: サブユースケースを包含できる箇所

ここでは，Design by Contract に基づくモジュラー検証のように，プログラムの正しさを確かめているのではなくて，単なる包含可能性を調べているだけである．ここでは，基底ユースケースの事前条件 ≫ サブユースケースの事前条件，基底ユースケースの事後条件 ≫ サブユースケースの事後条件が成立するかどうかを調べている．

勿論，シナリオの途中に包含できないわけではないが，途中に包含する場合は図 2.7 に示すように，包含する箇所における，シナリオの一部の事後条件を設定する必要がある．

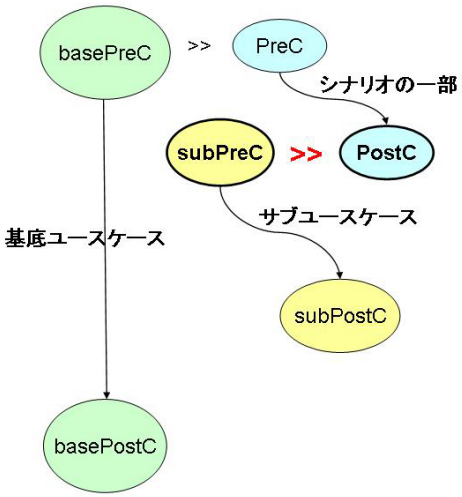

図 2.7: サブユースケースを途中に包含する場合

エレベータ制御システム [2] のユースケース記述「行き先を選択する」（図 2.3）を例に挙げるとシナリオ記述 1. の”利用者はエレベータボタンを押す．”がシナリオの一部に該当する．また，図 2.3 では，省略されているが，シナリオ記述 3. の後にシナリオの一部として，”利用者はエレベータから降りる”が入ると考えられる．図 2.3 の基底ユースケース「行き先を選択する」の構造は準含意関係を用いて，図 2.8 のように表すことができる．

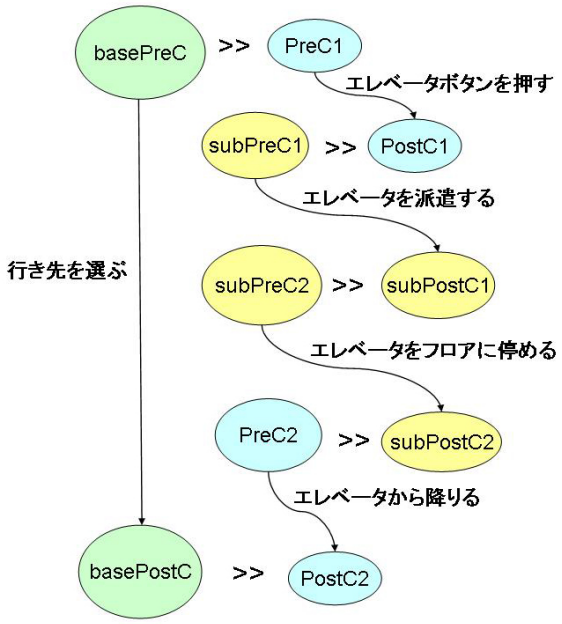

図 2.8: 準含意関係を用いて表した基底ユースケースの構造

6.2 でも述べるが，本研究で定義する準含意関係が成立してもサブユースケースを基底ユースケースに必ず include（包含）できるとは限らない．すなわち，準含意関係は基底ユースケースがサブユースケースと include（包含）関係を持つための必要条件であるが，必要十分条件ではない．

第3章　ユースケースから一階述語論理式への変換

本章では，自然言語で記述されたユースケースから一階述語論理の論理式への変換について説明する．本章では，エレベータ制御システム [2] を例に取り上げて説明する．

3.1　ユースケースにおける一階述語論理表現

まず，本稿では自然言語で記述されたユースケースの事前条件，事後条件を論理式で表現したものを事前条件，事後条件の制約と定義する．そして，事前条件，事後条件の制約を構成する各条件を部分制約と定義し，各部分制約を and でつないで事前条件，事後条件の制約を表す．さらに，部分制約を構成する各要素を事実と定義し，事実を and, or ,→ と組み合わせて部分制約を表す．事実は述語により表現される．以下にユースケースにおける一階述語論理表現のシンタックスを BNF 記法で示す．

⟨ 事前条件，事後条件の制約 ⟩ ::=	⟨ 部分制約を and で結んだもの ⟩
⟨ 部分制約を and で結んだもの ⟩ ::=	⟨ 部分制約 ⟩ |
	⟨ 部分制約 ⟩ and ⟨ 部分制約を and で結んだもの ⟩
⟨ 部分制約 ⟩ ::=	⟨ 事実の組み合わせ ⟩
⟨ 事実の組み合わせ ⟩ ::=	⟨→ 節 ⟩ | ⟨or 節 ⟩ | ⟨and 節 ⟩
⟨→ 節 ⟩ ::=	⟨or 節 ⟩ → ⟨or 節 ⟩
⟨or 節 ⟩ ::=	⟨and 節 ⟩ | ⟨and 節 ⟩ or ⟨or 節 ⟩
⟨and 節 ⟩ ::=	⟨ 事実 ⟩ | ⟨ 事実 ⟩ and ⟨and 節 ⟩
⟨ 事実 ⟩::=	述語 | ⟨ 否定 ⟩
⟨ 否定 ⟩ ::=	not 述語

3.1.1　述語

述語は述語名，引数のセットから構成され，述語名 (引数 1, 引数 2,..., 引数 N) と表記する．引数はいくつでも指定することができる．次に，事実を表現する際の，述語の引数と述語名の意味を説明する．

引数

ユースケース図におけるアクター，もしくは事実を決める重要概念を表す．事実を決める重要概念とは，事実を述語で記述する際のキーポイントになる名詞である．表3.1 に例を示す.

表 3.1: 引数の例

引数の種類	引数の例	意味
アクター	user	ユースケース図のアクターである利用者
事実を決める重要概念	elevator	エレベータ
	userFloorNumber	user がエレベータを要求した階
	userDestinationNumber	user の目的階

述語名

引数の状態，引数の役割を事実として表現する.

表 3.2 に述語名の例を示す.

表 3.2: 述語名の例

述語名の種類	述語名の例	意味
引数 (アクター) の状態	beOnTheFloor(user, userFloorNumber)	user がエレベータを要求した階にいる
引数 (事前条件，事後条件の制約を決める重要概念) の状態	motorOff(elevator)	elevator のモータは停止している
役割	fastElevator(elevator)	elevator は高速エレベータである

3.2　表現方法

3.2.1　述語の表現

引数の洗い出し

事実は述語により表現されるが，初めに述語の引数を洗い出す．引数の洗い出しは，要求分析で設計されたユースケース図とシステム分析で設計されたクラス図を用いて，以下の手順で行う．

1. クラス図の集約とコンポジション集約に注目し，全体－部分の関係において，全体を表すクラスを引数の候補とする．なぜなら，本稿の表現方法では，部分を表すクラスの状態や役割を，全体を表すクラスを用いて表すからである．もし，集約とコンポジション集約の関係がクラス図になければ，関連の多重度において他のクラスが多に対し，1 になっているクラスを候補とする．

2. ユースケース図のアクターを引数の候補に入れる．しかし，アクターとなるクラスが，1 で候補としたクラスと，集約又コンポジション集約関係において部分を表すクラスになっているか，1 対多の関連を持つ場合，候補からはずす．なぜなら，本稿の表現方法では，アクターとなるクラスの状態や役割を，全体を表すクラスを用いて表すからである．

3. 1 と 2 で引数の候補に入れたものの中から，複数の意味を持つものは，それぞれ別の名前にする．

例として，エレベータ制御システムのクラス図（図 3.1）を用いて説明する．

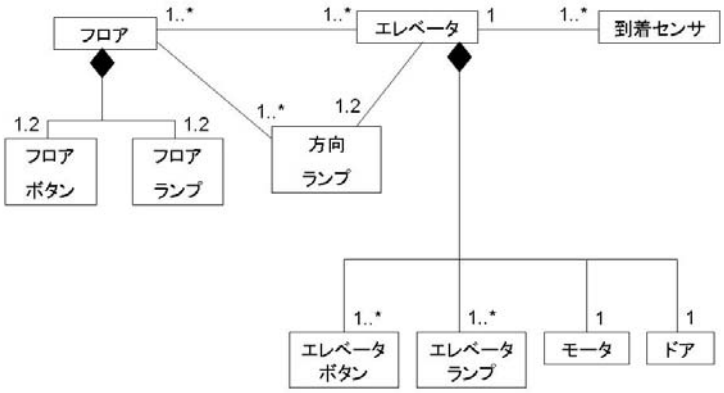

図 3.1: エレベータ制御システム クラス図

図 3.1 より，まずフロアクラスとエレベータクラスがコンポジットクラスになっているので，これらを引数の候補にする．次にユースケース図（図 2.2）より，アクターである利用者と到着センサが引数の候補として考えられるが，到着センサは図 3.1 でエレベータクラスと 1 対多の関係を持つので，候補からはずす．よって，エレベータ制御システムによる引数の候補はフロア，エレベータ，利用者となる．ここで，フロアに関しては，利用者がエレベータを要求した階，利用者の目的階，エレベータが到着している階と 3 種類の意味を持つので，それぞれ別の名前で候補にする．

述語の記述

各ユースケースの事前条件，事後条件において，クラス図を用いて，各クラスが満たすべき状態と役割を表現する．この際，引数に対して述語名を決定する．例えば，モータクラスにおいて電源が入っていなければならない状態を述語で表現する場合，motorOn(elevator) と記述する．

3.2.2 事実（述語と述語を否定したもの）の表現

事実は，

$$\langle \text{事実} \rangle ::= \text{述語} \mid \langle \text{否定} \rangle$$

で表現される．述語の表現は3.2.1で説明した．⟨否定⟩は，述語の否定を表現する．例えば，エレベータのドアが閉まっていてはいけない事実を表現する場合，not doorClosed(elevator)と記述する．

3.2.3 部分制約（事実の組み合わせ）の表現

部分制約は，

⟨部分制約⟩ ::= ⟨事実の組み合わせ⟩
⟨事実の組み合わせ⟩ ::= ⟨→節⟩ | ⟨or 節⟩ | ⟨and 節⟩

で表現される．

or 節

「A の事実または B の事実」を表現するのに用いる．例えば，「利用者がフロアから上昇エレベータまたは下降エレベータを要求している」を部分制約で表現する場合，needAscentElevator(user, userFloorNumber) or needDescentElevator(user, userFloorNumber と記述する．

and 節

「A の事実かつ B の事実」を表現するのに用いる．例えば，「利用者の目的階のエレベータボタンが押されていて，エレベータランプが点灯している」を部分制約で表現する場合，pushedElevatorButton(elevator, userDestinationNumber) and lightElevatorLamp(elevator, userDestinationNumber) と記述する．

→ 節

「A の事実ならば B の事実」を表現するのに用いる．この節は．条件付きの部分制約を表現する際に用いる．例えば，「エレベータが利用者からの要求がない場合，方向ランプを消灯する」を部分制約で表現する場合，not haveDestination(elevator, user) → not lightDirectionLamp(elevator) と記述する．

3.2.4 事前条件，事後条件の制約の表現

事前条件，事後条件の制約は

⟨ 事前条件，事後条件の制約 ⟩ ::= ⟨ 部分制約を and で結んだもの ⟩
⟨ 部分制約を and で結んだもの ⟩ ::= ⟨ 部分制約 ⟩ |
⟨ 部分制約 ⟩ and ⟨ 部分制約を and で結んだもの ⟩

で表現され，3.2.3 で説明した部分制約を and で結んで表現する．

第4章　サブユースケース抽出アルゴリズム

4.1　アルゴリズムの概要

2.3.1で述べた，準含意（≫）関係を定理証明技法で調べることにより，基底ユースケースに包含できる可能性があるサブユースケースを抽出するアルゴリズムを提案する．既存のシステムに新たなユースケースを追加する際，そのユースケースを基底ユースケースとして，包含できる可能性があるサブユースケースの組み合わせを，このアルゴリズムにより抽出できる．初めに，アルゴリズムの入力と出力について述べる．入力は新たに追加するユースケースの事前条件，事後条件を一階述語論理の論理式で表したものである．出力は新たに追加するユースケースが包含できる可能性がある既存のサブユースケースの組み合わせである．なお，可能な限り全ての候補を出力する．図4.1にアルゴリズムの概要を示す．

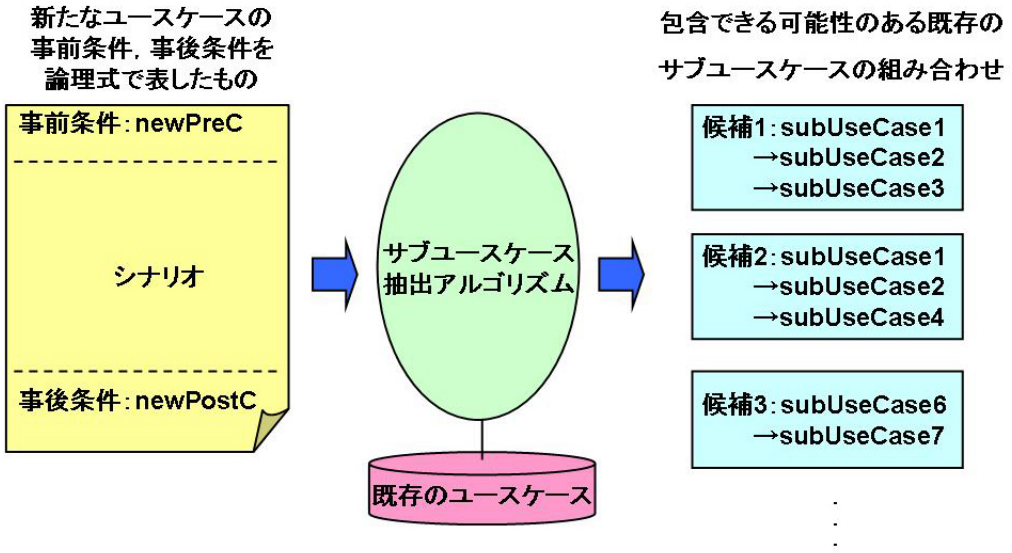

図 4.1: アルゴリズムの概要

4.2　アルゴリズムの詳細

サブユースケース抽出アルゴリズムは，事前条件からの抽出と事後条件からの抽出の両方を行う．どちらも，深さ優先探索により，以下の各項で説明する一つ目のサブユースケース抽出処理，二つ目以降のサブユースケース抽出処理を順番に行っていき，それぞれの処理において準含意関係が成立するか調べていく．そして，サブユースケースの可能な限りの組み合わせを全て候補として保持し，最後に全候補を出力する．以下で，事前条件からの抽出における，アルゴリズムの一つ目のサブユースケース抽出処理，二つ目以降のサブユースケース抽出処理について説明する．

4.2.1　一つ目のサブユースケース抽出処理

新たなユースケースの事前条件の論理式 X について，X ≫ Y を満たす論理式 Y を事前条件として持つ既存のユースケースを抽出する（図 4.2）．抽出に成功すれば，そのユースケースを候補として保持する．

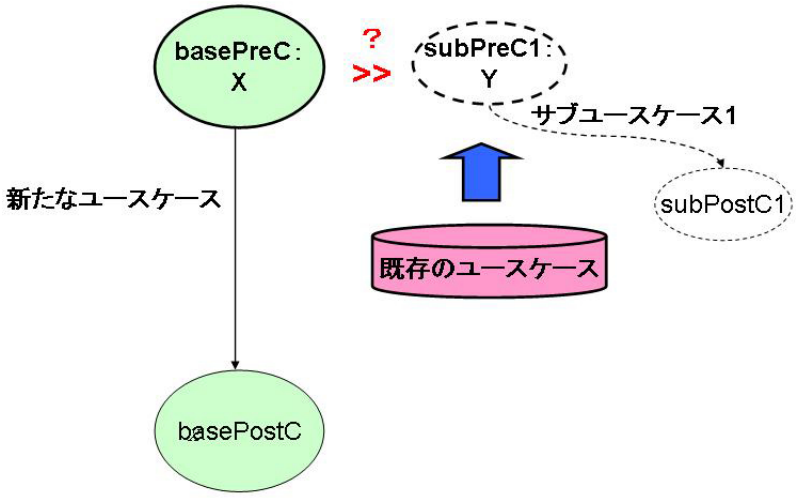

図 4.2: 一つ目のサブユースケース抽出処理

4.2.2 二つ目以降のサブユースケース抽出処理

4.2.1 で抽出したユースケースの事後条件の論理式を Y' としたとき，$X' \gg Y'$ を満たす論理式 X' を事前条件として持つ既存のユースケースを抽出する（図 4.3）.

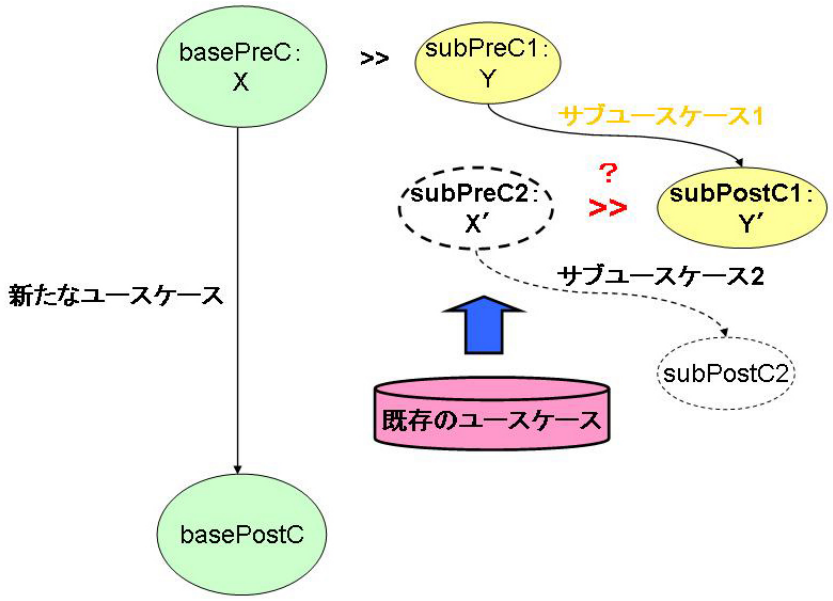

図 4.3: 二つ目以降のサブユースケース抽出処理

以降は，抽出したユースケースに対して，同様に準含意関係による既存のユースケースの抽出を行っていき，ユースケースを抽出できなくなるまで続ける．抽出に成功する毎に，それまでに抽出したユースケースの組み合わせを候補として保持する．

4.2.3 事後条件からの抽出

事後条件からの抽出については，一つ目のサブユースケース抽出処理で新たなユースケースの事後条件の論理式を X としたとき，X ≫ Y を満たす論理式 Y を事後条件として持つ既存のユースケースを抽出する（図 4.4）.

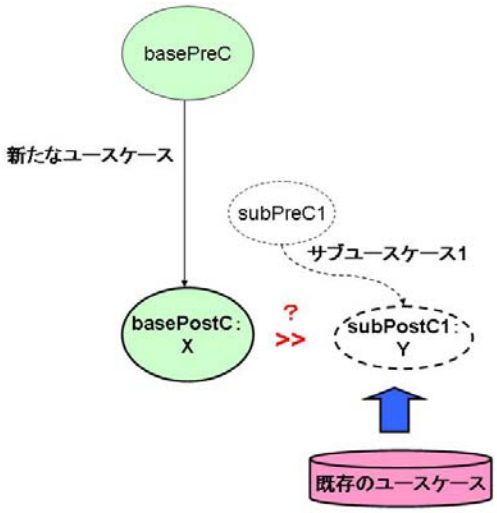

図 4.4: 事後条件からの抽出における，一つ目のサブユースケース抽出処理

二つ目以降のサブユースケース抽出処理では，まず一つ目のサブユースケース抽出処理で抽出したユースケースの事前条件の論理式を X′ としたとき，X′ ≫ Y′ を満たす論理式 Y′ を事後条件として持つ既存のユースケースを抽出する（図 4.5）.

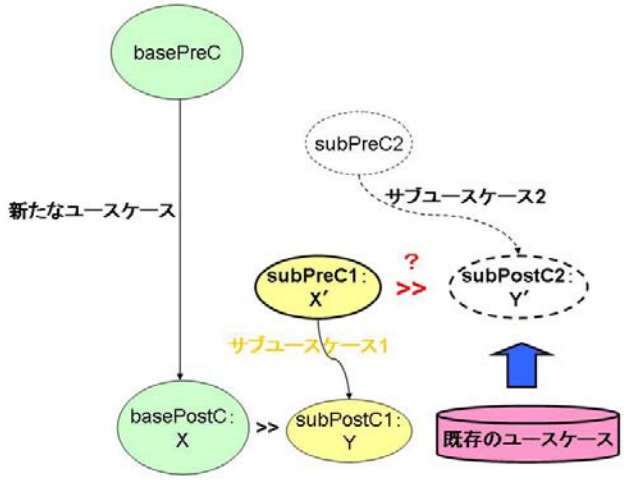

図 4.5: 事後条件からの抽出における，二つ目以降のサブユースケース抽出処理

以降は，抽出したユースケースに対して，同様に準含意関係による既存のユースケースの抽出を行っていき，ユースケースを抽出できなくなるまで続ける．抽出に成功する毎に，それまでに抽出したユースケースの組み合わせを候補として保持する．

第5章　ツールの開発

5.1　サブユースケース抽出アルゴリズムの実現

4 章で述べたサブユースケース抽出アルゴリズムを実現するツールを論理型言語 prolog で開発した（付録 B 参照）．prolog の処理系は SWI-Prolog[5] を用いた．prolog で開発した理由は，4 で述べた準含意関係を調べる処理で，論理式を扱うため，論理型言語を用いるのが適切だと考えたからである．また，サブユースケースを抽出する際に，再帰処理が必要となるので，これに関しても論理型言語が適していると考えた．

5.2　ツールの仕様

5.2.1　実行環境

表 5.1 にツールの実行環境を示す．

表 5.1: システムの実行環境

prolog 処理系	SWI-Prolog version 5.6.25
OS	Linux, most Unix platforms and Windows NT/2000/XP.

5.2.2　機能の概要

本ツールにより，既存のシステムに新たなユースケースを追加する際，そのユースケースを基底ユースケースとして，包含できる可能性があるサブユースケースの組み合わせを抽出できる．このツールを用いてサブユースケースの抽出を試みる際，既存のユースケースの情報（ユースケース名，事前条件の制約，事後条件の制約）をあらかじめ規則としてソースプログラム内に記述しておく必要がある（付録 B 参照）．5.2.3 に登録を行う際のシンタックスを説明し，5.2.4 にツールの実行方法について説明する．

5.2.3　ユースケースの登録

この項では，ユースケースの情報（ユースケース名，事前条件の制約，事後条件の制約）の登録を行う際のシンタックスを説明する．

ユースケース名

ユースケース名は半角英数字で記述し，頭文字を英小文字にする．頭文字以外の文字は，英大文字でも数字でもよい．これは，頭文字を英大文字にするとprologの処理系がプログラム内の変数と解釈してしまうためである．以下に正しいユースケース名の例と誤ったユースケース名の例を示す．

正しいユースケース名の例：　oldUseCase1, selectElevator, requestElevator
誤ったユースケース名の例：　OldUseCase1, 101UseCase

事前条件の制約，事後条件の制約

本研究のアプローチでは，ユースケースの事前条件，事後条件を一階述語論理の論理式で記述するが，本ツールを使用する際，ツールの仕様に従って記述する必要がある．

事前条件，事後条件として，入力する事前条件，事後条件の制約のシンタックスをBNF記法で以下に示す．3.1と照らし合わせて参照されたい．

⟨ 事前条件，事後条件の制約 ⟩ ::= '['⟨ 部分制約を and で結んだもの ⟩']'
⟨ 部分制約を and で結んだもの ⟩ ::= ⟨ 部分制約 ⟩ |
⟨ 部分制約 ⟩','⟨ 部分制約を and で結んだもの ⟩
⟨ 部分制約 ⟩ ::= '['⟨ 部分制約 ⟩']' |
⟨and 節 ⟩ |
⟨or 節 ⟩
⟨or 節 ⟩ ::= '['⟨or 節 ⟩']' |
⟨and 節 ⟩ |
⟨and 節 ⟩ '##' ⟨or 節 ⟩
⟨and 節 ⟩ ::= '['⟨and 節 ⟩']' |
⟨ 事実 ⟩ |
⟨ 事実 ⟩ '&&' ⟨and 節 ⟩
⟨ 事実 ⟩ ::= 述語 |
⟨ 否定 ⟩
⟨ 否定 ⟩ ::= '~' 述語

なお，3.1 で述べた ⟨→ 節 ⟩ は A→B を~A##B と変換して登録する必要がある．述語を構成する述語名と引数は，ユースケース名と同じシンタックスで記述する．

ユースケースの登録例

prolog のソースプログラム内にユースケースの情報（ユースケース名，事前条件の制約，事後条件の制約）を登録する際，Name にユースケース名，Precondition に事前条件の制約，Postcondition に事後条件の制約で単一化代入を行う．単一化代入とは，prolog でそれらの項を同一にする代入のことである [8]．以下の図 5.1 に prolog のソースプログラム内での登録例を示す．

```
/*oldUseCase1*/
usecase(Name, Precondition, Postcondition):-
 Name=oldUseCase1,
 Precondition=[predicate1(x),predicate2(y),predicate4(x, z)],
 Postcondition=[predicate8(x),predicate9(y)].
/*oldUseCase2*/
usecase(Name, Precondition, Postcondition):-
 Name=oldUseCase2,
 Precondition=[predicate8(x),predicate9(y),~predicate4(x, z)],
 Postcondition=[predicate3(y),predicate4(y, z)].
/*oldUseCase3*/
usecase(Name, Precondition, Postcondition):-
 Name=oldUseCase3,
 Precondition=[predicate3(y),predicate4(y, z)##predicate7(z),predicate8(x)],
 Postcondition=[predicate5(x, y),predicate7(z)].
/*oldUseCase4*/
usecase(Name, Precondition, Postcondition):-
 Name=oldUseCase4,
 Precondition=[predicate3(y),predicate4(y, z),predicate8(x)],
 Postcondition=[predicate5(x, y),~predicate4(x, z)##predicate7(z)].
```

図 5.1: ソースプログラム内でのユースケースの登録例

5.2.4　ツールの実行

本節では，SWI-Prolog を用いた実行方法について説明する．なお，OS は Windows XP で，SWI-Prolog のインストールが済んでいる状態を想定して説明する．

1. プログラムの起動

初めに，SWI-Prolog を起動する．図 5.2 に SWI-Prolog を起動後に表示されるウィンドウを示す．

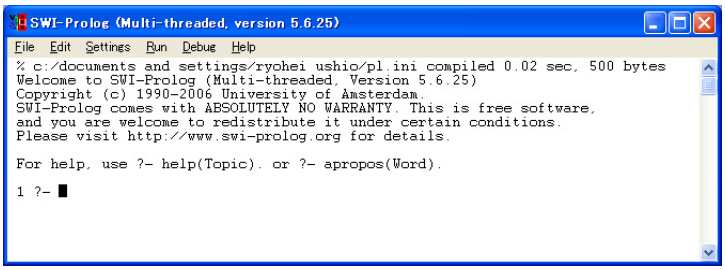

図 5.2: 起動後に表示されるウィンドウ

2. プログラムファイルの読み込み

起動後のウィンドウからメニューバーの File をマウスでクリックし，ドロップダウンリストから Consult をクリックする．クリック後，ファイル選択のダイアログボックスが表示されるので，拡張子 pl の付いたプログラムファイルを選択し，実行ボタンを押す．図 5.3 にクリック後に表示されるダイアログボックスを示す．

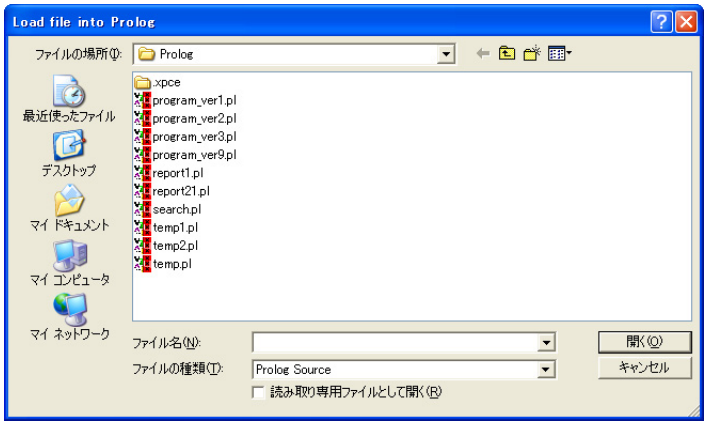

図 5.3: ファイル選択ダイアログボックス

3. 実行

プログラムファイルの読み込み後，プログラムにエラーがなければ，読み込み成功となり，図 5.4 のようなメッセージが表示され質問文の入力が可能になる．

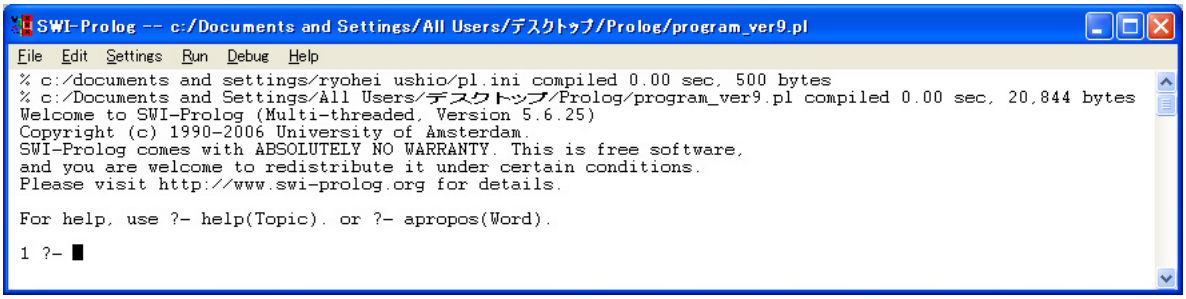

図 5.4: ファイル読み込み後画面

4. 質問文の入力

prolg は，対話型インタプリタなのでプログラムを実行した後,「?-」以降に質問文を入力して使用する．本ツールでは，プログラムファイルの読み込み後,「?-」以降に

main([新たなユースケースの事前条件の制約],[新たなユースケースの事後条件の制約]).

を入力して Enter ボタンを押す．新たなユースケースの事前条件の制約と新たなユースケースの事後条件の制約の表現方法は，5.2.3 で述べたユースケースを登録する場合と同じである．本ツールの出力は，新たなユースケースに対する包含可能なサブユースケースの抽出に成功した場合，

useCaseList_SearchedFromPrecondition = [[候補 1], [候補 2], [候補 3],...,[候補 N]]
useCaseList_SearchedFromPostcondition = [[候補 1], [候補 2], [候補 3],...,[候補 N]]

と出力され，useCaseList_SearchedFromPrecondition には事前条件からの抽出，useCaseList_SearchedFromPostcondition には事後条件からの抽出により，可能な限りの候補が全部出力される．[候補 N] の中身は [一つ目のサブユースケース名，二つ目のサブユースケース名，三つ目のサブユースケース名，...] と表され，新たなユースケースに包含できる可能性があるサブユースケースの組み合わせを包含する順番に出力する．さらに，出力として，useCaseList_SearchedFromPrecondition と useCaseList_SearchedFromPostcondition で発見できたユースケースの情報（ユースケース名, 事前条件の制約，事後条件の制約）を useCaseInformation の下に表示する．

もし，新たなユースケースに対する包含可能なサブユースケースの抽出に失敗し，一つも候補が見つからなかった場合，No. と出力する．

例として，図 5.1 で示したユースケースの情報がソースプログラム中に登録されている場合に，新たなユースケースの事前条件の制約を

[predicate1(x),predicate2(y)##predicate3(y),predicate4(x, z)],

新たなユースケースの事後条件の制約を

[predicate5(x, y),˜predicate6(y, z)&&predicate7(z)]

とし，

main([predicate1(x),predicate2(y)##predicate3(y),predicate4(x, z)], [predicate5(x, y),˜predicate6(y, z)&&predicate7(z)]).

と実行した場合の実行結果を図 5.5 に示す.

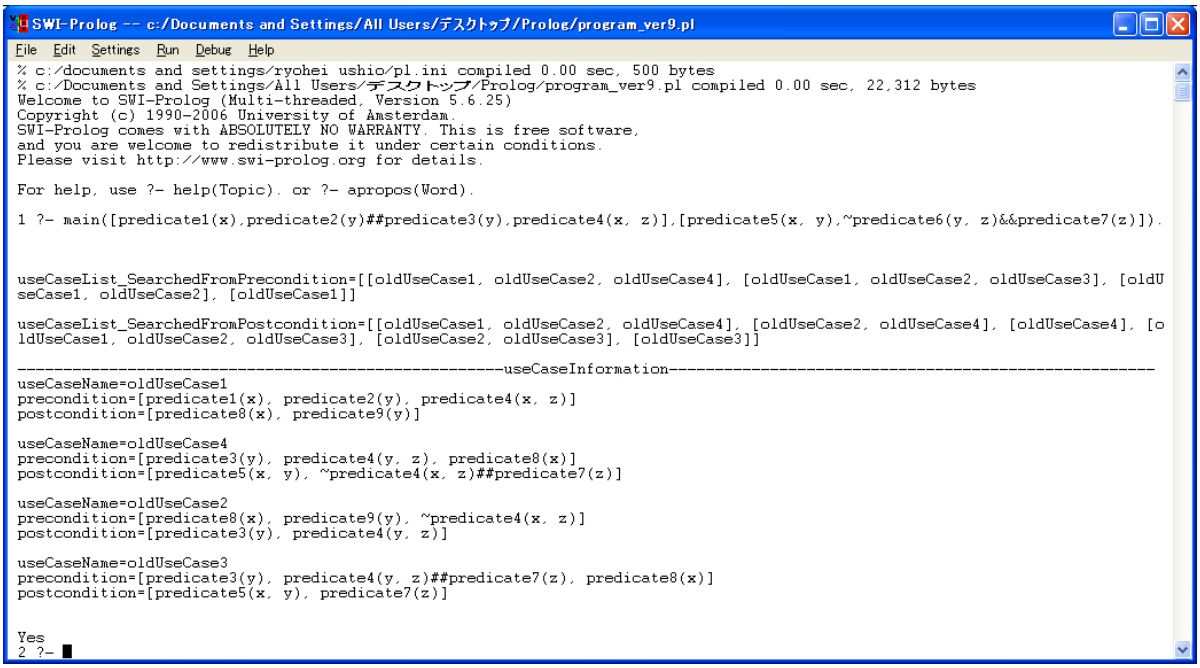

図 5.5: 質問文の実行結果

5. 終了

ウィンドウの閉じるボタンを押すか，質問文で halt. と入力し，Enter ボタンを押す.

第6章　適用例

4章で述べたサブユースケース抽出アルゴリズムをエレベータ制御システムの事例研究[2]を題材に適用を試みた．以下に，エレベータ制御システムの事例研究[2]のユースケース図（図6.1）とユースケース記述「エレベータを要求する」（図6.2），ユースケース記述「エレベータで行き先を選択する」（図6.3），ユースケース記述「エレベータを派遣する」（図6.4），ユースケース記述「エレベータをフロアに停める」（図6.5）を示す．

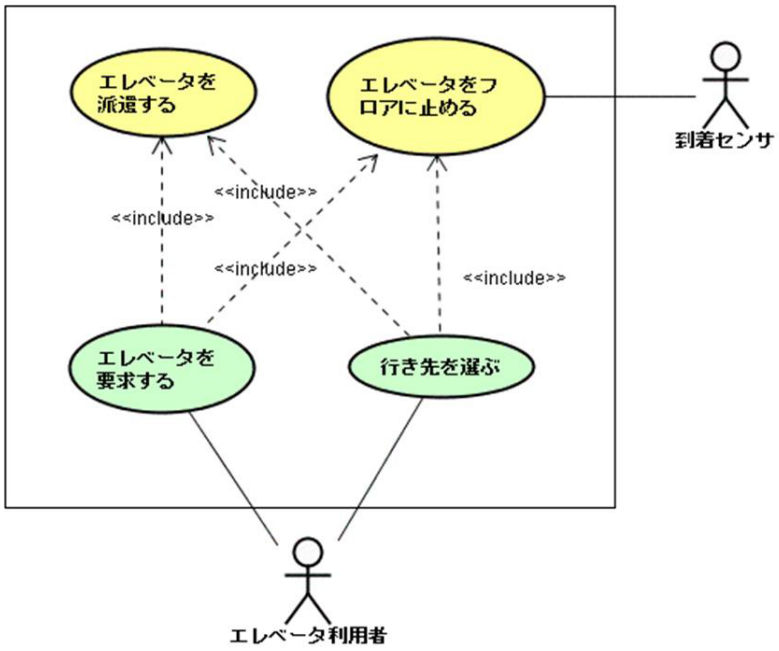

図 6.1: エレベータ制御システムのユースケース図

ユースケース名：エレベータを要求する
概要：ある階にいる利用者が，エレベータを呼ぶため上昇または下降フロアボタンを押す．
アクター：エレベータ利用者，到着センサ
事前条件：利用者がある階にいて，エレベータを必要としている．
記述：

１．利用者は上昇フロアボタンを押す．フロアボタンセンサは，ボタンが押された階を識別しつつ，システムに利用者要求を送る．
２．システムはその階に行くべきエレベータを選択する．新しい要求が行き先階リストに追加される．include ”エレベータを派遣する ”
３．include ”エレベータをフロアに停める ”
— 他に要求があるときには，エレベータは利用者によって要求されたこれらの階を訪問する．最終的にはエレベータは利用者によって選択された目的階に到着する．

例外記述：

１．利用者が下降エレベータボタンを押す．システムの応答は主系列と同じである．
２．もし，エレベータが目的階にいて，移動すべき階がない場合には，システムはその階に，ドアを開けたまま留まる．

事後条件：エレベータは利用者によって選択された階に到着している．

図 6.2: ユースケース記述「エレベータを要求する」

ユースケース名：行き先を選ぶ
概要：エレベーターの中にいる利用者は，移動すべき目的階を選ぶためにエレベータボタンを押す．
アクター：エレベータ利用者，到着センサ
事前条件：利用者がエレベータ内にいる．
記述：

１．利用者はエレベータボタンを押す．エレベータボタンセンサは，利用者が行きたい目的階を識別しつつ，システムにエレベータボタン要求を送る．
２．新しい要求が行き先階リストに追加される．もし，エレベータがアイドル状態の場合，include ”エレベータを派遣する ”
３． include ”エレベータをフロアに停める ”
— 他に要求があるときには，エレベータは利用者によって要求されたこれらの階を訪問する．最終的にはエレベータは利用者によって選択された目的階に到着する．

例外記述：

１．利用者が下降エレベータボタンを押す．システムの応答は主系列と同じである．
２．もし，エレベータが目的階にいて，移動すべき階がない場合には，システムはその階に，ドアを開けたまま留まる．

事後条件：エレベータは利用者によって選択された階に到着している．

図 6.3: ユースケース記述「行き先を選択する」

ユースケース名：エレベータを派遣する
概要：
アクター：
事前条件：エレベータは最低一つ訪問すべき階を持つ.
記述：

システムは，次の要求を満たすために，エレベータがどちらの方向に動くべきかを決定する．システムはドアを閉めるように指示する．ドアが閉まったとき，システムは，上昇または下降のためにエレベータを動かすため，モータに指示する.

例外記述：
事後条件：エレベータは指示された階に移動中である.

図 6.4: ユースケース記述「エレベータを派遣する」

ユースケース名：エレベータをフロアに停める
概要：
アクター：到着センサ
事前条件：エレベータは移動中である.
記述：

エレベータがある階から別の階に移動中に，到着センサは移動先に接近中であることを検知し，システムに通知する．システムはその階で停止すべきか否かを決定する．もし，停止すべきときは，システムはモータに停止を指示する．エレベータが停止したとき，システムはエレベータドアを開くように指示する.

例外記述：エレベータがその階で停まる必要がない場合はその階を通過する.
事後条件：エレベータはその階に停止し，ドアを開けている.

図 6.5: ユースケース記述「エレベータをフロアに停める」

6.1 適用内容

初めに，基底ユースケース「エレベータを要求する」，基底ユースケース「行き先を選択する」より，シナリオの一部を洗い出し，事前条件，事後条件を設定して一階述語論理の論理式で表す．次に，ユースケース図（図 6.1）の全てのユースケースに対して，事前条件，事後条件を論理式で記述し，シナリオの一部とユースケースを 5 章で述べたツールのソースプログラム内に記述した．次に，エレベータ制御システムの全てのユースケースに対して，事前条件，事後条件を一階述語論理の論理式で記述し，5 章で述べたツールのソースプログラム内に記述した．次に，エレベータ制御システムに新たなユースケースとして，A 階から B 階へは停まらず，B 階から各階に停まる高速エレベータに関するユースケース「高速エレベータを要求する」「高速エレベータで行き先を選択する」を追加した．そして，それらの事前条件，事後条件を一階述語論理の論理式で表し，5 のツールに入力として与え，サブユースケースの抽出を試みた．以下に，新たなユースケース追加後のユースケース図（図 6.6）と新たなユースケースのユースケース記述「高速エレベータを要求する」（図 6.7），ユースケース記述「高速エレベータで行き先を選択する」（図 6.8）を示す．

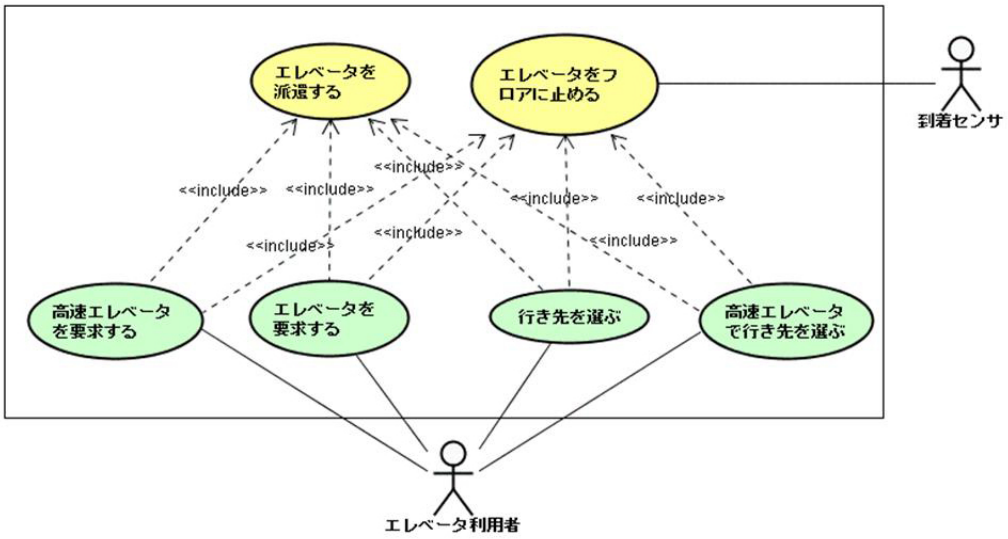

図 6.6: 新たなユースケース追加後のエレベータ制御システムのユースケース図

ユースケース名：高速エレベータを要求する
概要：ある階にいる利用者が，高速エレベータを呼ぶため上昇または下降フロア
　　ボタンを押す.
アクター：エレベータ利用者，到着センサ
事前条件：利用者が高速エレベータが止まる階にいて，高速エレベータを必要としている.
記述：
　　1．利用者は上昇フロアボタンを押す．フロアボタンセンサは，ボタン
　　が押された階を識別しつつ，システムに利用者要求を送る.
　　2．システムはその階に行くべきエレベータを選択する．新しい要求が
　　行き先階リストに追加される．include ”エレベータを派遣する ”
　　3．include ”エレベータをフロアに停める ”
　　— 他に要求があるときには，エレベータは利用者によって要求さ
　　れたこれらの階を訪問する．最終的にはエレベータは利用者に
　　よって選択された目的階に到着する.
例外記述：
　　1．利用者が下降エレベータボタンを押す．システムの応答は主
　　系列と同じである.
　　2．もし，エレベータが目的階にいて，移動すべき階がない場合に
　　は，システムはその階に，ドアを開けたまま留まる.
事後条件：高速エレベータは，利用者によって選択された階に到着している.

図 6.7: ユースケース記述「高速エレベータを要求する」

ユースケース名：高速エレベータで行き先を選ぶ
概要：高速エレベーターの中にいる利用者は，移動すべき目的階を選ぶためにエレベータボタンを押す．
アクター：エレベータ利用者，到着センサ
事前条件：利用者が高速エレベータ内にいる．
記述：

1．利用者はエレベータボタンを押す．エレベータボタンセンサは，利用者が行きたい目的階を識別しつつ，システムにエレベータボタン要求を送る．
2．新しい要求が行き先階リストに追加される．もし，エレベータがアイドル状態の場合，include ”エレベータを派遣する ”
3． include ”エレベータをフロアに停める ”

— 他に要求があるときには，エレベータは利用者によって要求されたこれらの階を訪問する．最終的にはエレベータは利用者によって選択された目的階に到着する．

例外記述：

1．利用者が下降エレベータボタンを押す．システムの応答は主系列と同じである．
2．もし，エレベータが目的階にいて，移動すべき階がない場合には，システムはその階に，ドアを開けたまま留まる．

事後条件：高速エレベータは，利用者によって選択された階に到着している．

図 6.8: ユースケース記述「高速エレベータで行き先を選ぶ」

6.1.1 適用結果

新たなユースケース「高速エレベータを要求する」「高速エレベータで行き先を選択する」の事前条件，事後条件を論理式で表現する際,「高速エレベータを要求する」のユースケースには,「エレベータは高速エレベータである」という意味の論理式 fastElevator(elevator) と「利用者がいる階は，高速エレベータの禁止階リストに入っていない」という意味の論理式 not(haveProhibitionFloorNumber(elevator, userFloorNumber)) をユースケース「エレベータを要求する」の事前条件，事後条件に追加して設定した．また,「高速エレベータで行き先を選択する」のユースケースには,「エレベータは高速エレベータである」という意味の論理式 fastElevator(elevator) と「利用者の行き先階は，高速エレベータの禁止階リストに入っていない」という意味の論理式 not(haveProhibitionFloorNumber(elevator, userDestinationNumber)) をユースケース「行き先を選択する」の事前条件，事後条件に追加して設定した．

ツールの出力結果より，一部抜粋したものを表6.1～表6.4に示す．図6.6で新たなユースケースが既存のユースケースとinclude（包含）関係にあるように，本適用では新たなユースケースのシナリオを設計した．つまり，あらかじめ包含可能なサブユースケースを把握していたので，ツールの出力結果について，サブユースケースの組み合わせが適切か不適切かを判断することができた．なお，表中の「上昇フロアボタンを押す」,「下降フロアボタンを押す」,「エレベータボタンを押す」,「エレベータから降りる」は新たなユースケースのシナリオの一部である．

表 6.1: 新たなユースケース「高速エレベータを要求する」の事前条件からの抽出による出力結果

		順番1	順番2	順番3	順番4
適切	候補1	上昇フロアボタンを押す	エレベータを派遣する	エレベータをフロアに停める	
	候補2	下降フロアボタンを押す	エレベータを派遣する	エレベータをフロアに停める	
不適切	候補1	上昇フロアボタンを押す	エレベータをフロアに停める		
	候補2	下降フロアボタンを押す	エレベータをフロアに停める		
	候補3	上昇フロアボタンを押す	エレベータを派遣する	エレベータをフロアに停める	エレベータから降りる
	候補4	下降フロアボタンを押す	エレベータを派遣する	エレベータをフロアに停める	エレベータから降りる

表 6.2: 新たなユースケース「高速エレベータを要求する」の事後条件からの抽出による出力結果

		順番1	順番2	順番3
適切	候補1	上昇フロアボタンを押す	エレベータを派遣する	エレベータをフロアに停める
	候補2	下降フロアボタンを押す	エレベータを派遣する	エレベータをフロアに停める
不適切	候補1	上昇フロアボタンを押す	エレベータをフロアに停める	
	候補2	下降フロアボタンを押す	エレベータをフロアに停める	
	候補3	エレベータボタンを押す	エレベータを派遣する	エレベータをフロアに停める

表 6.3: 新たなユースケース「高速エレベータで行き先を選択する」の事前条件からの抽出による出力結果

		順番1	順番2	順番3	順番4
適切	候補1	エレベータボタンを押す	エレベータを派遣する	エレベータをフロアに停める	エレベータから降りる
不適切	候補1	エレベータボタンを押す	エレベータをフロアに停める	エレベータから降りる	

表 6.4: 新たなユースケース「高速エレベータで行き先を選択する」の事後条件からの抽出による出力結果

		順番1	順番2	順番3	順番4
適切	候補1	エレベータボタンを押す	エレベータを派遣する	エレベータをフロアに停める	エレベータから降りる
不適切	候補1	エレベータボタンを押す	エレベータをフロアに停める	エレベータから降りる	
	候補2	エレベータボタンを押す	エレベータを派遣する	エレベータをフロアに停める	
	候補3	上昇フロアボタンを押す	エレベータを派遣する	エレベータをフロアに停める	エレベータから降りる
	候補4	下降フロアボタンを押す	エレベータを派遣する	エレベータをフロアに停める	エレベータから降りる

6.2 考察

ツールの適用の結果，あらかじめ把握していた包含可能なサブユースケースを抽出できた．すなわち，本研究で提案した手法で，新たなユースケースに対して包含できる可能性のあるサブユースケースを抽出できることを確認できた．不適切な候補が抽出されたことに関して，表 6.1〜表 6.4 よりそれらの候補を分析すると，以下のことがわかった．

- 適切な候補に比べて，途中の順番で出力されるはずのユースケースが抽出されていない．
- 適切な候補に比べて，余分なユースケースが抽出されているか，途中から別のユースケースが抽出されている．

前者については，途中に出力されるべきサブユースケースをサブユースケース N とすると，基底ユースケースの事前条件 ≫ サブユースケース N+1 の事前条件（図 6.9），基底ユースケースの事後条件 ≫ サブユースケース N-1 の事後条件（図 6.10），サブユース

ケース N+1 の事前条件 ≫ サブユースケース N-1 の事後条件（図 6.11）のいずれかが成り立ったと考えられる．

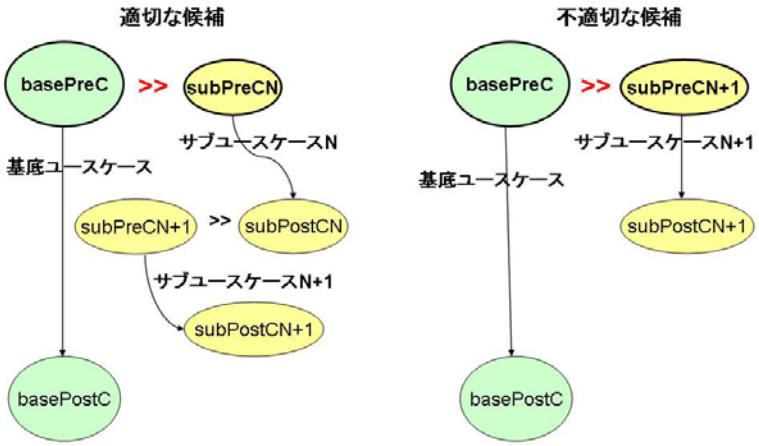

図 6.9: 途中に出力されるべきサブユースケース N が出力されなかった例 1

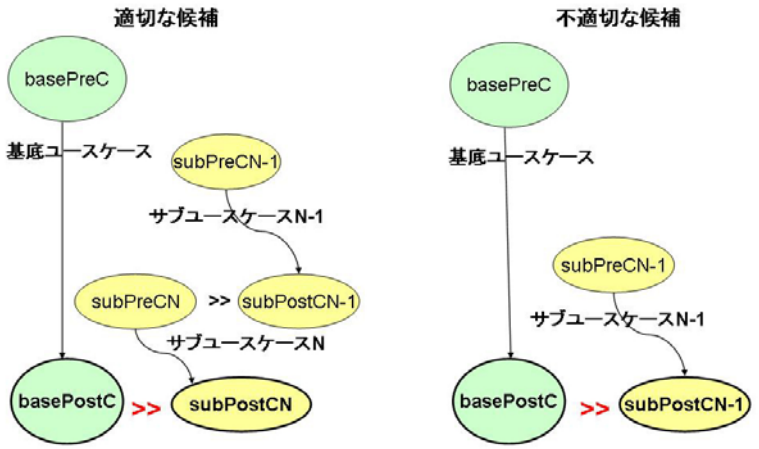

図 6.10: 途中に出力されるべきサブユースケース N が出力されなかった例 2

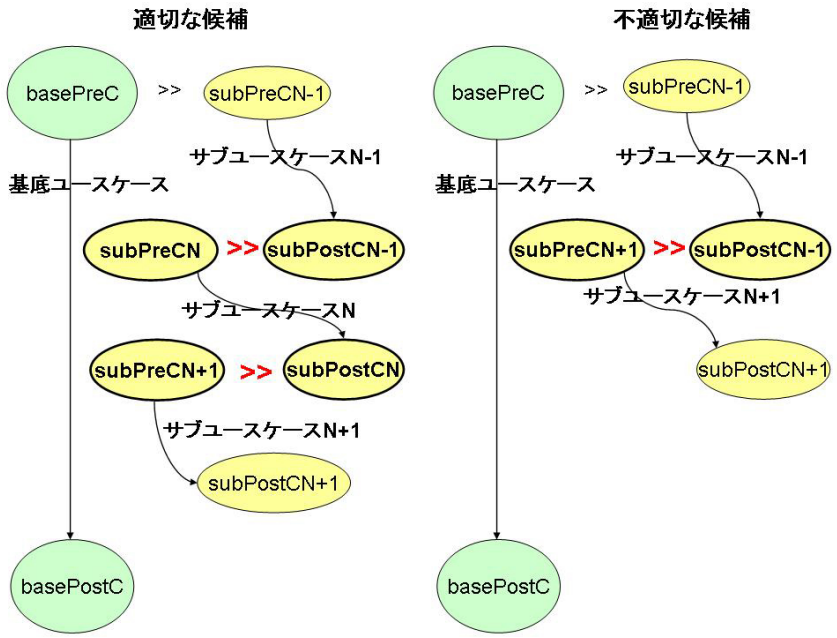

図 6.11: 途中に出力されるべきサブユースケース N が出力されなかった例 3

後者については，サブユースケース抽出アルゴリズムで，可能な限りのサブユースケースの組み合わせを出力した結果と考える．

前者，後者いずれにおいても，サブユースケース抽出アルゴリズムでは準含意関係を調べるので，不適切な候補が出力されるのは理論上避けられない問題である．今回の適用では，あらかじめ適切な候補を予想できた．しかし，一般的には既存のシステムに新たなユースケースを追加する際，関連する既存のユースケースを予想することはできない．よって，利用者が事前条件からの抽出による出力結果と事後条件からの抽出による出力結果を基に，出力された候補が適切か不適切かを検討する必要がある．

第7章　おわりに

7.1　まとめ

本研究では新たなユースケースの追加を行う方式を，利用者が既存のユースケースを再利用しつつ，定理証明システムを適用するものとして開発した．これを行うにあたり，ユースケースの事前条件，事後条件を一階述語論理の論理式で表現したものから，基底ユースケースとサブユースケースの論理式の間で成立すべき論理的関係（準含意関係）を定義した．この関係を用いて，新たなユースケースに包含できる可能性があるサブユースケースを抽出するアルゴリズムを提案し，そのアルゴリズムを論理型言語で実現した．

7.2　今後の課題

本研究で開発したツールは，新たなユースケースが既存のユースケースの組み合わせで実現できる場合，再利用可能なユースケースを抽出できるが，既存のユースケースの一部を変更して使用する場合には対応できない．よって，既存のユースケースの変更に伴う事前条件，事後条件の変更が他のユースケースにどのような影響を及ぼすかが調べられるようなシステムも実現したい．図 7.1 に既存のユースケースの事後条件を変更した場合の例を示す．

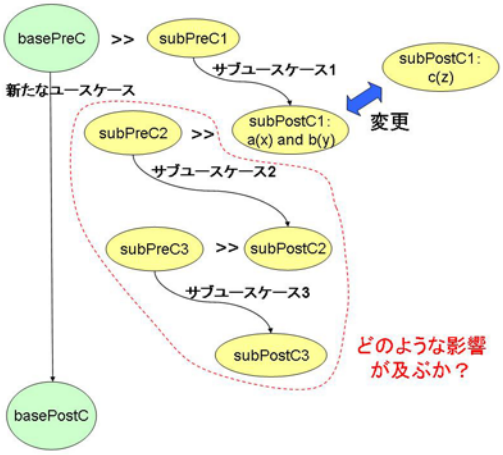

図 7.1: 既存のユースケースの一部の変更

謝辞

本研究の遂行にあたり，目的の設定から終わりまで貴重なご指導とご助言をいただいた北陸先端科学技術大学院大学 情報科学研究科 落水 浩一郎教授 工学博士に深く感謝致します.

本研究の審査員として，貴重なご意見とご助言をいただた片山 卓也教授 工学博士，鈴木 正人助教授 工学博士に深く感謝致します.

本研究の方針で迷ったときや数理論理学に関する内容の質問をしたとき，的確なご助言をいただいた北陸先端科学技術大学院大学 情報科学研究科 服部 哲助手 工学博士に深く感謝致します.

本研究を進めるにあたり，数々のご助言をいただいた北陸先端科学技術大学院大学 情報科学研究科 藤枝 和宏助手 工学博士に深く感謝致します.

日頃より研究に関する相談に乗っていただき，貴重なご助言をいただいたた北陸先端科学技術大学院大学 情報科学研究科 小谷 正行氏，早坂 良氏に深く感謝致します.

また，研究を進めるに当たり，貴重な意見をいただきましたた北陸先端科学技術大学院大学 情報科学研究科 ソフトウェア計画構成学講座の皆様に心より感謝いたします.

最後に学業の理解と学生生活を支えてくださった家族，親戚，ならびに，私生活で支えてくださった友人に深く感謝致します.

参考文献

[1] 竹政昭利, はじめて学ぶUML, ナツメ社, 2003.

[2] H.Gomma, Designing Concurrent, Distributed, and Real-time Application with UML, Addison-Wesley, 2000.

[3] 小野寛晰, 情報科学における論理, 日本評論社, 1994.

[4] C.Nebut, F.Fleurey, Y.Le Traon, J.-M.Jezequel, ”Automatic Test Generation : A Use Case Driven Approach ”, IEEE TRANSACTIONS ON SOFTWARE ENGENEERING, NO.3, 2006.

[5] SWI-Prolog's Home, http://www.swi-prolog.org/

[6] Project Triskell, http://www.irisa.fr/triskell/Softwares/protos/ucts/

[7] A.Cockburn, ユースケース実践ガイド, 山岸耕二, 矢崎博英, 水谷雅宏, 篠原明子訳, 翔泳社, 2001.

[8] L.Sterling, E.Shapiro, 松田利夫訳, Prologの技芸, 共立出版, 1988.

付 録A　ユースケースの一階述語論理による表現

A.1　ユースケース

```
ユースケース名：エレベータを要求する
事前条件：beOnTheFloor(user, userFloorNumber)  and
          ( needAscentElevator(user, userFloorNumber) or
            needDescentElevator(user, userFloorNumber) )
事後条件：arrived(elevator, userFloorNumber)  and
          doorOpened(elevator, userFloorNumber)  and
          arrivalSensorDetectedApproach(elevator, userFloorNumber)  and
          motorOff(elevator)  and
          ( not pushedAscentFloorButton(userFloorNumber) or
            not pushedDescentFloorButton(userFloorNumber) )  and
          not arrived(elevator, elevatorFloorNumber)  and
          not doorOpened(elevator, elevatorFloorNumber)  and
          not doorClosed(elevator)  and
          not arrivalSensorDetectedApproach(elevator, elevatorFloorNumber)  and
          not motorOn(elevator)  and
          not move(elevator, userFloorNumber, elevatorFloorNumber)  and
          ( needAscentElevator(user, userFloorNumber) →
                     not needAscentElevator(user, userFloorNumber) )  and
          ( needDescentElevator(user, userFloorNumber) →
                     not needDescentElevator(user, userFloorNumber) )  and
          ( not haveDestination(elevator, otherUserFloorNumber) →
                     not lightDirectionLamp(elevator) )  and
          ( needAscentElevator(otherUser, userFloorNumber) →
                     not needAscentElevator(otherUser, userFloorNumber) )  and
          ( needDescentElevator(otherUser, userFloorNumber) →
                     not needDescentElevator(otherUser, userFloorNumber) )
```

ユースケース名：行き先を選ぶ
事前条件：beInTheElevator(user, elevator) and
needToGo(user, userDestinationNumber) and
not pushedElevatorButton(elevator, userDestinationNumber)
事後条件：arrived(elevator, userDestinationNumber) and
doorOpened(elevator, userDestinationNumber) and
arrivalSensorDetectedApproach(elevator,userDestinationNumber) and
motorOff(elevator) and
beOnTheFloor(user, userDestinationNumber) and
not doorClosed(elevator) and
not motorOn(elevator) and
not beInTheElevator(user, elevator) and
not needToGo(user, userDestinationNumber) and
not move(elevator, userFloorNumber, userDestinationNumber) and
not arrived(elevator, userFloorNumber) and
not doorOpened(elevator, userFloorNumber) and
not arrivalSensorDetectedApproach(elevator, userFloorNumber) and
(haveDestination(elevator, otherUserFloorNumber) →
not lightDirectionLamp(elevator))

ユースケース名：エレベータを派遣する
事前条件：(haveDestination(elevator, userFloorNumber) or
haveDestination(elevator, userDestinationNumber)) and
motorOff(elevator) and
((pushedAscentFloorButton(userFloorNumber) and
lightAscentFloorLamp(userFloorNumber)) or
(pushedDescentFloorButton(userFloorNumber) and
lightDescentFloorLamp(userFloorNumber)) or
(pushedElevatorButton(elevator, userDestinationNumber) and
lightElevatorLamp(elevator, userDestinationNumber)))
事後条件：doorClosed(elevator) and
motorOn(elevator) and
lightDirectionLamp(elevator) and
not motorOff(elevator) and
(doorOpened(elevator, elevatorFloorNumber) →
(move(elevator, elevatorFloorNumber, userFloorNumber) and
not arrived(elevator, elevatorFloorNumber) and
not arrivalSensorDetectedApproach(elevator, elevatorFloorNumber) and
not doorOpened(elevator, elevatorFloorNumber))) and
(doorOpened(elevator, userFloorNumber) →
(move(elevator, userFloorNumber, userDestinationNumber) and
not arrived(elevator, userFloorNumber) and
not arrivalSensorDetectedApproach(elevator, userFloorNumber) and
not doorOpened(elevator, userFloorNumber)))

ユースケース名：エレベータをフロアに停める
事前条件：(haveDestination(elevator, userFloorNumber) or
haveDestination(elevator, userDestinationNumber)) and
(move(elevator, elevatorFloorNumber, userFloorNumber) or
move(elevator, userFloorNumber, userDestinationNumber)) and
doorClosed(elevator) and
motorOn(elevator) and
not motorOff(elevator) and
lightDirectionLamp(elevator) and
((pushedAscentFloorButton(userFloorNumber) and
lightAscentFloorLamp(userFloorNumber)) or
(pushedDescentFloorButton(userFloorNumber) and
lightDescentFloorLamp(userFloorNumber)) or
(pushedElevatorButton(elevator, userDestinationNumber) and
lightElevatorLamp(elevator, userDestinationNumber)))
事後条件：motorOff(elevator) and
not doorClosed(elevator) and
not motorOn(elevator) and
(haveDestination(elevator, userFloorNumber) →
(arrived(elevator, userFloorNumber) and
doorOpened(elevator, userFloorNumber) and
arrivalSensorDetectedApproach(elevator, userFloorNumber) and
not move(elevator, elevatorFloorNumber, userFloorNumber) and
not haveDestination(elevator, userFloorNumber))) and
(haveDestination(elevator, userDestinationNumber) →
(arrived(elevator, userDestinationNumber) and
doorOpened(elevator, userDestinationNumber) and
arrivalSensorDetectedApproach(elevator, userDestinationNumber) and
not move(elevator, userFloorNumber, userDestinationNumber) and
not haveDestination(elevator, userDestinationNumber))) and
(needAscentElevator(user, userFloorNumber) →
(not needAscentElevator(user, userFloorNumber) and
not pushedAscentFloorButton(userFloorNumber) and
not lightAscentFloorLamp(userFloorNumber))) and
(needDescentElevator(user, userFloorNumber) →
(not needDescentElevator(user, userFloorNumber) and
not pushedDescentFloorButton(userFloorNumber) and
not lightDescentFloorButton(userFloorNumber))) and
(needToGo(user, userDestinationNumber) →
(not pushedElevatorButton(elevator, userDestinationNumber) and
not lightElevatorLamp(elevator, userDestinationNumber))) and
(not haveDestination(elevator, otherUserFloorNumber) →
not lightDirectionLamp(elevator))

A.2　新たに追加したユースケース

```
ユースケース名：高速エレベータを要求する
事前条件：beOnTheFloor(user, userFloorNumber)  and
          ( needAscentElevator(user, userFloorNumber) or
            needDescentElevator(user, userFloorNumber) )  and
          fastElevator(elevator)  and
          not haveProhibitionFloorNumber(elevator, userFloorNumber)
事後条件：arrived(elevator, userFloorNumber)  and
          doorOpened(elevator, userFloorNumber)  and
          arrivalSensorDetectedApproach(elevator, userFloorNumber)  and
          motorOff(elevator)  and
          ( not pushedAscentFloorButton(userFloorNumber) or
            not pushedDescentFloorButton(userFloorNumber) )  and
          not arrived(elevator, elevatorFloorNumber)  and
          not doorOpened(elevator, elevatorFloorNumber)  and
          not doorClosed(elevator)  and
          not arrivalSensorDetectedApproach(elevator, elevatorFloorNumber)  and
          not motorOn(elevator)  and
          not move(elevator, userFloorNumber, elevatorFloorNumber)  and
          ( needAscentElevator(user, userFloorNumber) →
                        not needAscentElevator(user, userFloorNumber) )  and
          ( needDescentElevator(user, userFloorNumber) →
                        not needDescentElevator(user, userFloorNumber) )  and
          ( not haveDestination(elevator, otherUserFloorNumber) →
                        not lightDirectionLamp(elevator) )  and
          ( needAscentElevator(otherUser, userFloorNumber) →
                    not needAscentElevator(otherUser, userFloorNumber) )  and
          ( needDescentElevator(otherUser, userFloorNumber) →
                    not needDescentElevator(otherUser, userFloorNumber) )  and
          fastElevator(elevator)  and
          not haveProhibitionFloorNumber(elevator, userFloorNumber)
```

ユースケース名：高速エレベータで行き先を選ぶ
事前条件：beInTheElevator(user, elevator) and
needToGo(user, userDestinationNumber) and
not pushedElevatorButton(elevator, userDestinationNumber) and
fastElevator(elevator) and
not haveProhibitionFloorNumber(elevator, userDestinationNumber)
事後条件：arrived(elevator, userDestinationNumber) and
doorOpened(elevator, userDestinationNumber) and
arrivalSensorDetectedApproach(elevator,userDestinationNumber) and
motorOff(elevator) and
beOnTheFloor(user, userDestinationNumber) and
not doorClosed(elevator) and
not motorOn(elevator) and
not beInTheElevator(user, elevator) and
not needToGo(user, userDestinationNumber) and
not move(elevator, userFloorNumber, userDestinationNumber) and
not arrived(elevator, userFloorNumber) and
not doorOpened(elevator, userFloorNumber) and
not arrivalSensorDetectedApproach(elevator, userFloorNumber) and
(haveDestination(elevator, otherUserFloorNumber) →
not lightDirectionLamp(elevator)) and
fastElevator(elevator) and
not haveProhibitionFloorNumber(elevator, userDestinationNumber)

A.3 シナリオの一部

上昇フロアボタンを押す
事前条件：beOnTheFloor(user, userFloorNumber) and
needAscentElevator(user, userFloorNumber) and
not pushedAscentFloorButton(userFloorNumber)
事後条件：pushedAscentFloorButton(userFloorNumber) and
haveDestination(elevator, userFloorNumber) and
lightAscentFloorLamp(userFloorNumber)

下降フロアボタンを押す
事前条件：beOnTheFloor(user, userFloorNumber) and
needDescentElevator(user, userFloorNumber) and
not pushedDescentFloorButton(userFloorNumber)
事後条件：pushedDescentFloorButton(userFloorNumber) and
haveDestination(elevator, userFloorNumber) and
lightDescentFloorLamp(userFloorNumber)

エレベータボタンを押す
事前条件：beInTheElevator(user, elevator) and
needToGo(user, userDestinationNumber) and
not pushedElevatorButton(elevator, userDestinationNumber)
事後条件：haveDestination(elevator, userDestinationNumber) and
pushedElevatorButton(elevator, userDestinationNumber) and
lightElevatorLamp(elevator, userDestinationNumber)

エレベータから降りる
事前条件：beInTheElevator(user,elevator) and
arrived(elevator, userDestinationNumber) and
needToGo(user, userDestinationNumber) and
doorOpened(elevator, userDestinationNumber) and
motorOff(elevator) and
arrivalSensorDetectedApproach(elevator, userDestinationNumber) and
not doorClosed(elevator) and
not motorOn(elevator) and
not move(elevator, userFloorNumber, userDestinationNumber) and
not haveDestination(elevator, userDestinationNumber)
事前条件：beOnTheFloor(user, userDestinationNumber) and
not beInTheElevator(user, elevator) and
not needToGo(user, userDestinationNumber)\verb—

付 録B　開発ツール

B.1　ツールのソースプログラム

```
/* 演算子の定義 上から否定, 連言, 選言 */
:- op(900, fy, ~).
:- op(910, xfy, &&).
:- op(910, xfy, ##).

/*** 各ユースケースの情報 ***/
/* エレベータを要求する */
usecase(Name, Precondition, Postcondition):-
Name=requestElevator,
Precondition=[beOnTheFloor(user, userFloorNumber),
                needAscentElevator(user, userFloorNumber) ##
                    needDescentElevator(user, userFloorNumber),
                ~pushedAscentFloorButton(userFloorNumber) &&
                    ~pushedDescentFloorButton(userFloorNumber)],
Postcondition=[arrived(elevator, userFloorNumber),
                doorOpened(elevator, userFloorNumber) ,
                arrivalSensorDetectedApproach(elevator, userFloorNumber),
                motorOff(elevator), ~arrived(elevator, elevatorFloorNumber),
                ~doorOpened(elevator, elevatorFloorNumber),
                ~doorClosed(elevator),
                ~arrivalSensorDetectedApproach(elevator, elevatorFloorNumber),
                ~motorOn(elevator),
                ~move(elevator, userFloorNumber, elevatorFloorNumber),
                ~pushedAscentFloorButton(userFloorNumber) ##
                    ~pushedDescentFloorButton(userFloorNumber),
                ~needAscentElevator(user, userFloorNumber) ##
                    ~needDescentElevator(user, userFloorNumber),
                haveDestination(elevator, otherUserFloorNumber) ##
                    ~lightDirectionLamp(elevator),
                ~needAscentElevator(otherUser, userFloorNumber) ##
                    ~needAscentElevator(otherUser, userFloorNumber),
                ~needDescentElevator(otherUser, userFloorNumber) ##
                    ~needDescentElevator(otherUser, userFloorNumber)].
/* 行き先を選ぶ */
usecase(Name, Precondition, Postcondition):-
```

```
Name=selectDestination,
Precondition=[beInTheElevator(user, elevator),
                needToGo(user, userDestinationNumber),
                ~pushedElevatorButton(elevator, userDestinationNumber)],
Postcondition=[arrived(elevator, userDestinationNumber),
                doorOpened(elevator, userDestinationNumber),
                arrivalSensorDetectedApproach(elevator, userDestinationNumber),
                motorOff(elevator),
                beOnTheFloor(user, userDestinationNumber),
                ~doorClosed(elevator), ~motorOn(elevator),
                ~beInTheElevator(user, elevator),
                ~needToGo(user, userDestinationNumber),
                ~move(elevator, userFloorNumber, userDestinationNumber),
                ~arrived(elevator, userFloorNumber),
                ~doorOpened(elevator, userFloorNumber),
                ~arrivalSensorDetectedApproach(elevator, userFloorNumber),
                haveDestination(elevator, otherUserFloorNumber) ##
                    ~lightDirectionLamp(elevator)].
/* エレベータを派遣する */
usecase(Name, Precondition, Postcondition):-
Name=dispatchElevator,
Precondition=[haveDestination(elevator, userFloorNumber) ##
                    haveDestination(elevator, userDestinationNumber),
                motorOff(elevator),
                pushedAscentFloorButton(userFloorNumber) &&
                    lightAscentFloorLamp(userFloorNumber) ##
                pushedDescentFloorButton(userFloorNumber) &&
                    lightDescentFloorLamp(userFloorNumber) ##
                pushedElevatorButton(elevator, userDestinationNumber) &&
                    lightElevatorLamp(elevator, userDestinationNumber)],
Postcondition=[motorOn(elevator),
                lightDirectionLamp(elevator),
                ~motorOff(elevator),
                ~doorOpened(elevator, elevatorFloorNumber) ##
                     move(elevator, elevatorFloorNumber, userFloorNumber) &&
                     ~arrived(elevator, elevatorFloorNumber) &&
                     ~arrivalSensorDetectedApproach(elevator, elevatorFloorNumber) &&
                     ~doorOpened(elevator, elevatorFloorNumber),
                ~doorOpened(elevator, userFloorNumber) ##
                     move(elevator, userFloorNumber, userDestinationNumber) &&
                     ~arrived(elevator, userFloorNumber) &&
                     ~arrivalSensorDetectedApproach(elevator, userFloorNumber) &&
                     ~doorOpened(elevator, userFloorNumber)].
/* エレベータをフロアに停める */
usecase(Name, Precondition, Postcondition):-
Name=stopElevatorAtFloor,
```

```
Precondition=[haveDestination(elevator, userFloorNumber) ##
                  haveDestination(elevator, userDestinationNumber),
              move(elevator, elevatorFloorNumber, userFloorNumber) ##
                  move(elevator, userFloorNumber, userDestinationNumber),
              doorClosed(elevator),
              motorOn(elevator),
              ~motorOff(elevator),
              lightDirectionLamp(elevator),
              pushedAscentFloorButton(userFloorNumber) &&
                  lightAscentFloorLamp(userFloorNumber) ##
              pushedDescentFloorButton(userFloorNumber) &&
                  lightDescentFloorLamp(userFloorNumber) ##
              pushedElevatorButton(elevator, userDestinationNumber) &&
                  lightElevatorLamp(elevator, userDestinationNumber)],
Postcondition=[motorOff(elevator),
               ~doorClosed(elevator),
               ~motorOn(elevator),
               ~haveDestination(elevator, userFloorNumber) ##
                    arrived(elevator, userFloorNumber) &&
                    doorOpened(elevator, userFloorNumber) &&
                    arrivalSensorDetectedApproach(elevator, userFloorNumber) &&
                    ~move(elevator, elevatorFloorNumber, userFloorNumber) &&
                    ~haveDestination(elevator, userFloorNumber),
               ~haveDestination(elevator, userDestinationNumber) ##
                    arrived(elevator, userDestinationNumber) &&
                    doorOpened(elevator, userDestinationNumber) &&
                    arrivalSensorDetectedApproach(elevator, userDestinationNumber) &&
                    ~move(elevator, userFloorNumber, userDestinationNumber) &&
                    ~haveDestination(elevator, userDestinationNumber),
               ~needAscentElevator(user, userFloorNumber) ##
                    ~needAscentElevator(user, userFloorNumber) &&
                    ~pushedAscentFloorButton(userFloorNumber) &&
                    ~lightAscentFloorLamp(userFloorNumber),
               ~needDescentElevator(user, userFloorNumber) ##
                    ~needDescentElevator(user, userFloorNumber) &&
                    ~pushedDescentFloorButton(userFloorNumber) &&
                    ~lightDescentFloorButton(userFloorNumber),
               ~needToGo(user, userDestinationNumber) ##
                    ~pushedElevatorButton(elevator, userDestinationNumber) &&
                    ~lightElevatorLamp(elevator, userDestinationNumber),
               haveDestination(elevator, otherUserFloorNumber) ##
                    ~lightDirectionLamp(elevator)].

/*** シナリオの一部 ***/
/* 上昇フロアボタンを押す */
usecase(Name, Precondition, Postcondition):-
```

```
Name=pushAscentFloorButton,
Precondition=[beOnTheFloor(user, userFloorNumber),
                needAscentElevator(user, userFloorNumber),
               ~pushedAscentFloorButton(userFloorNumber)],
Postcondition=[pushedAscentFloorButton(userFloorNumber),
                haveDestination(elevator, userFloorNumber),
                lightAscentFloorLamp(userFloorNumber)].
/* 下降フロアボタンを押す */
usecase(Name, Precondition, Postcondition):-
Name=pushDescentFloorButton,
Precondition=[beOnTheFloor(user, userFloorNumber),
                needDescentElevator(user, userFloorNumber),
               ~pushedDescentFloorButton(userFloorNumber)],
Postcondition=[pushedDescentFloorButton(userFloorNumber),
                haveDestination(elevator, userFloorNumber),
                lightDescentFloorLamp(userFloorNumber)].
/* エレベータボタンを押す */
usecase(Name, Precondition, Postcondition):-
Name=pushElevatorButton,
Precondition=[beInTheElevator(user, elevator),
                needToGo(user, userDestinationNumber),
               ~pushedElevatorButton(elevator, userDestinationNumber)],
Postcondition=[haveDestination(elevator, userDestinationNumber),
                pushedElevatorButton(elevator, userDestinationNumber),
                lightElevatorLamp(elevator, userDestinationNumber)].
/* エレベータボタンから降りる */
usecase(Name, Precondition, Postcondition):-
Name=getOffElevator,
Precondition=[motorOff(elevator),
~motorOn(elevator),
doorOpened(elevator, userDestinationNumber),
~doorClosed(elevator),
beInTheElevator(user,elevator),
arrived(elevator, userDestinationNumber),
arrivalSensorDetectedApproach(elevator, userDestinationNumber),
needToGo(user, userDestinationNumber),
~move(elevator, userFloorNumber, userDestinationNumber),
~haveDestination(elevator, userDestinationNumber)],
Postcondition=[beOnTheFloor(user, userDestinationNumber),
                ~beInTheElevator(user, elevator),
                ~needToGo(user, userDestinationNumber)].


/**** 実行のメイン ****/
main(New_Pre, New_Post):-
searchFromPrecondition(New_Pre, New_Post, [], UseCaseList_FromPre),
```

```
searchFromPostcondition(New_Pre, New_Post, [], UseCaseList_FromPost),
make_List(UseCaseList_FromPost, UseCaseList_FromPre, Unique_UseCaseList),
nl,nl,
print(useCaseList_SearchedFromPrecondition = UseCaseList_FromPre),
nl,nl,
print(useCaseList_SearchedFromPostcondition = UseCaseList_FromPost),
nl,nl,
make_UseCaseList(Unique_UseCaseList, Unique_UseCases),
print(————————————————————————),
print(useCaseInformation),
print(————————————————————————),nl,
printUseCaseInformation(Unique_UseCases).

/*** 事前条件からの抽出 ***/
searchFromPrecondition(New_Pre, New_Post, Temp, UseCaseList_FromPre):-
execute_searchFromPrecondition(New_Pre, New_Post, Result),
not(check_double(Result, Temp)),
!,
searchFromPrecondition(New_Pre, New_Post,
                       [Result | Temp], UseCaseList_FromPre).
searchFromPrecondition(_, _, Temp, Temp).

/** 事前条件からの抽出（実行部分） **/
execute_searchFromPrecondition(New_Pre, New_Post, [Result_Head | Result_Rest]):-
newPre_QuasiImply_to(New_Pre, Old_Post, Result_Head),
quasiImply_to_OldPost(Old_Post, New_Post, Result_Rest).

/** New_Pre ≫ Old_Pre を満たす論理式 Old_Pre を
                                        事前条件に持つユースケースを探す。 **/
newPre_QuasiImply_to(New_Pre, Old_Post, Name):-
usecase(Name, Old_Pre, Old_Post),
satisfy(New_Pre, Old_Pre).

/** OLD_Pre ≫ Old_Post を満たす論理式 OLD_Pre を
                                        事前条件に持つユースケースを探す。 **/
/*終了条件*/
quasiImply_to_OldPost(_,_, []).

quasiImply_to_OldPost(Old_Post, New_Post, [Name | Rest]):-
usecase(Name, OLD_Pre, OLD_Post),
satisfy(OLD_Pre, Old_Post),
quasiImply_to_OldPost(OLD_Post, New_Post, Rest).

/*** 事後条件からの抽出 ***/
searchFromPostcondition(New_Pre, New_Post, Temp, UseCaseList_FromPost):-
execute_searchFromPostcondition(New_Pre, New_Post, Result),
```

```
reverse(Result, Reversed_Result),
not(check_double(Reversed_Result, Temp)),
!,
searchFromPostcondition(New_Pre, New_Post,
                                [Reversed_Result | Temp], UseCaseList_FromPost).
searchFromPostcondition(_, _, Temp, Temp).

/** 事後条件からの抽出（実行部分） **/
execute_searchFromPostcondition(New_Pre, New_Post, [Result_Head | Result_Rest]):-
newPost_QuasiImply_to(New_Post, Old_Pre, Result_Head),
oldPre_QuasiImply_to(Old_Pre, New_Pre, Result_Rest).

/** New_Post ≫ Old_Post を満たす論理式 Old_Post を
                                                事後条件に持つユースケースを探す。 **/
newPost_QuasiImply_to(New_Post, Old_Pre, Name):-
usecase(Name, Old_Pre, Old_Post),
satisfy(New_Post, Old_Post).

/** Old_Pre ≫ OLD_Post を満たす論理式 OLD_Post を
                                                事後条件に持つユースケースを探す。 **/
/*終了条件*/
oldPre_QuasiImply_to(_,_, []).

oldPre_QuasiImply_to(Old_Pre, New_Pre, [Name | Rest]):-
usecase(Name, OLD_Pre, OLD_Post),
satisfy(Old_Pre, OLD_Post),
oldPre_QuasiImply_to(OLD_Pre, New_Pre, Rest).

/** Elem が List の要素になっているか確認する。（satisfy(List1, List2) で使用） **/
part_of(Elem, List):-
List=[[A] | List_Rest],
(part_of(Elem, [A]);part_of(Elem, List_Rest)),!.
part_of(Elem, List):-
List=[A##B | List_Rest],
(part_of(Elem, [A]);part_of(Elem, [B]);part_of(Elem, List_Rest)),!.
part_of(Elem, List):-
List=[A&&B | List_Rest],
(part_of(Elem, [A]);part_of(Elem, [B]);part_of(Elem, List_Rest)),!.
part_of(Elem, List):-
List=[List_Head | List_Rest],
(Elem==List_Head;part_of(Elem, List_Rest)).

/** List1 が List2 を満たしているか確認する。 **/
/* 終了条件 */
satisfy(_,A):-
A=[],!.
```

```
satisfy(List1, List2):-
List2=[[A] | List2_Rest],
satisfy(List1, [A]),
satisfy(List1, List2_Rest).
satisfy(List1, List2):-
List2=[A##B | List2_Rest],
(satisfy(List1, [A]);satisfy(List1, [B])),!,
satisfy(List1, List2_Rest).
satisfy(List1, List2):-
List2=[A&&B | List2_Rest],
(satisfy(List1, [A]),satisfy(List1, [B])),
satisfy(List1, List2_Rest).
satisfy(List1, List2):-
List2=[~A | List2_Rest],
not(satisfy(List1, [A])),
satisfy(List1, List2_Rest).
satisfy(List1, List2):-
List2=[List2_Head | List2_Rest],
part_of(List2_Head, List1),
satisfy(List1, List2_Rest).

/*** 重複のないデータで構成されるリストを作成する。 ***/
make_List(New, Temp, Unique_DataList):-
New=[A|B],
not(check_double(A, Temp)),
!,
make_List(B, [A | Temp], Unique_DataList).
make_List(New, Temp, Unique_DataList):-
New=[_|A],
make_List(A, Temp, Unique_DataList).
make_List(_, Temp, Temp).

/** Elem が List の要素になっているか確認する。(出力するデータの作成用) **/
check_double(Elem, List):-
List=[List_Head | List_Rest],
(Elem==List_Head;check_double(Elem, List_Rest)),!.

/*** 再利用できる可能性のあるユースケースのリストを作成する。 ***/
make_UseCaseList(Unique_UseCaseList, Unique_UseCases):-
execute_Make_UseCaseList(Unique_UseCaseList, [], Unique_UseCases).

/** 再利用できる可能性のあるユースケースのリストを作成する (実行部分)。 **/
/* 終了条件 */
execute_Make_UseCaseList([], UseCaseList, UseCaseList).
```

```
execute_Make_UseCaseList(Unique_UseCaseList, Temp, UseCaseList):-
Unique_UseCaseList=[Unique_UseCaseList_Head | Unique_UseCaseList_Rest],
make_List(Unique_UseCaseList_Head, Temp, UseCaseList1),
execute_Make_UseCaseList(Unique_UseCaseList_Rest, UseCaseList1, UseCaseList).

/*** 再利用できる可能性のあるユースケースの情報
                                （Name，Precondition, Postcondition）を表示する。 ***/
/* 終了条件 */
printUseCaseInformation([]).

printUseCaseInformation(Unique_UseCases):-
Unique_UseCases=[Unique_UseCases_Head | Unique_UseCases_Rest],
usecase(Name, Old_Pre, Old_Post),
Name==Unique_UseCases_Head,
print(useCaseName = Name),nl,
print(precondition = Old_Pre),nl,
print(postcondition = Old_Post),nl,nl,
printUseCaseInformation(Unique_UseCases_Rest).
```

B.2 実行時に引数とする新たなユースケースの事前条件，事後条件

```
/*** 実行形式 ***/
main([新たなユースケースの Precondition][新たなユースケースの Postcondition]).


/*** 新たなユースケースの情報 ***/
/* 高速エレベータを要求する */
Precondition=[beOnTheFloor(user, userFloorNumber),
                needAscentElevator(user, userFloorNumber) ##
                     needDescentElevator(user, userFloorNumber),
                fastElevator(elevator),
                ~haveProhibitionFloorNumber(elevator, userFloorNumber)],
Postcondition=[arrived(elevator, userFloorNumber),
                doorOpened(elevator, userFloorNumber) ,
                arrivalSensorDetectedApproach(elevator, userFloorNumber),
                motorOff(elevator),
                ~arrived(elevator, elevatorFloorNumber),
                ~doorOpened(elevator, elevatorFloorNumber),
                ~doorClosed(elevator),
                ~arrivalSensorDetectedApproach(elevator, elevatorFloorNumber),
                ~motorOn(elevator),
                ~move(elevator, userFloorNumber, elevatorFloorNumber),
                ~pushedAscentFloorButton(userFloorNumber) ##
                     ~pushedDescentFloorButton(userFloorNumber),
```

```
                    ~needAscentElevator(user, userFloorNumber) ##
                        ~needDescentElevator(user, userFloorNumber),
                    haveDestination(elevator, otherUserFloorNumber) ##
                        ~lightDirectionLamp(elevator),
                    ~needAscentElevator(otherUser, userFloorNumber) ##
                        ~needAscentElevator(otherUser, userFloorNumber),
                    ~needDescentElevator(otherUser, userFloorNumber) ##
                        ~needDescentElevator(otherUser, userFloorNumber),
                    fastElevator(elevator),
                    ~haveProhibitionFloorNumber(elevator, userFloorNumber)].
/* 高速エレベータで行き先を選ぶ */
Precondition=[beInTheElevator(user, elevator),
                   needToGo(user, userDestinationNumber),
                   fastElevator(elevator),
                  ~haveProhibitionFloorNumber(elevator, userDestinationNumber)],
Postcondition=[arrived(elevator,
                    userDestinationNumber),
                    doorOpened(elevator, userDestinationNumber),
                    arrivalSensorDetectedApproach(elevator, userDestinationNumber),
                    motorOff(elevator), beOnTheFloor(user, userDestinationNumber),
                    ~beInTheElevator(user, elevator),
                    ~needToGo(user, userDestinationNumber),
                    ~move(elevator, userFloorNumber, userDestinationNumber),
                    ~arrived(elevator, userFloorNumber),
                    ~doorOpened(elevator, userFloorNumber),
                    ~arrivalSensorDetectedApproach(elevator, userFloorNumber),
                    haveDestination(elevator, otherUserFloorNumber) ##
                    ~lightDirectionLamp(elevator),
                    fastElevator(elevator),
                    ~haveProhibitionFloorNumber(elevator, userDestinationNumber)].
```