ソフトウェア工学 64－13
（１９８９．２．２）

時区間論理に基づく動作仕様記述の構成法

宇山政志　　米崎直樹
東京工業大学工学部情報工学科

本稿では、時区間論理による動作仕様記述の構成法と、それに基づく動作仕様記述言語を提案する。仕様は、要求イベントとサービスイベントとの時間的関係に対する制約として記述され、イベントの詳細化を階層的に行うことにより、単純な仕様から所望の仕様記述を得る。また制御的情報を他の情報の記述と分離することにより、記述の簡略化と差分記述を可能にしている。また、オブジェクト間の関係に注目し、それを表す述語の仕様を考える事により、従来とは異なるモジュール化を行っている。ここでは、このような述語の成立する時区間を詳細化していく段階的な設計法を、エレベータ問題を例に説明している。

A method of constructing specifications for dynamic systems based on interval logic

Masashi Uyama and Naoki Yonezaki

Department of Computer Science,
Tokyo Institute of Technology,
O-okayama,Meguro-ku,Tokyo,152,Japan

A method of constructing specifications for dynamic systems based on interval logic and a specification definition language for the method are presented. Specifications are described as the constraint of the temporal relation between "request" and "service". A desired specification is obtained from a simple specification by stepwise refinement which divides an event into some sequence or combination of events. Information of control is described separately from other information,so that this enables differential and orthogonal description. Considering the relation among objects , we introduced modularization which differs from conventional one into this specification language. The refining process of the specification is explained with an example of the elevator problem.

1 まえがき

動的システムを記述する際に、記述された動作仕様に論理的矛盾がないかといった性質を検証したり、その仕様が直接または間接的に実行可能であるためには、その動作仕様が形式的に記述されていなければならない。また、仕様記述を容易に行うため、あるいは検証を効率的に支援するためにも、了解性の良い仕様記述言語が用いられるべきである。時間論理は仕様を記述したり、各種の検証を行なうのに適当である [1,2] が、仕様記述言語としては、抽象化・階層化の機能を備えていない。抽象化や階層化は大規模システムをわかりやすく記述するために、仕様記述言語にとっては不可欠の機能である。時区間論理は、時区間の概念が導入されており、通常の $\Box, \Diamond, until$ に基づく時間論理より、時間概念を構造化して捉えることができる[3,4]。ここでは、過去・現在・未来という人間の直感的な時間認識を考慮した時区間論理を導入し、動的システムの記述に適した記述形式と設計スタイルを提案する。

2 時区間論理

ここでは、現在を起点とした過去の事象・未来の事象を扱うのに都合の良い時区間論理を定義する。

2.1 syntax

1. 一階述語論理の式は、式である。
2. 式を一階述語論理の連結子でつないだもの、あるいは、一階述語論理の限定子で束縛したものも式である。
3. A, B, Cが式であるとき、
   $(A \Rightarrow B)C, [A \Rightarrow B]C,$
   $(B \Leftarrow A)C, [B \Leftarrow A]C,$
   $\Box A$
   は式である。但し、$\Rightarrow$ は、$\mapsto, \multimap$のいずれか、$\Leftarrow$ は、$\leftarrow\!\!\bullet, \multimap$のいずれかである。

2.2 abbreviation

$< A \Rightarrow B > C \equiv\sim [A \Rightarrow B] \sim C$
$< B \Leftarrow A > C \equiv\sim [B \Leftarrow A] \sim C$
$\Diamond A \equiv\sim \Box \sim A$

2.3 semantics

ここでは、この時区間論理のセマンティクスを形式的に詳述することはしない。各式の直感的な意味は、以下の通りである。

式は、現在を表す時刻と、その前後の区間を表す時刻列とで評価される。

1. A が一階述語論理の式であれば、現在の時刻で A が成り立つ
2. $\Box A$ は、評価中の区間内の現在以降の区間Iで、任意の時刻を現在とするとき、Iで A が成り立つ
3. $[A \Rightarrow B]C$は、評価中の区間の現在以降の区間で、どんな「 A の成立の立ち上がりの時刻から始まり、Bが初めて成り立つまでの区間(Bが永久に成り立たなくてもよい)」をとっても、その区間で現在をその区間の始点においた時に Cが成り立つ
4. $[B \Leftarrow A]C$は、評価中の区間の、現在以前の区間をさかのぼって考え、どんな「 A の成立の立ち上がりの時刻から始まり、Bが初めて成り立つまでの区間(Bが永久に成り立たなくてもよい)」をとっても、その区間で現在をその区間の最過去の点においた時に Cが成り立つ
5. $(A \Rightarrow B)C$は、評価中の区間の、現在以降の区間で、「初めて A が成り立ってから、初めて Bが成り立つまでの区間(Bが永久に成り立たなくてもよい)」が存在し、その区間で現在をその区間の始点においた時に Cが成り立つ
6. $(B \Leftarrow A)C$は、評価中の区間の、現在以前の区間をさかのぼって考え、「初めて A が成り立ってから、初めて Bが成り立つまでの区間(Bが永久に成り立たなくてもよい)」が存在し、その区間で現在をその区間の最過去においた時に Cが成り立つ

ことを各々示している。

ここで、$\mapsto, \multimap, \leftarrow\!\!\bullet, \multimap$は、C の成立を調べる時に、区間の始点、すなわち、A が成り立つところを含めて考えるかどうかを表す。$\mapsto, \leftarrow\!\!\bullet$は含める。$\multimap, \multimap$は含めない。どれも Bのところは含めない。

3 動的システムの分析

動的システムでは、「要求に対するサービス」が基本的な考え方である。その要求と現在の状態、システムの物理的な制約から、そのサービスを行なうため必要な「動作列」に対する制約事項を規定していくのが動作仕様の考え方である。ここでは、エレベータ問題を取り上げ、動作仕様記述言語に必要なスタイルについて考察する。

3.1 オブジェクトとオブジェクト間の関係

動作を記述する際に、

・その動作の主体となっているのは何か？

・その動作の制御対象となっているのは何か？

を意識せずに仕様を記述することは不可能であろう。また、その動作が行なわれるのに必要な条件が、

・その動作主体内部の状態のみから得られるのか

・外部からの非同期的な信号を必要としているのか

にも注意が必要である。

このような条件から、広義のオブジェクト・ベース[5] の考え方を用いる。さらに各オブジェクト間の関係の記述という考え方を用いている。つまり、他のオブジェクトからメッセージを受けて動作をするという考え方でなく、オブジェクト間の関係を表す述語があり、その述語が真のときの動作を規定していくという考えである。

例えば、エレベータシステムの例では、リフトiがフロアjにいるということは、ｏｂｊｅｃｔ「リフト」とｏｂｊｅｃｔ「フロア」の関係であり、述語として例えばP(i,j) のように書く。メッセージが瞬間的で、一方向的なのに対し、関係による定義は、双方向的で、時間的な広がりを持つのが特徴である。述語が偽から真に変わる瞬間に着目した記述を行えば、メッセージパッシングと同等な記述もできる。後述の時区間の階層化等考えたときに、「オブジェクト・ベース」&「関係による定義」は、メッセージパッシングより柔軟性がある。

3.2 動作記述の基本単位

システムの動作を記述する基本単位としては、

(現在の状態) + (過去の履歴)

→ (未来に関する制約) —(1)

という形式が、人間の直感に近いと考えられる[6,2] 。ここで言う「過去の履歴」は、状態遷移図では何らかの状態遷移列として、手続き型言語では、変数の何らかの値として現在の状態の一部として表されてきたものである。履歴という見方をすれば、システムの状態として本質的ではない履歴保存を行うための名前を陽に用いずにすむ。section2で定義した時区間論理では「過去の履歴」は、

$(B \Leftarrow A)C, < B \Leftarrow A > C, [B \Leftarrow A]C$

のような論理式として表される。

現存の手続きプログラムのスタイルは

(現在の状態) → (次の時刻に関する制約)

であり、(1)のスタイルの時区間論理式から、このようなプログラムレベルに翻訳することは、プログラム合成に対応する。

3.3 時区間の階層化モデルと物理的制約

「そのシステムの動作を表すのに、どういった状態が必要とされるか」つまり論理型の言語の場合ならば「どのようなシステムの状態に、基本的述語(他の述語から定義できない述語)を割り当てるか」ということが問題となる。

その設定がまずければ、仕様の記述は難しく、できたとしても読みにくいものになってしまう。よって動作仕様を書くにあたって、仕様記述者は、まず、システムの状態を端的に表す述語をいくつか選定し、それらの間の時間的包含関係(つまり、ある述語の成立を前提にして起きる述語)、時間的順序関係等を考慮したシステムのモデルを作ることが望ましい。本稿では、これを時区間の階層化モデルと呼んでいる。この段階における、そのモデルが良いものであるかの唯一の判断基準は、人間の直感に近いかどうかということである。

例えば、エレベータ問題におけるリフトの動作記述において、「リフトがi階にいる」ということを示す述語は必須であろう。しかし、「リフトがi階とi＋1階の間を上昇している」ということは、他の述語から記述でき、独立に述語を割り振る必要があるかどうかは疑問である。また、「現在を含めて最近いた階がi階である」ことを示す「状態i」という述語を導入すれば、

「状態ｉで、かつ止まっている」という連言で「ｉ階にいる」を表すことも可能である。しかし、この「状態ｉ」のような直感的に理解しづらい概念を基本述語と捉えることはしない。

このような、時間的包含関係、時間的な順序関係の考えられたモデルは、そのシステムの持つ制約として、時区間論理の論理式で記述することが可能である。こういった関係を最初に書き出しておくことによって、動作を記述する論理式各々の持つ情報量を少なくすることが出来、その動作本来の意味が明確になる。

このモデル作成の持つもう一つのメリットは時区間の詳細化と情報隠蔽の働きである。例えば、「ｉ階でサービス」が行なわれるということは、実際にはオブジェクトｄｏｏｒに関する述語 open,close 等がその間成立するはずであるが、要求に対するサービスの仕様としては、その「ｉ階でサービス」の成立する区間での出来事に目をくばる必要は全く無い。この部分は、実現レベルの仕様でドアの開け閉め、閉まりかかったドアの開け直し等、記述する必要が生じた時に追加すれば良い。

4 仕様記述言語のスタイル

前章を考慮して仕様記述言語のスタイルを以下のように定める。

仕様は、モジュールの集まりであり、各モジュールは、

- 構成要素となるオブジェクトの宣言。—[a]
- 各オブジェクト間の関係を表す述語の宣言。—[b]
- オブジェクト固有の状態を表す述語の宣言。—[c]
- イベントのモデル化に応じた制約。—[d]
- 記述の簡略化のためのマクロの定義。—[e]
- 初期条件。—[f]
- 基本的な要求に対するサービスの記述。—[g]
- 制御の記述。—[h]

とで構成される。Appendix A,B に仕様の記述例を示す。[a]等は仕様例の各該当部分を示す。#で始まる部分は、コメント文である。それ以外の日本語文は説明の都合上のものであり、仕様の一部ではない。全ての自由変数は全称束縛されていると考える。

5 時区間の詳細化による動作仕様の段階的設計

5.1 エレベータ問題への適用

ここでは、時区間の詳細化を用いた仕様設計法をエレベータ問題の仕様を例にとり紹介する。

動作仕様記述を難しくしている要因の一つとして各オブジェクトが非同期的に動作することが挙げられる。そこで、設計の初期段階には「ある述語の成立時には、他の述語全ての真偽がその間一定である」という特殊な仮定の仕様から記述していくことにする。
つまり、述語が真になってる区間は、実際には時間的な広がりを持っているのであるが、あたかもそれが一瞬の時刻であるかのように考えて仕様を記述していく。

このような仕様の、述語の瞬間的な成立区間を、徐々にさまざまな述語の組み合わせに細分化していくことにより、より現実的な仕様に展開していくことができる。

> エレベータ問題の捉え方。
> オブジェクトの選択。基本となるオブジェクトは、
> lift,button,floor,door
> である。
>
> 基本となる述語は、button が押されている事を示す on(button), ｉ番目の lift がｊ階にいることを示す P(i,j),そこでサービスが行なわれている事を示す S(i,j)だけあれば良い。(S(i,j)は具体的には、door の開け閉めに対応する。)
>
> このとき、各ボタンの述語 on の真偽の時間的な系列に対して、仕様を満たすような P,S の系列が、エレベータシステムの動作列に対応する。

MODEL1 は、リフトｉがｊ階にいるという状態を一時点に圧縮して、考えたモデルである。つまり、P(i,j)が瞬間であり、その間あらゆる状態変化は起こらないと仮定する。P,S の他にリフトｉの次に取る行動を示

す述語up(i),down(i)を導入している。MODEL 1 に基づく動作系列の例をfig.1 に示す。＊で表された時刻と時刻の間には、高々有限個の時刻が存在してよい。

```
 P(i,j) P(i,j+1) P(i,j)  P(i,j-1) P(i,j-2)
---*-------*------*--------*--------*---->
 S(i,j) S(i,j+1)          S(i,j-1)
  up(i)  down(i) down(i) down(i)   up(i)
```

fig.1

このモデルを時区間論理の式に直すと、Appendix A の[d]のようになる。Appendix A は、このモデルに基き書かれた仕様である。

MODEL2 は、P(i,j)をいくつかの部分区間に詳細化したものである。つまり、サービス中を表すS とドアの閉まった状態を表すC とに分解する。MODEL2 に基づく動作系列の例をfig.2 に示す。

```
 P(i,j) P(i,j)  P(i,j)  P(i,j-1) P(i,j-2)
--*------*------*-------*--------*----- -
 S(i,j) S(i,j)  C(i,j)  C(i,j-1) C(i,j-2)
 up(i)  down(i) down(i) down(i)  up(i)

     P(i,j)  P(i,j)   P(i,j)  P(i,j+1)
- - ---*------*-------*-------*--------->
     S(i,j)  C(i,j)   C(i,j)  S(i,j+1)
     stay(i  stay(i)  up(i)   up(i)
```

fig.2

MODEL2 は、MODEL1 では考えられていなかったサービスを実行している最中の

他のリフトの行動に起因する要求の減少、

各ボタンが押されることに起因する要求の増加

によるサービス前とサービス後の要求変化に対応した物で、より現実的な仕様となっている。また、MODEL1 ではリフトiがj階にいる状態を一時点と捉えたため、全く要求が無いケースを考えることが出来なかったが、ここでは *stay*という述語でその状態が表されている。Appendix B は、このモデルに基づき書かれた仕様である。

5.2 仮定に基づく仕様と現実の動作

MODEL1,2 ともに、実際には、その間に他のオブジェクトの独自の動作により何らかの変化が起こり得る区間を、一瞬であると仮定した上での仕様なので、そのままでは完全に現実を反映した仕様とならない。例えば、

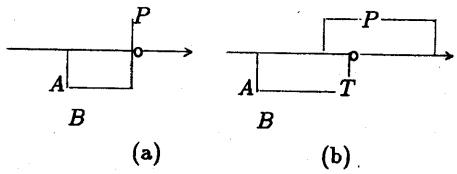

fig.3

fig.3(a)を意図した論理式

$P \wedge (A \Leftarrow\!\circ T)B$ —(2)

は、現実的にはfig.3(b)のように、Pの成立している時間は幅のある区間なのでその中のどこの時刻に現在をとるかで(2)式の評価は、変わってくる。そこで、最初に意図した通りの意味を得るには、

$P \wedge (A \Leftarrow\!\bullet \sim P)B$

のようにPに依存した式に書き直さなければならない。このような記述の修正は繁雑であり、ここでの時区間論理の限界を示すものと考えられる。時区間論理の限界については、更にsection 7で言及する。

6 述語のモジュール性

論理型の仕様記述言語のメリットとして述語のモジュール性を挙げておく。

例えば、エレベータ仕様の $aboveR, belowR$を次のように換えておく。

$$aboveR_{equest}(i,j) \leftrightarrow (\exists k(Reserved(i,k) \vee inR(i,k)) \wedge j < k)$$

$$belowR_{equest}(i,j) \leftrightarrow (\exists k(Reserved(i,k) \vee inR(i,k)) \wedge j > k)$$

$$Reserved(i,j) \leftrightarrow outR(j)$$

このように分解しておけば、種々のエレベータシステムの仕様が、新たに定義された述語 *Reserved* の書換えだけで得られる。例えば、

$$Reserved(1,j) \leftrightarrow outupR(j) \wedge j > n/2$$
$$Reserved(2,j) \leftrightarrow outupR(j) \wedge j \leq n/2$$

のように各フロアの要求を拾いに行くリフトを指定したシステムにも出来る。

7 あとがき

時区間の階層モデル、人間の直感に近い記述の基本単位、種々のレベルのモジュール性、等を採用した時区間論理に基づく仕様記述言語および、それによる時区間の詳細化の手法を用いた段階的な仕様設計のスタイルを提案した。

section 5.2で指摘した通り、その設計段階において、一時刻に凝縮していると仮定していた時区間を、実際に区間とすると記述に繁雑さが生ずる。もうひとつ、この時区間論理で記述できない例を挙げる。時区間論理はある区間に限定して成立する事実を規定するのに都合がよい。例えば、

$[P(i,j) \Leftrightarrow \sim P(i,j)]\Box(opening \rightarrow \Diamond closed)$

のような記述ができる。しかし、$[P(i,j) \Leftrightarrow \sim P(i,j)]$ 等で限定された区間からは、その区間の外を参照・言及することができない。

このようなことから、時間的包含関係、時間的順序関係をより明確な形にし、時間を木構造として捉えた論理が、動作仕様を記述するのに有効ではないかと考えている。これは、今後の研究課題である。

References

[1] Naoki Yonezaki and Takao Katayama. Functional specification of synchronized process baced on modal logic. *JARECT[12],Computer Science & Technology*, 1984.

[2] 米崎直樹、佐伯元司、荒俣博、西昭仁. Tell/NSL における動的記述の検証. , 第 *30* 回情報処理学会全国大会, 497 ページ, 1985 .

[3] 米崎直樹、新 淳、蓬莱 尚幸. 時間論理プログラミング言語 templog . , 日本ソフトウェア科学会第 *1* 回全国大会論文集, 77–80 ページ, 1986 .

[4] 米崎直樹、新 淳、蓬莱 尚幸. 時区間を基礎とする時間論理プログラミング. 情報処理学会 知識工学と人工知能研究会報告書, 39-11 , 1985 .

[5] Peter Wegner. Dimensions of object-based language design. In *OOPSLA '87 Conference Proceedings*, pages 168–182, 1987.

[6] 桜川 貴司、竹中 一起、中島 玲二、新出 尚之、服部 隆志. RACCO:実時間プロセス制御システムのモデル記述のための様相論理プログラミング言語. コンピュータソフトウェア, Vol.5,No.3 , 1988 .

Appendix A: MODEL1

#Constraint specification for dynamic system
elevator system
  objects
    lift(1..2):box;
    floor(1..n):floor —[a]
  end;
  relation
    $P_{osition}$(lift-id, floor-id); S_{ervice}(lift-id, floor-id)
  end; —[b]
  constraint of model
    $\sim (P(i,j) \wedge P(i,k) \wedge j \neq k)$
    $P(i,j) \rightarrow up(i) \vee down(i)$
    $up(i) \rightarrow \exists j P(i,j)$; $down(i) \rightarrow \exists j P(i,j)$
    $\sim (up(i) \wedge down(i))$
    $S(i,j) \rightarrow P(i,j)$
    $up(i) \wedge P(i,j) \rightarrow (T \circ\!\Rightarrow P(i,j+1)) \Box \forall l \sim P(i,l)$
    $down(i) \wedge P(i,j) \rightarrow (T \circ\!\Rightarrow P(i,j-1)) \Box \forall l \sim P(i,l)$
    $P(i,j) \rightarrow \Diamond \exists l P(i,l)$ —[d]
  end;
  macro —[e]
    $cameup(i) \leftrightarrow P(i,1) \vee$
    $\exists j(P(i,j) \wedge 1 < j < n \wedge (P(i,j-1) \Leftarrow\!\circ T) \Box \forall l \sim P(i,l))$
    $camedown(i) \leftrightarrow P(i,n) \vee$
    $\exists j(P(i,j) \wedge 1 < j < n \wedge (P(i,j+1) \Leftarrow\!\circ T) \Box \forall l \sim P(i,l))$
    $inR_{equest}(i,j) \leftrightarrow$
    $(S(i,j) \Leftarrow\!\circ T) \Diamond on$(lift[i].destination-button(j))
    $outupR_{equest}(j) \leftrightarrow$
    $(\exists i(up(i) \wedge S(i,j)) \Leftarrow\!\circ T) \Diamond on$(floor[$j$].floor-button(up))
    $outdownR_{equest}(j) \leftrightarrow$
    $(\exists i(down(i) \wedge S(i,j)) \Leftarrow\!\circ T)$
      $\Diamond on$(floor[j].floor-button(down))
    $outR_{equest}(j) \leftrightarrow outupR(j) \vee outdownR(j)$
    $aboveR_{equest}(i,j) \leftrightarrow \exists k((outR(k) \vee inR(i,k)) \wedge j < k)$
    $belowR_{equest}(i,j) \leftrightarrow \exists k((outR(k) \vee inR(i,k)) \wedge j > k)$
      以下、$\cdots R_{equest} N_{ow}$は対応する$\cdots R_{equest}$の
      $\Leftarrow\!\circ$を$\Leftarrow\!\bullet$で置き換えたものであり定義は省略。
  end;

  Initial —[f]
    $Init \rightarrow$
    $P(1,1) \wedge S(1,1) \wedge up(1) \wedge$
    $P(2,n) \wedge S(2,n) \wedge down(2)$
  end;
  service
    $inR(i,j) \rightarrow \Diamond S(i,j)$
    $outupR(j) \rightarrow \exists i \Diamond S(i,j)$
    $outdownR(j) \rightarrow \exists i \Diamond S(i,j)$ —[g]
  control —[h]
    #service timing
    $P(i,j) \wedge inR(i,j) \rightarrow S(i,j)$
    $P(i,j) \wedge outupR(j) \wedge$
      $(cameup(i) \vee \sim belowR(i,j)) \rightarrow S(i,j)$
    $P(i,j) \wedge outdownR(j) \wedge$
      $(camedown(i) \vee \sim aboveR(i,j)) \rightarrow S(i,j)$
    $P(i,j) \wedge \sim inR(i,j) \wedge \sim outR(j) \rightarrow \sim S(i,j)$
    #direction set
    $P(i,j) \wedge cameup(i) \wedge aboveRN(i,j) \rightarrow up(i)$
    $P(i,j) \wedge camedown(i) \wedge belowRN(i,j) \rightarrow down(i)$
    # direction change
    $P(i,j) \wedge cameup(i) \wedge$
      $\sim aboveRN(i,j) \wedge belowRN(i,j) \rightarrow down(i)$
    $P(i,j) \wedge camedown(i) \wedge$
      $\sim belowRN(i,j) \wedge aboveRN(i,j) \rightarrow up(i)$
  end;
box
  objects
    destination-button(floor-id):button;
    #door:door
  end;
  state up; $down$ end; —[c]
end;
floor
  objects floor-button({up,down}):button end;
end;
button
  state on end;
end;

Appendix B: MODEL2

#Constraint specification for dynamic system
モデル1と異なるところのみ記述。
relation
$P_{osition}$(lift-id, floor-id);S_{ervice}(lift-id, floor-id);
C_{losed}(lift-id, floor-id)
end;
Constraint of model
$\sim(P(i,j)\wedge P(i,k)\wedge j\neq k)$
#
$S(i,j)\rightarrow P(i,j);C(i,j)\rightarrow P(i,j)$
$\sim(S(i,j)\wedge C(i,j))$
$P(i,j)\rightarrow S(i,j)\vee C(i,j)$
#
$up(i)\rightarrow P(i,j);down(i)\rightarrow P(i,j);stay(i)\rightarrow P(i,j)$
$\sim(up(i)\wedge down(i))\wedge$
$\sim(down(i)\wedge stay(i))\wedge\sim(stay(i)\wedge up(i))$
$P(i,j)\rightarrow up(i)\vee down(i)\vee stay(i)$
#
$S(i,j)\rightarrow(T\circ\!\!\Rightarrow P(i,j))\Box\forall l\sim P(i,l)$
$C(i,j)\wedge up(i)\rightarrow(T\circ\!\!\Rightarrow P(i,j+1))\Box\forall l\sim P(i,l)$
$C(i,j)\wedge down(i)\rightarrow(T\circ\!\!\Rightarrow P(i,j-1))\Box\forall l\sim P(i,l)$
$C(i,j)\wedge stay(i)\rightarrow(T\circ\!\!\Rightarrow P(i,j))\Box\forall l\sim P(i,l)$
#
$S(i,j)\rightarrow\Diamond C(i,j)$
$C(i,j)\rightarrow\Diamond\exists lP(i,l)$
#駆動条件
end;
macro
$cameup(i)\leftrightarrow P(i,1)\vee$
$\exists j(P(i,j)\wedge 1<j<n\wedge(up(i)\Leftarrow\!\circ T)\Box\sim down(i))$
$camedown(i)\leftrightarrow P(i,n)\vee$
$\exists j(P(i,j)\wedge i<j<n\wedge(down(i)\Leftarrow\!\circ T)\Box\sim up(i))$
以下のマクロ、イニシャルの記述はそのまま。
但し、詳細化の結果、$\cdots R_{equest}N_{ow}$は不要。

service
$inR(i,j)\rightarrow\Diamond S(i,j)$
$outupR(j)\rightarrow\exists i\Diamond(up(i)\wedge S(i,j))$
$outdownR(j)\rightarrow\exists i\Diamond(down(i)\wedge S(i,j))$
control
#service timing
$P(i,j)\wedge inR(i,j)\rightarrow S(i,j)$
$P(i,j)\wedge outupR(j)\wedge$
$(cameup(i)\vee\sim belowR(i,j))\rightarrow S(i,j)$
$P(i,j)\wedge outdownR(j)\wedge$
$(camedown(i)\vee\sim aboveR(i,j))\rightarrow S(i,j)$
$P(i,j)\wedge\sim inR(i,j)\wedge\sim outR(j)\rightarrow C(i,j)$
#その階でサービスするか。
$P(i,j)\wedge cameup(i)\wedge$
$(aboveR(i,j)\vee outupR(j))\rightarrow up(i)$
$P(i,j)\wedge camedown(i)\wedge$
$(belowR(i,j)\vee outdownR(j))\rightarrow down(i)$
#更に同じ方向に進む予定。
$P(i,j)\wedge cameup(i)\wedge$
$\sim(aboveR(i,j)\vee outupR(j))\wedge$
$(belowR(i,j)\vee outdownR(j))\rightarrow down(i)$
$P(i,j)\wedge camedown(i)\wedge$
$\sim(belowR(i,j)\vee outdownR(j))\wedge$
$(aboveR(i,j)\vee outupR(j))\rightarrow up(i)$
#方向の変換。
$P(i,j)\wedge$
$\sim(aboveR(i,j)\vee belowR(i,j)\vee outR(j))\rightarrow stay(i)$
#停止して何らかの要求を待つ。
end;
end;
box
objects
destination-button(floor-id):button;
# door:door
end;
state up; $down$; $stay$ end;
end;