7M-5

partial な状態による呼処理の仕様記述*

中島 俊介, 橋本 正明, 門田 充弘†
ATR 通信システム研究所‡

1 はじめに

筆者等は, 記述性, 実行可能性に着目し, 仕様記述法の研究を進めている. 本稿では, 状態遷移規則における状態の記述と, その解釈について考察する. まず, total な状態記述に起因する問題点を挙げ, 次に, partial な状態による仕様記述法を提案する. 最後に, 呼処理の仕様記述と解釈を例示する.

2 total な状態による仕様記述法の問題点

通常, 呼処理の仕様は状態遷移規則の集まりとして与えられる. 状態遷移規則は, Event, Situation, および Task の 3 項組である. Event は状態遷移を引き起こすきっかけとなる Action であり, Task は状態遷移において実行される Action の集まりである. Situation は Event の生起に対して, Task を実行するか否かを判定するための State の集まりである.

従来の仕様の解釈では, Situation として, total な状態が与えられることを前提としている. つまり, どんな Event が起きても, 唯一の状態遷移規則が選択されるように, Situation を定める必要がある.

そのため, 仕様記述は次の手順で行われる. まず, 仕様全体を吟味し, 機能上必要な Situation を掌握する. 次に, 各々の Situation に対し, その場合に起こり得る Event を挙げる. 最後に, Situation と Event の組に対し, その場合に実行する Task を決める.

このように, total な状態記述を前提とする限り, 記述に先立つ仕様全体の分析を必要とする. また, そのような分析自体が難しいので, 仕様の記述性, 保守性が悪く, 部品化できないなどの問題が生ずる.

3 partial な状態による仕様記述法

partial な状態記述を前提とする仕様記述法の形式と, その解釈を次のように定義する.

3.1 形式

1) 状態遷移規則は次のような 3 項述語である.

pst(Event, Situation, Task)

2) Event は唯一の Action である.

3) Situation は State の集合である.

4) Task は Action の集合である.

*Specification Description using Partial Situation in Call Processing

†Shunsuke NAKAJIMA, Masaaki HASHIMOTO, Michihiro MONDEN
‡ATR Communication Systems Research Laboratories

3.2 解釈

Event e と, その時点の Situation S_0 に対して,

1) e, S_0 に対して以下の展開規則を適用し, 活性化 Task 集合 A を求める.

2) A に対して以下の集約規則を適用し, 不活性化 Task 集合 D を求める.

3) $A - D$ がその時点の実行 Task 集合である.

*) 活性化 Task, 不活性化 Task, および実行 Task は $t(T_1, S_1)$ の形式で表される. 実行する場合の解釈は Task T_1 と同等であるが, 実行可能となった Situation S_1 が付加されている.

**) S_0 に対して, State s の評価を行うための Action $eval(s)$ は, 仕様解釈の時点で実行されるものとする.

展開規則

Event e, Situation S_0 に対する活性化 Task 集合 A を定義する:

1) $pst(e, S_1, T_1)$ かつ $S_1 \subset S_0$ ならば, $t(T_1, S_1) \in A$.

2) $t(T_1, S_1) \in A$ かつ $a \in T_1$ かつ $pst(a, S_2, T_2)$ かつ $S_2 \subset S_0$ ならば, $t(T_2, S_1 \cup S_2) \in A$.

集約規則

活性化 Task 集合 A に対する不活性化 Task 集合 D を次に定義する:

相違なる $t(T_1, S_1), t(T_2, S_2) \in A$ に対して, T_1 と T_2 が衝突し, かつ $S_1 \subset S_2$ ならば, $t(T_1, S_1) \in D$.

4 呼処理の仕様

通常通話の仕様記述の一部を図 1 に示し, 2 つの場合の解釈について述べる. なお, いずれの場合においても, 変わることのない Situation を次に示しておく.

```
[callNumber(p1, 11), callNumber(p2, 12),
 idleTone(it), dialTone(dt),
 ringbackTone(rbt),
 ringTone(rgt), busyTone(bt)]
```

dial した相手が busy の時

1) Event `dial(p1, 12)` が起きる.

2) その時の Situation を次のように仮定する.

```
[busy(p1), offhook(p1), connected(p1, dt),
 busy(p2), offhook(p2), connected(p2, dt)]
```

3) 展開規則を適用する. 次の 3 つの活性化 Task が得られる.
(規則 1 により pst1, 3 から t1, 2 が得られ,
規則 2 により t1 と pst2 から t3 が得らる)

```
% t1.
t([assert(busy(p2)),
   assert(calling(p1, p2)),
   connect(p1, rbt), connect(p2, rgt)],
  [dialTone(dt), connected(p1, dt),
   callNumber(p2, 12)]),
% t2.
t([connect(p1, bt)],
  [dialTone(dt), connected(p1, dt)]),
% t3.
t([connect(p1, bt)],
  [dialTone(dt), connected(p1, dt),
   callNumber(p2, 12), busy(p2)])
```

4) 集約規則を適用する. t1, 2 が不活性化 Task となる.
(Action connect(p1, _) が互いに衝突し,
t3 の Situation が最も大きいため)

5) 従って, t2 が実行 Task となる.

dial された場合には, 番号が誤っているか否か, 相手が busy か否かにより実行する Task が異なる. 従って, total な状態記述を前提とする仕様記述法では, それらの State を Situation に明記しなければならない. ところが, pst1 には相手が busy ではないという State がなく, pst3 には番号が誤っているという State が記述されていない. つまり, pst1, 3 はそれらの State を考慮せずに記述されたもので, 特別な場合を除いて, それぞれの Task を実行せよと解釈される. また, pst2 は pst1 の Task 中の Action を Event としており, pst1 に対する特別な場合として解釈される.

通話中に一方が onhook した時

1) Event onhook(p1) が起きる.

2) その時の Situation を次のように仮定する.

```
[calling(p1, p2),
 busy(p1), offhook(p1), connected(p1, p2),
 busy(p2), offhook(p2), connected(p2, p1)]
```

3) 展開規則を適用する. 2 つの活性化 Task が得られる.
(規則 1 により pst4, 5 から t4, 5 が得られる)

```
% t4.
t([retract(busy(p1)), connect(p1, it)],
  []),
% t5.
t([retract(calling(p1, p2)),
   connect(p2, bt)],
  [calling(p1, p2)])
```

4) 集約規則を適用する. t4, 5 とも不活性化 Task とならない.

5) 従って, t4, 5 が実行 Task となる.

onhook された場合には, どんな状態であっても idle 状態へ遷移する必要があり, その時の状態によって実行する Task が異なる. 従って, total な状態記述を前提とする仕様記述法では, すべての Situation に対して, onhook の状態遷移規則を与えなければならない. ところが, pst4 の Situation には何の State も記述されておらず, どんな Situation であっても実行する Action が Task として与えられている. また, pst5 は通話相手に関する状態遷移であり, 状態により実行が異なる Action が別の状態遷移規則として記述されている.

```
/* 通話 */
% pst1.
pst(dial(P1, N),
    [dialTone(T1), connected(P1, T1),
     callNumber(P2, N)],
    [assert(busy(P2)),
     assert(calling(P1, P2)),
     eval(ringbackTone(T2)), connect(P1, T2),
     eval(ringTone(T3)), connect(P2, T3)]).

/* 話中 */
% pst2.
pst(assert(calling(P1, P2)),
    [busy(P2)],
    [eval(busyTone(T1)), connect(P1, T1)]).

/* 番号誤り */
% pst3.
pst(dial(P1, _),
    [dialTone(T1), connected(P1, T1)],
    [eval(busyTone(T2)), connect(P1, T2)]).

/* 放棄 */
% pst4.
pst(onhook(P1),
    [],
    [retract(busy(P1)),
     eval(idleTone(T1)), connect(P1, T1)]).
% pst5.
pst(onhook(P1),
    [calling(P1, P2)],
    [retract(callingP1, P2)),
     eval(busyTone(T1)), connect(P2, T1)]).
```

図 1: 通常通話の仕様 (一部)

5 おわりに

本稿では, partial な状態による仕様記述法を提案した. また, 呼処理の仕様記述と解釈の例を通して, partial な状態記述を前提とする考え方の有用性を示した.