ホテル

佐原伸
（株）CSK システムズ
通信グループ
VDM 担当

2006 年 10 月 5 日

概要

参考文献 [1] 付録 E のホテル部屋の鍵掛けの例題の欠陥を修正し、日本語化し、実行可能仕様とした。

目次

1　問題の概要

ホテルの客は、チェックイン時に、フロントで、部屋の鍵となるカードを受け取る。
カードは 2 つの鍵を持ち、この鍵はその後変わることはない。
フロントは、これまでに発行したカードと、使用済みのカードを記録している。
客が、カードを使って部屋に入ると、前の客のカードでは部屋に入れなくなる。
客は、再度チェックインすることで、新たな鍵となるカードを受け取ることができるが、新しいカードを使うと、古いカードでは部屋に入れなくなる。

1.1 Hotel

ホテルの状態を、状態変数 Hotel で表す。

発行済の「鍵」の集合（発行済）と、どの部屋にどの鍵が使用されたかの情報（使用済）は、フロントが管理している。どの部屋にどの鍵が対応しているかは、「部屋」から「鍵」への写像（部屋の鍵）が記録している。どの客がどのカードを持っているかは、「客」から 「カード」集合への写像（客）が記録している。新しい鍵の候補は、keys が持っている。

参考文献 [1] では、新しい鍵を割り当てるために型束縛を使っているが、仕様を実行可能とするため、本仕様では、新しい鍵は 1 から 9999 までの値を取るようにして、集合束縛により実行が可能なようにした。この値を、あまり大きくすると、仕様実行の際、初期設定で時間がかかってしまう。

もちろん、「鍵」や 「部屋」の型を nat にしているのは、実装のための指定ではない。型の詳細をまだ決めたくない場合、一般的には token 型を使うのだが、テストケースの記述が長くなるのを避けるため、便宜上、nat 型を使っているだけである。

module『ホテル』

exports all

definitions

types

1.0 「鍵」= ℕ;
2.0 「部屋」= ℕ;
3.0 「客」= token;
4.0 「カード」:: 第一 :「鍵」
.1 第二 :「鍵」;
5.0 「フロント」:: 発行済 :「鍵」-set
.1 使用済 :「部屋」$\xrightarrow{m}$「鍵」
.2 inv $f \triangleq$ rng f.使用済 $\subseteq f$.発行済

1.1.1 チェックイン

新しい鍵を持つ新カードを発行し、その鍵を発行済・使用済の情報に記録する。新カードの情報を、客写像に追加する。

6.0 state $Hotel$ of
.1 フロント :「フロント」
.2 部屋の鍵 :「部屋」$\xrightarrow{m}$「鍵」
.3 客 :「客」$\xrightarrow{m}$「カード」-set
.4 $keys$: ℕ-set
.5 inv $h \triangleq$
.6 dom h.フロント.使用済 $\subseteq$ dom h.部屋の鍵 $\wedge$
.7 $\bigcup\{\{c.第一, c.第二\} \mid c \in \bigcup$ rng h.客 $\} \subseteq h$.フロント.発行済
.8 init $h \triangleq h =$
.9 mk-$Hotel$
.10 (
.11 mk-「フロント」$(\{\}, \{\mapsto\})$,
.12 $\{\mapsto\}$,
.13 $\{\mapsto\}$,
.14 $\{1, \ldots, 9999\}$)
.15 end

operations

```
7.0  チェックイン : 「客」× 「部屋」 →^o 「カード」
.1   チェックイン (guest, room) ≜
.2     let 新鍵 ∈ keys be st 新鍵 ∉ フロント.発行済 in
.3     let 新カード = mk-「カード」(フロント.使用済 (room), 新鍵) in
.4     (   フロント.発行済 := フロント.発行済 ∪ {新鍵 };
.5         フロント.使用済 := フロント.使用済 † {room ↦ 新鍵 };
.6         客 := if guest ∈ dom 客
.7               then 客 † {guest ↦ 客 (guest) ∪ {新カード }}
.8               else 客 ⊌ {guest ↦ {新カード }};
.9         return 新カード
.10    )
.11  pre  room ∈ dom フロント.使用済
.12  post ∃新鍵 ∈ keys ·
.13            新鍵 ∉ ←フロント.発行済 ∧
.14            let 新カード = mk-「カード」(←フロント.使用済 (room), 新鍵) in
.15            フロント.発行済 = ←フロント.発行済 ∪ {新鍵 } ∧
.16            フロント.使用済 = ←フロント.使用済 † {room ↦ 新鍵 } ∧
.17            if guest ∈ dom ←客
.18            then 客 = ←客 † {guest ↦ ←客 (guest) ∪ {新カード }}
.19            else 客 = ←客 ⊌ {guest ↦ {新カード }} ;
```

1.1.2 チェックイン陰仕様

```
8.0  チェックイン陰仕様 : 「客」× 「部屋」 →^o ()
.1   チェックイン陰仕様 (guest, room) ≜
.2     is not yet specified
.3   pre  room ∈ dom フロント.使用済
.4   post ∃新鍵 : 「鍵」·
.5             新鍵 ∉ ←フロント.発行済 ∧
.6             let 新カード = mk-「カード」(←フロント.使用済 (room), 新鍵) in
.7             フロント.発行済 = ←フロント.発行済 ∪ {新鍵 } ∧
.8             フロント.使用済 = ←フロント.使用済 † {room ↦ 新鍵 } ∧
.9             if guest ∈ dom ←客
.10            then 客 = ←客 † {guest ↦ ←客 (guest) ∪ {新カード }}
.11            else 客 = ←客 ⊌ {guest ↦ {新カード }} ;
```

1.1.3 入る

客が部屋にはいると、部屋の鍵がカードの第二の鍵になるので、前の客の鍵は使えなくなる。

```
9.0  入る : 「部屋」× 「客」× 「カード」 →^o 𝔹
.1   入る (room, guest, c) ≜
.2     (   if c.第一 = 部屋の鍵 (room)
.3         then (   部屋の鍵 := 部屋の鍵 † {room ↦ c.第二 };
.4                  return true
.5              )
```

```
.6          elseif c.第二 = 部屋の鍵(room)
.7          then return true
.8          else return false
.9      )
.10 pre  room ∈ dom 部屋の鍵 ∧ guest ∈ dom 客 ∧
.11      c ∈ 客(guest)
.12 post if c ∈ 客(guest) ∧
.13          (c.第一 = 部屋の鍵(room) ∧ 部屋の鍵 = 部屋の鍵 † {room ↦ c.第二} ∨
.14           c.第二 = 部屋の鍵(room) ∧ 部屋の鍵 = 部屋の鍵)
.15       then RESULT = true
.16       else RESULT = false ;
```

1.1.4 入る陰仕様

```
10.0 入る陰仕様 :「部屋」×「客」 →o ℬ
.1   入る陰仕様(room, guest) ≜
.2      is not yet specified
.3   pre  room ∈ dom 部屋の鍵 ∧ guest ∈ dom 客 ∧
.4        ∃c ∈ 客(guest) · c.第一 = 部屋の鍵(room) ∨ c.第二 = 部屋の鍵(room)
.5   post if ∃c ∈ 客(guest) ·
.6             c.第一 = 部屋の鍵(room) ∧ 部屋の鍵 = 部屋の鍵 † {room ↦ c.第二} ∨
.7             c.第二 = 部屋の鍵(room) ∧ 部屋の鍵 = 部屋の鍵
.8        then RESULT = true
.9        else RESULT = false ;
```

1.1.5 ホテル状態を設定する

```
11.0 ホテル状態を設定する :「フロント」×「部屋」 →m「鍵」×「客」 →m「カード」-set →o Hotel
.1   ホテル状態を設定する(a フロント, 部屋の鍵写像, 客写像) ≜
.2      (   atomic (   フロント := a フロント;
.3                     部屋の鍵 := 部屋の鍵写像;
.4                     客 := 客写像
.5                 ) ;
.6                 return Hotel
.7      );
```

1.1.6 ホテル状態を得る

```
12.0 ホテル状態を得る : () →o Hotel
.1   ホテル状態を得る () ≜
.2      return Hotel
end 『ホテル』
```

Test Suite : vdm.tc

Module : 『ホテル』

Name	#Calls	Coverage
『ホテル』‘入る	10	√

Name	#Calls	Coverage
『ホテル』‘入る陰仕様	0	0%
『ホテル』‘チェックイン	9	√
『ホテル』‘ホテル状態を得る	9	√
『ホテル』‘チェックイン陰仕様	0	0%
『ホテル』‘ホテル状態を設定する	6	√
Total Coverage		**62%**

1.2 Hotel の回帰テストケース

module Hotel の回帰テストケース。
t1(), t2(), ... などの各テストケースが返す TestDriver‘TestCase 型の 2 番目の引数は、true を返す式でなければならない。

module $HotelT$

imports

13.0 from『ホテル』all ,

14.0 from $TestDriver$ all

exports all

definitions

values

15.0 チェックイン =『ホテル』‘チェックイン;

16.0 入る =『ホテル』‘入る;

17.0 ホテル状態を得る =『ホテル』‘ホテル状態を得る;

18.0 ホテル状態を設定する =『ホテル』‘ホテル状態を設定する

functions

19.0 $run : () \to \mathbb{B}$
.1 $run\,() \triangleq$
.2 let $testcases = [t1\,(), t2\,(), t3\,(), t4\,(), t5\,(), t6\,()]$ in
.3 $TestDriver`run\,(testcases)$

1.2.1 初めてのチェックイン

operations

20.0 $t1 : () \xrightarrow{o} TestDriver`TestCase$
.1 $t1\,() \triangleq$
.2 let $key1 = 1$,
.3 $key2 = 2$,
.4 $key3 = 3$,
.5 $room1 = 101$,
.6 $room2 = 102$,
.7 佐原 = mk-token ("佐原"),
.8 $oldhotel =$
.9 ホテル状態を設定する
.10 (
.11 mk-『ホテル』‘「フロント」$(\{key1, key2, key3\}, \{room2 \mapsto key2\})$,
.12 $\{room1 \mapsto key1, room2 \mapsto key2\}$,
.13 $\{\mapsto\})$,
.14 - = チェックイン (佐原, $room2$),

```
.15         hotel = ホテル状態を得る () in
.16     return mk-TestDriver‘TestCase
.17                 (
.18                  "HotelT01 : \t   初めてのチェックイン",
.19                  ∃新鍵 ∈ oldhotel.keys ·
.20                      新鍵 ∉ oldhotel.フロント.発行済 ∧
.21                      let 新カード = mk-『ホテル』‘「カード」(oldhotel.フロント.使用済 (room2), 新
鍵) in
.22                      hotel.フロント.発行済 = oldhotel.フロント.発行済 ∪ {新鍵 } ∧
.23                      hotel.フロント.使用済 = oldhotel.フロント.使用済 † {room2 ↦ 新鍵 } ∧
.24                      hotel.客 = oldhotel.客 ⩊ {佐原 ↦ {新カード }});
```

1.2.2 2 回目以後のチェックイン

```
21.0    t2 : () →o TestDriver‘TestCase
.1      t2 () ≜
.2        let key1 = 1,
.3            key2 = 2,
.4            key3 = 3,
.5            room1 = 101,
.6            room2 = 102,
.7            佐原 = mk-token ("佐原"),
.8            - =
.9                ホテル状態を設定する
.10                   (
.11                    mk-『ホテル』‘「フロント」({key1, key2, key3}, {room2 ↦ key2}),
.12                    {room1 ↦ key1, room2 ↦ key2},
.13                    {↦}),
.14           - = チェックイン (佐原, room2),
.15           oldhotel = ホテル状態を得る (),
.16           - = チェックイン (佐原, room2),
.17           hotel = ホテル状態を得る () in
.18       return mk-TestDriver‘TestCase
.19                   (
.20                    "HotelT02 : \t   2 回目以後のチェックイン",
.21                    ∃新鍵 ∈ oldhotel.keys ·
.22                        新鍵 ∉ oldhotel.フロント.発行済 ∧
.23                        let 新カード = mk-『ホテル』‘「カード」(oldhotel.フロント.使用済 (room2), 新
鍵) in
.24                        hotel.フロント.発行済 = oldhotel.フロント.発行済 ∪ {新鍵 } ∧
.25                        hotel.フロント.使用済 = oldhotel.フロント.使用済 † {room2 ↦ 新鍵 } ∧
.26                        hotel.客 = oldhotel.客 † {佐原 ↦ oldhotel.客 (佐原) ∪ {新カード }});
```

1.2.3 入る (第一鍵に一致)

```
22.0    t3 : () →o TestDriver‘TestCase
.1      t3 () ≜
.2        let key1 = 1,
.3            key2 = 2,
.4            key3 = 3,
.5            room1 = 101,
```

```
.6          room2 = 102,
.7          佐原 = mk-token ("佐原"),
.8          - =
.9              ホテル状態を設定する
.10                 (
.11                  mk-『ホテル』‘「フロント」({key1, key2, key3}, {room2 ↦ key2}),
.12                  {room1 ↦ key1, room2 ↦ key2},
.13                  {↦}),
.14         c = チェックイン (佐原, room2),
.15         oldhotel = ホテル状態を得る (),
.16         - = 入る (room2, 佐原, c),
.17         hotel = ホテル状態を得る () in
.18     return mk-TestDriver‘TestCase
.19                 (
.20                  "HotelT03 : \t    入る (第一鍵に一致)",
.21                  c ∈ hotel.客 (佐原) ∧
.22                  (c.第一 = oldhotel.部屋の鍵 (room2) ∧
.23                   hotel.部屋の鍵 = oldhotel.部屋の鍵 † {room2 ↦ c.第二 } ∨
.24                   c.第二 = oldhotel.部屋の鍵 (room2) ∧
.25                   hotel.部屋の鍵 = oldhotel.部屋の鍵));
```

1.2.4 入る (第二鍵に一致)

```
23.0    t4 : () →o TestDriver‘TestCase
.1      t4 () ≜
.2        let key1 = 1,
.3            key2 = 2,
.4            key3 = 3,
.5            room1 = 101,
.6            room2 = 102,
.7            佐原 = mk-token ("佐原"),
.8            - =
.9                ホテル状態を設定する
.10                   (
.11                    mk-『ホテル』‘「フロント」({key1, key2, key3}, {room2 ↦ key2}),
.12                    {room1 ↦ key1, room2 ↦ key2},
.13                    {↦}),
.14           c = チェックイン (佐原, room2),
.15           - = 入る (room2, 佐原, c),
.16           oldhotel = ホテル状態を得る (),
.17           - = 入る (room2, 佐原, c),
.18           hotel = ホテル状態を得る () in
.19       return mk-TestDriver‘TestCase
.20                   (
.21                    "HotelT04 : \t    入る (第二鍵に一致)",
.22                    c ∈ hotel.客 (佐原) ∧
.23                    (c.第一 = oldhotel.部屋の鍵 (room2) ∧
.24                     hotel.部屋の鍵 = oldhotel.部屋の鍵 † {room2 ↦ c.第二 } ∨
.25                     c.第二 = oldhotel.部屋の鍵 (room2) ∧
.26                     hotel.部屋の鍵 = oldhotel.部屋の鍵));
```

1.2.5 前の客のカードでは入れない。

```
24.0  t5 : () ^o→ TestDriver`TestCase
  .1  t5 () ≜
  .2    let key1 = 1,
  .3        key2 = 2,
  .4        key3 = 3,
  .5        room1 = 101,
  .6        room2 = 102,
  .7        佐原 = mk-token ("佐原"),
  .8        酒匂 = mk-token ("酒匂"),
  .9        - =
  .10             ホテル状態を設定する
  .11                 (
  .12                  mk-『ホテル』`「フロント」({key1, key2, key3}, {room2 ↦ key2}),
  .13                  {room1 ↦ key1, room2 ↦ key2},
  .14                  {↦}),
  .15       c = チェックイン (佐原, room2),
  .16       - = 入る (room2, 佐原, c),
  .17       c1 = チェックイン (酒匂, room2),
  .18       - = 入る (room2, 酒匂, c1),
  .19       - = ホテル状態を得る (),
  .20       err = 入る (room2, 佐原, c) in
  .21   return mk-TestDriver`TestCase
  .22                (
  .23                 "HotelT05 : \t　前の客のカードでは入れない。",
  .24                 ¬ err);
```

1.2.6 自分自身でも、前のカードでは入れないが、新しいカードでは入れる。

```
25.0  t6 : () ^o→ TestDriver`TestCase
  .1  t6 () ≜
  .2    let key1 = 1,
  .3        key2 = 2,
  .4        key3 = 3,
  .5        room1 = 101,
  .6        room2 = 102,
  .7        佐原 = mk-token ("佐原"),
  .8        - =
  .9             ホテル状態を設定する
  .10                 (
  .11                  mk-『ホテル』`「フロント」({key1, key2, key3}, {room2 ↦ key2}),
  .12                  {room1 ↦ key1, room2 ↦ key2},
  .13                  {↦}),
  .14       c = チェックイン (佐原, room2),
  .15       - = 入る (room2, 佐原, c),
  .16       c1 = チェックイン (佐原, room2),
  .17       - = 入る (room2, 佐原, c1),
  .18       - = ホテル状態を得る (),
  .19       oldcard = 入る (room2, 佐原, c),
```

```
.20          newcard = 入る (room2, 佐原, c1) in
.21      return mk-TestDriver‘TestCase
.22                  (
.23                   "HotelT05 : \t　自分自身でも、前のカードでは入れないが、新しいカードでは入れる。",
.24                   ¬ oldcard ∧ newcard)
end HotelT
```

Test Suite : vdm.tc

Module : HotelT

Name	**#Calls**	**Coverage**
HotelT‘t1	1	√
HotelT‘t2	1	√
HotelT‘t3	1	83%
HotelT‘t4	1	88%
HotelT‘t5	1	√
HotelT‘t6	1	√
HotelT‘run	1	√
Total Coverage		**95%**

1.3 TestDriver

回帰テストを実行するモジュール。
TestCase 型は、テストケース１件を表す。

```
module TestDriver

        imports

26.0      from TestLogger all

        exports all

definitions
types
27.0    TestCase :: testCaseName : char*
  .1                testResult : ℬ
```

run は、与えられたテストケース列から結果列を得る。結果がすべて true ならば全体成功メッセージを表示し、１つでも失敗があれば全体失敗メッセージを表示する。

```
functions

28.0    run : TestCase* → ℬ
  .1    run (t) ≜
  .2      let m = "Result-of-testcases.",
  .3          r = [isOK (t (i)) | i ∈ inds t] in
  .4      if ∀ i ∈ inds r · r (i)
  .5      then TestLogger`SuccessAll (m)
  .6      else TestLogger`FailureAll (m);
```

isOK は、与えられたテストケースのテスト結果を確認し、true ならば成功メッセージを表示し、false ならば失敗メッセージを表示する。

```
29.0    isOK : TestCase → ℬ
  .1    isOK (t) ≜
  .2      if GetTestResult (t)
  .3      then TestLogger`Success (t)
  .4      else TestLogger`Failure (t);
```

GetTestResult は、テスト結果を得る。

```
30.0    GetTestResult : TestCase → ℬ
  .1    GetTestResult (t) ≜
  .2      t.testResult;
```

GetTestName は、テスト名を得る。

```
31.0    GetTestName : TestCase → char*
  .1    GetTestName (t) ≜
  .2      t.testCaseName
end TestDriver
```

Test Suite : vdm.tc

Module : TestDriver

Name	#Calls	Coverage
TestDriver`run	1	86%
TestDriver`isOK	6	70%

Name	**#Calls**	**Coverage**
TestDriver‘GetTestName	6	√
TestDriver‘GetTestResult	6	√
Total Coverage		**83%**

1.4 TestLogger

テストのログを管理する関数を提供する。

```
module TestLogger
        imports
32.0      from IO all ,
33.0      from TestDriver
34.0        types TestCase
35.0        functions GetTestName
        exports all
definitions
values
```

```
36.0    historyFileName = "VDMTESTLOG.TXT"
```

Success は、成功メッセージをファイルに追加し、標準出力に表示し、true を返す。

```
operations
```

```
37.0    Success : TestDriver`TestCase ->o B
  .1    Success (t) ==
  .2      let message = TestDriver`GetTestName (t) ^ "\tOK.\n" in
  .3      (   Fprint(message) ;
  .4          Print(message) ;
  .5          return true
  .6      );
```

Failure は、失敗メッセージをファイルに追加し、標準出力に表示し、false を返す。

```
38.0    Failure : TestDriver`TestCase ->o B
  .1    Failure (t) ==
  .2      let message = TestDriver`GetTestName (t) ^ "\tNG.\n" in
  .3      (   Fprint(message) ;
  .4          Print(message) ;
  .5          return false
  .6      );
```

SuccessAll は、全体成功メッセージをファイルに追加し、標準出力に表示し、true を返す。

```
39.0    SuccessAll : char* ->o B
  .1    SuccessAll (m) ==
  .2      let message = m ^ "\tOK!!\n" in
  .3      (   Fprint(message) ;
  .4          Print(message) ;
  .5          return true
  .6      );
```

FailureAll は、全体失敗メッセージをファイルに追加し、標準出力に表示し、false を返す。

```
40.0    FailureAll : char* ->o B
  .1    FailureAll (m) ≜
  .2      let message = m ↷ "\tNG!!\n" in
  .3      (     Fprint(message) ;
  .4            Print(message) ;
  .5            return false
  .6      );
```

Print は、標準出力に文字列を表示する。

```
41.0    Print : char* ->o ()
  .1    Print (s) ≜
  .2      def - = IO`echo (s) in
  .3      skip;
```

Fprint は、現在ディレクトリの historyFileName で示されるファイルに文字列を表示する。

```
42.0    Fprint : char* ->o ()
  .1    Fprint (s) ≜
  .2      def - = IO`fecho (historyFileName, s, APPEND) in
  .3      skip
end TestLogger
```

Test Suite : vdm.tc

Module : TestLogger

Name	#Calls	Coverage
TestLogger`Print	7	√
TestLogger`Fprint	7	√
TestLogger`Failure	0	0%
TestLogger`Success	6	√
TestLogger`FailureAll	0	0%
TestLogger`SuccessAll	1	√
Total Coverage		**60%**

参考文献

参考文献

[1] Daniel Jackson. *Software Abstraction : Logic, Language, Analysis*. The MIT Press, Cambridge, Massachusetts, London, 2006.