Contents

1 The Problem **3**

2 What you must do **4**

3 Elicitation Questions and Answers **5**

4 Grammar specification of input commands **6**

5 Abstract state **7**

6 Three important use cases **8**
6.1 Use Case 1: Normal scenario 8
6.2 Use Case 2: ERROR CASES 12
6.3 Use Case 3: Stressing the Program 17

7 Function Tables **24**

List of Figures

1 **invoice-definitions.txt** 6

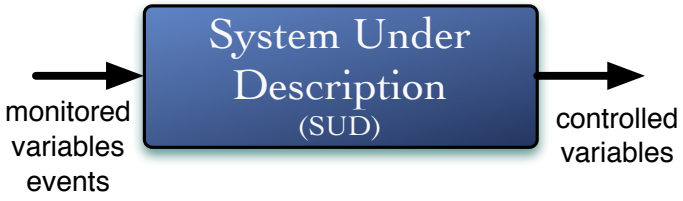
List of Tables

1 add type command function table 24
2 Error table for add type command function table 24
3 add product command function table 25
4 Error table for add product command function table 25
5 Definitions for add order command 25
6 add order function command function table 26
7 Error table for add order command function table 27
8 invoice command function table 27
9 Error table for invoice command function table 27
10 Cancel order function table 27
11 Error cancel order command function table 28
12 nothing command function table 28
13 initial state function table 28
14 invoice commands function table 29

1 The Problem

The problem: Our customer provided us with the following statement of their needs.

> The subject is to invoice orders. To invoice is to change the state of an order (to change it from the state pending to invoiced). On an order, we have one and one only reference to an ordered product of a certain quantity. The quantity can be different to other orders. The same reference can be ordered on several different orders. The state of the order will be changed into invoiced if the ordered quantity is either less or equal to the quantity which is in stock according to the reference of the ordered product. You have to take into account new orders, cancellations of orders, and entries of quantities in the stock.



A requirement is a separately *verifiable* contractual statement stating a need of the customer. A precise requirements document describes everything necessary to produce a safe and correct system—one that fulfills the needs of the customer—nothing more. At the same time the specification must not over-constrain developers by venturing into design and implementation detail.

2 What you must do

- Carefully read the instructions for the the Project and the details for Phase 1(a) of the project.[1]
- The deliverables are:
  1. (D1) Ask and answer further questions to elicit the requirements and to clarify for yourself the real customer needs. Provide the three most important questions and their answers. Ask questions from the customer (instructor/-TAs) on the forum.
  2. (D2) Complete the invoice-definitions.txt, i.e. the grammar that specifies all possible inputs of monitored events
  3. (D3) Choose the abstract state (the controlled variables)
  4. (D4) Provide at least three complete use cases (acceptance tests) that describe the output in terms of the input (i.e. relate the controlled variables to the the monitored events). You may provide more use cases, but we will be grading the first three. So make sure that the first three use cases are a good sample of possible inputs and capture success cases as well as error cases.

Provide the above information structured as show below.

[1] `https://wiki.eecs.yorku.ca/course_archive/2014-15/F/4312/protected:assignments:project:start`

3 Elicitation Questions and Answers

If our customer's needs are vague, we need to talk with them to elicit their requirements with precision and clarity. Here is an example of questions we might ask in order to clarify the role of the System Under Development (SUD):

Question 1: Once an order has been placed and is pending, is a customer allowed to request any changes be made to their order?

Answer: No. Once an order is made, no changes can be made. It can only be cancelled and a new order must be made.

Question 2: can an order be placed on hold so that another customer can place their order first?

Answer: No. The current order must be completed before another order can be placed.

Question 3: Should we allow the invoiced order Ids to be available for re-use as 1000 order ids may all be used up?. This may restrict the customer to wait for an order to be cancelled so that an order ID is available for them to place and order.

Answer: No. We want the system to be as simple as possible; we do not want any mixing or loss of orders.

4 Grammar specification of input commands

Basing yourself on your elicitation questions and answers, provide the grammar of the input commands (i.e. monitored events) of the SUD. Provide comments to justify your choice of commands

```
- invoice_definitions.txt
system invoice

nothing
  --skip

add_type(product_id: STRING)
  -- e.g. add("nuts")
  -- product types must be declared in advance of orders

add_product(a_product: STRING; quantity: INTEGER)
  -- e.g. add_product("nuts", 1000)
  -- adds 1000 nuts to stock

add_order (a_order: ARRAY[ TUPLE[ pid:STRING; no:INTEGER ] ] )
  -- e.g. place_order(<<["nuts", 5], ["bolts", 12]>> )
  -- system assigns a unique order ID seen at the output
  -- if successful, items removed from stock

invoice(order_id: INTEGER)
  -- e.g. do_invoice(1)
  -- change order ID 1 from pending to invoiced

cancel_order(order_id: INTEGER)
  -- e.g. cancel_order(1)
  -- items are returned to stock
  -- order_id is freed up
```

Figure 1: **invoice-definitions.txt**

5 Abstract state

You must complete the table below so as to describe the outputs (controlled variables) of the SUD. Provide a mathematical type, and the meaning of each variable.[2]

Variable	**Type**	**Details**
report	STRING	report ok, otherwise report error
id	$1 \cdots n$	Order ID
products	$SET[PRODUCT]$	Collection on unique products.
stock	$[(products)(i) \rightarrow NAT]$	quantity associated with a product
orders	$SET[1 \cdots n]$	set of order ids that placed an order.
carts	$[(orders)(i) \rightarrow SET[PRODUCT \rightarrow NAT]]$	set of orders placed associated with order id
order_status	$[(orders)(i) \rightarrow status]$	status of a placed order

[2]Use `http://www.tablesgenerator.com` to produce Latex tables

6 Three important use cases

You must provide three important use cases. We got you started on the first one.

6.1 Use Case 1: Normal scenario

Description: Pre Condition :

1. The DB is empty
2. No products present
3. No orders have been made so far
4. Stocks are empty

User Actions:

1. Add a product to the database
2. Add a few more unique and non empty products
3. Increase the stock of the products that were added
4. Place an order without any duplicates and non negative quantity greater than zero
5. Cancels the order that is present
6. Place two more order
7. Invoice the order that is present

Post-Condition:

1. The database is not empty
2. Successfully made orders
3. Successfully added products
4. Successfully added stocks

```
report:      ok
  id:          0
  products:
  stock:
  orders:
  carts:
  order_state:
->add_type("nuts")
  report:      ok
  id:          0
  products:    nuts
  stock:
  orders:
  carts:
  order_state:
->add_type("bolts")
  report:      ok
  id:          0
  products:    bolts,nuts
  stock:
  orders:
  carts:
  order_state:
->add_type("HAMMERS")
  report:      ok
  id:          0
  products:    bolts,hammers,nuts
  stock:
  orders:
  carts:
  order_state:
->add_product("bolts",100)
  report:      ok
  id:          0
  products:    bolts,hammers,nuts
  stock:       bolts->100
  orders:
  carts:
  order_state:
->add_product("hammers",12)
  report:      ok
```

```
  id:          0
  products:    bolts,hammers,nuts
  stock:       bolts->100,hammers->12
  orders:
  carts:
  order_state:
->add_product("nuts",2000)
  report:      ok
  id:          0
  products:    bolts,hammers,nuts
  stock:       bolts->100,hammers->12,nuts->2000
  orders:
  carts:
  order_state:
->add_product("hammers",7)
  report:      ok
  id:          0
  products:    bolts,hammers,nuts
  stock:       bolts->100,hammers->19,nuts->2000
  orders:
  carts:
  order_state:
->add_order(<<["bolts", 5], ["hammers", 1]>>)
  report:      ok
  id:          1
  products:    bolts,hammers,nuts
  stock:       bolts->95,hammers->18,nuts->2000
  orders:      1
  carts:       1: bolts->5,hammers->1
  order_state: 1->pending
->cancel_order(1)
  report:      ok
  id:          1
  products:    bolts,hammers,nuts
  stock:       bolts->100,hammers->19,nuts->2000
  orders:
  carts:
  order_state:
->add_order(<<["hammers", 1], ["bolts", 1], ["nuts", 9]>>)
  report:      ok
  id:          1
```

```
  products:    bolts,hammers,nuts
  stock:       bolts->99,hammers->18,nuts->1991
  orders:      1
  carts:       1: bolts->1,hammers->1,nuts->9
  order_state: 1->pending
->add_order(<<["hammers", 2], ["nuts", 10]>>)
  report:      ok
  id:          2
  products:    bolts,hammers,nuts
  stock:       bolts->99,hammers->16,nuts->1981
  orders:      1,2
  carts:       1: bolts->1,hammers->1,nuts->9
               2: hammers->2,nuts->10
  order_state: 1->pending,2->pending
->invoice(1)
  report:      ok
  id:          2
  products:    bolts,hammers,nuts
  stock:       bolts->99,hammers->16,nuts->1981
  orders:      1,2
  carts:       1: bolts->1,hammers->1,nuts->9
               2: hammers->2,nuts->10
  order_state: 1->invoiced,2->pending
-> ...
```

6.2 Use Case 2: ERROR CASES

Description: Pre Condition :

1. The DB is empty
2. No products present
3. No products present
4. Stocks are empty

User Actions:

1. Add a product to the database called chocolate successfully accepts
2. Add the same product but using different case letters
3. Add a different product
4. Increase the stock of a product that does not exists
5. Increase the stock of the products that were added
6. Input a negative stock quantity while adding to the stock
7. Place an order with a product that doesnt exists
8. Place an order with existing products but with duplicates
9. Place an order with negative quantity
10. Invoice an order that has already been invoiced
11. Invoice an order which doesnt exists
12. Add an empty string into the product
13. Add an order where the quantity order is greater than the stock

Post-Condition:

1. The database is not empty
2. Products have been added
3. All errors were handled succesfully with appropriate Messages

4. orders have been placed

5. All appropriate orders invoiced

```
  report:      ok
  id:          0
  products:
  stock:
  orders:
  carts:
  order_state:
->add_type("chocolate")
  report:      ok
  id:          0
  products:    chocolate
  stock:
  orders:
  carts:
  order_state:
->add_type("ChocoLate")
  report:      product type already in database
  id:          0
  products:    chocolate
  stock:
  orders:
  carts:
  order_state:
->add_type("water")
  report:      ok
  id:          0
  products:    chocolate,water
  stock:
  orders:
  carts:
  order_state:
->add_product("pepsi",100)
  report:      product not in database
  id:          0
  products:    chocolate,water
  stock:
  orders:
  carts:
```

```
  order_state:
->add_product("water",1005)
  report:      ok
  id:          0
  products:    chocolate,water
  stock:       water->1005
  orders:
  carts:
  order_state:
->add_product("chocolate",191000000)
  report:      ok
  id:          0
  products:    chocolate,water
  stock:       chocolate->191000000,water->1005
  orders:
  carts:
  order_state:
->add_product("chocolate",-5)
  report:      quantity added must be positive
  id:          0
  products:    chocolate,water
  stock:       chocolate->191000000,water->1005
  orders:
  carts:
  order_state:
->add_order(<<["nuts", 29], ["bolts", 31], ["pepsi", 300]>>)
  report:      some products in order not valid
  id:          0
  products:    chocolate,water
  stock:       chocolate->191000000,water->1005
  orders:
  carts:
  order_state:
->add_order(<<["waTer", 22], ["chOcoLate", 5], ["chOcoLate", 5]>>)
  report:      duplicate products in order array
  id:          0
  products:    chocolate,water
  stock:       chocolate->191000000,water->1005
  orders:
  carts:
  order_state:
```

```
->add_order(<<["waTer", 22], ["chOcoLate", -5]>>)
  report:      quantity added must be positive
  id:          0
  products:    chocolate,water
  stock:       chocolate->191000000,water->1005
  orders:
  carts:
  order_state:
->add_type("chips")
  report:      ok
  id:          0
  products:    chips,chocolate,water
  stock:       chocolate->191000000,water->1005
  orders:
  carts:
  order_state:
->add_product("chips",3000)
  report:      ok
  id:          0
  products:    chips,chocolate,water
  stock:       chips->3000,chocolate->191000000,water->1005
  orders:
  carts:
  order_state:
->add_order(<<["chips", 100]>>)
  report:      ok
  id:          1
  products:    chips,chocolate,water
  stock:       chips->2900,chocolate->191000000,water->1005
  orders:      1
  carts:       1: chips->100
  order_state: 1->pending
->invoice(1)
  report:      ok
  id:          1
  products:    chips,chocolate,water
  stock:       chips->2900,chocolate->191000000,water->1005
  orders:      1
  carts:       1: chips->100
  order_state: 1->invoiced
->invoice(1)
```

```
  report:      order already invoiced
  id:          1
  products:    chips,chocolate,water
  stock:       chips->2900,chocolate->191000000,water->1005
  orders:      1
  carts:       1: chips->100
  order_state: 1->invoiced
->cancel_order(10)
  report:      order id is not valid
  id:          1
  products:    chips,chocolate,water
  stock:       chips->2900,chocolate->191000000,water->1005
  orders:      1
  carts:       1: chips->100
  order_state: 1->invoiced
->invoice(10)
  report:      order id is not valid
  id:          1
  products:    chips,chocolate,water
  stock:       chips->2900,chocolate->191000000,water->1005
  orders:      1
  carts:       1: chips->100
  order_state: 1->invoiced
->add_type("")
  report:      product type must be non-empty string
  id:          1
  products:    chips,chocolate,water
  stock:       chips->2900,chocolate->191000000,water->1005
  orders:      1
  carts:       1: chips->100
  order_state: 1->invoiced
->add_order(<<["chips", 100000000]>>)
  report:      not enough in stock
  id:          1
  products:    chips,chocolate,water
  stock:       chips->2900,chocolate->191000000,water->1005
  orders:      1
  carts:       1: chips->100
  order_state: 1->invoiced
```

6.3 Use Case 3: Stressing the Program

Description: Pre Condition :

1. The DB is empty
2. No products present
3. No orders have been made so far
4. Stocks are empty

User Actions:

1. Add a product to the database
2. Add a few more unique and non empty products
3. Increase the stock of the products that were added
4. Place an order without any duplicates and non negative quantity greater than zero
5. Cancel orders successively
6. Place orders again
7. Invoice orders

Post-Condition:

1. The database is not empty
2. Successfully made orders
3. Successfully added products
4. Successfully added stocks
5. Successfully invoiced orders
6. Successfully cancels multiple orders
7. Order ID is maintained as highlighted in the requirements FIFO (first in first out)

```
  report:      ok
  id:          0
  products:
  stock:
  orders:
  carts:
  order_state:
->add_type("nuts")
  report:      ok
  id:          0
  products:    nuts
  stock:
  orders:
  carts:
  order_state:
->add_type("bolts")
  report:      ok
  id:          0
  products:    bolts,nuts
  stock:
  orders:
  carts:
  order_state:
->add_type("hammers")
  report:      ok
  id:          0
  products:    bolts,hammers,nuts
  stock:
  orders:
  carts:
  order_state:
->add_product("bolts",100)
  report:      ok
  id:          0
  products:    bolts,hammers,nuts
  stock:       bolts->100
  orders:
  carts:
  order_state:
->add_product("nuts",1005)
  report:      ok
```

```
  id:          0
  products:    bolts,hammers,nuts
  stock:       bolts->100,nuts->1005
  orders:
  carts:
  order_state:
->add_product("hammers",191000000)
  report:      ok
  id:          0
  products:    bolts,hammers,nuts
  stock:       bolts->100,hammers->191000000,nuts->1005
  orders:
  carts:
  order_state:
->add_order(<<["nuts", 29], ["bolts", 31]>>)
  report:      ok
  id:          1
  products:    bolts,hammers,nuts
  stock:       bolts->69,hammers->191000000,nuts->976
  orders:      1
  carts:       1: bolts->31,nuts->29
  order_state: 1->pending
->add_order(<<["hammers", 22], ["nuts", 5]>>)
  report:      ok
  id:          2
  products:    bolts,hammers,nuts
  stock:       bolts->69,hammers->190999978,nuts->971
  orders:      1,2
  carts:       1: bolts->31,nuts->29
               2: hammers->22,nuts->5
  order_state: 1->pending,2->pending
->add_order(<<["nuts", 100]>>)
  report:      ok
  id:          3
  products:    bolts,hammers,nuts
  stock:       bolts->69,hammers->190999978,nuts->871
  orders:      1,2,3
  carts:       1: bolts->31,nuts->29
               2: hammers->22,nuts->5
               3: nuts->100
  order_state: 1->pending,2->pending,3->pending
```

```
->add_order(<<["nuts", 10], ["hammers", 91000000]>>)
  report:      ok
  id:          4
  products:    bolts,hammers,nuts
  stock:       bolts->69,hammers->99999978,nuts->861
  orders:      1,2,3,4
  carts:       1: bolts->31,nuts->29
               2: hammers->22,nuts->5
               3: nuts->100
               4: hammers->91000000,nuts->10
  order_state: 1->pending,2->pending,3->pending,4->pending
->cancel_order(2)
  report:      ok
  id:          4
  products:    bolts,hammers,nuts
  stock:       bolts->69,hammers->100000000,nuts->866
  orders:      1,3,4
  carts:       1: bolts->31,nuts->29
               3: nuts->100
               4: hammers->91000000,nuts->10
  order_state: 1->pending,3->pending,4->pending
->cancel_order(4)
  report:      ok
  id:          4
  products:    bolts,hammers,nuts
  stock:       bolts->69,hammers->191000000,nuts->876
  orders:      1,3
  carts:       1: bolts->31,nuts->29
               3: nuts->100
  order_state: 1->pending,3->pending
->cancel_order(3)
  report:      ok
  id:          4
  products:    bolts,hammers,nuts
  stock:       bolts->69,hammers->191000000,nuts->976
  orders:      1
  carts:       1: bolts->31,nuts->29
  order_state: 1->pending
->cancel_order(1)
  report:      ok
  id:          4
```

```
  products:    bolts,hammers,nuts
  stock:       bolts->100,hammers->191000000,nuts->1005
  orders:
  carts:
  order_state:
->add_order(<<["nuts", 29], ["bolts", 31]>>)
  report:      ok
  id:          2
  products:    bolts,hammers,nuts
  stock:       bolts->69,hammers->191000000,nuts->976
  orders:      2
  carts:       2: bolts->31,nuts->29
  order_state: 2->pending
->cancel_order(5)
  report:      order id is not valid
  id:          2
  products:    bolts,hammers,nuts
  stock:       bolts->69,hammers->191000000,nuts->976
  orders:      2
  carts:       2: bolts->31,nuts->29
  order_state: 2->pending
->add_order(<<["hammers", 22], ["nuts", 5]>>)
  report:      ok
  id:          4
  products:    bolts,hammers,nuts
  stock:       bolts->69,hammers->190999978,nuts->971
  orders:      2,4
  carts:       2: bolts->31,nuts->29
               4: hammers->22,nuts->5
  order_state: 2->pending,4->pending
->add_order(<<["nuts", 100]>>)
  report:      ok
  id:          3
  products:    bolts,hammers,nuts
  stock:       bolts->69,hammers->190999978,nuts->871
  orders:      2,4,3
  carts:       2: bolts->31,nuts->29
               4: hammers->22,nuts->5
               3: nuts->100
  order_state: 2->pending,4->pending,3->pending
->add_order(<<["nuts", 10], ["hammers", 91000000]>>)
```

```
  report:      ok
  id:          1
  products:    bolts,hammers,nuts
  stock:       bolts->69,hammers->99999978,nuts->861
  orders:      2,4,3,1
  carts:       2: bolts->31,nuts->29
               4: hammers->22,nuts->5
               3: nuts->100
               1: hammers->91000000,nuts->10
  order_state: 2->pending,4->pending,3->pending,1->pending
->invoice(1)
  report:      ok
  id:          1
  products:    bolts,hammers,nuts
  stock:       bolts->69,hammers->99999978,nuts->861
  orders:      2,4,3,1
  carts:       2: bolts->31,nuts->29
               4: hammers->22,nuts->5
               3: nuts->100
               1: hammers->91000000,nuts->10
  order_state: 2->pending,4->pending,3->pending,1->invoiced
->invoice(2)
  report:      ok
  id:          1
  products:    bolts,hammers,nuts
  stock:       bolts->69,hammers->99999978,nuts->861
  orders:      2,4,3,1
  carts:       2: bolts->31,nuts->29
               4: hammers->22,nuts->5
               3: nuts->100
               1: hammers->91000000,nuts->10
  order_state: 2->invoiced,4->pending,3->pending,1->invoiced
->invoice(3)
  report:      ok
  id:          1
  products:    bolts,hammers,nuts
  stock:       bolts->69,hammers->99999978,nuts->861
  orders:      2,4,3,1
  carts:       2: bolts->31,nuts->29
               4: hammers->22,nuts->5
               3: nuts->100
```

```
                1: hammers->91000000,nuts->10
  order_state: 2->invoiced,4->pending,3->invoiced,1->invoiced
->invoice(4)
  report:      ok
  id:          1
  products:    bolts,hammers,nuts
  stock:       bolts->69,hammers->99999978,nuts->861
  orders:      2,4,3,1
  carts:       2: bolts->31,nuts->29
               4: hammers->22,nuts->5
               3: nuts->100
               1: hammers->91000000,nuts->10
  order_state: 2->invoiced,4->invoiced,3->invoiced,1->invoiced
```

7 Function Tables

	$cmds(i) = add\_type(p)$	
	$p \in products(i-1) \vee p = \epsilon$	$p \notin products(i-1) \wedge p \neq \epsilon$
$id(i)$	NC	NC
$product(i)$	NC	$products(i-1) \cup \{p\}$
$stock(i)$	NC	$stock(i-1) \cup \{p \mapsto 0\}$
$orders(i)$	NC	NC
$carts(i)$	NC	NC
$order\_state(i)$	NC	NC
$report(i)$	See table 2	ok

Table 1: add type command function table

$cmd(i) = add\_types(p) \wedge (p = \epsilon \vee p \in products(i-1))$		
condition	**error_code**	**description**
$p = \epsilon$	pd_emty	product type must be non-empty string
$p \in products(i-1)$	in_db	product type already in database
otherwise	ok	no problem

Table 2: Error table for add type command function table

	$cmds(i) = add\_prodcut(p, int)$		
	$int > 0$		$int \leqslant 0$
	$p \in products(i-1)$	$p \notin products(i-1)$	
$id(i)$	NC	NC	NC
$product(i)$	NC	NC	NC
$stock(i)$	stock(i-1) $\cup \{p \mapsto int\}$	NC	NC
$orders(i)$	NC	NC	NC
$carts(i)$	NC	NC	NC
$orderstate(i)$	NC	NC	NC
$report(i)$	ok	See table 4	See table 4

Table 3: add product command function table

$cmd(i) = add\_prodct(p, int) \wedge$ $(i > 0 \wedge p \notin products(i-1) \vee$ $i \leq 0)$		
condition	**error_code**	**description**
$p \notin products(i-1)$	p_notDB	product not in database
$i \leq 0$	qty_pos	quantity added must be positive
otherwise	ok	no problem

Table 4: Error table for add product command function table

Definitions	
$enoughstock(bag\_p)$	$\forall p \in dom(bag\_p) : bag\_p(p) - stock(i-1)(p) \geq 0$
$enoughStock$	$enoughstock(validbag?(i-1))(st)(i-1))$
$stockDEF$	$\{stock(i-1) \mid \forall\ t \in Range(arr) : stock(i)(dom(t)) = stock(i-1)(dom(t)) - t(dom(t))\}$
$orderDEF$	$\{id \in (1 \cdots n) \mid id \notin orders(i-1)\} \cup orders(i-1)$
$cartDEF$	$\{id \mapsto set[Range(arr)]\} \cup carts(i-1)$
$stateDEF$	$\{id \mapsto pending\} \cup order\_state(i-1)$
$validbag?(arr[[p, int]])$	$productsvalid \wedge no\_dup\_prod \wedge qty\_pos$
$validBag$	$validbag?(i-1)(arr)$
$cartEmpty$	$dom(arr) \in \varnothing$
$orderFull$	$\forall (1 \cdots n) \in orders(i-1)$
$products\_valid$	$\forall t \in Range(arr[[p, int]]) : dom(t) \in products(i-1)$
no_dup_prod	$\forall t, t2 \in Range(arr[[p, in]]) : dom(t) \neq dom(t2)$
qty_pos	$\forall t \in Range(arr[[p, in]]) : Range(t) \geq 0$

Table 5: Definitions for add order command

	$cmd(i) = addorder(arr[[p, int]])$				
	$\neg orderFull$				$orderFull$
	$\neg cartEmpty$			$cartEmpty$	
	$validBag$		$\neg validBag$		
	$enoughStock$	$\neg enoughStock$			
$id(i)$	$\in (1 \cdots n)$	NC	NC	NC	NC
$products(i)$	NC	NC	NC	NC	NC
$stock(i)$	$stockDEF$	NC	NC	NC	NC
$orders(i)$	$orderDEF$	NC	NC	NC	NC
$carts(i)$	$cartDEF$	NC	NC	NC	NC
$orderstate(i)$	$stateDEF$	NC	NC	NC	NC
$report(i)$	ok	see table 7	see table 7	see table 7	see table 7

Table 6: add order function command function table

$cmds(i) = add\_order(arr[[p, int]])\wedge$			
$\forall(1\cdots n) \in orders$			
$dom(arr) \in \varnothing\vee$			
$\neg validbag?(i-1)(arr)\vee$			
$\neg enoughstock(validbag?(i-1))(st)(i-1)\vee$			
condition		**error_code**	**description**
$\forall(1\cdots n) \in orders$		id_full	no more order ids left
$dom(arr) \in \varnothing$		c_ntempty	cart must be non-empty
$\neg validbag?(i-1)(arr)$	$\neg products\_valid$	p_invalid	some products in order not valid
	$\neg no\_dup\_prod$	p_dupli	duplicate products in order array
	$\neg qty\_pos$	qty_pos	quantity added must be positive
$\neg enoughstock(validbag?(i-1))(st)(i-1))$		not_instock	not enough in stock
otherwise		ok	no problem

Table 7: Error table for add order command function table

	$cmds(i) = invoice(id)$		
	$id \in orders(i-1)$		$id \notin orders(i-1)$
	$\{id \mapsto pending\} \in order\_state(i-1)$	$\{id \mapsto invoiced\} \in order\_state(i-1)$	
$id(i)$	NC	NC	NC
$product(i)$	NC	NC	NC
$stock(i)$	NC	NC	NC
$orders(i)$	NC	NC	NC
$carts(i)$	NC	NC	NC
$order_state(i)$	$\{id \mapsto invoiced\} \cup order\_state(i-1)$	NC	NC
$report(i)$	ok	see table 9	see table 9

Table 8: invoice command function table

$cmds(i) = invoice(id)\wedge$ $(id \in orders(i-1) \wedge \{id \mapsto invoiced\} \in order\_state(i-1)\vee$ $id \notin orders(i-1))$		
condition	**error_code**	**description**
$\{id \mapsto invoiced\} \in order\_state(i-1)$	id_inv	order already invoiced
$id \notin orders(i-1)$	id_invalid	order id is not valid
otherwise	ok	no problem

Table 9: Error table for invoice command function table

	$cmds(i) = cancel\_order(id)$	
	$id \in orders(i-1)$	$id \notin orders(i-1)$
$id(i)$	NC	NC
$products(i)$	NC	NC
$stock(i)$	$\{stock(i-1) \mid \forall\ p \in dom(carts(i-1)(id)) \cdot stock(i)(p) = stock(i-1)(p) + carts(i-1)(id)(p)\}$	NC
$orders(i)$	$orders(i-1) - \{id\}$	NC
$carts(i)$	$orders(i-1) - \{id \mapsto carts(i-1)(id)\}$	NC
$order\_state(i)$	$order\_state(i-1) - \{id \mapsto order\_state(i-1)(id)\}$	NC
$report(i)$	ok	see table 11

Table 10: Cancel order function table

$cmds(i) = cancel\_order(id) \wedge (id \notin orders(i-1))$		
condition	**error_code**	**description**
$(id \notin orders(i-1))$	id_invalid	order id is not valid
otherwise	ok	no problem

Table 11: Error cancel order command function table

$cmds(i) = nothing$	
$st(i)$	$st(i-1)$

Table 12: nothing command function table

init_state	
report	ok
id	0
products	emptyset
stock	emptybag
orders	emptyset
carts	emptyfun
order_state	emptyfun

Table 13: initial state function table

Invoice (st) (p ,int , arr[[p,int]] ,id)		
$i = 0$		$st(0) = init\_state$
$i > 0$	case $cmd(i) =$	
	$add\_types(p)$	$add\_types(p)(st)(i)$
	$add\_product(p, int)$	$add\_product(p, int)$
	$add\_order(arr[[p, int]])$	$add\_order(arr[[p, int]])$
	$inoice(id)$	$inoice(id)$
	$cancel\_order(id)$	$cancel\_order(id)$
	$nothing$	$nothing(st)(i)$

Table 14: invoice commands function table