INSO4101  
Homework #5

**8.1 Transportation Net Algebra**

**class**

**type**

Net, Sgm, Conn, Start

1. Net is the Net as specified in the book
2. Sgm is the segment as specified in the book
3. Conn is the connection as specified in the book.
4. Start is the starting position

**values**

seg\_to\_connection: Sgm Sgm Net 🡪 Net

1. Given two segments and a net get a new connection

rm\_segment: Sgm Net 🡪 Net

1. Given the segment to delete from a net obtain a new net that has the segment removed.

new\_connection: Conn Net 🡪 Net

1. Given the connection you want to inset obtain a new net that has the connection that was added

rm\_connection: Conn Net 🡪 Net

1. Given the connection that you want to remove from the net obtain as output a new net.

is\_empty: Net 🡪 **Bool**

1. Given a net return true if and only if empty.

empty: **Unit 🡪** Net

1. Return a empty net

get\_start: Net 🡪 Start

1. Given a net get the starting position

**axioms**

s:Sgm, n: Net, c: Conn, start: Start

is\_empty(empty())

* Checking if an empty net is empty is always true by definition

rm\_connection(c new\_connection(c net) ) c

* Removing a connection from a new connection gives you the connection

rm\_segment(s new\_segment(s net) ) s

* Removing a segment from a new segment

rm\_connection(c empty) **chaos**

* Removing a connection from an empty net is not possible

rm\_segment(s empty)  **chaos**

* Removing a segment from an empty net is not possible

seg\_to\_connection(s1 s2 net) new\_connection(c net)

* Adding 2 segments to a net is equivalent to adding a connection.

**8.2 Container Logistics Algebra**

**class**

**types**

Cont\_Ship, Cont, Cont\_Storage A, Q, B, Row, Stack\_id, Stack, Loc

* Cont\_Ship is the container ship
* Cont is the container
* Cont Storage is the container storage
* A is the Area
* B is the Bay
* Row is the row as described in the book
* Stack is the stack as the defined in the book
* Stack\_id is the stack identifies as defined in the book

**values**

load\_cont\_to\_cont\_ship: (Cont Cont\_Ship) 🡪 Cont\_Ship

* Load container to container ship an get a new container ship state.

unload\_cont\_to\_cont\_ship: (Cont Cont\_Ship) 🡪 Cont\_Ship

* Unload container to container ship an get a new container ship state.

load\_stack\_to\_quay(StackQuay) 🡪 Quay

* Load stack to quay

unload\_stack\_to\_quay((StackQuay)🡪 Quay

* Unload stack from quay

cont\_to\_cont\_storage(Cont Cont\_Storage) 🡪 Cont\_Storage **Nat**

* Move a container to a container storage, once a container is added we obtain a new container storage and a natural number for the number of containers present in the container storge.

cont\_ship\_to\_bay(Cont\_Ship) 🡪 A🡪B🡪Loc

* A container ship moves to an area, bay and then to a location.

**axioms**

1. Loading and Unloading a container from a container ship will give you the container itself.
2. Unloading a container from an empty container ship is chaos
3. Loading and Unloading a stack from a container storage gives you the stack itself. You don’t do much with this.
4. An area, a row and a bay must exist in order for the ship to dock properly.
5. Loading a container to an empty container ship causes chaos. If it does not exist it is not possible to add something to it
6. Adding a stack to an empty quay causes chaos. Again, a quay must exist for a stack to be added.

**8.3 Financial Service System Algebra**

**class**

**types**

Cust, Bank Acct, Book,Curr, Check, Stat

* Cust is the customer
* Bank is the bank entity
* Acct is the account in the bank
* Book is the book entity
* Curr is the currency
* Check is the Check entity
* Stat is the status of the account (En la cuenta un par de 0)

**values**

close\_acct: Bank Cust Acct 🡪 Curr

* Given that you want to close the account, this function takes a bank a customer and a account, closes the account, and returns the currency in that account.

open\_acct: Bank Cust 🡪 Acct

* A customer wants to open an account, the function takes a bank and a customer and returns an account.

exchange\_currency: Curr 🡪 **Nat**  Curr.

* Given a certain currency the exchange function gives you a new currency and the amount equivalent to the exchange currency.

get\_status:Bank Acct 🡪 Stat

* Given the Bank and the account, this function allows the account owner to get his account status.

add\_to\_acct: Bank Acct Curr🡪Stat

* A user can add money to his/her account with this function. The function returns a new status that reflects the amount of money added to the account.

rm\_money\_from\_acct: Bank Acct Curr **Nat** 🡪 Stat Curr **Nat**

* A user can remove money from a bank account given the currency and the amount of money to remove. The function returns the new account status, the currency and the amount of money received.

**Axioms**

1. Checking the status of an account, that doesn’t exist returns chaos.
2. Opening and closing the account will return the same status, i.e. not altering the status.
3. Depositing funds to an account that exist returns chaos.
4. Adding an amount to an account and then removing does not alter the account status.
5. Closing an account that doesn’t exist returns chaos.
6. Exchanging currency in account that doesn’t exist returns chaos
7. Exchanging currency from one currency to another and then exchanging to the original currency does not affect the account status.

**9.1 Predicates over Transportation Net domain**

**class**

**type**

Net, Sgm, Conn, Start

* Net is the Net as specified in the book
* Sgm is the segment as specified in the book
* Conn is the connection as specified in the book.
* Start is the starting position

**values**

observe\_sgm\_from\_conn: Conn 🡪 Sgm**-set**

* **Given a connection obtain the set of segments that you can observe from it**

observe\_conn\_from\_sgm: Sgm 🡪 Conn

* Given a segment observe a connection from it.

empty\_con: **Unit 🡪** Conn

* Return a empty connection

empty\_smg: **Unit 🡪 Sgm**

* Return a empty segment

**axioms**

s:Sgm, n: Net, c: Conn, start: Start

is\_empty(empty())

* Checking if an empty net is empty is always true by definition

s:Sgm observe\_conn\_from\_sgm(s)

* There exists a segment such that you see a connection

c:Conn observe\_sgm\_from\_conn(c)

* There exists a connection such that you can see a set of segments

rm\_connection(c new\_connection(c net) ) c

* Removing a connection from a new connection gives you the connection

rm\_segment(s new\_segment(s net) ) s

* Removing a segment from a new segment

rm\_connection(c empty) **chaos**

* Removing a connection from an empty net is not possible

rm\_segment(s empty)  **chaos**

* Removing a segment from an empty net returns chaos.

observe\_sgm\_from\_conn(empty\_conn) **chaos**

* **One cannot observe a segment from an empty connection**

observe\_conn\_from\_sgm(empty\_sgm) **chaos**

* **One cannot observe a connection from an empty segment**

2. A constraint would be, if we see the net as graph, then there must two segments for a connection to exist

3. **class**

**type**

Net, Sgm, Conn, Start

* Net is the Net as specified in the book
* Sgm is the segment as specified in the book
* Conn is the connection as specified in the book.
* Start is the starting position

**values**

sgm\_exist: Net 🡪 **Bool**

new\_sgm: Net Sgm🡪 Net

contains\_new\_sgm: Net 🡪 **Bool**

seg\_to\_connection: Sgm Sgm Net 🡪 Net

* Given two segments and a net get a new connection

rm\_segment: Sgm Net 🡪 Net

* Given the segment to delete from a net obtain a new net that has the segment removed.

new\_connection: Conn Net 🡪 Net

* Given the connection you want to inset obtain a new net that has the connection that was added

rm\_connection: Conn Net 🡪 Net

* Given the connection that you want to remove from the net obtain as output a new net.

is\_empty: Net 🡪 **Bool**

* Given a net return true if and only if empty.

empty: **Unit 🡪** Net

* Return a empty net

get\_start: Net 🡪 Start

* Given a net get the starting position

observe\_sgm\_from\_conn: Conn 🡪 Sgm**-set**

* **Given a connection obtain the set of segments that you can observe from it**

observe\_conn\_from\_sgm: Sgm 🡪 Conn

* Given a segment observe a connection from it.

empty\_con: **Unit 🡪** Conn

* Return a empty connection

empty\_smg: **Unit 🡪 Sgm**

* Return a empty segment

**axioms**

s:Sgm, n: Net, c: Conn, start: Start

is\_empty(empty()) = True

* Checking if an empty net is empty is always true by definition

Precondition

n:Net sgm\_exist:n

Post-condition

s:Sgm contains\_new\_sgm(n)

s:Sgm observe\_conn\_from\_sgm(s)

* There exists a segment such that you see a connection

c:Conn observe\_sgm\_from\_conn(c)

* There exists a connection such that you can see a set of segments

rm\_connection(c new\_connection(c net) ) c

* Removing a connection from a new connection gives you the connection

rm\_segment(s new\_segment(s net) ) s

* Removing a segment from a new segment

rm\_connection(c empty) **chaos**

* Removing a connection from an empty net is not possible

rm\_segment(s empty)  **chaos**

* Removing a segment from an empty net returns chaos.

observe\_sgm\_from\_conn(empty\_conn) **chaos**

* **One cannot observe a segment from an empty connection**

observe\_conn\_from\_sgm(empty\_sgm) **chaos**

* One cannot observe a connection from an empty segment

4.

**class**

**type**

Net, Sgm, Conn, Start

* Net is the Net as specified in the book
* Sgm is the segment as specified in the book
* Conn is the connection as specified in the book.
* Start is the starting position

**values**

conn\_exist: Net 🡪 **Bool**

* Given a net check if it exists

new\_conn: Net Conn🡪 Net

* Add new connection to the net

contains\_new\_conn: Net 🡪 **Bool**

* Checks for new connections

empty\_smg: **Unit 🡪 Sgm**

* Return a empty segment

**axioms**

s:Sgm, n: Net, c: Conn, start: Start

is\_empty(empty())

* Checking if an empty net is empty is always true by definition

Precondition

n:Net conn\_exist:n

Post-condition

c:Conn contains\_new\_conn(n)

Precondition

n:Net sgm\_exist:n

Post-condition

s:Sgm contains\_new\_sgm(n)

s:Sgm observe\_conn\_from\_sgm(s)

* There exists a segment such that you see a connection

c:Conn observe\_sgm\_from\_conn(c)

* There exists a connection such that you can see a set of segments

**9.2 Predicate over Container Logistics Domain**

**class**

**types**

Cont\_Ship, Cont, Cont\_Storage A, Q, B, Row, Stack\_id, Stack, Loc, H

* Cont\_Ship is the container ship
* Cont is the container
* Cont Storage is the container storage
* A is the Area
* B is the Bay
* Row is the row as described in the book
* Stack is the stack as the defined in the book
* Stack\_id is the stack identifies as defined in the book
* H is the height of the stacks

**values**

max\_height 🡪 **Nat**

get\_max\_height: B Cont\_Storage 🡪 **Nat**

* Given the Bay or Container Storage return the Height of the Stack

load\_cont\_to\_cont\_ship: (Cont Cont\_Ship) 🡪 Cont\_Ship

* Load container to container ship an get a new container ship state.

unload\_cont\_to\_cont\_ship: (Cont Cont\_Ship) 🡪 Cont\_Ship

* Unload container to container ship an get a new container ship state.

**Axioms**

**b:B (get\_max\_height(b) < max\_height) = True**

**9.3 Predicate over the Financial Service Industry Domain.**

**class**

**types**

Cust, Bank Acct, Book,Curr, Check, Stat, T, I, P, sec\_exchange, Q

* T is the time
* I is the name
* P is the transaction price
* sec\_exchange is the security exchange.
* Q is the cumulative Cuantity

**values**

buy\_order: T I 🡪 P Q

* Given the time and name return the price for the buy order

sell\_order: T I 🡪 PQ

* Given the time and name return the price for the sell order

transaction: T I sec\_exchange 🡪 P**-set**

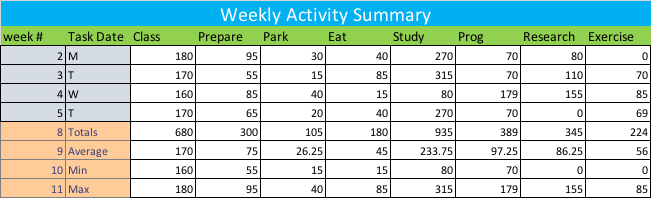
* Given the time, name and security exchange return the price interval set.

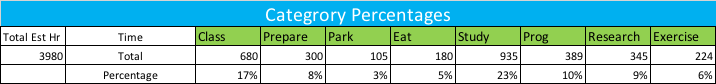
**Axioms**

t:T, i:I,sec:sec\_exchange (buy\_order: t i 🡪 p q V sell\_order: t i 🡪 p q | p transaction: T I sec

**Logs:**

****

****

****

****