INSO4101
Homework #5

8.1 Transportation Net Algebra

class

type

Net, Sgm, Conn, Start

a) Net is the Net as specified in the book
b) Sgm is the segment as specified in the book
c) Conn is the connection as specified in the book.
d) Start is the starting position

values

seg_to_connection: Sgm × Sgm ×Net → Net

a) Given two segments and a net get a new connection

rm_segment: Sgm × Net → Net

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

new_connection: Conn × Net → Net

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

rm_connection: Conn × Net → Net

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

is_empty: Net → **Bool**

e) Given a net return true if and only if empty.

empty: **Unit** → Net

a) Return a empty net

get_start: Net → Start

b) 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(Stack×Quay) → Quay

- Load stack to quay

unload_stack_to_quay((Stack×Quay)→ 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

a) Loading and Unloading a container from a container ship will give you the container itself.
b) Unloading a container from an empty container ship is chaos
c) Loading and Unloading a stack from a container storage gives you the stack itself. You don't do much with this.
d) An area, a row and a bay must exist in order for the ship to dock properly.
e) Loading a container to an empty container ship causes chaos. If it does not exist it is not possible to add something to it
f) 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

a) Checking the status of an account, that doesn't exist returns chaos.
b) Opening and closing the account will return the same status, i.e. not altering the status.
c) Depositing funds to an account that exist returns chaos.
d) Adding an amount to an account and then removing does not alter the account status.
e) Closing an account that doesn't exist returns chaos.
f) Exchanging currency in account that doesn't exist returns chaos

g) 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 → P×Q

- 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:

Time Log								
13/Sep/2018								
date	start	stop	interrupt	net	act	comment	Completed	Units
10-Sep	7:25	9:00		95	prepare	read news, breakfast		
	9:00	9:30		30	park	find parking space		
	9:30	10:20	10	40	class	lecture and waisting time on twitter		
	10:30	12:10		40	eat	lunch		
	12:30	1:20		50	class	lecture		
	1:30	3:00	10	80	research	read assigned papers	x	2
	3:00	5:00	30	90	study	read instructions for HW3, break, phone		
	5:00	6:00	30	30	study	read ch3	x	1
	6:00	7:30		90	class	lecture		
	7:00	8:30	20	70	prog	merge code research team & chat with team	x	2
	8:40	11:30	20	150	study	quiz prep, chat, leisure	x	1
date	start	stop	interrupt	net	act	comment	Completed	Units
11-Sep	7:25	8:30		55	prepare	read news, breakfast		
	8:30	8:55	10	15	park	parking time and chat with friends		
	9:00	10:20		80	class	lecture		
	10:30	12:20		110	research	research meeting		
	12:30	1:15		45	eat	lunch		
	1:15	4:30	30	165	study	quiz prep, began reading chapter	x	1
	4:30	4:55	15	15	prog	read requirements for project		
	5:00	6:30		90	class	lecture		
	6:40	7:20		40	eat	supper		
	7:30	10:40	40	150	study	read ppt before class		
	10:50	12:00		70	exercise	go for a run		
date	start	stop	interrupt	net	act	comment	Completed	Units
12-Sep	7:25	9:00	10	85	prepare	read news, breakfast		
	9:00	9:25		25	park	find parking space		
	9:30	10:20	2	48	class	lecture		
	10:30	12:05		95	eat	lunch with friends and colleagues		
	12:06	12:25	10	9	prog	research code and chat with friends		
	12:25	1:27		62	class	lecture		
	1:30	4:35	30	155	research	meeting and programming		
	4:35	4:50		15	eat	buy coffee and waist time on twitter		
	4:50	6:10		80	study	quiz prep, read notes before lecture	x	1
	6:30	7:20		50	class	lecture		
	7:20	10:30	20	170	prog	HW	x	1
	10:30	12:00	5	85	exercise	go for a run		
date	start	stop	interrupt	net	act	comment	Completed	Units
13-Sep	7:25	8:30		65	prepare	read news, breakfast		
	8:30	8:55	10	20	park	find parking space		
	9:00	10:20		80	class	lecture		
	10:30	12:20		110	study	study lecture notes	x	1
	12:30	1:15		45	eat	lunch		
	1:15	4:00	30	135	prog	began programming for project		
	4:00	4:55		55	prog	programming for reasearch	x	1
	5:00	6:30		90	class	lecture		
	6:45	7:20		35	eat	supper		
	7:30	10:40	30	160	study	study	x	1
	10:50	23:59		69	exercise	do some exercise with my friends		

Weekly Activity Summary

week #	Task Date	Class	Prepare	Park	Eat	Study	Prog	Research	Exercise
2	M	180	95	30	40	270	70	80	0
3	T	170	55	15	85	315	70	110	70
4	W	160	85	40	15	80	179	155	85
5	T	170	65	20	40	270	70	0	69
8	Totals	680	300	105	180	935	389	345	224
9	Average	170	75	26.25	45	233.75	97.25	86.25	56
10	Min	160	55	15	15	80	70	0	0
11	Max	180	95	40	85	315	179	155	85

Categrory Percentages

Total Est Hr	Time	Class	Prepare	Park	Eat	Study	Prog	Research	Exercise
3980	Total	680	300	105	180	935	389	345	224
	Percentage	17%	8%	3%	5%	23%	10%	9%	6%

Proposed Schedule for New Tasks

Time	Monday	Tuesday	Wednesday	Thursday	Friday	Saturday	Sunday
7:00	Prepare	Prepare	Prepare	Prepare	Prepare		
8:00	Lecture	Lecture	Lecture	Lecture	Lecture		
9:00	Lecture	Lecture	Lecture	Lecture	Lecture		
10:00	eat	Lecture	eat	Lecture	Code Res		
11:00	Code Res	research	Code Res	*Leisure*	Prog		
12:00	HW3	HW3	HW3	HW3	HW3		
13:00	HW3	Lecture	HW3	Lecture	HW3		
14:00	Lecture	Lecture	Lecture	Lecture	Study		
15:00	Lecture	*Leisure*	Lecture	*Leisure*	Study		
16:00	Study	eat	Prog	eat	Study		
17:00	Prog	Study	Prog	Study	*Leisure*		
18:00	Code Res	Study	Code Res	Study	*Leisure*		
19:00	eat	Lecture	eat	Lecture	*Leisure*		
20:00	Lecture	Lecture	Lecture	Lecture	*Leisure*		
21:00	Lecture	Lecture	Lecture	Lecture	*Leisure*		
22:00	Study	*Leisure*	Study	*Leisure*	*Leisure*		
23:00	Exercise	*Leisure*	Exercise	*Leisure*	Exercise		
0:00							
1:00							