Jahdiel Alvarez

802 – 15 – 0328

INSO 4101 – 071

Prof. Marko Schutz

Homework 3: CSP Channels

Exercise 21.1 System Channel Configurations:

Scheme A =

Class

Type

Info

Channel

ch

Value

Sys: **Nat Unit** → **Unit**, Sys(m) ≡ P() || (|| {Q() | x:{1..m} })

Get_info: **Unit** → Info

Analyze_info: Info → **Unit**

P: **Unit** → **in** ch **Unit**

P() ≡ **let** i = ch ? **in** analyze_info(i) **end**; P()

Q: **Unit** → **out** ch **Unit**

Q() ≡ **let** i = Get_info() **in** ch ! i **end;** Q()

End

Scheme B =

Class

Type

QIndex, Info

Channel

Ch[1..n]

Value

Sys: **Unit** → **Unit**, Sys(m) ≡ P() || (|| {Q(q) | q:QIndex })

Get_info: **Unit** → Info

Analyze_info: Info → **Unit**

P: **Unit** → **in** {ch[q] | q:QIndex } **Unit**

P() ≡ [] { **let** i = ch[q] ? **in** analyze_info(i) **end** | q:QIndex }; P()

Q: q:QIndex → **out** ch[q] **Unit**

Q(q) ≡ **let** i = Get_info() **in** ch[q] ! i **end;** Q()

End

Scheme C =

Class

Type

QIndex, Info

Channel

Ch[1..n]

Value

Sys: **Unit** → **Unit**, Sys(m) ≡ P() || (|| {Q(q) | q:QIndex })

Get_info: **Unit** → Info

Analyze_info: Info → **Unit**

P: **Unit** → **in** {ch[q] | q:QIndex } **Unit**

P() ≡ [] { **let** i = ch[q] ? **in** analyze_info(i) **end** | q:QIndex }; P()

Q: q:QIndex → **out** ch[q] **Unit**

Q(q) ≡ **let** i = Get_info() **in** ch[q] ! i **end;** Q()

End

Exercise 8.2 Suggest a Container Logistics Algebra:

Scheme Container Logistics =

Class

Type

C_Terminal == Container Terminal, C_Ship == Container Ship, Container, Quay, CSA == Container Storage Area, Bay, Row, Stack, BI == Bay Identifier, RI == Row Identifier, SI == Stack Identifier, Manifest

Value

Manifest ≡ BI × RI × SI

Load_onto_ship: Quay × Manifest → Container × C_Ship

Unload_from_ship: C_Ship × Manifest →Container × Quay

Load_onto_storage: Quay × Manifest → Container × CSA

Unload_from_storage: CSA × Manifest →Container × Quay

Axiom

Loading/Unloading onto/from a full CSA produces **chaos**.

Loading/Unloading onto/from a full C_Ship produces **chaos**.

Providing a SI which represents a full stack produces **chaos**.

Loading a container onto an empty stack and then unloading a container from that stack would move the same container.

End

Exercise 8.3 Suggest a Financial Service Industry Algebra:

Scheme Financial Service Industry =

Class

Type

Banks, Customer, Funds, B_Account == Bank Account, Insurance Company, Stocks, Security Instrument, BAR == Bank Account Record, Fund Amount

Value

Open_account: Customer × BAR → B_Account

Close_account: Customer × BAR → BAR

Set_shared_account: Customer-**set** × B_Account × BAR → B_Account

Deposit_fund: Customer × B_Account × Fund Amount → Funds × B_Account

Withdraw_funds: Customer × B_Account × Funds → B_Account

Transfer_funds: Customer-**set** × B_Account-**set** × Fund Amount → B_Account-**set**

Axiom

Withdrawing from empty account produces **chaos**.

If Customer and B_Account do not match then produce **chaos.**

Transfering funds from empty account produces **chaos**.

End

Exercise 9.1 Predicates over the Transportation Net Domain:

Type

Net, Segment, Connection, SI == Segment Identifier, CI == Connection Identifier, **Bool**

Value

Obs_Ss: Net → Segment-**set**

Obs_Cs: Net → Connection-**set**

Obs_Si: Segment → SI

Obs_Ci: Connection → CI

View_Cs: Segment → Connection-**set**

View_Ss: Connection → Segment-**set**

Axiom

∀ s:Segment ● ∀ c:Connection ● c ∈ View_Cs(s) ⇒ s ∈ View_Ss(c)

∀ c:Connection ● ∀ s:Segment ● s ∈ View_Ss(c) ⇒ c ∈ View_Cs(s)

Insert Segment –

Value

Insert_segment: Connection × Connection × Segment → Net

Axiom

∀ c,c':Connection, s:Segment ●

Insert_segment(c)(c')(s)

Pre succ(c) = c' ∧ press(c') = c

Post succ(c) = s ∧ press(c') = s

Insert Connection –

Value

Insert_connection: Segment × Connection → Net

Axiom

∀ c:Connection, s:Segment, n:Net ●

Insert_connection (s)(c)

Pre (s ∈ obs_Ss(n))= true

Post (c ∈ obs_Cs(n)) = true

Exercise 9.2 A Predicate over the Container Logistics Domain:

Type

Bay, Row, Stack, Height

Value

Obs_Rs: Bay → Row-**set**

Obs_Stacks: Row → Stack-**set**

Obs_max_height: Bay → Height

Obs_height: Stack → Height

Axom

∀ bay:Bay ● ∀ row:Row ● row ∈ obs_Rs(bay)

⇒ ∀ stack:Stack ● stack ∈ obs_Stacks(row)

⇒ Obs_Height(stack) ≤ Obs_max_height(bay)