

CONTEXT CCTx_Abstract_DLT_c1

SETS

GATEWAYS
SOURCE_TRANSACTIONS
CROSS_CHAIN_SMART_CONTRACTS
SMART_CONTRACT_EVENTS
TARGET_TRANSACTIONS

CONSTANTS

source_smart_contract
target_smart_contract
gateway

AXIOMS

axm1:: $source_smart_contract \in CROSS_CHAIN_SMART_CONTRACTS$
axm2:: $target_smart_contract \in CROSS_CHAIN_SMART_CONTRACTS$
axm3:: $gateway \in GATEWAYS$

END

MACHINE CCTx_Abstract_DLT.m1

SEES CCTx_Abstract_DLT_c1

VARIABLES

received_source_transactions
emitted_events
subscriptions
gateway_pending_events
received_target_transactions

INVARIANTS

inv1: $received_source_transactions \in CROSS_CHAIN_SMART_CONTRACTS \leftrightarrow SOURCE_TRANSACTIONS$

inv2: $emitted_events \in CROSS_CHAIN_SMART_CONTRACTS \leftrightarrow SMART_CONTRACT_EVENTS$

inv3: $subscriptions \in GATEWAYS \leftrightarrow CROSS_CHAIN_SMART_CONTRACTS$
inv4: $gateway_pending_events \in GATEWAYS \leftrightarrow TARGET_TRANSACTIONS$
inv5: $received_target_transactions \in CROSS_CHAIN_SMART_CONTRACTS \leftrightarrow TARGET_TRANSACTIONS$

EVENTS

Initialisation

begin

act1: $received_source_transactions := \emptyset$
act2: $emitted_events := \emptyset$
act3: $subscriptions := \emptyset$
act4: $gateway_pending_events := \emptyset$
act6: $received_target_transactions := \emptyset$

end

Event SUBSCRIBE_SMART_CONTRACT_EVENTS $\langle ordinary \rangle \hat{=}$

when

grd1: $gateway \mapsto source_smart_contract \notin subscriptions$
The gateway is not already subscribed to the smart contract events

then

act1: $subscriptions := subscriptions \cup \{gateway \mapsto source_smart_contract\}$
The gateway is subscribed to listen to the smart contract events

end

Event INITIATE_CC_TX $\langle \text{ordinary} \rangle \hat{=}$

any

transaction

where

grd1: $transaction \in SOURCE_TRANSACTIONS$

grd3: $transaction \notin received_source_transactions[\{source_smart_contract\}]$

The transaction was not received by the smart contract

then

act1: $received_source_transactions := received_source_transactions \cup \{source_smart_contract \mapsto transaction\}$

Add the transaction to the received transactions of the smart contract

end

Event EMIT_EVENT $\langle \text{ordinary} \rangle \hat{=}$

any

transaction

emitted_event

where

grd1: $source_smart_contract \mapsto transaction \in received_source_transactions$

The smart contract has a pending transaction to process

grd2: $emitted_event \notin emitted_events[\{source_smart_contract\}]$

The smart contract will always emit a new event

then

act1: $emitted_events := emitted_events \cup \{source_smart_contract \mapsto emitted_event\}$

The smart contract emits a new event related to the transaction processing

act2: $received_source_transactions := received_source_transactions \setminus \{source_smart_contract \mapsto transaction\}$

The smart contract processed the transaction

end

Event LISTEN_EVENT $\langle \text{ordinary} \rangle \hat{=}$

any

emitted_event

pending_event

where

grd1: $source_smart_contract \mapsto emitted_event \in emitted_events$

The smart contract has emitted an event

grd2: $gateway \mapsto source_smart_contract \in subscriptions$

Exist a subscription to the smart contract events

grd3: $gateway \mapsto pending_event \notin gateway_pending_events$

The event was not already listened

then

act1: $gateway_pending_events := gateway_pending_events \cup \{gateway \mapsto pending_event\}$

The event is added to the pending events to be processed by the gateway

act2: $emitted_events := emitted_events \setminus \{source_smart_contract \mapsto emitted_event\}$

The event is removed from the pending events to listen

end

Event SUBMIT_TX *<ordinary>* $\hat{=}$

any

pending_event
transaction

where

grd1: $gateway \mapsto pending_event \in gateway_pending_events$

There is one pending event to be processed

grd2: $transaction \in TARGET_TRANSACTIONS$

then

act1: $received_target_transactions := received_target_transactions \cup \{target_smart_contract \mapsto transaction\}$

The transaction is received by the target smart contract

act2: $gateway_pending_events := gateway_pending_events \setminus \{gateway \mapsto pending_event\}$

Remove the event from the pending events of the gateway

end

END

CONTEXT CCTx.Ethereum.Fabric.c2**EXTENDS** CCTx.Abstract.DLT.c1**SETS**

PERMISSIONS

USERS

CONSTANTS

read

write

gateway_user

AXIOMS**axm11:** $partition(PERMISSIONS, \{read\}, \{write\})$

Only two types of permissions exist: read and write (REQ1)

axm12: $gateway_user \in USERS$

The gateway has a Fabric user

END**MACHINE** CCTx.Ethereum.Fabric.m2**REFINES** CCTx.Abstract.DLT.m1**SEES** CCTx.Ethereum.Fabric.c2**VARIABLES**

received_source_transactions

emitted_events

subscriptions

gateway_pending_events

received_target_transactions

authenticated_users

authenticated_transactions

grants

INVARIANTS**inv11:** $authenticated_users \subseteq USERS$

Authenticated users are a subset of all the possible users

inv12: $authenticated_transactions \in received_target_transactions \rightarrow authenticated_users$

Authenticated transactions are received transactions submitted by an authenticated user

inv13: $\forall tx \cdot tx \in received_target_transactions \Rightarrow (\exists u \cdot u \in authenticated_users \wedge tx \mapsto u \in authenticated_transactions)$

Every submitted transaction to Fabric must be authenticated (REQ4)

inv14: $grants \in authenticated_users \leftrightarrow PERMISSIONS$

Users with read or write permissions

inv15: $\forall u \cdot u \in authenticated_transactions[received_target_transactions] \Rightarrow u \mapsto write \in grants$

Authenticated users that submitted a transaction must have write permissions (REQ5)

EVENTS**Initialisation** ⟨extended⟩**begin****act1:** $received_source_transactions := \emptyset$ **act2:** $emitted_events := \emptyset$ **act3:** $subscriptions := \emptyset$ **act4:** $gateway_pending_events := \emptyset$ **act6:** $received_target_transactions := \emptyset$ **act11:** $authenticated_users := \emptyset$ **act12:** $authenticated_transactions := \emptyset$ **act14:** $grants := \emptyset$ **end**

```

Event SUBSCRIBE_SMART_CONTRACT_EVENTS_IN_ETHEREUM  $\langle \text{ordinary} \rangle \hat{=}$ 
extends SUBSCRIBE_SMART_CONTRACT_EVENTS
  when
    grd1: gateway  $\mapsto$  source_smart_contract  $\notin$  subscriptions
      The gateway is not already subscribed to the smart contract events
  then
    act1: subscriptions := subscriptions  $\cup$  {gateway  $\mapsto$  source_smart_contract}
      The gateway is subscribed to listen to the smart contract events
  end

Event INITIATE_CC_TX_IN_ETHEREUM  $\langle \text{ordinary} \rangle \hat{=}$ 
extends INITIATE_CC_TX
  any
    transaction
  where
    grd1: transaction  $\in$  SOURCE_TRANSACTIONS
    grd3: transaction  $\notin$  received_source_transactions[{source_smart_contract}]
      The transaction was not received by the smart contract
  then
    act1: received_source_transactions := received_source_transactions  $\cup$  {source_smart_contract  $\mapsto$ 
      transaction}
      Add the transaction to the received transactions of the smart contract
  end

Event EMIT_EVENT_IN_ETHEREUM  $\langle \text{ordinary} \rangle \hat{=}$ 
extends EMIT_EVENT
  any
    transaction
    emitted_event
  where
    grd1: source_smart_contract  $\mapsto$  transaction  $\in$  received_source_transactions
      The smart contract has a pending transaction to process
    grd2: emitted_event  $\notin$  emitted_events[{source_smart_contract}]
      The smart contract will always emit a new event
  then
    act1: emitted_events := emitted_events  $\cup$  {source_smart_contract  $\mapsto$  emitted_event}
      The smart contract emits a new event related to the transaction processing
    act2: received_source_transactions := received_source_transactions  $\setminus$  {source_smart_contract  $\mapsto$ 
      transaction}
      The smart contract processed the transaction
  end

Event LISTEN_EVENT_IN_ETHEREUM  $\langle \text{ordinary} \rangle \hat{=}$ 
extends LISTEN_EVENT
  any
    emitted_event
    pending_event
  where
    grd1: source_smart_contract  $\mapsto$  emitted_event  $\in$  emitted_events
      The smart contract has emitted an event
    grd2: gateway  $\mapsto$  source_smart_contract  $\in$  subscriptions
      Exist a subscription to the smart contract events
    grd3: gateway  $\mapsto$  pending_event  $\notin$  gateway_pending_events
      The event was not already listened
  then
    act1: gateway_pending_events := gateway_pending_events  $\cup$  {gateway  $\mapsto$  pending_event}
      The event is added to the pending events to be processed by the gateway
    act2: emitted_events := emitted_events  $\setminus$  {source_smart_contract  $\mapsto$  emitted_event}
      The event is removed from the pending events to listen
  end

```

Event SUBMIT_TX_TO_FABRIC $\langle \text{ordinary} \rangle \hat{=}$

extends SUBMIT_TX

any

pending_event

transaction

user

where

grd1: *gateway* \mapsto *pending_event* \in *gateway_pending_events*

There is one pending event to be processed

grd2: *transaction* \in *TARGET_TRANSACTIONS*

grd11: *user* \in *authenticated_users*

Only allow authenticated users (REQ6)

grd12: *user* \mapsto *write* \in *grants*

Only allow authorized users (REQ7)

then

act1: *received_target_transactions* $:=$ *received_target_transactions* \cup {*target_smart_contract* \mapsto *transaction*}

The transaction is received by the target smart contract

act2: *gateway_pending_events* $:=$ *gateway_pending_events* \setminus {*gateway* \mapsto *pending_event*}

Remove the event from the pending events of the gateway

act11: *authenticated_transactions(target_smart_contract \mapsto transaction)* $:=$ *user*

Every submitted transaction to Fabric must be authenticated

end

Event CREATE_GATEWAY_USER $\langle \text{ordinary} \rangle \hat{=}$

when

grd1: *gateway_user* \notin *authenticated_users*

then

act1: *authenticated_users* $:=$ *authenticated_users* \cup {*gateway_user*}

Create credentials for a user (e.g. gateway) to authenticate them (REQ2)

end

Event GRANT_PERMISSION $\langle \text{ordinary} \rangle \hat{=}$

any

permission

user

where

grd1: *permission* \in *PERMISSIONS*

grd2: *user* \in *authenticated_users*

grd3: *user* \mapsto *permission* \notin *grants*

then

act1: *grants* $:=$ *grants* \cup {*user* \mapsto *permission*}

Grant permissions (e.g. write) to users (e.g. gateway) in Fabric (REQ3)

end

END

CONTEXT CCTx.Fabric.Ethereum.c2**EXTENDS** CCTx.Abstract.DLT.c1**SETS**

ADDRESS

CONSTANTS

gateway_address

AXIOMS**axm11**: $gateway_address \in ADDRESS$ **END****MACHINE** CCTx.Fabric.Ethereum.m2**REFINES** CCTx.Abstract.DLT.m1**SEES** CCTx.Fabric.Ethereum.c2**VARIABLES**

received_source_transactions

emitted_events

subscriptions

gateway_pending_events

received_target_transactions

accounts

INVARIANTS**inv11**: $accounts \in ADDRESS \rightarrow \mathbb{N}$

The balance of each address must be equal or greater than zero (REQ4)

EVENTS**Initialisation** $\langle \text{extended} \rangle$ **begin****act1**: $received_source_transactions := \emptyset$ **act2**: $emitted_events := \emptyset$ **act3**: $subscriptions := \emptyset$ **act4**: $gateway_pending_events := \emptyset$ **act6**: $received_target_transactions := \emptyset$ **act11**: $accounts := \emptyset$ **end****Event** SUBSCRIBE_SMART_CONTRACT_EVENTS_IN_FABRIC $\langle \text{ordinary} \rangle \hat{=}$ **extends** SUBSCRIBE_SMART_CONTRACT_EVENTS**when****grd1**: $gateway \mapsto source_smart_contract \notin subscriptions$

The gateway is not already subscribed to the smart contract events

then**act1**: $subscriptions := subscriptions \cup \{gateway \mapsto source_smart_contract\}$

The gateway is subscribed to listen to the smart contract events

end**Event** INITIATE_CC_TX_IN_FABRIC $\langle \text{ordinary} \rangle \hat{=}$ **extends** INITIATE_CC_TX**any***transaction***where****grd1**: $transaction \in SOURCE_TRANSACTIONS$ **grd3**: $transaction \notin received_source_transactions[\{source_smart_contract\}]$

The transaction was not received by the smart contract

then**act1**: $received_source_transactions := received_source_transactions \cup \{source_smart_contract \mapsto transaction\}$

Add the transaction to the received transactions of the smart contract

end

```

Event EMIT_EVENT_IN_FABRIC  $\langle$ ordinary $\rangle \hat{=}$ 
extends EMIT_EVENT
  any
    transaction
    emitted_event
  where
    grd1: source_smart_contract  $\mapsto$  transaction  $\in$  received_source_transactions
      The smart contract has a pending transaction to process
    grd2: emitted_event  $\notin$  emitted_events[{source_smart_contract}]
      The smart contract will always emit a new event
  then
    act1: emitted_events := emitted_events  $\cup$  {source_smart_contract  $\mapsto$  emitted_event}
      The smart contract emits a new event related to the transaction processing
    act2: received_source_transactions := received_source_transactions  $\setminus$  {source_smart_contract  $\mapsto$ 
      transaction}
      The smart contract processed the transaction
  end
Event LISTEN_EVENT_IN_FABRIC  $\langle$ ordinary $\rangle \hat{=}$ 
extends LISTEN_EVENT
  any
    emitted_event
    pending_event
  where
    grd1: source_smart_contract  $\mapsto$  emitted_event  $\in$  emitted_events
      The smart contract has emitted an event
    grd2: gateway  $\mapsto$  source_smart_contract  $\in$  subscriptions
      Exist a subscription to the smart contract events
    grd3: gateway  $\mapsto$  pending_event  $\notin$  gateway_pending_events
      The event was not already listened
  then
    act1: gateway_pending_events := gateway_pending_events  $\cup$  {gateway  $\mapsto$  pending_event}
      The event is added to the pending events to be processed by the gateway
    act2: emitted_events := emitted_events  $\setminus$  {source_smart_contract  $\mapsto$  emitted_event}
      The event is removed from the pending events to listen
  end
Event SUBMIT_TX_TO_ETHEREUM  $\langle$ ordinary $\rangle \hat{=}$ 
extends SUBMIT_TX
  any
    pending_event
    transaction
    fee
  where
    grd1: gateway  $\mapsto$  pending_event  $\in$  gateway_pending_events
      There is one pending event to be processed
    grd2: transaction  $\in$  TARGET_TRANSACTIONS
    grd11: gateway_address  $\in$  dom(accounts)
    grd12: accounts(gateway_address)  $\geq$  fee
      The gateway has enough balance to pay the validators fee (REQ5)
    grd13: fee  $>$  0
  then
    act1: received_target_transactions := received_target_transactions  $\cup$  {target_smart_contract  $\mapsto$ 
      transaction}
      The transaction is received by the target smart contract
    act2: gateway_pending_events := gateway_pending_events  $\setminus$  {gateway  $\mapsto$  pending_event}
      Remove the event from the pending events of the gateway
    act11: accounts(gateway_address) := accounts(gateway_address)  $-$  fee
      The fee is subtracted from the gateways account (REQ5)
  end

```


Event CREATE_ADDRESS_IN_ETHEREUM $\langle \text{ordinary} \rangle \hat{=}$

Users can create their address on Ethereum (REQ1 and REQ2)

any

address

where

grd1: $address \in ADDRESS$

grd2: $address \notin \text{dom}(\text{accounts})$

then

act1: $\text{accounts} := \text{accounts} \cup \{address \mapsto 0\}$

end

Event DEPOSIT_CRYPTOCURRENCY_IN_ETHEREUM $\langle \text{ordinary} \rangle \hat{=}$

Users (including the gateway) can deposit Ethers on their address (REQ3)

any

amount

address

where

grd1: $amount > 0$

grd2: $address \in \text{dom}(\text{accounts})$

then

act1: $\text{accounts}(address) := \text{accounts}(address) + amount$

end

Event SUBMIT_TRANSFER_TRANSACTION_IN_ETHEREUM $\langle \text{ordinary} \rangle \hat{=}$

The user can transfer an estimated fee to the gateways account (REQ1)

any

validator_fee

user_address

transfer_amount

where

grd1: $user_address \in \text{dom}(\text{accounts})$

The user has an account

grd2: $\text{accounts}(user_address) \geq validator_fee + transfer_amount$

The user has enough balance to do the transfer and pay the validator's fee

grd3: $validator_fee > 0$

grd4: $transfer_amount > 0$

grd5: $gateway_address \in \text{dom}(\text{accounts})$

The gateway has an account

grd6: $(\{user_address, gateway_address\} \triangleleft \text{accounts}) \cup \{gateway_address \mapsto \text{accounts}(gateway_address) + transfer_amount\} \cup \{user_address \mapsto \text{accounts}(user_address) - transfer_amount - validator_fee\} \in ADDRESS \rightarrow \mathbb{N}$

then

act1:

$\text{accounts} := (\{user_address, gateway_address\} \triangleleft \text{accounts}) \cup$

$\{gateway_address \mapsto \text{accounts}(gateway_address) + transfer_amount\} \cup$

$\{user_address \mapsto \text{accounts}(user_address) - transfer_amount - validator_fee\}$

Subtracts the transfer amount and fee from the user's account and add the transfer amount to the gateways account

end

END

CONTEXT CCTx_Preserve_Balance_Gateway_c3

EXTENDS CCTx_Fabric_Ethereum_c2

END

MACHINE CCTx_Preserve_Balance_Gateway_m3

REFINES CCTx_Fabric_Ethereum_m2

SEES CCTx_Fabric_Ethereum_c2

VARIABLES

received_source_transactions

emitted_events

subscriptions

gateway_pending_events

received_target_transactions

accounts

estimated_cross_chain_cost

INVARIANTS

inv31: $estimated_cross_chain_cost \in \mathbb{N}$

EVENTS

Initialisation $\langle \text{extended} \rangle$

begin

act1: $received_source_transactions := \emptyset$

act2: $emitted_events := \emptyset$

act3: $subscriptions := \emptyset$

act4: $gateway_pending_events := \emptyset$

act6: $received_target_transactions := \emptyset$

act11: $accounts := \emptyset$

act31: $estimated_cross_chain_cost := 0$

end

Event SUBSCRIBE_SMART_CONTRACT_EVENTS_IN_FABRIC $\langle \text{ordinary} \rangle \hat{=}$

extends SUBSCRIBE_SMART_CONTRACT_EVENTS_IN_FABRIC

when

grd1: $gateway \mapsto source_smart_contract \notin subscriptions$

The gateway is not already subscribed to the smart contract events

then

act1: $subscriptions := subscriptions \cup \{gateway \mapsto source_smart_contract\}$

The gateway is subscribed to listen to the smart contract events

end

Event INITIATE_CC_TX_IN_FABRIC $\langle \text{ordinary} \rangle \hat{=}$

extends INITIATE_CC_TX_IN_FABRIC

any

transaction

where

grd1: $transaction \in SOURCE_TRANSACTIONS$

grd3: $transaction \notin received_source_transactions[\{source_smart_contract\}]$

The transaction was not received by the smart contract

then

act1: $received_source_transactions := received_source_transactions \cup \{source_smart_contract \mapsto transaction\}$

Add the transaction to the received transactions of the smart contract

end

Event EMIT_EVENT_IN_FABRIC $\langle \text{ordinary} \rangle \hat{=}$

extends EMIT_EVENT_IN_FABRIC

any

transaction

emitted_event

where

grd1: *source_smart_contract* \mapsto *transaction* \in *received_source_transactions*

The smart contract has a pending transaction to process

grd2: *emitted_event* \notin *emitted_events*[{*source_smart_contract*}]

The smart contract will always emit a new event

then

act1: *emitted_events* := *emitted_events* \cup {*source_smart_contract* \mapsto *emitted_event*}

The smart contract emits a new event related to the transaction processing

act2: *received_source_transactions* := *received_source_transactions* \setminus {*source_smart_contract* \mapsto *transaction*}

The smart contract processed the transaction

end

Event LISTEN_EVENT_IN_FABRIC $\langle \text{ordinary} \rangle \hat{=}$

extends LISTEN_EVENT_IN_FABRIC

any

emitted_event

pending_event

where

grd1: *source_smart_contract* \mapsto *emitted_event* \in *emitted_events*

The smart contract has emitted an event

grd2: *gateway* \mapsto *source_smart_contract* \in *subscriptions*

Exist a subscription to the smart contract events

grd3: *gateway* \mapsto *pending_event* \notin *gateway_pending_events*

The event was not already listened

then

act1: *gateway_pending_events* := *gateway_pending_events* \cup {*gateway* \mapsto *pending_event*}

The event is added to the pending events to be processed by the gateway

act2: *emitted_events* := *emitted_events* \setminus {*source_smart_contract* \mapsto *emitted_event*}

The event is removed from the pending events to listen

end

Event SUBMIT_TX_TO_ETHEREUM $\langle \text{ordinary} \rangle \hat{=}$
extends SUBMIT_TX_TO_ETHEREUM
any
pending_event
transaction
fee
where
 grd1: *gateway* \mapsto *pending_event* \in *gateway_pending_events*
 There is one pending event to be processed
 grd2: *transaction* \in *TARGET_TRANSACTIONS*
 grd11: *gateway_address* \in *dom(accounts)*
 grd12: *accounts(gateway_address)* \geq *fee*
 The gateway has enough balance to pay the validators fee (REQ5)
 grd13: *fee* > 0
 grd31: *fee* \leq *estimated_cross_chain_cost*
 Check that the estimated fee is greater or equal to the validators fee (RQ2)
then
 act1: *received_target_transactions* $:=$ *received_target_transactions* \cup {*target_smart_contract* \mapsto *transaction*}
 The transaction is received by the target smart contract
 act2: *gateway_pending_events* $:=$ *gateway_pending_events* \setminus {*gateway* \mapsto *pending_event*}
 Remove the event from the pending events of the gateway
 act11: *accounts(gateway_address)* $:=$ *accounts(gateway_address)* $-$ *fee*
 The fee is subtracted from the gateways account (REQ5)
end
Event CREATE_ADDRESS_IN_ETHEREUM $\langle \text{ordinary} \rangle \hat{=}$
extends CREATE_ADDRESS_IN_ETHEREUM
any
address
where
 grd1: *address* \in *ADDRESS*
 grd2: *address* \notin *dom(accounts)*
then
 act1: *accounts* $:=$ *accounts* \cup {*address* $\mapsto 0$ }
end
Event DEPOSIT_CRYPTOCURRENCY_IN_ETHEREUM $\langle \text{ordinary} \rangle \hat{=}$
extends DEPOSIT_CRYPTOCURRENCY_IN_ETHEREUM
any
amount
address
where
 grd1: *amount* > 0
 grd2: *address* \in *dom(accounts)*
 grd31: *address* \neq *gateway_address*
 The gateway cannot deposit in its own account (RQ3)
then
 act1: *accounts(address)* $:=$ *accounts(address)* $+$ *amount*
end

Event SUBMIT_TRANSFER_TRANSACTION_IN_ETHEREUM *(ordinary)* $\hat{=}$
extends SUBMIT_TRANSFER_TRANSACTION_IN_ETHEREUM
any
validator_fee
user_address
transfer_amount
where
grd1: *user_address* $\in \text{dom}(\text{accounts})$
The user has an account
grd2: *accounts*(*user_address*) $\geq \text{validator_fee} + \text{transfer_amount}$
The user has enough balance to do the transfer and pay the validator's fee
grd3: *validator_fee* > 0
grd4: *transfer_amount* > 0
grd5: *gateway_address* $\in \text{dom}(\text{accounts})$
The gateway has an account
grd6: $(\{ \text{user_address}, \text{gateway_address} \} \triangleleft \text{accounts}) \cup \{ \text{gateway_address} \mapsto \text{accounts}(\text{gateway_address}) + \text{transfer_amount} \} \cup \{ \text{user_address} \mapsto \text{accounts}(\text{user_address}) - \text{transfer_amount} - \text{validator_fee} \} \in \text{ADDRESS} \mapsto \mathbb{N}$
then
act1:
accounts $:= (\{ \text{user_address}, \text{gateway_address} \} \triangleleft \text{accounts}) \cup \{ \text{gateway_address} \mapsto \text{accounts}(\text{gateway_address}) + \text{transfer_amount} \} \cup \{ \text{user_address} \mapsto \text{accounts}(\text{user_address}) - \text{transfer_amount} - \text{validator_fee} \}$
Subtracts the transfer amount and fee from the user's account and add the transfer amount to the gateways account
act31: *estimated_cross_chain_cost* $:= \text{transfer_amount}$
end
END