Contents

Core data Types

Data Type	**Description**
string	strings
nat	natural numbers
int	integers
bytes	bytes
bool	booleans (True or False)
unit	The only value is Unit (used as a placeholter)
list (t)	immutable, homogeneous linked list
pair (l) (r)	A pair of two values (a) and (b) of type (l) and (r): (Pair a b)
option (t)	Optional value of type (t): None or (Some v).
or (l) (r)	A union of two types: a value holding either of (l) or (r)
set (t)	Immutable sets of values of type (t)
map (k) (v)	Immutable maps from keys of type (k) of values of type (v)
big_map (k) (v)	Lazily deserialized maps from keys of type (k) of values of type (v)

Examples for Unfamiliar Datatypes

Type: or (l) (r)

A value of this type can be either a value of type (l) or of type (r).
Technically a **sum type**.
Example: Value of type **or** string **bool** can be Left "foo" or Right False.
Instructions

- LEFT :: 'a : 'S → or 'a 'b : 'S
- RIGHT :: 'b : 'S → or 'a 'b : 'S
- IF_LEFT code1 code2 :: or 'a 'b : 'S
  - removes top value,
  - executes code1 on ('a : 'S) if it was a Left 'a,
  - otherwise executes code2 on ('b : 'S) if it was a Right 'b

Examples for Unfamiliar Datatypes II

Type: set (t)

Immutable set with elements of type t. Elements must be comparable. (cf. frozenset in Python)
Instructions

- EMPTY_SET
- ITER code :: set 'a : 'S → 'S
  apply code :: 'a : 'S → S to each element of the set (cf. Python for)
- MEM
- SIZE
- UPDATE :: 'a : bool : set 'a : 'S → set 'a : 'S
  - applied to (v : b : s :...) return a new set (s' : ...) such that
  - s' has the same elements as s except
  - if b=True, then $v \in s'$
  - if b=False, then $v \notin s'$

The Set Datatype

Example: Sum of the elements of a set

```
1 # set int : S
2 PUSH int 0; SWAP;       #initialize the sum
3 # set int : int : S
4 ITER {
5 # int : int : S
6     ADD
7 # int : S
8 };
9 # int : S               # leave sum on stack top
```

Examples for Unfamiliar Datatypes III

Type: big_map (k) (v)

Instructions on big_maps have higher gas costs than those over standard maps, as data is lazily deserialized. However, a big_map has a lower storage cost than a standard map of the same size.

Stack Instructions

Instruction	**Description**
DROP	Drop the top element of the stack
DUP	Duplicate the top element of the stack
SWAP	Exchange the top two elements of the stack
PUSH 't x	Push a value of x of type t onto the stack
UNIT	Push unit value
LAMBDA 'a 'r code	Push function given by code; arg type 'a; return type 'r
NIL t	Push an empty list of type (list (t)

Instructions on pairs

Instruction	**Description**
PAIR	Build a pair from the stack's top two elements
CAR	Push the left part of a pair onto stack
CDR	Push the right part of a pair onto stack
COMPARE	Lexicographic comparison

Comparison

Instruction	**Description**
EQ	Checks that the top element of the stack is equal to zero
NEQ	Checks that the top element of the stack is not equal to zero
LT	Checks that the top element of the stack is less than zero
GT	Checks that the top element of the stack is greater than zero
LE	Checks that the top element of the stack is less than or equal to zero
GE	Checks that the top of the stack is greater than or equal to zero

Instructions on Strings

Instruction	**Description**
CONCAT	String concatenation
SIZE	The number of characters in a string
SLICE	String access
COMPARE	Lexicographic comparison

Domain specific data types

Data Type	**Description**
timestamp	Dates in real world
mutez	specific type for manipulating tokens
operation	internal operation emitted by a contract
contract	contract
address	untyped contract address
key	public cryptography key
key_hash	hash of a public cryptography key
signature	cryptographic signature

Examples for Domain-Specific Datatypes

operation

- A Tezos contract cannot manipulate the blockchain directly.
- It returns a list of operations to be executed **after** the contract terminates.
- An instance of **metaprogramming**: The contract **generates** these operations, but they are executed after termination of the generating contract.
- Values of type `operation` are created by three instructions
  - CREATE_CONTRACT
  - SET_DELEGATE
  - TRANSFER_TOKENS

TRANSFER_TOKENS Instruction

- constructs an operation to send a specified amount of tokens to a contract
- TRANSFER_TOKENS :: 'a : mutez : contract 'a : 'S
- Example: suppose the contract's address is a and it expects a parameter of type string

  TRANSFER_TOKENS :: "hello" : 1000 : a : S $\mapsto$ `transfer_tokens` "hello" 1000 a : S
- if the target contract is an implicit account, the parameter type is unit

Implicit Account

An implicit account is not associated with a specific contract. Rather, its contract is implicit, which mean it just updates the account balance with the tokens received.

TRANSFER_TOKEN Instruction Example

Schedule Transfer to Bob

- Suppose Bob's address is "tz1KqTpEZ7Yob7QbPE4Hy4Wo8fHG8LhKxZSx" (do not use this address)
- IMPLICIT_ACCOUNT transforms a key hash into its implicit account

```
parameter unit;
storage unit;
code {
  DROP; # ignore parameter and storage
  PUSH key_hash "tz1KqTpEZ7Yob7QbPE4Hy4Wo8fHG8LhKxZSx"; # key_hash : A
  IMPLICIT_ACCOUNT;          # contract unit : A
  AMOUNT;                    # mutez : contract unit : A
  UNIT;                      # unit : mutez : contract unit : A
  TRANSFER_TOKENS;           # transfer mutez->contract : A
  DIP{UNIT; NIL operation}   # transfer mutez->contract : NIL : unit : A
  CONS; PAIR                 # ([transfer mutez->contract], unit)
}
```

SET_DELEGATE Instruction

- construct operation to add, update or remove a delegation (i.e., baking right).
- SET_DELEGATE :: option key_hash : 'S → operation : 'S
- None withdraws delegation
- Some kh delegates to (implicit account of) kh
- Crashes if delegation is not possible.

CREATE_CONTRACT Instruction

- construct operation to create a contract
- suppose that code is a contract, i.e.
  code :: pair ty2 ty1 : [ ] → pair (list operation) ty1 : [ ]
- then CREATE_CONTRACT ty1 ty2 code ::
  option key_hash : mutez : ty1 : 'S → operation : address : 'S
  - the optional key_hash determines a delegate
  - the initial amount stored with the contract (taken from currently executing)
  - the initial value of the storage (ty1)
- later on, the contract can be invoked via the returned *address*, but first, the returned *operation* to create it must be executed!

Remark

On a stack machine, an operation can return more than one result!

CREATE_CONTRACT Instruction Example

```
1  parameter unit;
2  storage (option address);
3  code {
4         DROP;
5         UNIT;              # Initial storage for contract
6         AMOUNT;            # Push initial balance
7         NONE key_hash;     # No delegate
8         CREATE_CONTRACT    # Create the contract
9           { parameter unit ;
10            storage unit ;
11            code
12              { CDR;
13                NIL operation;
14                PAIR; }
15          };
16        DIP {SOME;NIL operation};CONS ; PAIR  # Epilogue
17 }
```

Let's figure out the final storage...

CREATE_CONTRACT Instruction Example II

```
1       CREATE_CONTRACT {...}    # Create the contract
2 # [ cc-operation , address ]
3       DIP {SOME;NIL operation};
4 # [ cc-operation , NIL , SOME address ]
5       CONS;
6 # [ [cc-operation], SOME address ]
7       PAIR  # Epilogue
8 # [ pair [cc-operation] (SOME address) ]
```

- returns a single operation cc-operation
- final storage: SOME address of the new contract to be created
- BUT: we cannot call the new contract in this list of operations
- To do so, we'd have to create a TRANSFER_TOKENS operation with this address, but it's not yet valid

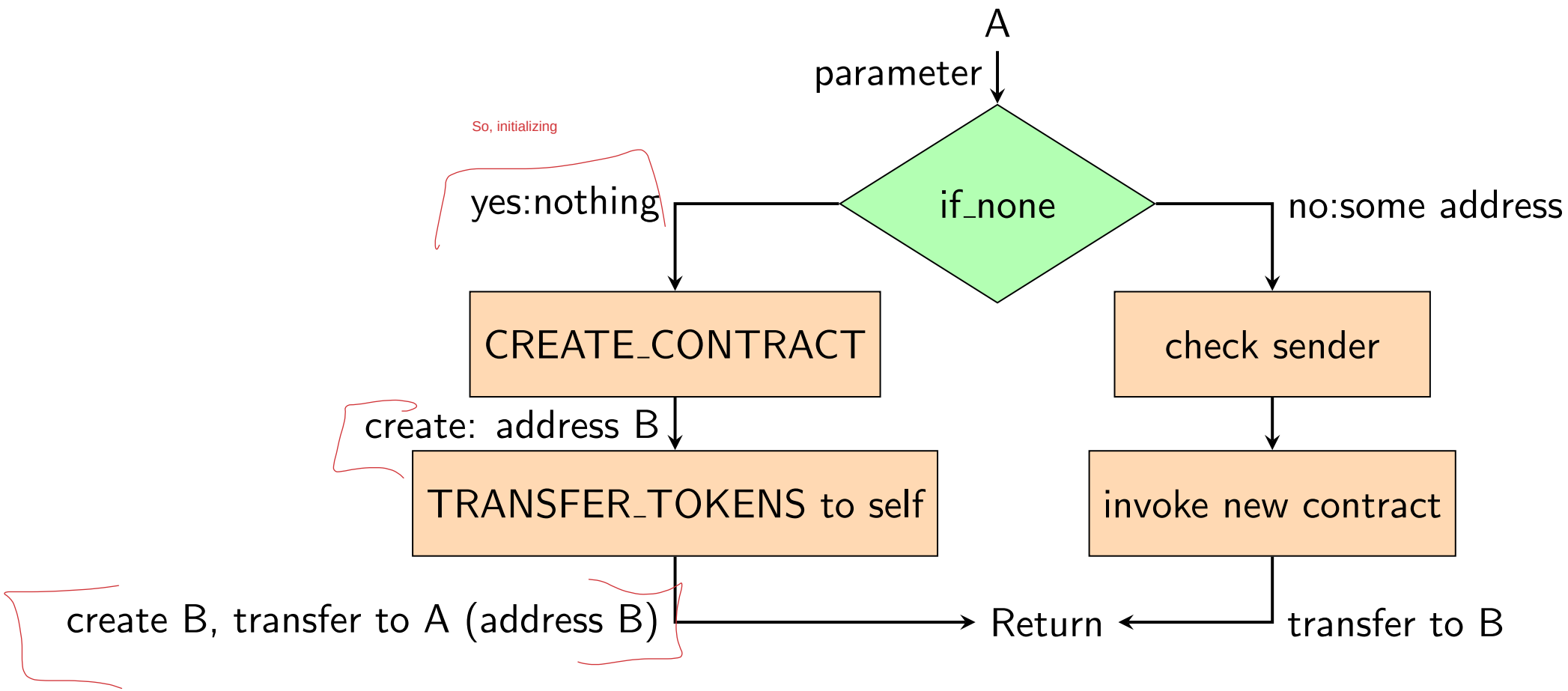
CREATE_CONTRACT Instruction Example III

Create a contract and call it

```
parameter (option address) ;
storage unit ;
code { CAR ;
       IF_NONE
         { PUSH string "dummy" ;
           PUSH mutez 100000000 ; NONE key_hash ;
           CREATE_CONTRACT
             { parameter string ;
               storage string ;
               code { CAR ; NIL operation ; PAIR } } ;
           DIP { SOME ; DIP { SELF ; PUSH mutez 0 } ; TRANSFER_TOKENS ;
                 NIL operation ; SWAP ; CONS } ;
           CONS ; UNIT ; SWAP ; PAIR }
         { SELF ; ADDRESS ; SENDER ; IFCMPNEQ { FAIL } {} ;
           CONTRACT string ; IF_NONE { FAIL } {} ;
           PUSH mutez 0 ; PUSH string "abcdefg" ; TRANSFER_TOKENS ;
           NIL operation ; SWAP ; CONS ; UNIT ; SWAP ; PAIR } } ;
```

CREATE_CONTRACT Instruction Example IV

What's going on?



CREATE_CONTRACT Instruction Example V

Parameter is None

```
1        { PUSH string "dummy" ;
2          PUSH mutez 100000000 ; NONE key_hash ;
3          CREATE_CONTRACT
4            { parameter string ;
5              storage string ;
6              code { CAR ; NIL operation ; PAIR } } ;
7  # cc-operation : address
8          DIP { SOME ; DIP { SELF ; PUSH mutez 0 } ;
9  # cc-operation : SOME address ; mutez 0; my-address
10         TRANSFER_TOKENS ;
11 # cc-operation : tt-operation
12               NIL operation ; SWAP ; CONS } ;
13 # cc-operation : [tt-operation]
14         CONS ; UNIT ; SWAP ; PAIR }
15 # pair [cc-operation, tt-operation] unit
```

CREATE_CONTRACT Instruction Example VI

Parameter is Some address

```
1  # stack: new−address
2  # check if I'm the sender
3          { SELF ; ADDRESS ; SENDER ; IFCMPNEQ { FAIL } {} ;
4  # stack: new−address
5  # check if this is a contract with string input
6          CONTRACT string ; IF_NONE { FAIL } {} ;
7  # stack: contract
8  # schedule invocation of the contract
9          PUSH mutez 0 ; PUSH string "abcdefg" ; TRANSFER_TOKENS ;
10 # stack: tt−operation
11         NIL operation ; SWAP ; CONS ; UNIT ; SWAP ; PAIR } } ;
12 # stack: pair [tt−operation] unit
```

More Information on Michelson

https://michelson.nomadic-labs.com/