kadena

scalable blockchain | smarter contracts

Pact property checking system

Pact: introduction

- Intentionally simple language
- No loops; non-Turing complete
- Data stored in tables owned by the contract (think: SQL)
- Simple, first-class authorization constructs (“keysets”)
- Transactional semantics (and enforce)

Pact: tables

```
(defschema account
  “A user account”
  balance:integer
  ks:keyset)



(deftable accounts:{account})
```

	balance	ks
alice	10	[“d37126b”]
bob	5	[“5fca08d”]

Pact: a function

```
(defun transfer (from to amount)
 "Transfer money between accounts"
 (let ((from-bal (at 'balance (read accounts from)))
    (to-bal   (at 'balance (read accounts to))))
  (update accounts from { "balance": (- from-bal amount) })
  (update accounts to   { "balance": (+ to-bal amount) })))
```

Pact: optional types

```
(defun transfer (from:string to:string amount:integer)
  "Transfer money between accounts"
  (let ((from-bal (at 'balance (read accounts from)))
        (to-bal   (at 'balance (read accounts to))))
    (update accounts from { "balance": (- from-bal amount) })
    (update accounts to   { "balance": (+ to-bal amount) })))
```

Pact: authorization

```
(defun transfer (from:string to:string amount:integer)
  "Transfer money between accounts"
  (let ((from-bal (at 'balance (read accounts from)))
        (to-bal   (at 'balance (read accounts to))))
    (update accounts from { "balance": (- from-bal amount) })
    (update accounts to   { "balance": (+ to-bal amount) })))
```

Pact: authorization

```
(defun transfer (from:string to:string amount:integer)
  "Transfer money between accounts"
  (let ((from-bal (at 'balance (read accounts from)))
        (from-ks  (at 'ks      (read accounts from)))
        (to-bal   (at 'balance (read accounts to))))
    (enforce-keyset from-ks)
    (update accounts from { "balance": (- from-bal amount) })
    (update accounts to   { "balance": (+ to-bal amount) })))
```

Pact: enforcing an invariant

```
(defun transfer (from:string to:string amount:integer)
  "Transfer money between accounts"
  (let ((from-bal (at 'balance (read accounts from)))
        (from-ks  (at 'ks      (read accounts from)))
        (to-bal   (at 'balance (read accounts to))))
    (enforce-keyset from-ks)
    (update accounts from { "balance": (- from-bal amount) })
    (update accounts to   { "balance": (+ to-bal amount) })))
```

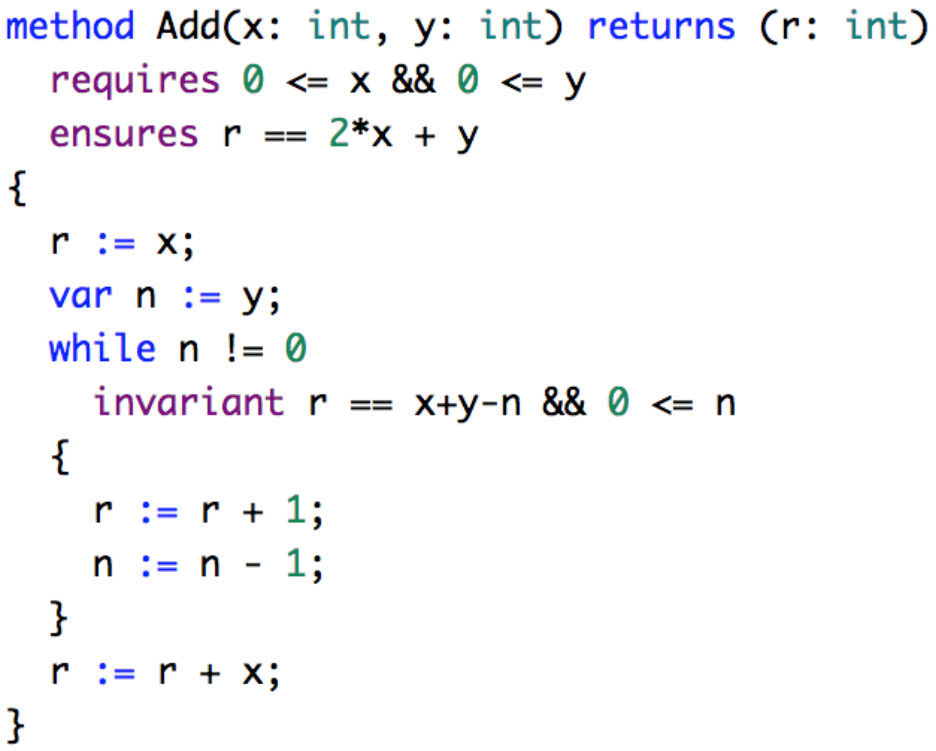
Pact: enforcing an invariant

```
(defun transfer (from:string to:string amount:integer)
  "Transfer money between accounts"
  (let ((from-bal (at 'balance (read accounts from)))
        (from-ks  (at 'ks      (read accounts from)))
        (to-bal   (at 'balance (read accounts to))))
    (enforce-keyset from-ks)
    (enforce (>= from-bal amount) "Insufficient Funds")
    (update accounts from { "balance": (- from-bal amount) })
    (update accounts to   { "balance": (+ to-bal amount) })))
```

Property checker motivation

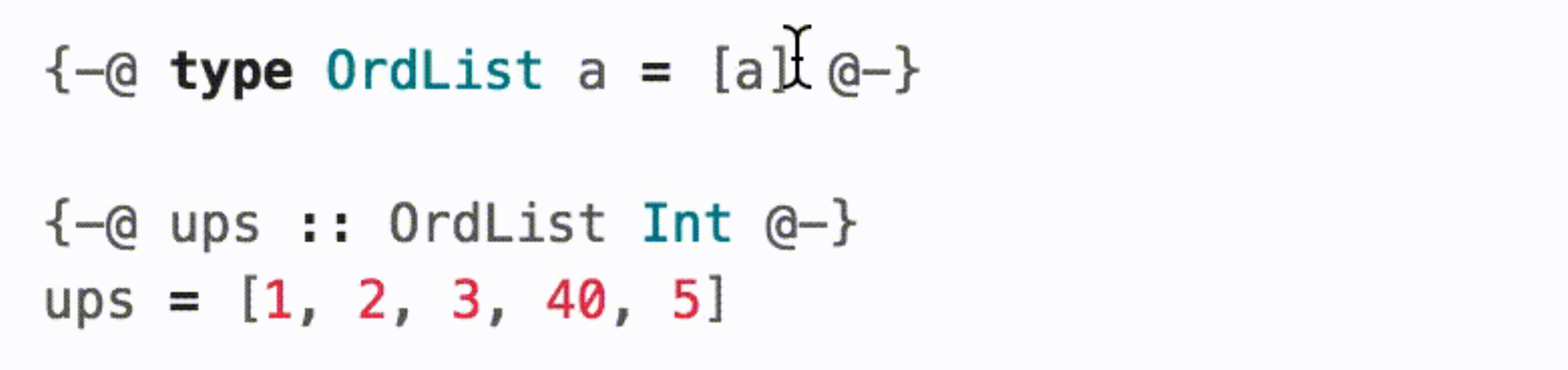
- Smart contracts have been repeatedly exploited
- On Ethereum: DAO attack, Parity multisig wallet bug, batchOverflow, etc.
- Security is holding us back from nontrivial distributed apps
- Pact is safer than Solidity, but smart contract authors still make mistakes!
- Unit tests are not sufficient vs adversaries

Dafny



```
method Add(x: int, y: int) returns (r: int)
  requires 0 <= x && 0 <= y
  ensures r == 2*x + y
{
  r := x;
  var n := y;
  while n != 0
    invariant r == x+y-n && 0 <= n
  {
    r := r + 1;
    n := n - 1;
  }
  r := r + x;
}
```

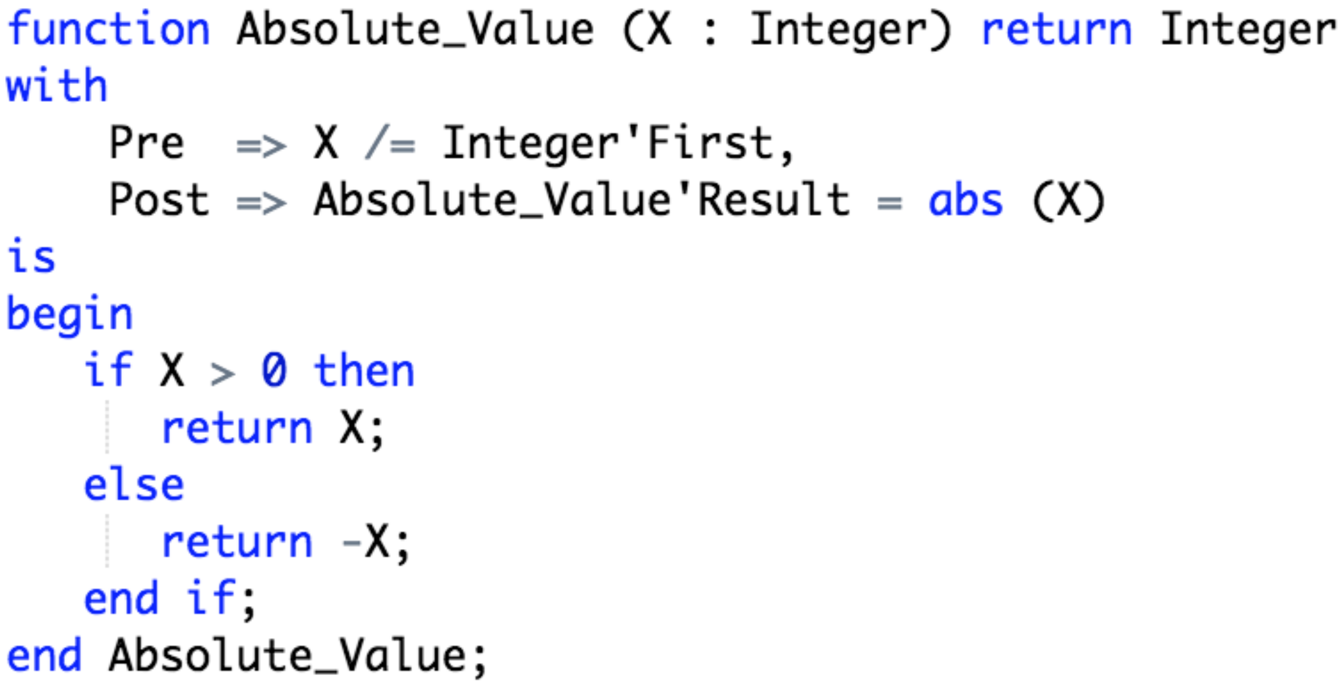
Liquid Haskell?



```
{-@ type OrdList a = [a] @-}

{-@ ups :: OrdList Int @-}
ups = [1, 2, 3, 40, 5]
```

Ada / SPARK



```
function Absolute_Value (X : Integer) return Integer
with
    Pre  => X /= Integer'First,
    Post => Absolute_Value'Result = abs (X)
is
begin
   if X > 0 then
      return X;
   else
      return -X;
   end if;
end Absolute_Value;
```

REDFIN

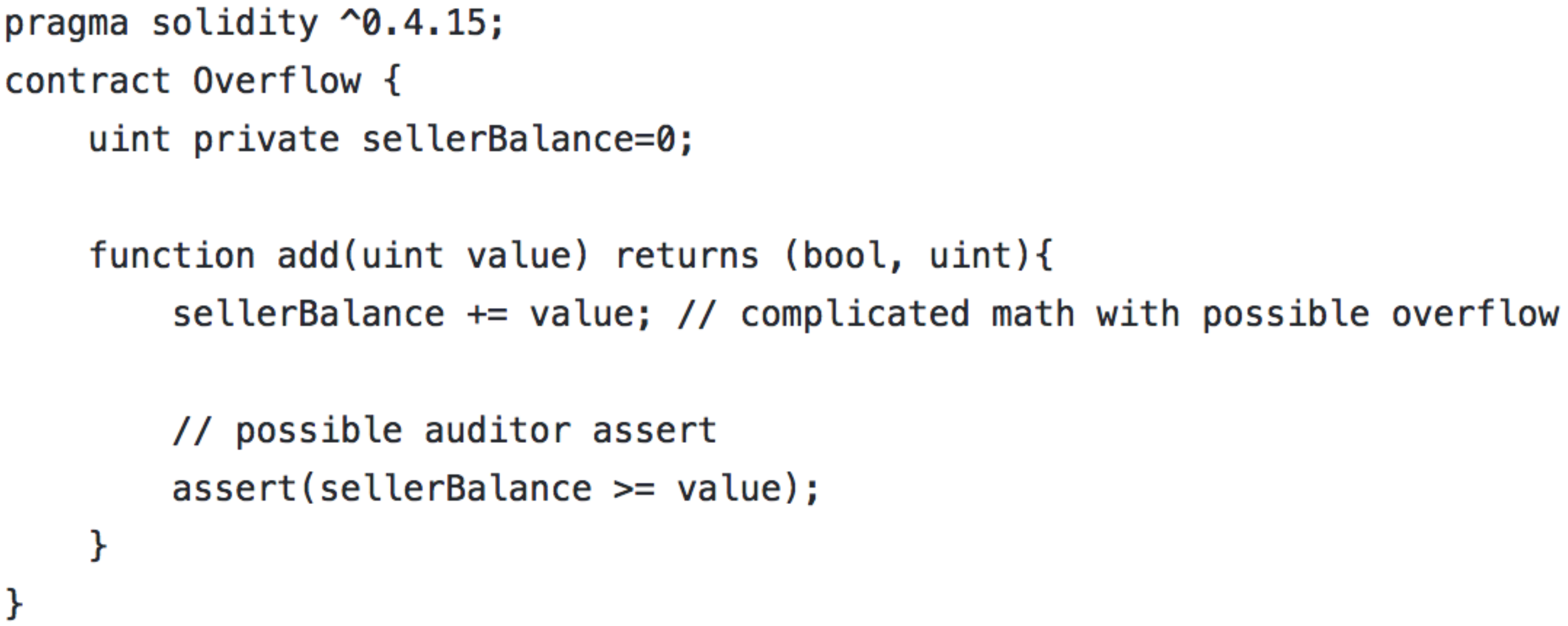
Formal Verification of Spacecraft Control Programs Using a Metalanguage for State Transformers

Andrey Mokhov[1](✉), Georgy Lukyanov[1], Jakob Lechner[2]

[1]Newcastle University, UK [2]RUAG Space Austria GmbH

✉ andrey.mokhov@ncl.ac.uk

Manticore



```
pragma solidity ^0.4.15;
contract Overflow {
    uint private sellerBalance=0;

    function add(uint value) returns (bool, uint){
        sellerBalance += value; // complicated math with possible overflow

        // possible auditor assert
        assert(sellerBalance >= value);
    }
}
```

The Pact property checker

- A static analysis tool built into Pact
- Uses Microsoft’s Z3 theorem prover
- Enforce **schema invariants** & **function properties** for *all* possible inputs and program states
- No background in formal verification required
- Not an interactive theorem prover -- enforces *contracts* on functions

Schema invariants

Declared on the fields of a table’s schema

```
(defschema account
  “A user account”

  balance:integer
  ks:keyset)
```

Schema invariants

Declared on the fields of a table’s schema

```
(defschema account
  (“A user account”
    (invariant (> balance 0)))
  balance:integer
  ks:keyset)
```

Function properties

Checker verifies that the desired property on a function always holds

```
(defun abs:integer (x:integer)
   “Returns the absolute value of an integer”



 (if (< x 0)
   (- x)
   x))
```

Function properties

Checker verifies that the desired property on a function always holds

```
(defun abs:integer (x:integer)
  ("Returns the absolute value of an integer"
    (properties
      [(>= result 0)]))
 (if (< x 0)
   (- x)
   x))
```

Function properties

Checker verifies that the desired property on a function always holds

```
(defun abs:integer (x:integer)
  (“Returns the absolute value of an integer”
    (properties
      [(>= result 0)
       (not (table-read ‘accounts))
       (not (table-write ‘accounts))]))
 (if (< x 0)
   (- x)
   x))
```

The mass conservation property

- For fungible assets
- Applied to a particular column
- Nothing “created or destroyed”

	balance	ks
alice	10	[“d37126b”]
bob	5	[“5fca08d”]
carol	85	[“b900da0”]

TOTAL	100	

The mass conservation property

- For fungible assets
- Applied to a particular column
- Nothing “created or destroyed”

	balance	ks
alice	10	[“d37126b”]
bob	5	[“5fca08d”]
carol	85	[“b900da0”]
TOTAL	100	

The mass conservation property

- For fungible assets
- Applied to a particular column
- Nothing “created or destroyed”

	balance	ks
alice	0	[“d37126b”]
bob	15	[“5fca08d”]
carol	85	[“b900da0”]
TOTAL	100	

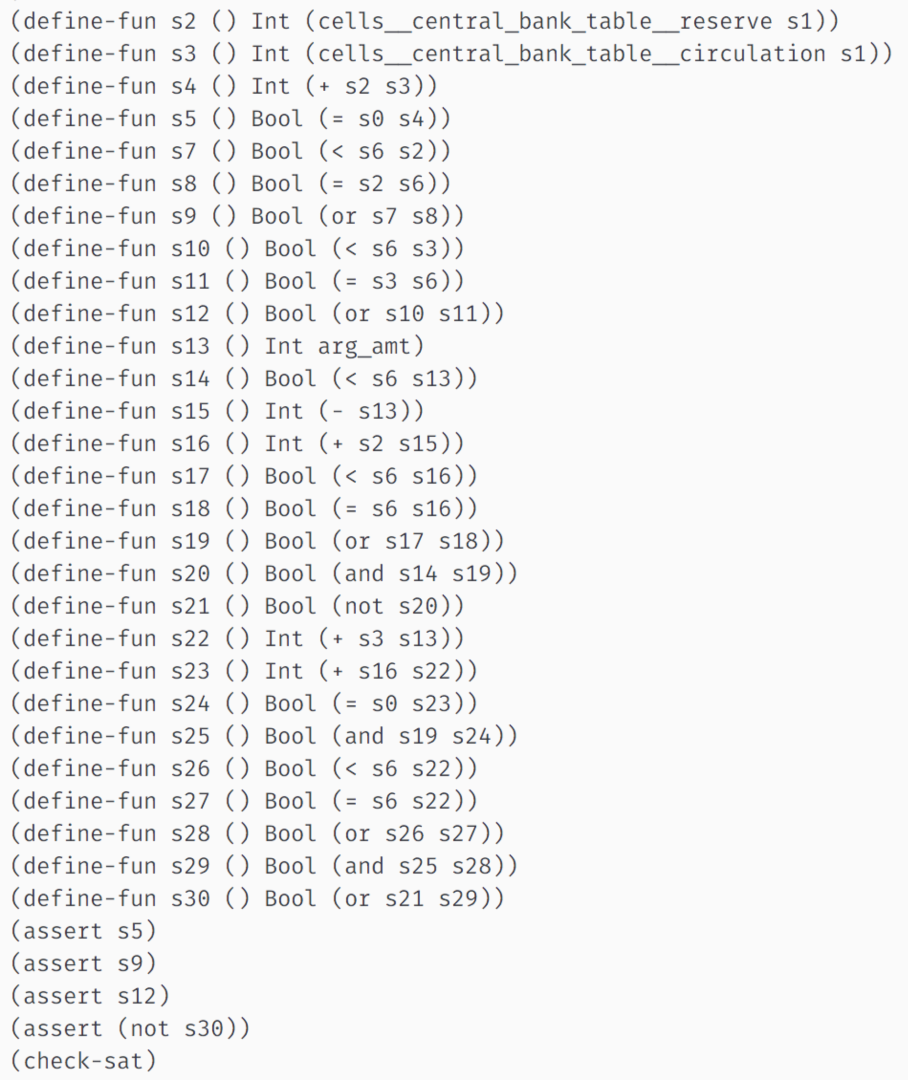
Demo

How does it work?

- Translate Pact code to SMTLib
- Translate desired property to SMTLib
- *Assume* schema invariants hold on DB reads
- *Check* invariants on DB writes
- Ask Z3 to produce an input that violates an invariant or a property
- Can’t generate an input? Invariants are maintained; Property is valid

How does it work?

Lots of lisp



```
(define-fun s2 () Int (cells__central_bank_table__reserve s1))
(define-fun s3 () Int (cells__central_bank_table__circulation s1))
(define-fun s4 () Int (+ s2 s3))
(define-fun s5 () Bool (= s0 s4))
(define-fun s7 () Bool (< s6 s2))
(define-fun s8 () Bool (= s2 s6))
(define-fun s9 () Bool (or s7 s8))
(define-fun s10 () Bool (< s6 s3))
(define-fun s11 () Bool (= s3 s6))
(define-fun s12 () Bool (or s10 s11))
(define-fun s13 () Int arg_amt)
(define-fun s14 () Bool (< s6 s13))
(define-fun s15 () Int (- s13))
(define-fun s16 () Int (+ s2 s15))
(define-fun s17 () Bool (< s6 s16))
(define-fun s18 () Bool (= s6 s16))
(define-fun s19 () Bool (or s17 s18))
(define-fun s20 () Bool (and s14 s19))
(define-fun s21 () Bool (not s20))
(define-fun s22 () Int (+ s3 s13))
(define-fun s23 () Int (+ s16 s22))
(define-fun s24 () Bool (= s0 s23))
(define-fun s25 () Bool (and s19 s24))
(define-fun s26 () Bool (< s6 s22))
(define-fun s27 () Bool (= s6 s22))
(define-fun s28 () Bool (or s26 s27))
(define-fun s29 () Bool (and s25 s28))
(define-fun s30 () Bool (or s21 s29))
(assert s5)
(assert s9)
(assert s12)
(assert (not s30))
(check-sat)
```

Limitations

- The property checker itself is not formally verified
- Unknown performance on large / difficult contracts
- The properties we can express are somewhat limited
- A contract is only as correct as its spec

Roadmap

- defproperty
- Module-level properties
- Support as much of Pact as possible
- Method visibility
- Standard library
- Improved tooling & UX
- Verified implementation of the property checker

Joel Burget - joel@monic.co

kadena

scalable blockchain | smarter contracts