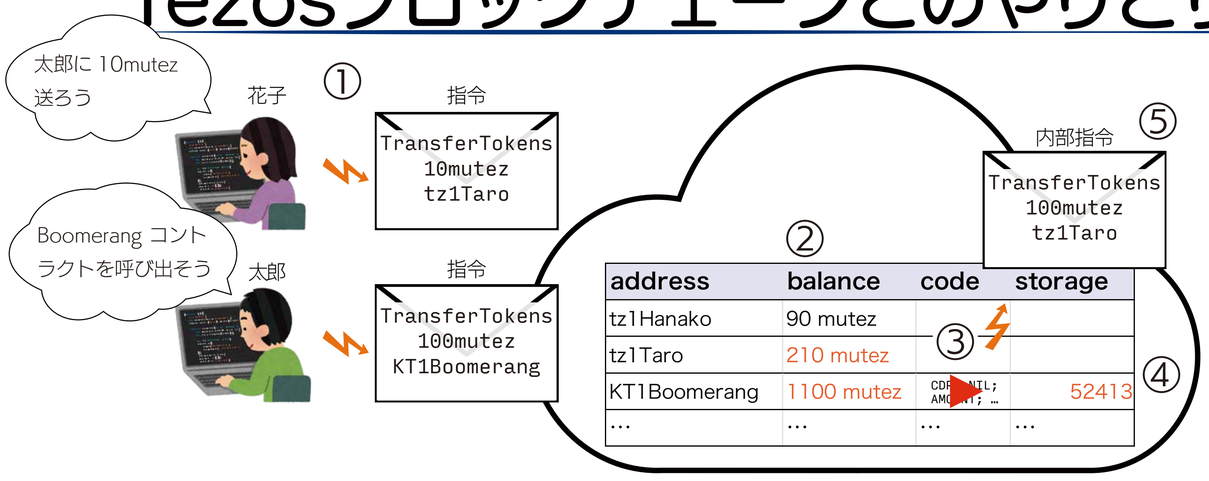
iCon: Tezos スマートコントラクト群に対する未知の存在する下での協調動作検証器

京都大学大学院情報学研究科

西田 雄気　　末永 幸平　　五十嵐 淳

Tezosブロックチェーンとのやりとり



1. ユーザーが”指令”を送る
2. 残高が書き換わる
（以下コントラクトアカウント）
3. コントラクトが実行される
新しいストレージの値と内部指令が計算される
4. ストレージが書き換わる
5. 内部指令が処理される

① 送金指令処理の様子

```
1  // self denotes the account receiving XTZ
2  self.balance ← self.balance + amount;  ②
3  // If the account is a smart contract
4  if ∃ self.code then
5      // Run michelson VM
6      let õ, s′ = mVM (self.code, p, self.storage) ;  ③
7      self.storage ← s′;  ④
8      // Process the list from head
9      foreach o in õ do  ⑤
10         switch o do
11             // Transaction operation
12             case Xfer(v, m, a) do
13                 self.balance ← self.balance - m;
14                 send (v,m,a) ;
15             end
16             // Origination operation
17             case Orig(c, v, m, a) do
18                 self.balance ← self.balance - m;
19                 create (c,v,m,a) ;
20             end
21             // Delegation operation
22             case Sdel(a) do
23                 self.delegate ← a;
24             end
25         end
26     end
27 end
```

iConで検証できること

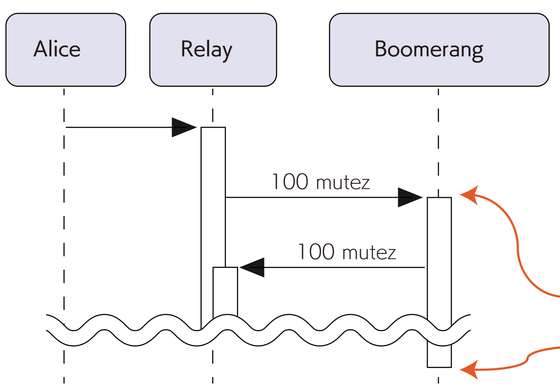
入力：事前条件・事後条件・コードの関数仕様
出力：入力３つの整合性が取れているか否か

実際のコードが関数仕様を満たしているかは別途検証する

検証作業

Why3 上で
- 自動証明
- 対話証明

検証例：Boomerang



送られてきたトークンの量をストレージに加算して
トークンは送り返す

受け取ったトークン量　送り主のアドレス

関数仕様 : s′ = s + amount
∧ õ = [(TransferTokens amount sender)]
事前条件 : True
事後条件 : Boomerang.balance = pre(Boomerang.balance)
∧ Boomerang.storage ≥ pre(Boomerang.storage)

難しいところ
- 仕様が不明かつ強制できない**未知**のコントラクトが絡んでくる
- 内部指令が動的に生成される

✕ 成り立たない事後条件 : Boomerang.storage = pre(Boomerang.storage) + amount

その他の検証例

Auction: シングルオークション

- 現在の落札予定額以上の入札金を送ると入札できる
- 落札予定者以外はいつでも入札金を引き出せる
- 検証項目：Auction の残高がいつも十分存在する

Dexter2: 分散型交換所

- 自律的に交換レートの調整されるトークン交換所
- 実際に稼働している（いた）サービス
- ver.1 は関数仕様は検証されていたが協調動作時に欠陥がみつかった
- 検証項目 : 流動性の担保が意図通り管理されている

今回の検証手法のポイント

システム特有の性質

- アカウントの状態は明示的に送金（コントラクトが実行）されることによってのみ変更される
- 一度配置されたコントラクトは変更できない

検証対象の制限

- 未知コントラクトの状態には言及しない
- 既知コントラクトに関して
  - 有限個に限る
  - 生成する内部指令の長さに上限がある
  - 検証後新たに配置されない Future work

検証アプローチ：ホーア論理からの検証条件の導出

指令列に対する推論規則

非自明なポイント

$$\frac{x_1\,x_2\,x_3\,x_4 \,\#\, FV(\mathcal{P}) \cup FV(P) \cup FV(Q) \qquad (\mathcal{C},\mathcal{P},\mathcal{Q}) = K(a_2) \qquad \models P[\texttt{ledger}[a_1 \mathrel{+}= m]/\texttt{ledger}] \to \mathcal{P}() \qquad K \vdash a_1 \Rsh \{P[x_1[a_1 \mathrel{+}= m]/\texttt{ledger}, x_2/\texttt{stores}] \wedge \mathcal{Q}(x_1,x_2,a_1,a_2,m,i) \wedge x_3 = \texttt{ledger} \wedge x_4 = \texttt{stores}\}\vec{o}\{Q\}}{K \vdash a_1 \Rsh \{P\}\texttt{Xfer}\,(i,m,a_2) :: \vec{o}\{Q\}}\ [\text{T-XFER}]$$

3つ組の整合性をチェックする推論規則

各アドレスから検証する 3 つ組への写像 $\mathcal{C}$: 関数仕様 $\mathcal{P}$: 事前条件 $\mathcal{Q}$: 事後条件

$$\frac{x_1\,x_2\,x_3\,x_4\,x_5\,x_6 \,\#\, FV(K) \qquad (\mathcal{C},\mathcal{P},\mathcal{Q}) = K(a) \qquad K \vdash a \Rsh \{\mathcal{P}()[x_1/\texttt{ledger}, x_2/\texttt{stores}] \wedge \mathcal{C}(x_3,a,x_4,x_5,x_6,\vec{o},\texttt{stores}[a]) \wedge x_1 = \texttt{ledger}[a \mathrel{-}= x_4] \wedge x_2 = \texttt{stores}[a \mapsto x_6]\}\vec{o}\{\mathcal{Q}(x_1,x_2,x_3,a,x_4,x_5)\}\ \text{for any } \vec{o}}{K \vdash a}\ [\text{CONSIST}]$$

任意の内部指令列について証明しないといけない

検証時に将来配置されるコントラクトも含めて決めないといけない

（可算）無限個のアドレスについて導出しないといけない

未知コントラクトの扱い

- 関数仕様は True に固定
- 事前事後条件は同じものを共有
  未知コントラクトは無数に存在しうるが検証すべき 3 つ組は一つに
- 指令列の任意性は列についての帰納法で証明
  制限事項から証明に必要な条件（検証条件）が簡単に求められる

既知コントラクトの扱い

- 制限により有限個の 3 つ組を検証すれば OK
- 内部指令列は上限の長さまでの全パターンを素朴に検証する