Mondex

Miroslav Kovář

December 5, 2017

Úvod do problému

Mondex karta je elektronická karta (peněženka) vyvinutá v roce 1994 sloužící pro uchování a převodu monetární hodnoty. Narozdíl od kreditní karty je veškerá hodnota uložená přímo na kartě samotné, není tedy vyžadován přístup k internetu a vzdáleným bankovním databázím.
Zřejmě je zcela kritické, aby během převodů nedocházelo ke ztrátě nebo duplikaci peněžní hodnoty. Pochopitelně tedy bylo vyvinuto úsilí ověřit pomocí formálních verifikačních metod robustnost a bezpečnost systému *Mondex*. V roce 2000 byla publikována formální verifikace modelu pomocí specifikačního jazyka Z. Tato verifikace je založena na převdu mezi dvěma modely: abstraktním a konkrétním modelem. Verikace v jazyku Alloy kopíruje tento postup.

1. V *abstraktním* modelu je převod atomický - každá ze stran odešle / přijme požadovanou hodnotu okamžitě, přenosové prostředí není modelováno. Přechody mezi abstraktními stavy jsou možné pouze pomocí dvou operací: `AbTransferOkay` (převod proběhl úspěšně) a `AbTransferLost` (odesilatel odečte částku ze svého zůstatku, ale příjemce ji nepřijme).

2. *Konkrétní* model odpovídá skutečné situaci, kde se převod realizuje na nespolehlivém kanálu, kde přeovod probíhá v pěti krocích.

Nejdříve je dokázáno, že bezpečnostní požadavky platí pro abstraktní model, a ten se převede na konkrétní model, aniž by se porušily bezpečnostní požadavky. [1]
Zmíněných pět kroků vypadá následujícím způsobem:

1. Odesilatel odešle inicializační zprávu.

2. Příjemce přijme inicializační zprávu a odešle požadavek.

3. Odesilatel přijme požadavek, sníží svůj zůstatek a odešle hodnotu.

4. Příjemce přijme hodnotu, zvýší svůj zůstatek a odešle potvrzení.

5. Odesilatel přijme potvrzení.

V případě, že je tento proces z jakéhokoli důvodu přerušen, obě peněženky musí zaznamenat ztracenou hodnotu ve svém lokálním logu, který může být v libovolném okamžiku (i v průběhu transakce) sdílen s globálním logem nevydařených transakcí, a následně vyprázdněn následováním tříkrokového protokolu. Tento proces je ale mimo rozsah tohoto reportu.
Nespolehlivostí kanálu se myslí, že:

- Zpráva může v prostředí zůstat a být tedy přijmuta více než jednou.
- Zpráva může z prostředí zmizet dříve, než je přečtena.
- Zpráva může být přečtena více než jednou peněženkou. Je třeba zajistit, aby byla zpracována pouze příslušnou peněženkou.
- Může se objevit falešná zpráva nesouvisející s protokolem.

[1] Tohoto převodu je docíleno pomocí mezipřevodu využívající mezimodel (between world), který se od konkrétního modelu liší tím, že komunikační kanál mezi peněženkami není ztrátový. Jinými slovy, mezimodel sice také modeluje přenosové prostředí, ale aplikuje na ně volnější omezení, jako např. v prostředí se objeví pouze zprávy vyslané peněženkami a odpovídající jejich stavům, zalogované zprávy odpovídají zprávám z prostředí, atd., kdežto konkrétní model povoluje vznik nespolehlivostí kanálu zmíněných dále. Detailnější diskuse odpovídajících převodů je ovšem mimo rozsah tohoto reportu.

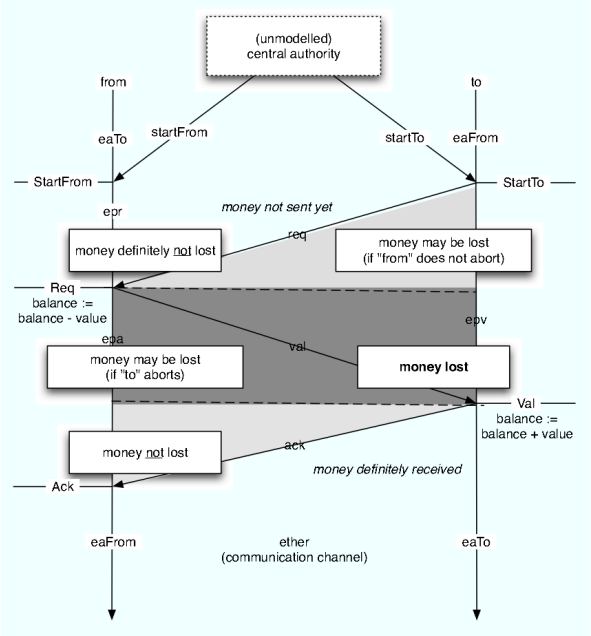

Figure 1: Ilustrace pětikrokového transakčního protokolu, společně s ilustrací `maybeLost()` a `definitelyLost()`. Čtverce reprezentují stav mincí příslušných dané transakci v případě, že daná strana v daném intervalu přeruší transakci.

Reprezentace mincí v Alloy

V době, kdy byla specifikace z článku modelována byla implementace celých čísel v Alloy pomocí převodů do SAT formulí velmi pomalá. Proto byly navrženy alternativní způsoby reprezentace transakcí a mincí.
Uspořádání transakcí je v Alloy realizováno pomocí modulu `ordering`.

```
sig SEQNO {}
open util/ordering [SEQNO]
```

Peněžní hodnoty jsou reprezentovány pomocí množin mincí, kde hodnota není kardinalita této množiny, ale součet hodnot jednotlivých mincí, kde každá mince je individuálním objektem v rámci každ peněženky. Operace s mincemi jsou tedy realizovány následujícím způsobem:

Součet je disjunktní sjednocení dvou množin.

Rozdíl je množinový rozdíl s přidanou podmínkou, že odečítaná množina musí být podmnožinou množiny, od které se odečítá.

Porovnání je inkluze množin.

Pro reprezentaci mincí se využívá faktu, že se nikdy neporovnávají zůstatky *různých* peněženek.

```
sig NAME {} -- Jmeno uzivatele
sig Coin {}
sig PayDetails {
  from, to: NAME,
  fromSeqNo, toSeqNo: SEQNO,
  value: set Coin
}
```

Omezení modelu `abstract`

Z výše uvedéno vyplývají následující jednoduché podmínky na abstraktní model.

```
sig AbPurse {
  balance: set Coin,
  lost: set Coin -- Mince zalogovane jako ztracene
}
sig AbWorld { abAuthPurse: NAME -> AbPurse }
fact noCoinSharing {
  all w: AbWorld {
    no disj n1, n2: NAME {
      some n1.(w.abAuthPurse).(balance + lost) & n2.(w.abAuthPurse).(balance + lost)
        -- 1
    }
    no p: AbPurse {
      p in NAME.(w.abAuthPurse)
      some p.balance & p.lost -- 2
    }
  }
}
```

1. Pro žádné dva uživatele neplatí, že by jejich peněženky z abstraktního modelu sdílely mince, a to ať už v zůstatku nebo mezi mincemi zalogovanými jako ztracenými.
2. Pro žádnou peněženku z abstraktního modelu neplatí, že by obsahovala nějakou minci zároveň ve svém zůstatku a v logu ztracených mincí.

Omezení modelu `concrete`

Konkrétní model je nutné zatížit ekvivalentními omezeními. V první iteraci konkrétního modelu byly vytvořeny následující podmínky.

```
sig ConPurse {
  name: NAME,
  balance: set Coin,
  pdAuth: PayDetails, -- Transakce cekajici na zpracovani
  exLog: set PayDetails, -- Lokalni log nevydarenych transakci
  nextSeqNo: SEQNO, -- Cislo dalsi transakce
  status: STATUS -- Status penezenky, viz Fig. 1
}
sig ConWorld {
  conAuthPurse: NAME -> lone ConPurse,
  ether: set MESSAGE, -- Komunikacni kanal
  archive: NAME -> PayDetails -- Globalni archiv nevydarenych transakci
}

fact noCoinSharingConcrete {
  all p: ConPurse {
    no p.exLog.value & p.balance -- 1
  }
  all w: ConWorld {
    no disj n1, n2: NAME {
       some n1.(w.conAuthPurse).balance & n2.(w.conAuthPurse).balance -- 2
    }
    no p: ConPurse, pd: PayDetails {
      p in NAME.(w.conAuthPurse)
      pd in NAME.archive
      some p.balance & pd.value -- 3
    }
  }
```

}

1. Žádná peněženka z konkrétního modelu neobsahuje minci zároveň ve svém zůstatku a v lokálním logu nevydařených transakcí.
2. Žádné dvě peněženky z konkrétního modelu neobsahují dvě stejné mince ve svém zůstatku.
3. Žádná peněženka z konkrétního modelu nesdílí minci ze svého zůstatku s globálním archivem nevydařených transakcí.

Ukazuje se ale, že pouze požadavek 2 je správný, a požadavky 1 a 3 jsou příliš silné. Například:

- Předpokládejme, že příjemce přijal částku, odeslal potvrzení, ale odesilatel ukončí transakci před přijetím tohoto potvrzení, a zaloguje transakci jako nevydařenou.Pak požadavek 3 zamezí odesilateli tuto transakci zapsat jako nevydařenou.
- Předpokládejme, že příjemce odešle požadavek, ukončí transakci, a zaznamená ji do svého logu nevydařench transakcí. Pokud odesilatel ukončí transakci před přijetím požadavku, pak zanechá příslušné mince ve svém zůstatku. Pak požadavek 3 zamezí příjemci zapsat tuto transakci do globálního logu nevydařených transakcí.
- Předpokládejme, situaci z předchozího bodu. Pak požadavek 1 zamezí příjemci přijmout příslušné mince, neboť mince, které by měl přijmout, už má ve svém logu nevydařených transakcí.

Řešení spočívá v zavední dvou funkcí - `definitelyLost()` a `maybeLost()`. První odpovídá transakcím zalogovaným oběma peněženkami a transakcím, kde odesilatel odeslal částku, nedostal potvrzení, a příjemce ukončil transakci. Druhá odpovídá transakcím, kde příjemce očekává částku a odesilatel ji už odeslal a buď čeká na potvrzení, nebo ukončil transakci před tím, než příjemce částku přijal.
V obou případech byla částka (úspěšně nebo neúspěšně) odeslána odesilatelem. Podmínka 3 je tedy nahrazena následující podmínkou, která vynucuje, aby žádná peněženka z konkrétního modelu ve svém zůstatku neobsahovala mince, které jsou mezi `definitelyLost()` nebo `maybeLost()` příslušného modelu.

```
all w: ConWorld {
  no p: ConPurse {
    p in NAME.(w.conAuthPurse)
    some p.balance & (definitelyLost (w) + maybeLost (w)) -- new 3
  }
}
```

Podrobnosti ohledně definice funkcí `definitelyLost()` a `maybeLost()` a řešení dalších problémů je bohužel mimo rozsah tohoto krátkého reportu.