

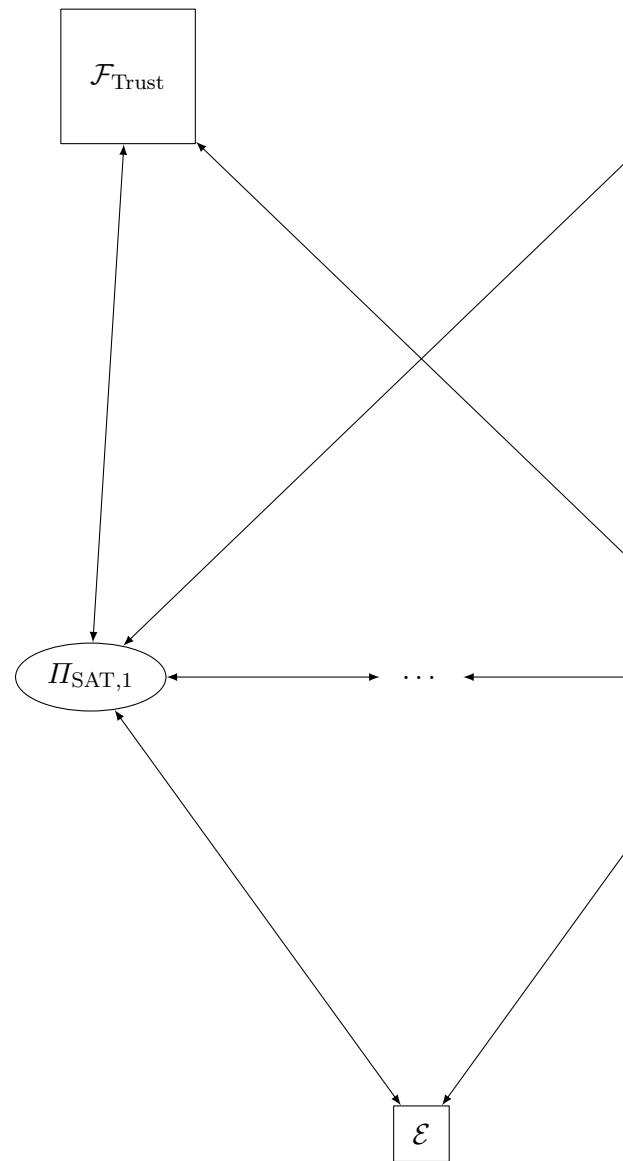
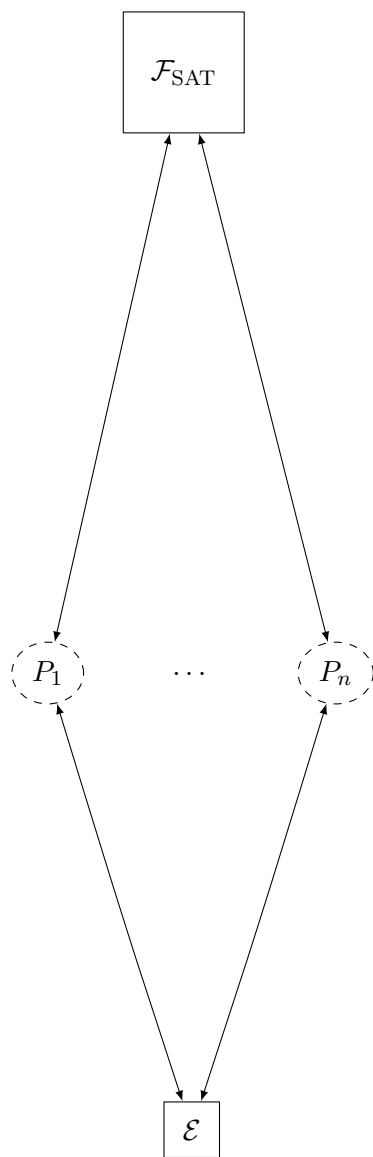
# What is Trust

Orfeas Stefanos Thyfronitis Litos

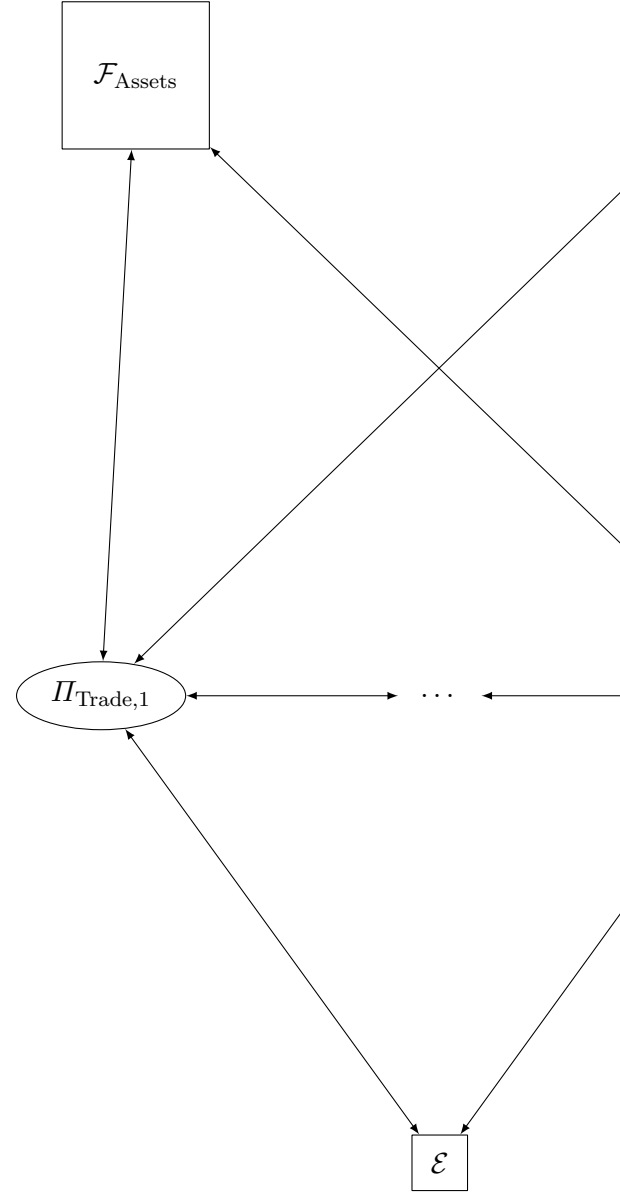
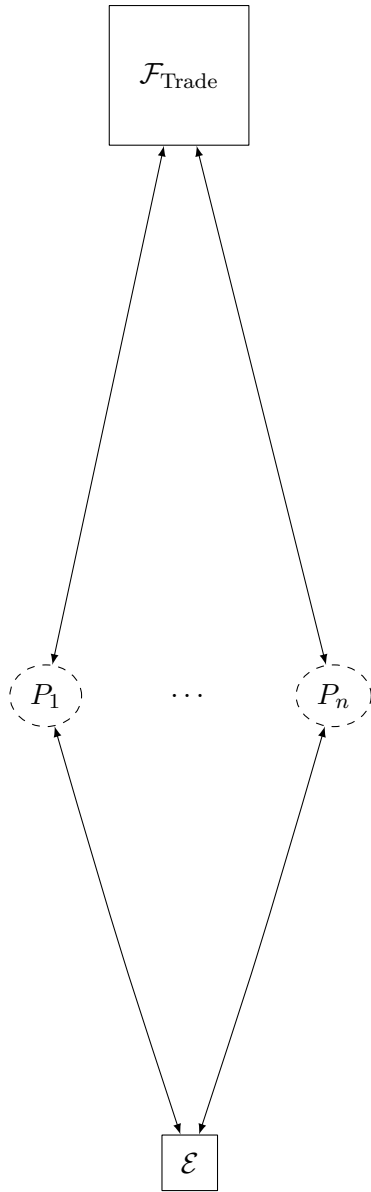
University of Edinburgh  
o.thyfronitis@ed.ac.uk

**Abstract.** We will try to define all the abstract properties that we would like "Trust" to have.

# 1 High-level idea



**Fig. 1:** (Almost) all functionalities



**Fig. 2:** Trade functionality and protocol

## 2 Utility Function Properties

Before *Alice* (an ITM that executes  $\Pi_{\text{SAT}}$ ) can take any action, she must be assigned a utility function  $U_{\text{Alice}} : \text{Time} \times \text{Money} \times \text{multiset}(\text{Asset}) \rightarrow \mathbb{R}$ , where  $\text{Money} = \text{Time} = \mathbb{N}$ . The utility function is strictly increasing with respect to *Money* and to the quantity of any single *Asset* or combination of *Assets*.

$\mathcal{E}$  can decide to conclude the game at any time  $t \in \text{Time}$ ; *Alice*'s utility will be then determined by

$$U_{\text{Alice}}(t, \mathcal{G}_{\text{Ledger}}.\text{money}(\text{Alice}), \mathcal{G}_{\text{Assets}}.\text{assets}(\text{Alice})) .$$

## 3 Protocol

In this section we will describe the real protocol executed by the players in the absence of the  $\mathcal{F}_{\text{SAT}}$  ideal functionality. The description of this protocol does not need any utility function; all the "important" decisions are taken by the environment.

Consider an environment  $\mathcal{E}$ , and adversary  $\mathcal{A}$  and  $n$  players executing copies of the same protocol  $\Pi_1, \dots, \Pi_n$ .  $\mathcal{E}$  can send the following messages to  $\Pi_i$ :

1. Manage player desires
  - (a) Satisfy  $d \in \mathcal{D}$  through a player in  $L \subseteq [n]$
  - (b) Abort attempt to satisfy  $d \in \mathcal{D}$
2. Manage offered desires satisfaction
  - (a) Gain the ability to satisfy  $d \in \mathcal{D}$  for players in  $L \subseteq [n]$  for a price  $x \in \mathbb{N}$
  - (b) Lose the ability to satisfy  $d \in \mathcal{D}$  for players in  $L \subseteq [n]$  for a price  $x \in \mathbb{N}$
3. Satisfy another player's desire
  - (a) Satisfy player's  $i \in [n]$  desire  $d \in \mathcal{D}$  with the corresponding satisfaction string  $s$
  - (b) Satisfy player's  $i \in [n]$  desire  $d \in \mathcal{D}$  with the satisfaction string  $s'$  (normally suitable for satisfying  $d' \neq d, d' \in \mathcal{D}$ )
  - (c) Ignore player's  $i \in [n]$  desire  $d \in \mathcal{D}$
4. Manage direct trusts
  - (a) Increase direct trust to player  $i \in [n]$  by  $x \in \mathbb{N}$
  - (b) Decrease direct trust to player  $i \in [n]$  by  $x \in \mathbb{N}$

(c) Steal direct trust  $x \in \mathbb{N}$  from player  $i \in [n]$

Some of these messages (e.g. 1b) are meaningful only when some other messages have been delivered previously (e.g. 1a).  $\mathcal{E}$  may send such messages even when they are not meaningful; the protocol should take care to reject/ignore such messages.

Let  $i \in [n]$ .  $\Pi_i$  can send the following messages to  $\mathcal{E}$ :

1. No player in  $L$  can satisfy my desire  $d \in \mathcal{D}$
2. Desire  $d \in \mathcal{D}$  made available for satisfaction amongst  $L \subseteq [n]$  for price  $x \in \mathbb{N}$
3. Desire  $d \in \mathcal{D}$  made unavailable for satisfaction amongst  $L \subseteq [n]$  for price  $x \in \mathbb{N}$
4. Payment  $x \in \mathbb{N}$  has been sent to player  $j \in [n]$  for the satisfaction of desire  $d \in \mathcal{D}$
5. Correct Payment  $x \in \mathbb{N}$  from player  $j \in [n]$  for the satisfaction of desire  $d \in \mathcal{D}$  has been received
6. Wrong Payment  $x \in \mathbb{N}$  from player  $j \in [n]$  for the satisfaction of desire  $d \in \mathcal{D}$  has been received
7. Player  $j \in [n]$  has satisfied my desire  $d \in \mathcal{D}$  with satisfaction string  $s$
8. Player  $j \in [n]$  has partially satisfied my desire  $d \in \mathcal{D}$  with satisfaction string  $s'$  (normally suitable for satisfying  $d' \neq d, d' \in \mathcal{D}$ )
9. Player  $j \in [n]$  has ignored my desire  $d \in \mathcal{D}$
10. Direct trust to player  $i \in [n]$  increased by  $x \in \mathbb{N}$
11. Direct trust to player  $i \in [n]$  decreased by  $x \in \mathbb{N}$
12. Stole  $x \in \mathbb{N}$  from player's  $i \in [n]$  direct trust

$\Pi_i$  should only send these messages when  $\mathcal{E}$  is expecting them.

Let  $i, j \in [n]$  The messages that can be sent between  $\Pi_i$  and  $\Pi_j$  are the following:

1. Can you satisfy  $d \in \mathcal{D}$ ?
2. I can satisfy  $d \in \mathcal{D}$  for a price  $x \in \mathbb{N}$
3. I cannot satisfy  $d \in \mathcal{D}$
4. Payment of  $x \in \mathbb{N}$  for satisfaction of  $d \in \mathcal{D}$  sent
5. Satisfaction string  $s'$ , response to payment of  $x \in \mathbb{N}$  for  $d \in \mathcal{D}$

$\Pi_i$  is supposed to send 4 when it has already paid through  $\mathcal{F}_{\text{Ledger}}$ . Similarly, it is supposed to send 5 when it has verified that the other party has sent the corresponding payment on  $\mathcal{F}_{\text{Ledger}}$ .

Going in more detail, the actual protocol is as follows:

```

 $\Pi_{\text{SAT}}$ 
1 Initialization:
2   util =  $\perp$ 
3
4 Upon receiving (utility,  $U$ ) from  $\mathcal{E}$ :
5   util =  $U$ 
6
7 Upon receiving message (satisfy,  $d$ ,  $L$ ) from  $\mathcal{E}$  at time  $t$ :
8   If util ==  $\perp$ :
9     send message (utilityNotSet) to  $\mathcal{E}$ 
10    go to Idle state
11    aux =  $L$ 
12     $m = \mathcal{G}_{\text{Ledger}}.\text{money}(\text{Alice})$ 
13     $a = \mathcal{G}_{\text{Assets}}.\text{assets}(\text{Alice})$ 
14    While (aux  $\neq \emptyset$ ):
15      send message (chooseBestSeller,  $d$ , aux) to  $\mathcal{F}_{\text{Trust}}$ 
16      wait for response1 from  $\mathcal{F}_{\text{Trust}}$ 
17      If response1 == (bestSeller,  $d$ ,  $L$ ,  $Bob$ ):
18        send message (canYouSatisfy,  $d$ ) to  $Bob$ 
19        wait for response2 from  $Bob$ 
20        If response2 == (IcanSatisfy,  $d$ ,  $x$ ,  $s$ ):
21          If util( $t+1$ ,  $m - x$ ,  $a \cup \{s\}$ ) > util( $t$ ,  $m$ ,  $a$ ) and  $s \in$ 
22             $d$ 
23            send message (trade,  $x$ ,  $s$ ,  $Bob$ ) to  $\mathcal{F}_{\text{Trade}}$ 
24            wait for response3 from  $\mathcal{F}_{\text{Trade}}$ 
25            If response3 == (traded,  $x$ ,  $s$ ,  $Bob$ ):
26              send message (satisfied,  $d$ ,  $x$ ,  $s$ ,  $Bob$ ) to  $\mathcal{F}_{\text{Trust}}$ 
27              # maybe send only utility difference to  $\mathcal{F}_{\text{Trust}}$ 
28              send message (satisfied,  $d$ ,  $L$ ) to  $\mathcal{E}$ 
29              go to Idle state
30            Else If response3 == (cheated,  $x$ ,  $s$ ,  $Bob$ ):
31              send message (cheated,  $d$ ,  $x$ ,  $s$ ,  $Bob$ ) to  $\mathcal{F}_{\text{Trust}}$ 
32              send message (cheated,  $d$ ,  $L$ ) to  $\mathcal{E}$ 
33              go to Idle state
34            Else: # if response1 ==  $\perp$ 
35              send message (unsatisfied,  $d$ ,  $L$ ) to  $\mathcal{E}$ 
36              go to Idle state
37            aux = aux  $\setminus \{Bob\}$  # only when response2 is not good
38            send message (unsatisfied,  $d$ ,  $L$ ) to  $\mathcal{E}$ 

```

```

39 Upon receiving message (obtain, s) from  $\mathcal{E}$ :
40   send message (obtain, s) to  $\mathcal{F}_{\text{Trade}}$ 
41   wait for response from  $\mathcal{F}_{\text{Trade}}$ 
42   If response == (obtained, s):
43     send message (obtained, s) to  $\mathcal{E}$ 
44   Else:
45     send message (notObtained, s) to  $\mathcal{E}$ 
46
47 Upon receiving message (lose, s) from  $\mathcal{E}$ :
48   send message (lose, s) to  $\mathcal{F}_{\text{Trade}}$ 
49   wait for response from  $\mathcal{F}_{\text{Trade}}$ 
50   If response == (lost, s):
51     send message (lost, s) to  $\mathcal{E}$ 
52   Else:
53     send message (notLost, s) to  $\mathcal{E}$ 
54
55 Upon receiving message (canYouSatisfy, d) from Alice at
    time t:
56   If util ==  $\perp$ :
57     ignore request, go to Idle state
58    $m = \mathcal{G}_{\text{Ledger}}.\text{money}(\textit{Alice})$ 
59    $a = \mathcal{G}_{\text{Assets}}.\text{assets}(\textit{Alice})$ 
60   If  $\exists s \in a \cap d, x \in \textit{Money}$ :
61     (util(t + 1, m + x, a \setminus \{s\}) > util(t, m, a) or
62     (util(t + 1, m + x, a) > util(t, m, a) and I do not
        care for the bad rep))
63     send message (IcanSatisfy, d, x, s) to Alice
64   Else:
65     send message (IcannotSatisfy, d) to Alice
66
67 Upon receiving message (willWeCheat, x, s, Alice) from
     $\mathcal{F}_{\text{Trade}}$ :
68   If util ==  $\perp$ : # (Unreachable since we've already engaged)
69     ignore request, go to Idle state
70   If cheating is a bad idea:
71     send message (doNotCheat, x, s, Alice) to  $\mathcal{F}_{\text{Trade}}$ 
72   Else:
73     send message (cheat, x, s, Alice) to  $\mathcal{F}_{\text{Trade}}$ 

```

## 4 Desire Satisfaction Ideal Functionality

Following the UC paradigm, in this section we define the ideal functionality for desire satisfaction,  $\mathcal{F}_{\text{SAT}}$ . In this setting, all the desires that are generated by the environment and are input to the players are immediately forwarded to  $\mathcal{F}_{\text{SAT}}$ ; the functionality decides which desires to satisfy. Since the players are dummy and all desires are satisfied by the functionality, no trust semantics amongst the players are necessary.

Nevertheless, given that all desires have a minimum cost, the cost semantics are still necessary, as we show with the following example: Consider a set of desires  $D$  with more elements than the total number of tokens all players have.  $D$  could never be satisfied by the players because of the high total cost, but a  $\mathcal{F}_{\text{SAT}}$  with no consideration for cost could in principle satisfy all desires in  $D$ .

The functionality can calculate the properties and functions defined in ??, ?? and ?? for all inputs at any moment in time.

Without knowledge of the utilities the environment is going to give to each satisfied desire, the functionality may fail spectacularly. So knowledge of the utility of each desire, or at least some function of the utility given the desires is needed. We can assume that  $\mathcal{F}_{\text{SAT}}$  knows  $U$  or an approximation of it.

Going into more detail,  $\mathcal{F}_{\text{SAT}}$  is a stateful process that acts as a market and as a bank for the players. The market does not offer a particular product for the same price to all users; For some users it may be cheaper than for others, reflecting the fact that some players can realize some desires more efficiently than others.

$\mathcal{F}_{\text{SAT}}$  stores a number for each player that represents the amount of tokens this player has and a table with the price of each desire for each player. It provides the functions  $\text{cost}(u, d)$  which returns the cost of the desire  $d$  for player  $u$  with no side effects,  $\text{sat}(u, d)$  that returns the string that satisfies the desire  $d$  to  $u$  and reduces the amount of the tokens belonging to  $u$  by  $\text{cost}(u, d)$ . There exists also the function  $\text{transfer}(u_1, u_2, t)$  which reduces the amount of tokens  $u_1$  has by  $t$  and increases the tokens of  $u_2$  by  $t$ , given that initially the tokens belonging to  $u_1$  were equal or more than  $t$ . This function is private to the functionality, thus can be used only internally.

$\mathcal{F}_{\text{SAT}}$

```

1 Initialisation:
2    $\forall \text{Alice} \in \mathcal{P}$ ,
3    $\text{util}(\text{Alice}) = \perp$ 
```



```

4     assets(Alice) =  $\perp$ 
5
6 Upon receiving (type, t) from Alice:
7     util(Alice) = t
8
9 Upon receiving (satisfy, d, L) from Alice:
10    If util(Alice) ==  $\perp$ :
11        send message (utilityNotSet) to Alice
12        go to Idle state
13    find list =  $\{(Bob, x, s) \in L \times \mathbb{R} \times Assets :$ 
14        s  $\in$  assets(Bob) and s  $\in$  d and x  $\geq$  0 and
15        Alice has at least x coins available and
16        util(Alice)(stateAlice  $\cup$  s - x) > util(Alice)(stateAlice)
17        and
18        util(Bob)(stateBob  $\setminus$  s + x)  $\geq$  util(Bob)(stateBob)  $\}$ 
19    send (chooseBestSeller, d, list, Alice) to  $\mathcal{A}$ 
20    wait for response from  $\mathcal{A}$ 
21    If response  $\neq$  (bestSeller, list, Bob, x, s),
22        Bob  $\in \mathcal{P}$ , x  $\in$  Money, s  $\in$  Asset:
23        # e.g. Bob =  $\perp$ 
24        (Bob, x, s) =  $\underset{(Bob, x, s) \in \text{list}}{\operatorname{argmax}} \{ \text{util}(\text{Alice})(\text{state}_{\text{Alice}} \cup s - x) \}$ 
25    Else:
26        parse response as (bestSeller, list, Bob, x, s)
27        pay x from Alice to Bob
28        assets(Bob) = assets(Bob)  $\setminus$  {s}
29        assets(Alice) = assets(Alice)  $\cup$  {s}
30        send message (satisfied, d, L) to Alice
31
32 Upon receiving (obtain, s) from Alice:
33     assets(Alice) = assets(Alice)  $\cup$  {s}
34     send message (obtained, s) to Alice
35
36 Upon receiving (lose, s) from Alice:
37     assets(Alice) = assets(Alice)  $\setminus$  {s}
38     send message (lost, s) to Alice

```

$\mathcal{F}_{\text{Trade}}$

```

1 Initialisation:
2      $\forall Alice \in \mathcal{P}$ ,
3     assets(Alice) =  $\perp$ 

```

```

4
5 Upon receiving (trade, ours, theirs, Bob) from Alice:
6   If not isAvailable(ours, Alice):
7     send message (youDontHave, ours) to Alice
8     go to Idle state
9   If transfer(ours, Alice, Bob) == True:
10    send message (willWeCheat, ours, theirs, Alice) to Bob
11    wait for response from Bob
12    If (response == (complete,  $x, s, Alice$ ) and
13       not isAvailable( $s, Bob$ )) or
14       response  $\neq$  (complete,  $x, s, Alice$ ):
15      send message (youDontHave,  $s$ ) to Bob
16      send message (cheated, ours, theirs, Bob) to Alice
17      go to Idle state
18    Else If (transfer, theirs, Bob, Alice) == True:
19      send message (traded, ours, theirs, Bob) to Alice
20  Else # failed to give (Unreachable for a good  $\mathcal{F}_{\text{Ledger}}$ )
21    send message (failed, ours, theirs, Bob) to Alice
22
23 isAvailable(object, player):
24   If object is money:
25     send (doIhaveBalance, object) to  $\mathcal{F}_{\text{Ledger}}$  as player
26     wait for response from  $\mathcal{F}_{\text{Ledger}}$ 
27     return response
28   Else: # object is asset
29     If object  $\in$  assets(player):
30       return True
31     Else:
32       return False
33
34 transfer(object, sender, receiver):
35   If isAvailable(object):
36     If object is money:
37       send (pay, object, receiver) to  $\mathcal{F}_{\text{Ledger}}$  as sender
38       wait for response from  $\mathcal{F}_{\text{Ledger}}$ 
39       Upon receiving (paymentDone, object, receiver):
40         return True
41     Else: # object is asset
42       assets(sender) = assets(sender)  $\setminus$  {object}
43       assets(receiver) = assets(receiver)  $\cup$  {object}

```

```

44         return True
45     return False
46
47 Upon receiving (obtain,  $s$ ) from Alice:
48     assets(Alice) = assets(Alice)  $\cup$   $\{s\}$ 
49     send message (obtained,  $s$ ) to Alice
50
51 Upon receiving (lose,  $s$ ) from Alice:
52     assets(Alice) = assets(Alice)  $\setminus$   $\{s\}$ 
53     send message (lost,  $s$ ) to Alice

```

$\mathcal{F}_{\text{Assets}}$

```

1 Initialisation:
2      $\forall \text{Alice} \in \mathcal{P}$ ,
3         assets(Alice) =  $\perp$ 
4
5 Upon receiving (add, asset) from Alice:
6     assets(Alice) = assets(Alice)  $\cup$   $\{\text{asset}\}$ 
7     send (added, asset) to Alice
8
9 Upon receiving (remove, asset) from Alice:
10    If  $\text{asset} \in \text{assets}(\text{Alice})$ :
11        assets(Alice) = assets(Alice)  $\setminus$   $\{\text{asset}\}$ 
12        send (removed, asset) to Alice
13    Else:
14        send (unableToRemove, asset) to Alice
15
16 Upon receiving (howManyDoIhave, asset) from Alice:
17     send (youHave, assets(Alice).count(asset)) to Alice
18
19 Upon receiving (transfer, asset, Bob) from Alice:
20    If  $\text{asset} \in \text{assets}(\text{Alice})$ :
21        assets(Alice) = assets(Alice)  $\setminus$   $\{\text{asset}\}$ 
22        assets(Bob) = assets(Bob)  $\cup$   $\{\text{asset}\}$ 
23        send (transferred, asset, Bob) to Alice
24    Else:
25        send (unableToTransfer, asset, Bob) to Alice

```

$\Pi_{\text{Trade}}$

```

1 Upon receiving (trade, ours, theirs, Bob) from  $\mathcal{E}$ :
2     Send (letsTrade, ours, theirs) to Bob

```

```

3   If transfer(ours, Bob) == True:
4       Send (transferred, ours, Bob) to Bob and  $\mathcal{E}$ 
5       Wait for response from Bob
6       If response == (transferred, theirs, Bob):
7           send message (traded, ours, theirs, Bob) to  $\mathcal{E}$ 
8       Else:
9           send message (cheated, ours, theirs, Bob) to  $\mathcal{E}$ 
10
11  Upon receiving (letsTrade, theirs, ours) from Bob:
12      Send (willWeCheat, theirs, ours, Bob) to  $\mathcal{E}$ 
13      Wait for response from  $\mathcal{E}$ 
14      If response is (doNotCheat, theirs, ours, Bob):
15          If (transfer, ours, Bob) == True:
16              Send (transferred, ours, Bob) to Bob and  $\mathcal{E}$ 
17
18  transfer(object, receiver):
19      If isAvailable(object):
20          If object is money:
21              send (pay, object, receiver) to  $\mathcal{F}_{\text{Ledger}}$ 
22              wait for response from  $\mathcal{F}_{\text{Ledger}}$ 
23              Upon receiving (paymentDone, object, receiver):
24                  return True
25          Else: # object is asset
26              send (transfer, object, receiver) to  $\mathcal{F}_{\text{Assets}}$ 
27              wait for response from  $\mathcal{F}_{\text{Assets}}$ 
28              Upon receiving (transferDone, object, receiver):
29                  return True
30      return False
31
32  isAvailable(object):
33      If object is money:
34          send (doIHaveBalance, object) to  $\mathcal{F}_{\text{Ledger}}$ 
35          wait for response from  $\mathcal{F}_{\text{Ledger}}$ 
36          return response
37      Else: # object is asset
38          Send (doIHave, object) to  $\mathcal{F}_{\text{Assets}}$ 
39          wait for response from  $\mathcal{F}_{\text{Assets}}$ 
40          If response == (youHave, object):
41              return True
42      Else:

```

```

43         return False
44
45     Upon receiving message (obtain,  $s$ ) from  $\mathcal{E}$ :
46         send message (add,  $s$ ) to  $\mathcal{F}_{\text{Assets}}$ 
47         wait for response from  $\mathcal{F}_{\text{Assets}}$ 
48         If response == (added,  $s$ )
49             send message (obtained,  $s$ ) to  $\mathcal{E}$ 
50         Else
51             send message (notObtained,  $s$ ) to  $\mathcal{E}$ 
52
53     Upon receiving message (lose,  $s$ ) from  $\mathcal{E}$ :
54         send message (remove,  $s$ ) to  $\mathcal{F}_{\text{Assets}}$ 
55         wait for response from  $\mathcal{F}_{\text{Assets}}$ 
56         If response == (removed,  $s$ )
57             send message (lost,  $s$ ) to  $\mathcal{E}$ 
58         Else
59             send message (notLost,  $s$ ) to  $\mathcal{E}$ 

```

## 5 $\mathcal{F}_{\text{SAT}}$ and $\Pi_{\text{SAT}}$ are potentially distinguishable

Consider the hybrid world of Fig. 1 (right) with  $n$  ITMs executing  $\Pi_{\text{SAT}}$ , where  $\mathcal{F}_{\text{Trust}}$  is replaced by  $\mathcal{F}'_{\text{Trust}}$ :

```

 $\mathcal{F}'_{\text{Trust}}$ 
1 Upon receiving (chooseBestSeller,  $d$ ,  $L$ ) from Alice:
2    $Bob \xleftarrow{R} L \cup \{\perp\}$ 
3   send message (bestSeller,  $d$ ,  $L$ , Bob) to Alice

```

We will show here that  $\mathcal{E}$  can distinguish between  $\mathcal{F}_{\text{SAT}}$  and  $\Pi_{\text{SAT}}^{\mathcal{F}'_{\text{Trust}}}$ .

*Distinguishability.* Consider the following adversary and environment:

```

 $\mathcal{A}$ 
1 Upon receiving (chooseBestSeller,  $d$ , list, Alice):
2   If  $|\text{list}| \neq 1 \vee |d| \neq 1$ :
3     go to Idle State
4    $Bob = p : (p, x, s) \in \text{list}$ 
5    $s = \text{asset} : \text{asset} \in d$ 
6   return (bestSeller, list, Bob, 1,  $s$ )

```

$\mathcal{E}$  distinguisher

```

1   $Alice \xleftarrow{R} \mathcal{P}$ 
2   $Bob \xleftarrow{R} \mathcal{P} \setminus \{Alice\}$ 
3   $U_{Alice}(t, x, a) = x + 2|a|$ 
4   $U_{Bob}(t, x, a) = 2x + |a|$ 
5   $\forall p \in \{Alice, Bob\}$ :
6    send message (type,  $U_p$ ) to  $p$ 
7   $s \xleftarrow{R} Asset$ 
8  send message (obtain, 1,  $Alice$ ) to  $G_{Ledger}$ 
9  send message (obtain,  $s$ ,  $Bob$ ) to  $G_{Assets}$ 
10 send message (satisfy,  $\{s\}$ ,  $\{Bob\}$ ) to  $Alice$ 
11
12 Upon receiving message ( $x$ ,  $\{s\}$ ,  $\{Bob\}$ ) from  $Alice$ :
13   If  $x == \text{satisfied}$ :
14     return 1 # functionality
15   Else: # if  $x \in \{\text{cheated}, \text{unsatisfied}\}$ 
16     return 0 # protocol

```

Because of the way  $\mathcal{E}$  is built, there always exists a seller ( $Bob$ , line 2) who has an asset (line 9) that can satisfy the desire (line 10) of the buyer ( $Alice$ , line 1).

In case  $\mathcal{E}$  interacts with  $\mathcal{F}_{SAT}$ , let  $\mathcal{S}$  simulator that tries to simulate  $\mathcal{A}$ .  $\mathcal{F}_{SAT}$  will always send the message (`chooseBestSeller`,  $\{s\}$ ,  $\{(Bob, 1, s)\}$ ,  $Alice$ ) to  $\mathcal{S}$  because:

1.  $Alice$  has one coin according to  $\mathcal{E}$ , line 8, as required by  $\mathcal{F}_{SAT}$ , line 15.
2. It is in  $Alice$ 's benefit for the trade to go through, since she values acquiring one asset more than one coin ( $\text{util}(Alice)(\{s\}, 0) = 2 > 1 = \text{util}(Alice)(\emptyset, 1)$  as can be seen in  $\mathcal{E}$ , lines 3 and 6), as required in  $\mathcal{F}_{SAT}$ , line 16.
3. It is in  $Bob$ 's benefit for the trade to go through, since he values acquiring one coin more than one asset ( $\text{util}(Bob)(\{s\}, 0) = 1 < 2 = \text{util}(Bob)(\emptyset, 1)$  as can be seen in  $\mathcal{E}$ , lines 4 and 6), as required in  $\mathcal{F}_{SAT}$ , line 18.

$\mathcal{S}$  should always match the buyer and the seller because of the way  $\mathcal{A}$  is built. More precisely,  $\mathcal{S}$  must always respond to (`chooseBestSeller`,  $\{s\}$ ,  $\{(Bob, 1, s)\}$ ,  $\_$ ) with (`bestSeller`,  $\{(Bob, 1, s)\}$ ,  $Bob$ , 1,  $s$ ) in order to correctly simulate  $\mathcal{A}$  (lines 4-6).

Furthermore,  $\mathcal{F}_{SAT}$  never cheats on a trade and always chooses a suitable seller, price and asset (given that there exists one, which is the case

here as we saw earlier) (lines 21-29), thus the exchange will always complete correctly and  $\mathcal{E}$  will receive **satisfied** as response.  $\mathcal{E}$  will always correctly output 1 (which corresponds to the functionality, line 14).

On the other hand, in case  $\mathcal{E}$  interacts with  $\Pi_{\text{SAT}}$ , then we observe that  $\mathcal{F}'_{\text{Trust}}$  does not choose players depending on their reputation (line 2), thus in this particular setting the utility of the players does not depend on their reputation. Thus, if  $\mathcal{F}'_{\text{Trust}}$  does not respond with  $\perp$ , it is always in *Bob's* interest to cheat, since keeping the asset is preferable to giving it ( $\mathcal{E}$ , lines 4 and 6). Thus *Alice's* response to  $\mathcal{E}$  will always be **cheated**. If  $\mathcal{F}'_{\text{Trust}}$  responds with  $\perp$  (line 2), the trade will not go through ( $\Pi_{\text{SAT}}$ , lines 33-34) and  $\mathcal{E}$  will receive **unsatisfied** as a response. In all cases  $\mathcal{E}$  will correctly output 0 (which corresponds to the protocol, line 16).  $\square$

$\mathcal{F}_{\text{Trust}}$

```

1 Has oracle access to every player's Alice utility function
   $U_{\text{Alice}}$ 
2
3 Upon receiving (chooseBestSeller,  $d$ ,  $L$ ) from Alice at time
   $t$ :
4    $m_{\text{Alice}} = \mathcal{G}_{\text{Ledger}}.\text{money}(\text{Alice})$ 
5    $a_{\text{Alice}} = \mathcal{G}_{\text{Assets}}.\text{assets}(\text{Alice})$ 
6    $m_{\text{Bob}} = \mathcal{G}_{\text{Ledger}}.\text{money}(\text{Bob})$ 
7    $a_{\text{Bob}} = \mathcal{G}_{\text{Assets}}.\text{assets}(\text{Bob})$ 
8    $\text{Bob} = \underset{\text{Bob} \in L}{\text{argmax}} \{ U_{\text{Alice}}(t, m_{\text{Alice}} - x, a_{\text{Alice}} \cup \{s\}) :$ 
9      $s \in d \cap a_{\text{Bob}} \wedge m_{\text{Alice}} \geq x \wedge$ 
10     $U_{\text{Alice}}(t, m_{\text{Alice}} - x, a_{\text{Alice}} \cup \{s\}) > U_{\text{Alice}}(t, m_{\text{Alice}}, a_{\text{Alice}}) \wedge$ 
11     $U_{\text{Bob}}(t, m_{\text{Bob}} + x, a_{\text{Bob}} \setminus \{s\}) > U_{\text{Bob}}(t, m_{\text{Bob}}, a_{\text{Bob}}) \wedge$ 
12     $U_{\text{Bob}}(t, m_{\text{Bob}} + x, a_{\text{Bob}}) < U_{\text{Bob}}(t, m_{\text{Bob}}, a_{\text{Bob}}) \}$  # impossible
13   send message (bestSeller,  $d$ ,  $L$ , Bob) to Alice
```

Assumptions:

- Trades are atomic.  $\mathcal{F}_{\text{Trust}}$  can deterministically decide whether *Bob* will complete the trade.
- The internal workings of  $\mathcal{F}_{\text{Trust}}$  is common knowledge to the players. Their utility depends on it.

Note: It would be interesting to see how utilities (as algorithms) and  $\mathcal{F}_{\text{Trust}}$  have a "fixed point".

$\Pi_{\text{Trust}}$

```

1 Upon receiving (chooseBestSeller,  $d$ ,  $L$ ) from Alice:
```

```

2   Bob =
3   argmax(Bob,x) ∈ L {  $\mathcal{O}_{\text{Rep}}(\text{Alice}, \text{Bob}, s) \text{util}(\text{Alice})(\text{state}_{\text{Alice}} \cup s - x)$  +
4    $(1 - \mathcal{O}_{\text{Rep}}(\text{Alice}, \text{Bob}, s)) \text{util}(\text{Alice})(\text{state}_{\text{Alice}} - x)$  :
5    $s \in d \wedge s \in \text{assets}(\text{Bob}) \wedge \text{Alice has } x \text{ coins}$  }
6   # drop "Bob has s"?
7   If  $\text{util}(\text{Alice})(\text{state}_{\text{Alice}}) \geq \text{util}(\text{Alice})(\text{state}_{\text{Alice}} \cup s - x)$ :
8     Bob =  $\perp$ 
9   send message (bestSeller, d, L, Bob) to Alice

```

## 6 An open source reputation management algorithm is impossible

**Theorem 1.** *Let  $U_{\text{Alice}} : \text{Money} \times \text{multiset}(\text{Assets}) \times \text{Rep} \rightarrow \mathbb{R}$ , where  $\text{Rep} \in \mathbb{N}$ . ( $\mathcal{F}_{\text{Trust}}$  does not have access to any oracle.) Then  $\nexists \mathcal{F}_{\text{Trust}} : \Pi_{\text{SAT}}$  UC-realizes  $\mathcal{F}_{\text{SAT}}$ .*

*Proof.* We will prove this by contradiction. Let  $\mathcal{F}_{\text{Trust}}$  be such a functionality. If it does not allow for cheats (grim trigger for seller on buyer reporting a cheat), then a malicious buyer (also prospective seller) can try buying from a competitor, falsely report a cheat and beat the competition (thus increasing their utility). If it allows for cheats, then it does not emulate  $\mathcal{F}_{\text{SAT}}$ . TODO: complete  $\square$

**Definition 1 (Cheating Security).** *We say that a protocol is secure against cheating if  $\forall \text{PPT } \mathcal{E}, \Pr[\mathcal{E} \text{ receiving } (\text{cheated}, \_, \_)]$  is negligible.*

Question: how to express security against false (cheated,  $\_, \_$ ) messages?

## 7 A case against honest and malicious parties

Satisfying Definition 1 in case there exist honest parties and an Adversary seems impossible. When an honest buyer pays a malicious seller, then the malicious seller can always cheat and the honest buyer will report it to  $\mathcal{E}$ . It seems improbable that we can create an  $\mathcal{F}_{\text{Trust}}$  that returns a malicious party with negligible probability, especially since  $\mathcal{F}_{\text{Trust}}$  cannot know from the outset which parties are corrupted and which are not.

Similarly, if a malicious buyer trades with an honest seller, then the seller will go through with the trade but the buyer will be able to falsely report a cheat to  $\mathcal{E}$ . This as well cannot be avoided, since  $\mathcal{E}$  does not



know whether the player to which it sends a (**satisfy**, **d**, **L**) message is corrupted.

Nevertheless, if we completely change perspective and have all players be rational, then it is possible that for certain utility functions no player will cheat or report false cheating for fear of future punishment.

## 8 Interaction with $F_{\text{Trust}}$

*Alice* receives an offer from *Bob* for price  $p$ . If  $\text{Trust}_{\text{Alice} \rightarrow \text{Bob}} \geq p$ , then she pays *Bob* by telling  $F_{\text{Trust}}$  to redistribute the  $p$  of her direct trust to him so that we end up with  $\text{Trust}_{\text{Alice} \rightarrow \text{Bob}} = \text{DirectTrust}_{\text{Alice} \rightarrow \text{Bob}}$ .

If the trade completes correctly and *Bob* takes  $p'$  coins from *Alice*'s direct trust, where  $0 \leq p' \leq p$ , *Alice* undoes the initial redistribution and on top of this she adds another  $p - p'$  as direct trust to *Bob*. If she has spare exclusive coins, she uses them, otherwise she decreases her direct trust to the least recently used direct trust. LRU direct trust is the oldest direct trust that helped her make a decision — for that she has to locally timestamp the direct trusts that are leveraged whenever she decides to trust someone.

If the trade fails to complete and *Bob* takes  $p'$  coins from *Alice*'s direct trust, where  $0 \leq p' \leq p$ , she takes her  $p - p'$  coins back and also reduces her trust to the players towards whom she had direct trusts that pointed her to *Bob* by another total  $p$ , keeping this money for exclusive use.

How can *Bob* exploit this strategy? He would need the collaboration of the players *Alice* trusts directly. They have to steal from her at the same time as he does. If we assume adaptive corruptions, then this is rather easy.

*Bob* seems to be able to cheat on half of the trades and still he will have some incoming direct trust. If he has say  $100p$  incoming direct trust, then he can maintain the same balance by selling honestly one item of value  $p$  and then cheating on the next trade of the same item alternatively. It is not directly obvious that this strategy will eventually leave *Bob* alone — indeed, he may build a lucrative mafia-like scheme this way; he just has to find constantly new first-line cheaters.

In order to mitigate this attack, a "know your customer" scheme may be necessary. Indeed, if honest sellers only sell to *Bob* only if they trust him enough, this scheme does not work because nobody will accept his money. Exclusive coins will only be useful as reserve to be directly trusted to others. This however completely overhauls the way money works today...

## 9 Single Corruption Game

---

**Algorithm 1**  $\text{singleCorruptionGame}_{\mathcal{A}}(G = (V, E, W), CP, D, w, \text{end})$

---

```

1: for all  $v \in V$  do
2:   if  $v \neq w$  then
3:     start  $\Pi_{\text{SAT}}$  on behalf of  $v$ 
4:   else
5:     start  $\mathcal{A}(v)$ 
6:   end if
7:    $U(v) \leftarrow CP$ 
8:   send (utility,  $U(v)$ ) to  $v$ 
9: end for
10: for  $t \leftarrow 1$  to  $\text{end}$  do
11:    $v \xleftarrow{r} V; d \xleftarrow{r} D; L \subseteq_r V$ 
12:   send (satisfy,  $d, L$ ) to  $v$ 
13:   wait for (satisfied,  $d, L$ ) or (cheated,  $d, L$ ) from  $v$ 
14: end for
15: output  $U(w)$ 

```

---

## References