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 Introduction

Consider the UC setting, with an environment $\mathcal{E}$, an adversary $\mathcal{A}$ and a set of ITIs that follow a given protocol Π.

Definition 1 (Player). *A player is an ITM that follows Π. Let $\mathcal{P}$ be the set of all players.*

Intuitively, players spontaneously feel different desires of varying intensities and seek to satisfy them, either on their own, consuming part of their input tokens in the process, or by delegating the process to other players and paying them for their help with part of their input tokens. The choice depends on the perceived difference in price and whether other players are trustworthy enough to satisfy the desire as promised. Each player plays rationally, always attempting to maximize her utility.

2 Mechanics

More precisely, let $\mathcal{D}$ be a set containing all possible desires. At arbitrary moments during execution, $\mathcal{E}$ can provide input to any player $Alice \in \mathcal{P}$ in the form (idx, d), where $idx \in \mathbb{N}, d \in \mathcal{D}, u \in \mathbb{R}^+$. idx represents an index number that is unique for each input generated by $\mathcal{E}$ and d represents the desire. d is satisfied when $Alice$ learns the string $s\left(idx, d, Alice\right)$, either by directly calculating it or by receiving it as subroutine output from another player. Some of the players, given as input the tuple $(idx, d, Alice)$, can calculate $s\left(idx, d, Alice\right)$ more efficiently than $Alice$, which means that they need to consume less input tokens than $Alice$ for this calculation. $Alice$ can choose to delegate this calculation to a more efficient player Bob and provide the necessary input tokens for his computation with a surplus to compensate *Bob* for his effort. Both players are better off, because *Alice* spent less tokens than she would if she had calculated $s\,(idx, d, Alice)$ herself, whilst *Bob* obtained some tokens which can in turn be used to satisfy some of his future desires.

Definition 2 (Cost of desire). *The cost of Alice's indexed desire, say* $(idx, d) \in (\mathbb{N}, \mathcal{D})$*, when satisfied by Bob is equal to the input tokens that Alice is required by Bob to give to him in order for him to calculate* $s\,(idx, d, Alice)$ *and is represented by* $c\,(idx, d, Alice, Bob)$*. The cost of satisfying this desire herself is represented by* $c\,(idx, d, Alice)$ *and is equal to the number of input tokens Alice must consume in order to calculate* $s\,(idx, d, Alice)$ *herself. Let* $c\,(idx, d, Alice, Alice) = c\,(idx, d, Alice)$*.*

It is reasonable to assume that there exists an absolute minimum of tokens that must be spent for the satisfaction of a desire, no matter how efficient the calculating party is.

Definition 3 (Minimum cost of desire). *The minimum cost of Alice's indexed desire, say* $(idx, d) \in (\mathbb{N}, \mathcal{D})$*, is equal to the theoretical minimum of input tokens required for the calculation of* $s\,(idx, d, Alice)$ *and is represented by* $c_{min}\,(idx, d, Alice)$*.*

Note that $1 \leq c_{min}\,(idx, d, Alice) \leq \min\limits_{v \in \mathcal{P}} c\,(idx, d, Alice, v)$.

Definition 4 (Player of desire function).
The function $Player : \mathbb{N} \times \mathcal{D} \to \mathcal{P}$ *takes as input an indexed desire and returns the player to which this desire was input by the environment.*

Definition 5 (Is the desire satisfied? property).
The property $isSatisfied : \mathbb{N} \times \mathcal{D} \to \{True, False\}$ *takes as input an indexed desire and returns whether it has been satisfied.*

Definition 6 (Desires of Player function).
The function $Desires : \mathcal{P} \to (\mathbb{N} \times \mathcal{D})^2$ *takes as input a player and returns the set of desires that the player was assinged throughout the game.*

$$Desires\,(v) = \bigcup_{(idx,d) \in \mathbb{N} \times \mathcal{D}} \{(idx, d) : Player\,(idx, d) = v\}$$

We also define $SatDesires : \mathcal{P} \rightarrow (\mathbb{N} \times \mathcal{D})^2$ *and* $UnsatDesires : \mathcal{P} \rightarrow (\mathbb{N} \times \mathcal{D})^2$*:*

$$\begin{aligned} SatDesires\left(v\right) = \bigcup_{(idx,d) \in \mathbb{N} \times \mathcal{D}} \{(idx, d) : Player\left(idx, d\right) = v \wedge \\ \wedge\, isSatisfied\left(idx, d\right) = True\} \\ UnsatDesires\left(v\right) = \bigcup_{(idx,d) \in \mathbb{N} \times \mathcal{D}} \{(idx, d) : Player\left(idx, d\right) = v \wedge \\ \wedge\, isSatisfied\left(idx, d\right) = False\} \end{aligned}$$

It is straightforward to see that

$$\forall v \in \mathcal{P}, SatDesires\left(v\right) \cap UnsatDesires\left(v\right) = \emptyset \ .$$

If additionally we suppose that $isSatisfied\left(\right)$ is always a computable function, then

$$\forall v \in \mathcal{P}, Desires\left(v\right) = SatDesires\left(v\right) \cup UnsatDesires\left(v\right) \ .$$

$\mathcal{E}$ can calculate the functions and the property defined above for all inputs at any moment in time.

The game begins with all players being created by $\mathcal{E}$, each allocated a random amount (TODO: define distribution) of input tokens. The game ends at a moment specified by the $\mathcal{E}$, which is unknown to the players. At that moment $\mathcal{E}$ assigns a utility to each player depending on which desires were satisfied throughout the game.

Definition 7 (Utility).

$$\forall v \in \mathcal{P}, u_v = f\left(SatDesires\left(v\right), UnsatDesires\left(v\right)\right),$$
$$U = \bigcup_{v \in \mathcal{P}} \{u_v\}$$

3 Utility Function Properties

Let $v \in \mathcal{P}$. The only property that any utility function should satisfy is that having more desires satisfied and less desires unsatisfied leads to a higher utility. More formally, consider two executions of the game (1) and (2):

$$\left.\begin{matrix} SatDesires_1\left(v\right) \subseteq SatDesires_2\left(v\right) \\ UnsatDesires_2\left(v\right) \subseteq UnsatDesires_1\left(v\right) \end{matrix}\right\} \Rightarrow u_{v,1} \leq u_{v,2} \ .$$

If one of the two subset is strict, then the inequality becomes strict as well.

As an example, consider two executions of the game (1) and (2) in which v was input exactly the same desires and satisfied the same desires, except for (idx, d), which was satisfied only in game (2) and remained unsatisfied in game (1). Then $u_{v,1} < u_{v,2}$. Going in some detail:

$$\text{Let } Desires_1(v) = Desires_2(v) = \{(idx, d)\} \cup Rest \ .$$

$$\text{It is:} \quad \begin{array}{r} SatDesires_1(v) = Rest \ , \\ UnsatDesires_1(v) = Rest \cup \{(idx, d)\} \ , \\ SatDesires_2(v) = Rest \cup \{(idx, d)\} \ \text{and} \\ UnsatDesires_2(v) = Rest \ , \end{array}$$

$$\text{thus:} \quad \left. \begin{array}{c} SatDesires_1(v) \subset SatDesires_2(v) \\ UnsatDesires_2(v) \subset UnsatDesires_1(v) \end{array} \right\} \Rightarrow u_{v,1} < u_{v,2} \ .$$

4 Desire Satisfaction Ideal Functionality

Following the UC paradigm, in this section we define the ideal functionality for desire satisfaction, $\mathcal{F}_{SAT}$. In this setting, all the desires that are generated by the environment and are input to the players are immediately forwarded to $\mathcal{F}_{SAT}$; 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 input tokens all players have. D could never be satisfied by the players because of the high total cost, but a $\mathcal{F}_{SAT}$ with no consideration for cost could in principle satisfy all desires in D.

The functionality can calculate the properties and functions defined in 4, 5 and 6 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 $\mathcal{F}_{SAT}$ knows U or an approximation of it.

Going into more detail, $\mathcal{F}_{SAT}$ 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.

$\mathcal{F}_{SAT}$ 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 $cost\,(u, d)$ which returns the cost of the desire d for player u with no side effects, $sat\,(u, d)$ that returns the string that satisfies the desire d to u and reduces the amount of the tokens belonging to u by $cost\,(u, d)$. There exists also the function $transfer\,(u_1, u_2, t)$ 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.

$\mathcal{F}_{SAT}$

```
Declarations:
  boolean allSet
  set pendingDesires
  map isSet(𝒫 => boolean)
  map price(𝒫 => map (𝒟 => number))
       # Is just the best price enough?
  map tokens(𝒫 => number)
  map fulfillment(𝒫 => map (𝒟 => SAT))
  map utility(𝒫 => function(set, set): number)
       # This map has hardcoded values

Initialization:
  allSet = False
  pendingDesires = ∅
  ∀ Alice ∈ 𝒫
    isSet(Alice) = False
    price(Alice) = null
    tokens(Alice) = ℱ_Ledger.getTokens(Alice)
    ∀ desire ∈ 𝒟
      fulfillment(Alice)(desire) = null

Upon receiving message
     init(map prices(𝒟 => number)) from 𝒫 Alice:
  If isSet(Alice) == False
    isSet(Alice) = True
    price(Alice) = prices
```

```
    If ∀ Bob ∈ 𝒫, isSet(Bob) == True
      allSet = True

Upon receiving message cost(𝒫 Bob, 𝒟 desire) from 𝒫 Alice:
  If allSet == True
    send message price(Bob)(desire) to Alice

Upon receiving message satisfy(𝒟 desire) from 𝒫 Alice:
  If allSet == True
    number minPrice = min_{Bob∈𝒫} {price(Bob)(desire)}
    If tokens(Alice) ≥ minPrice
      tokens(Alice) = tokens(Alice) - minPrice
      fulfillment(Alice)(desire) = s(Alice, desire)
    Else If utility(Alice)(satDesires(Alice) ∪ desire,
        unsatDesires(Alice) \ desire) -
        utility(Alice)(satDesires(Alice), unsatDesires(Alice))
        > (utility that can be gained for other players for
        minPrice)
      support = choosePayers(minPrice)
      ∀ (Bob, partialPrice) ∈ support
        tokens(Bob) = tokens(Bob) - partialPrice
      fulfillment(Alice)(desire) = s(Alice, desire)
    Else # desire remains unsatisfied
      pendingDesires = pendingDesires ∪ (Alice, desire)
    Attempt to satisfy desires from pendingDesires
         # How? Also, could be ℱ_SAT idle action

Upon receiving message
     getFulfillment(𝒟 desire) from 𝒫 Alice:
  If allSet == True
    send message fulfillment(Alice)(desire) to Alice

choosePayers(number price) returns set of (𝒫, number)
# Chooses to charge players that are most likely not to
    benefit by their tokens

satDesires(𝒫 Alice) returns set:
  ret = ∅
  ∀ desire ∈ 𝒟
    if fulfillment(Alice)(desire) ≠ null
```

```
    ret = ret ∪ desire
  return ret

unsatDesires(𝒫 Alice) returns set:
  ret = ∅
  ∀ desire ∈ 𝒟
    if fulfillment(Alice)(desire) == null
    ret = ret ∪ desire
  return ret

# May be redundant
transfer(𝒫 Alice, 𝒫 Bob, number tokens):
  If allSet == True
    If tokens(Alice) ≥ tokens
      tokens(Alice) = tokens(Alice) - tokens
      tokens(Bob) = tokens(Bob) + tokens
```

5 Trust definitions

We define two kinds of trust: direct and indirect. Direct trust from *Alice* to *Bob* is represented by input tokens (initially belonging to *Alice*) actively put by her in a common account from which *Bob* can also take them. As long as *Bob* does not take these tokens, *Alice* directly trusts him equally to the amount of tokens deposited in the common account.

This information can be used by another player *Charlie* that directly trusts *Alice* in order to derive information regarding *Bob*'s trustworthiness, even if *Charlie* does not directly trust *Bob*. Through a transitive property, *Dean*, who directly trusts *Charlie*, can in turn derive information from the direct trust from *Alice* to *Bob*. This reasoning can be extended to an arbitrary number of players, as long as they have at least one trust path to *Alice*. This is called indirect trust.

Definition 8 (Direct Trust). *The direct trust from Alice to Bob, represented by $DTr_{Alice \rightarrow Bob}$, is equal to the total tokens that Alice has given as input to $\mathcal{F}_{Trust}$ with* `entrust(`Bob`, 1`$^{\texttt{tokens}}$`)` *and also equal to the available tokens count sent by $\mathcal{F}_{Trust}$ as a response to a message* `query_direct_trust(`$Alice$`, `Bob`)`.

Definition 9 (Indirect Trust). *The indirect trust from Alice to Bob, $Tr_{Alice \rightarrow Bob}$, is measured in input tokens and can be calculated deterministically given the existing direct trusts between all pairs of players.*

It is equal to the number sent by $\mathcal{F}_{Trust}$ as a response to a message `query_indirect_trust(`$Alice$`, `Bob`)`.

By convention $DTr_{Alice \to Alice} = Tr_{Alice \to Alice}$ and are both equal to the quantity of input tokens $Alice$ has.

We would like to provide players with an ideal functionality where they:

1. Directly trust tokens to another player
2. Steal tokens previously directly entrusted by another player
3. Retract tokens previously directly entrusted to another player
4. Query indirect trust towards another player

The following functionality provides such an interface:

$\mathcal{F}_{Trust}$

`Initialization:`
`for all` $Alice, Bob \in \mathcal{P}$
  $DTr_{Alice \to Bob}$ `= 0`
  $Tr_{Alice \to Bob}$ `= 0`

`Upon receiving input entrust(`Bob`, 1`$^{\texttt{tokens}}$`) from` $Alice$`:`
  $DTr_{Alice \to Bob}$ `+= tokens`

`Upon receiving message steal(`Bob`, tokens) from` $Alice$`:`
  `If` $DTr_{Bob \to Alice}$ `>= tokens`
    $DTr_{Bob \to Alice}$ `-= tokens`
    `input 1`$^{\texttt{tokens}}$ `to` $Alice$
  `Else discard request`

`Upon receiving message distrust(`Bob`, tokens) from` $Alice$`:`
  `If` $DTr_{Alice \to Bob}$ `>= tokens`
    $DTr_{Alice \to Bob}$ `-= tokens`
    `input 1`$^{\texttt{tokens}}$ `to` $Alice$
  `Else discard request`

`Upon receiving message query_direct_trust(`$Alice$`, `Bob`) from` $Charlie$`:`
  `If` $Charlie \in \{Alice, Bob\}$ `# Privacy`
    `send message` $DTr_{Alice \to Bob}$ `to` $Charlie$
  `Else discard request`

```
Upon receiving message query_indirect_trust(Alice, Bob)
    from Charlie:
  If Charlie == Alice # Privacy
    Tr_{Alice→Bob} = calculate_indirect_trust(Alice, Bob, all
        direct trusts)
    send message Tr_{Alice→Bob} to Charlie
  Else discard request
```

6 Desired Properties for Indirect Trust

1. $Tr_{Alice \rightarrow Bob} \geq DTr_{Alice \rightarrow Bob}$
2. If universe (1) and (2) are identical except for $DTr_{Alice \rightarrow Bob}$, then

$$Tr^2_{Alice \rightarrow Bob} = Tr^1_{Alice \rightarrow Bob} - DTr^1_{Alice \rightarrow Bob} + DTr^2_{Alice \rightarrow Bob} \ .$$

3. Consider an indexed desire (idx, d) $Alice$ has. Let $p\left(idx, d, Alice\right)$ be a function that returns the player that $Alice$ should rationally delegate the calculation of $s\left(idx, d, Alice\right)$ to.
   (a) If a player is cheaper and more trustworthy than all other players, delegate the calculation to him.

$$\begin{gathered} \exists Bob \in \mathcal{P} : \forall Charlie \in \mathcal{P} \setminus \{Bob\} \\ \left(c\left(idx, d, Alice, Bob\right) < c\left(idx, d, Alice, Charlie\right) \wedge \right. \\ \left. \wedge \ Tr_{Alice \rightarrow Bob} > Tr_{Alice \rightarrow Charlie}\right) \Rightarrow \\ \Rightarrow p\left(idx, d, Alice\right) = Bob \ . \end{gathered}$$

   (b) If there exists a player Bob that is both cheaper and more trustworthy than $Charlie$, do not delegate the calculation to $Charlie$.

$$\begin{gathered} \exists Bob, Charlie \in \mathcal{P} : \\ \left(c\left(idx, d, Alice, Bob\right) < c\left(idx, d, Alice, Charlie\right) \wedge \right. \\ \left. \wedge \ Tr_{Alice \rightarrow Bob} > Tr_{Alice \rightarrow Charlie}\right) \Rightarrow \\ \Rightarrow p\left(idx, d, Alice\right) \neq Charlie \ . \end{gathered}$$

Note that the first property can be deduced from the second.

Let $x = \left(idx, d, Alice\right)$. Several ideas exist as to what rules $p\left(x\right)$ should satisfy:

1. Indirect trust towards *Bob* is required to be greater than the cost of the calculation requested by *Bob* in order for *Alice* to delegate $s(idx, d, Alice)$ to him. If there exist multiple players that *Alice* indirectly trusts more than their cost, then the cheapest one is chosen. If multiple trustworthy enough players have the same cost, the most trustworthy one is chosen.

$$\begin{aligned}
&c(x, Charlie) > Tr_{Alice \to Charlie} && \Rightarrow p(x) \neq Charlie \\
&\text{For the following, } \forall v \in \{Bob, Charlie\}\ c(x, v) \leq Tr_{Alice \to v} && . \\
&c(x, Bob) < c(x, Charlie) && \Rightarrow p(x) \neq Charlie \\
&\left.\begin{aligned} &c(x, Bob) = c(x, Charlie) \\ &Tr_{Alice \to Bob} > Tr_{Alice \to Charlie} \end{aligned}\right\} && \Rightarrow p(x) \neq Charlie
\end{aligned}$$

2. Similarly to the previous idea, the indirect trust must exceed the cost. In this case however, in case of multiple trustworthy and cheap players, the indirect trust is considered before the cost.

$$\begin{aligned}
&c(x, Charlie) > Tr_{Alice \to Charlie} && \Rightarrow p(x) \neq Charlie \\
&\text{For the following, } \forall v \in \{Bob, Charlie\}\ c(x, v) \leq Tr_{Alice \to v} && . \\
&Tr_{Alice \to Bob} > Tr_{Alice \to Charlie} && \Rightarrow p(x) \neq Charlie \\
&\left.\begin{aligned} &Tr_{Alice \to Bob} = Tr_{Alice \to Charlie} \\ &c(x, Bob) < c(x, Charlie) \end{aligned}\right\} && \Rightarrow p(x) \neq Charlie
\end{aligned}$$

3. The player with the highest difference between indirect trust and cost is chosen.

$$p(x) = \operatorname*{argmax}_{v \in \mathcal{P}} (Tr_{Alice \to v} - c(x, v))$$

   This aims to maximize trust and minimize cost and, contrary to the previous two approaches, attaches equal importance to these two metrics.
4. The player with the lowest difference between indirect trust and cost is chosen.

$$p(x) = \operatorname*{argmin}_{v \in \mathcal{P}} (Tr_{Alice \to v} - c(x, v))$$

   Note that this last approach constitutes another direction, which departs from choosing the cheapest and most trustworthy vendor, opting for the one whose price and trustworthiness match. It evidently does not follow the property (3).

7 Motivation for our Trust model

Nevertheless, one can say that at first sight it is in *Bob*'s best interest to trick *Alice* into believing that he can efficiently calculate $s\left(idx, d, Alice\right)$ and skip the computation entirely after obtaining *Alice*'s input, thus keeping all the tokens of the defrauded player. Evidently *Alice* would avoid further interaction with *Bob*, but without any way to signal other players of this unfortunate encounter, *Bob* can keep defrauding others until the pool of players is depleted; if the players are numerous or their number is increasing, *Bob* may keep this enterprise very profitable for an indefinite amount of time. This being a rational strategy, every player would eventually follow it, which through a "tragedy of the commons" effect invariably leads to a world where each player must satisfy all her desires by herself, entirely missing out on the prospect of the division of labor.

One answer to that undesirable turn of events is a method through which *Alice*, prior to interacting with an aspiring helper *Bob*, consults the collective knowledge of her neighborhood of the network regarding him. There are several methods to achieve this, such as star-based global ratings. This method however has several drawbacks:

- Very good ratings cost nothing, thus convey little valuable information.
- Different players may have different preferences, global ratings fail to capture this. Arrow's impossibility theorem is possibly relevant here.
- Susceptible to Sybil attacks; mitigation techniques are ad-hoc and require (partial) centralization and secrecy/obfuscation of methods to succeed, thus undermining the decentralized, transparent nature of the system, a property that we actively seek.