# Game Theory and High-Level-Synthesis applications

Giuseppe Scalora
*Electronic Engineering*
*Hochschule Hamm-lippstadt*
Lippstadt, Germany
giuseppe.scalora@stud.hshl.de

***Abstract*—This paper´s goal is to understand what Game theory is and how it is used in applications for High level synthesis. It focuses on the main key concepts behind the general theory and algorithms, with some general knowledge about the background and the motivation. It discusses and explains some popular examples which are essential in order to understand and further proceed to read through the topic. It follows with two specific applications which are strictly related to the area of high level synthesis and more specifically in the area of Hardware systems design. The final part then sums up the key concepts and concludes with a comparison of the advantages and disadvantages of using this applied method in this specific field.**

***Index Terms*—games, high-level-synthesis, hardware, algorithm, hardware, design**

## I. Introduction

Mathematical models have always been used in order to provide aid for the specifications or development of new technologies and discoveries in the field of physics, economics, chemistry and we could go on forever listing all of them. Most importantly in the last century, with the latest implementations and improvements of new and old revisited mathematical models, the field of hardware and software engineering has been heavily affected in order to develop new strategies for design and construction purposes. More specifically for high level synthesis, there are many mathematical models which can be applied and be of interesting use. One of this is the so called Game theory, an idea that was conceived in the 18th century within the scope of economic applications. Since then the improvement and new discoveries on the field rendered the application of this mathematical model and algorithms useful in many other fields. The relevant one for our purpose will be the already mentioned field of hardware engineering/design. The latter is mainly focused on the verification and security of a hardware system, the improvement of design techniques to render the production more reliable and higly functional. Game theory is based upon the theory of rational choices, this implies that a decision maker chooses an action according to his/her preferences. The mathematical model of the game theory is only based upon decision making of some players acting within a system and maximization of outputs decisions according to the preferences and other players in the model.

## II. Background of game theory

The very beginning of some ideas regarding the game theory seem to be assigned to the 18th century while the actual and first development began in the early years of the 20th century (around 1920s) with the work of the mathematicians Emile Borel (1871–1956) and John von Neumann (1903–57). The latter mentioned, published a first work on game theory, alongside Oskar Morgenstein, called ”Theory of games and economic behaviour”. In the 1950s game-theoretic models began to be used in economic theory, political science and also by psychologists who began studying how human subjects behave in experimental games. Later during the 70s the theory started taking place and being applied into many others fields, one of this recently included, which is computer science [1].

## III. Game theory general model

This model based theory consists mainly of two components, a set A of actions that the decision maker can choose upon and specifications of the decision maker´s preferences. Normally, the decision maker is aware of the spectrum of decisions available and it is only allowed to choose one among each set of actions. The set A can include things such as deciding upon whether to eat fish, meat or vegetables, or deciding whether to drink wine, water or coke. Most of the times 2 sets go side by side which means the decision maker has to choose a pair of actions and not just a single one among multiple sets. In terms of preferences, instead, the decision maker can have a favourite option or some equally desirable ones. Let us take the example of a decision maker preferring coke over wine and wine over water, the outcome of this will imply that the decision maker will prefer coke over water. In order to formally define the preferences a payoff function, between actions and numbers, is represented. The numbers represent the value of the preference, the higher the number the more preferred the action [2].

## IV. The key concepts behind the theory

In order to understand how and why this algorithm has been developed it is necessary to get familiar with some important concepts used as means of understanding for everyone.

### A. Static games

Let us take into account an interactive decision problem which involves two or more individuals who have to come to a final decision according to which, the payoff for each person depends on the other individual decision. Commonly, such decision making problems can be assessed as "games" and the individuals making the decisions (2 or more) are called "players". The game does not necessarily need to have a winner and a loser but it can have restricted features, which will lead us to call such games as recreational. On the other hand, games that have winners and losers are called zero-sum games. Therefore the definition of a static game describes games in which the decisions are made simultaneously by the players and in ignorance of choices made by other players in the game. According to this they can be referred also as simultaneous decision games, simply because the order in which the decision must be taken is irrelevant [3]. A technical description of the main points a static game looks as follows:

1) A set of players must be defined in the form of a set:

$$i \in \{1, 2, 3...\}$$

2) A pure strategy set for all players, $\mathbf{S_i}$
3) Players payoffs, according to the decisions combinations.

A very popular example of such games is shown and explained in the following paragraph: "the prisoner´s dilemma" [4].

*1) The prisoner's dilemma:* Let us assume two different prisoners are going through a trial for a crime, whether it has occured for real or not. Both prisoners have a degree of freedom limited to 2 possible answers, confessing or remain silent, therefore the number of possible outcomes is bounded to 4. These, respectively, are: scenario 1 (For simplification prisoner 1 and 2 will be assessed by using simply P1 and P2). P1 confesses and P2 confesses, this outcome leads both prisoners to be sentenced to 4 years of prison. Scenario 2. P1 confesses and P2 remains silent which leads to P1 being sentenced to 1 year of prison while P2 is sentenced to 5 years. Scenario 3. P1 remains silent and P2 confesses, this will result in the opposite of what just mentioned, P1 will get 5 years and P2 only 1 year. Scenario 4. P1 and P2 remain silent, this will lead to both being sentenced to only 2 years. For a better understanding, the following matrix, in figure 1, will visually show the concept in a more schematic way.

Now as you can see from the 2X2 matrix, the most stable outcome would be the scenario 1, in which both prisoners confess the crime and are not, obviously, allowed to change their declaration at any given time. Meanwhile within the rest of the scenarios the prisoners might decide, at any point, to change their mind in order to obtain a better outcome for themselves, more on the aspect of being egoistic [5].

### B. Nash equilibrium

When dealing with game theory it is necessary to create and instantiate some equilibrium that helps reach the best possible outcome for a system of any nature. When talking about Nash equilibrium, it is only taken into consideration a non-cooperative game which involves 2 or more players. In this particular case, each player's decision should depend on the other players' decisions, therefore each of them must give an assumption or, better, create a belief of what the others will choose. The mainly basis for this derives from the assumption that the players have already gotten a past experience in the game in order to create reliable beliefs about the other players' possible behaviours. This is only possible in idealized circumnstances, although in many cases the players, even with past experiences, are only aware of the behaviours of other typical players but not of specific ones. This means that in the solution, given by the Nash equilibrium each player's beliefs are assumed to be correct hence the player's choices will be made according to other players' choices. *"A Nash equilibrium is an action profile $a^*$ with the property that no player **i** can do better by choosing an action different from $a^*_i$, given that every other player **j** adheres to $a^*_j$* [6].

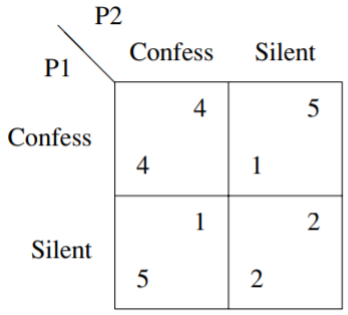


Fig. 1. Prisoner´s dilemma matrix

### C. Dynamic games

On the contrary of static games, dynamic games involve many situations of interest in which decisions are made at various times, not simultaneously, taking into account the choices made earlier. Therefore dynamic games introduce an explicit time-schedule which describes the exact time spans, or time points, during which the players make their decisions. Dynamic games can be represented by a game decision tree. The filled black circles show the time points at which decisions are taken. The connections between circles are self explanatory, they represent branches from which other action could occur. At the bottom of the tree, after every decision sequence has come to its end, there is the payoff, usually specified and written. By convention the tree is drawn "upside-down" which means that the time increases as the tree branches downwards.

*1) Dinner party game:* In order to understand the example, the tree will be shown and therefore analyzed.

Let us consider the tree shown in fig.2, in which the Husband will start the tree of decisions by buying either meat (M) or fish (F). Meanwhile the Wife will buy either Red wine (R) or White wine (W). We assume that both wife and husband prefer to drink red wine with meat and white wine with fish. Although the husband prefers to eat meat

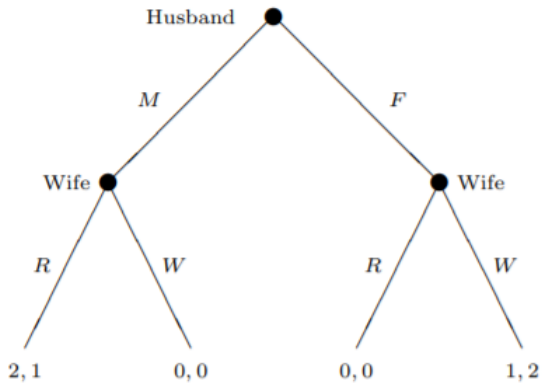


Fig. 2. dinner party game tree

for dinner and the wife prefers the opposite, the fish. The possible payoffs will be stated as following:

$$\pi_h \text{ (M, R)} = 2 \text{ , } \pi_h \text{ (F, W)} = 1$$
$$\pi_h \text{ (F, R)} = \pi_h \text{ (M, W)} = 0$$

$$\pi_w \text{ (M, R)} = 1 \text{ , } \pi_w \text{ (F, W)} = 2$$
$$\pi_w \text{ (F, R)} = \pi_w \text{ (M, W)} = 0$$

In order to solve this problem, we can assume that the husband has told the wife whether he bought fish or meat. Therefore, the discussion proceeds with a backward induction approach, if the husband has bought meat, for instance, then the wife will be aware of that and hence she will buy red wine in order to have a payoff = 1 rather than a payoff = 0 buying white wine. If the husband, instead, has bought fish, then the wife will proceed with buying white wine and have a payoff = 2 rather than = 0 with red wine and fish. It is pretty clear that in the best possible scenario, the husband will buy meat, in order to have a payoff = 2 and this will lead the wife in choosing red wine (payoff = 1) for the wife. In the end the dinner will be made of meat and red wine. [7]

### D. Stackelberg games

Another important concept, which will be helpful to understand the applications of game theory on specific fields, is the "Stackelberg games model". We suppose two firms are competing with each other, producing the same good. Firm $i$'s amount of product to produce is $q_i$ and the total output production cost, for all the units, will be assessed as $C_i$, meanwhile the selling price will be referred as $P_d$. According to a Stackelberg model, the two players (firms in this case) make their decisions sequentially, therefore, if firm i=1 makes the decision first, the second firm will follow knowing the decision of the first firm.
Given these initial information:

1) Set of 2 players (firms), $i$
2) Set of sequences ($q_1$, $q_2$ of firms outputs).

we can calculate the payoff for each firm which will look as such:

$$q_i P(q_1 + q_2) - C_i(q_i) \text{ for } i = 1,2$$

Firm 1 always moves first, generating an output strategy, and firm 2 always follows everytime firm 1 generates an output. Given that, using backward induction we can retrieve a solution.

1) For each firm 1 output, a firm 2 output is generated with the goal of maximizing the profits. The output of firm 2 is defined as $b_2$ for output of firm 1, defined $q_1$.
2) For each output, firm 1 must also decides its strategy upon maximization of its profit, so if firm 1 output is $q_1$ and firm 2 output is $b_2$, the total selling price $P_d$ is given by the sum of $q_1 + b_2$.

Therefore, if firm 2 best output is $b_2$ and firm 1 best output depends on firm 2 outputs, the new output for firm 1 will be defined as $q^*_1$ therefore, the pair $(q^*_1, b_2)$ will now lead firm 2 to behave accordingly to firm 1 new possible output, hence the new output for firm 2 will be $q^*_2$ coming, in conclusion, to a win-win situation for both firms, in which they perceive the maximum profits . This then leads to a subgame perfect equilibrium defined by the pair $(q^*_1, q^*_2)$. [8]

## V. Applications

### A. Validation and verification of hardware design using the Game theory

The concept explained and discussed in the following chapter, are taken from the conference paper. [9]

*1) Motivation:* When it comes to Hardware verification and validation, it is important to ensure that the design properties and goals, are specified accurately for the overall purpose of the final hardware design process. The accuracy is handled by a cooperation of the full spectrum of design teams which have to work on the project, therefore it requires a highly efficient communication among the teams. In terms of hardware verifications, it is necessary to apply mathematical models as proof of stability and accuracy for the implementation (reason why the Game Theory algorithm is feasible for this scope, being itself a mathematical model). The main focus of this application is the analysis and solution of problems regarding the security in Hardware systems and the correction of the desired behaviours of the already mentioned systems. Game theory, is applied here due to its useful aid in terms of securing physical aspects, strategic resources and power infrastructure. As already mentioned in the paper, more specifically, the Stackelberg game formulations will come in help. The main concept behind this is: the assumption that an attacker can observe all the security decisions placed by the defender. Verifying a hardware design as result of a security strategy implies that deployment schedules need to be as strict as possible, the state space size of the system, depends on its complexity and how critical the design is. .

*2) Approach:* In order to approach the problem of verification and validation of hardware design it is necessary to focus the interest into finding regions, of a complex hardware system, which are more vulnerable to possible malicious attacks. Given that the most vulnerable parts are composed by logic gates, the main feature that must be looked upon is a gate´s propensity to propagate failures or attacks to the final output of the whole system (which will lead to inevitable errors and faults compromising the behaviour and function of the system). Stackelberg Security games model is the way to go when security specific scenarios, in hardware design, have to be considered. Always in respect to the design´s specification, security means protection against any fault that can compromise the behaviour of the system or any malicious attack performed by third parts.

*3) Implementation:* To briefly describe how the system is implemented, the following chapter will analyze the behaviour of a hardware system, when a SSG model is applied. The main property to compute for security is the value of the defender of a vulnerable target. This is realized by the combination of how succesful an attack can be and the individual value of the target. Taking into account that a target will most likely be a netlist constructed by a Boolean network of logic gates, it is important to select the output wire for each flip-flop, within the target area of the hardware system, and we call it a Boolean node. Each output will then be the input of another flip-flop. Output nodes are defined as *j* and input nodes to a flip-flop are defined as *i*, these both, connected together, form an edge (*i*, *j*). The Boolean network G = (T, E), where T is the set of nodes or targets, and E is the set of edges, is shown in the following figure.

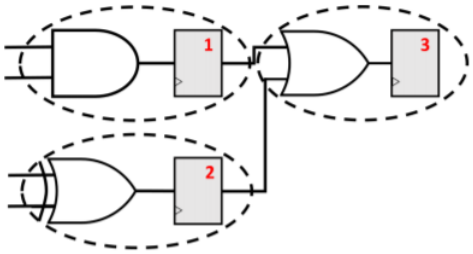


Fig. 3. Netlist of flip-flops and logic gates

Given that, the transfer function for each node can be represented as a truth table of the given logic circuit. Usually obtained through independent simulation. The computation of defenders values relies on two concepts called, respectively, *influence* and *independent cascade contagion*. Let us briefly explain what they are.

1) *Influence* defines the probability of a given bit value *i* belonging to an input vector *y* to change the output for the transfer function of a boolean node *j*. Technically speaking, low influence inputs can be overlooked during coverage tests as possible malicious insertion points, while high influence inputs are more propense to spread failures throughout the system, therefore they are considered in critical applications. Truth tables help to calculate the influence for given nodes and the complexity increases exponentially as the number of inputs increases.
2) *Independent cascade contagion* defines the determination process of the contagion failure rate throughout the system. The input graph takes the network G = (T, E), as explained earlier, and defines the probability $p_{i,i'}$ of the failure spread rate among the neighbors nodes, an example of the boolean network is shown as following:

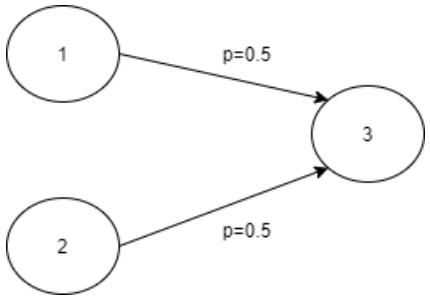


Fig. 4. Converted Boolean network with cascade contagion probability

The utility function of a defender at target *t*, is obtained, depending on three variables, such as: level of verification effort applied to target *t* defined as $q_t$, the evaluation of *t*, $v_t$, and spread contagion probability of *t* to propagate to *t'* for $t' \in T$.

$$U(t) = E[\sum_{t' \in T} (-v_{t'})\rho_{t'}(t)]$$

This utility function chooses the defender strategies to apply in order to minimize the losses. Accordingly, the attacker´s utility function is defined as $V_t = -U_t$.

*4) SSG applied and interpreted:* The concept of Stackelberg games has been introduced and discussed already. In this matter is important to apply the previous example in concern of a hardware system, therefore the 2 players acting in this scenario are: the attacker, A and the defender, D, who is the hardware designer/engineer. The solution applied to hardware verification with the aid of the Stackelberg games, aims to find equilibrium strategies which stabilize both attacker and defender on their own decisions. The attacker´s goal is to maximize the damage to the system but the defender is assumed to know the attacker utility function, which then leads the attacker to observe first which target nodes have been verified and secured by the defender, thus a Stackelberg equilibrium is created between both sides. Adopting the average-case Stackelberg equilibrium (ASE), makes sure that the attacker, in case it is ever indifferent of the node to attack, will decide upon a random one. This ASE helps the designer teams, to work on the same system although being non cooperative, because it considers uncertainty in the system, whenever the attacker is in front of multiple possible vulnerable nodes. In

order to compute ASE, the following mathematical constraints are necessary:

$$\max_{a,\mathbf{q},s,u,v} u - \sum_{t\in T} c_t q_t \quad (1)$$

s.t.

$$0 \le q_t \le 1 \qquad \forall\, t \in T \quad (2)$$

$$a_t \in \{0,1\} \qquad \forall\, t \in T \quad (3)$$

$$\sum_{t\in T} a_t \ge 1 \quad (4)$$

$$0 \le v - (1-q_t)V_t \le (1-a_t)M \qquad \forall\, t \in T \quad (5)$$

$$s_t = v - (1-q_t)V_t \qquad \forall\, t \in T \quad (6)$$

$$a_t + M s_t \ge 1 \qquad \forall\, t \in T \quad (7)$$

$$u = \frac{1}{\sum_{t\in T} a_t}\left(\sum_{t\in T} a_t(1-q_t)U_t\right), \quad (8)$$

(9)

where $M$ is some sufficiently large number, and $a_t = 1$ if node $t$ is attacked.

Fig. 5. ASE computational equations

Accordingly, equation 1 serves to find the maximization of the utility for the defender, the solutions must respect constraints 2-8. Constraints 4 and 7, meanwhile, enforce the maximization of the defender´s utility in respect to the attacker strategy. Constraint 4 forces the attacker to at least attack one node. Constraint 6 creates a gap variable between attacker optimal value and attacker target node value. Constraints 7 and 5 force the attacker to attack a node only if it can be done with maximum utility from the attacker. Lastly, constraint 8 is the final equation to find the maximum utility for the defender, given attacker strategy *a* and defender strategy *q*, once all the constraints have been satisfied. The results are then gonna be sorted from highest to lowest in to *q**. To read and interpret the results of the mathematical model, we take the results for *q** and compare if the verification efforts have been distribuited properly, if so, then the target node *t* is considered to be validated and secure, proportional to $q^*_t$. For example, a $q^*_t$ value of 1 is considered to fully adhere to the verification specifications, while a value of 0 is not considered so but it is still not neglected, just put on a lower prioritization.

### *B. Game theory on Trojan detection for HLS*

The following chapter focuses on a brief explanation of a research done about the topic of hardware trojan detection in HLS. All the concepts have been taken and re-elaborated from the cited conference paper. [10]

*1) Motivation:* As discussed in the previous application, when it comes to IC manifacturing, the developers might add by mistake some malicious trojans which have the capability of disrupting and compromising the behaviour of the circuit and therefore of the whole system surrounding it. At this point it is up to the company receiving the IC to test for trojans. It would realistically be too costly to test for every type of trojans therefore the purpose of this research is to develop a strategy that checks for what exact type of trojans to test for and how much damage can they bring to the system.

*2) A different approach for Hardware Trojans detection: Expected Utility Theory:* In order to do what mentioned, a static zero-sum noncooperative game is formulated. Its strategy looks like: N, $S_i$ $i \in N$, $u_i$ $i \in N$. In order, *N*:= a, d refers to the set of players,the so called attacker and defender. $S_i$ is the strategy space for each player, set of trojans to insert for the attacker and set of trojans to test for in the case of the defender, referred as with K. Lastly $u_i$ is the utility function. Let us define the defender utility function for each set K.

$$u_d(s_d, s_a) = \begin{cases} F_s a, & \text{if } s_a \in s_d \\ -V_s a, & \text{otherwise} \end{cases}$$

The utility function for the attacker instead will just be -$u_d$($s_d$, $s_a$).

Since the strategy for both defender and attacker is based upon probabilistic, the game is formulated under non deterministic mixed strategies. In game theory players are supposed to act rationally therefore, each player will choose upon their strategy objectively more specifically in this case they will choose upon a vector, defined as $\mathbf{p} = [\mathbf{p}_d, \mathbf{p}_a]$ this vector will give an expected utility value according to the Expected Utility Theory which defines the utility function as:

$$U_i^{EUT}(\mathbf{p}_d, \mathbf{p}_a) = \sum_{s\in S}(p_d(s_d)p_a(s_a))u_i(\mathbf{s})$$

Where **s** is simply a vector of attacker and defender strategies.

*3) Game solution:* The solution to the already proposed theory is based upon a mixed-strategy Nash equilibrium (MSNE). The definition for this looks as follows:
*A mixed strategy profile* ***p*** is said to be a mixed strategy Nash equilibrium if for the defender, d, and attacker, a, we have:*

$$U_d(\mathbf{p}*_d, \mathbf{p}*_a) \ge U_d(\mathbf{p}_d, \mathbf{p}*_a), \forall p_d \in P_d$$

$$U_a(\mathbf{p}*_a, \mathbf{p}*_d) \ge U_a(\mathbf{p}_a, \mathbf{p}*_d), \forall p_a \in P_a$$

In the MSNE the defender and the attacker, have no incentive to deviate from their current mixed strategy given that both of them are using the same MSNE strategy. Under rational choice, EUT, the defender will choose an optimal randomization over the testing strategies. Meanwhile, the attacker, under rational choice and according to MSNE, will choose its optimal randomization upon the trojans to insert. They both assume that the opposite side is using a rational choice and their utility is maximized according to EUT.

Considering the zero-sum nature of the game, the Von Neumann indifference principle comes in hand to find closed-form solutions. According to this, the expected utilities under MSNE are equal but opposite. The strategy space is very large therefore it might be difficult to find the solution to all the

important and relevant equations, in this concern, an algorithm must be developed from a starting point in the system. An example of the algorithm is shown in the figure below:

**Algorithm 1** Distributed Fictitious Play Learning Algorithm

**Input:** Action space of the defender, $\mathcal{S}_d$
Action space of the attacker, $\mathcal{S}_a$
Convergence parameter, $M$

**Output:** Equilibrium mixed strategy vector of each player, $\boldsymbol{p}_d^*$ and $\boldsymbol{p}_a^*$

Initialize $\boldsymbol{\sigma}_a^0$ and $\boldsymbol{\sigma}_d^0$
Initialize convergence tester: $C_{\text{test}} = 0$
Initialize iteration counter: $k = 1$
**while** Not Converged: $C_{\text{test}} == 0$ **do**
  Each player chooses its optimal strategy:
  $s_d^k = \arg\max_{s_d \in \mathcal{S}_d} U_d(s_d, \boldsymbol{\sigma}_d^{k-1})$
  $s_a^k = \arg\max_{s_a \in \mathcal{S}_a} U_a(s_a, \boldsymbol{\sigma}_a^{k-1})$
  Each player updates its observed empirical frequency:
  $\sigma_d^k(s_a) = \frac{k-1}{k} \cdot \sigma_d^{k-1}(s_a) + \frac{1}{k} \cdot \mathbf{1}_{\{s_a^{k-1} = s_a^k\}} \forall s_a \in \mathcal{S}_a,$
  $\sigma_a^k(s_d) = \frac{k-1}{k} \cdot \sigma_a^{k-1}(s_d) + \frac{1}{k} \cdot \mathbf{1}_{\{s_d^{k-1} = s_d^k\}}, \forall s_d \in \mathcal{S}_d$
  Check Convergence
  Calculate:
  $C_d(s_a) = |\sigma_d^k(s_a) - \sigma_d^{k-1}(s_a)| \, \forall s_a \in \mathcal{S}_a,$
  $C_a(s_d) = |\sigma_a^k(s_d) - \sigma_a^{k-1}(s_d)| \, \forall s_d \in \mathcal{S}_d$
  **if** Converged:
  $C_d(s_a) < \frac{1}{M}$ && $C_a(s_d) < \frac{1}{M}$ $\forall s_a \in \mathcal{S}_a$, $\forall s_d \in \mathcal{S}_d$ **then**
    $C_{\text{test}} = 1$
    Compute strategy vectors:
    $\boldsymbol{p}_d^* = [\sigma_a^k(s_1), \sigma_a^k(s_2), ..., \sigma_a^k(s_{|\mathcal{S}_d|})]$
    $\boldsymbol{p}_a^* = [\sigma_d^k(s_1), \sigma_d^k(s_2), ..., \sigma_d^k(s_{|\mathcal{S}_d|})]$
  **end if**
  Update Counter: k=k+1
**end while**
**return** Strategy vectors: $\boldsymbol{p}_d^*$ and $\boldsymbol{p}_a^*$

Fig. 6. Fictitious algorithm for strategy vectors

Going through the pseudo-coded algorithm step by step, the final result will give us in return the 2 strategy vectors necessary for the computation of the expected utilities for both attacker and defender.

*4) Final considerations:* The initial problem was conceived as a noncooperative game with two players, attacker and defender and the solution proposed to finding the potential inserted trojans, was based upon testing strategies using the Expected Utility Theory and the concept of Mixed Strategy Nash Equilibrium. Formulas have been derived in order to evaluate the strategies and the utilities for both players with also the help of an algorithm. Finally finding out how irrational behaviours and risks can impact the interactions between the two players present in the game.

## VI. Conclusion

Coming to a conclusion to the topic, it is clear how such flexible model, game theory, can be applied to many fields and also in the hardware design field. With the idea of involving techniques of decision making, through algorithms and mathematical constraints in order to seek for potential malicious elements within a system. With the aid of some examples it is also not difficult to understand the use of the concept, its realization and implementation in the real world applications. As discussed in this paper, which introduced concepts mainly relevant to High-level-synthesis, it is also important to fetch the utility and importance of the topic towards other disciplines without excluding them.

## Reference style

The reference and citation style used in this paper make sure that the reference is cited at the end of the discussed chapter, in order to state where is the concept coming from. If the reference has been put after an image then it will include the image source too.

## References


[1] M. Osborne, An introduction to game theory by Martin J. Osborne. Oxford: Oxford University Press, 2004, p. 3.
[2] M. Osborne, An introduction to game theory by Martin J. Osborne. Oxford: Oxford University Press, 2004, pp. 4-6.
[3] J. Webb, Game theory. [London]: Springer, 2007, p. 61.
[4] J. Webb, Game theory. [London]: Springer, 2007, pp. 63-64.
[5] N. Nisan, Algorithmic game theory. Cambridge: Cambridge University Press, 2011, pp. 3-4.
[6] M. Osborne, An introduction to game theory by Martin J. Osborne. Oxford: Oxford University Press, 2004, p. 32.
[7] J. Webb, Game theory. [London]: Springer, 2007, pp. 89-91.
[8] M. Osborne, An introduction to game theory by Martin J. Osborne. Oxford: Oxford University Press, 2004, pp. 184-185.
[9] Smith, Andrew, Mayo, Jackson, Kammler, Vivian, Armstrong, Robert, Vorobeychik, Yevgeniy. (2017). Using computational game theory to guide verification and security in hardware designs. 110-115.
[10] W. Saad, A. Sanjab, Y. Wang, C. A. Kamhoua and K. A. Kwiat, "Hardware Trojan Detection Game: A Prospect-Theoretic Approach," in IEEE Transactions on Vehicular Technology, vol. 66, no. 9, pp. 7697-7710, Sept. 2017, doi: 10.1109/TVT.2017.2686853.