Detecting the adherence of driving rules in an energy-efficient, safe and adaptive driving system

Emre Yay [a,*], Natividad Martínez Madrid [a], Juan Antonio Ortega Ramírez [b]

[a] School of Informatics, Reutlingen University, Alteburgstr. 150, 72762 Reutlingen, Germany
[b] Computer Languages and Systems Dept., University of Seville, 41012 Seville, Spain

Keywords:

Rule matching algorithm
Driving system
Energy-efficiency Safety

A B S T R A C T

An adaptive and rule-based driving system is being developed that tries to improve the driving behavior in terms of the energy-efficiency and safety by giving recommendations. Therefore, the driving system has to monitor the adherence of driving rules by matching the rules to the driving behavior. However, existing rule matching algorithms are not sufficient, as the data within a driving system is changing frequently. In this paper a rule matching algorithm is introduced that is able to handle frequently changing data within the context of the driving system. 15 journeys were used to evaluate the performance of the rule matching algorithms. The results showed that the introduced algorithm outperforms existing algorithms in the context of the driving system. Thus, the introduced algorithm is suited for matching frequently changing data against rules with a higher performance, why it will be used in the driving system for the detection of broken energy-efficiency or safety-relevant driving rules.

1. Introduction

When saving energy and protecting the environment became fundamental for politics and society, several laws were enacted to reduce the greenhouse gas emissions, like the CO_2 limitations, for passenger cars in the European Union. Another factor that became important during the past few decades is road safety, as the increasing number of cars led to more accidents and fatalities on the road. On the basis of the enacted laws with the goal to save the environment (Wessellink, Harmsen, & Eichhammer, 2010) and the increasing importance of road safety, car manufacturers are trying to optimize the car and its individual parts like the car body, the engine, or the power train. Furthermore, new methods are being invented to increase the energy-efficiency and safety of cars, like the regenerative brake (Patil, 2012), which converts the kinetic energy during a brake application to electric energy or the anti-lock brake system (Burton, Delaney, Newstead, Logan, & Fildes, 2004), which improves safety by preventing the wheels from locking up.

Besides the optimization of the car itself, there is also the potential to increase the energy-efficiency and improve road safety by adapting the individual driving behavior to the given driving situation. The studies (Xiaoqiu, Jinzhang, & Guoqiang, 2011) and (Chin & Quek, 1997) revealed that the driving behavior has an effect on road safety. This has also been verified in the accidents report of the German Statistical Office (German Statistical Office, 2014), which showed that 86% of the accidents with damage to persons in Germany in 2013 happened because of driver mistakes, see Fig. 1. Several studies have shown that energy savings up to 30% are possible with the adaptation of the driving behavior (Haworth & Symmons, 2001; Helms, Lambrecht, & Hanusch, 2010; van Mierlo, Maggetto, van de Burgwal, & Gense, 2004). However, according to Bongard (2007) this 30% savings is only possible with experienced drivers. The energy savings vary depending on the given driving practices. In short-term driving practices, like energy-efficiency training or contests, an energy saving of about 24% is possible as shown in the tests conducted by the car manufacturer Ford (Spencer, 2008). In contrast, Barkenbus (2009) calculates the energy savings in sustained driving practices at 5% when the drivers have no continuous energy-efficiency related feedback after the initial training. Continuous feedback is defined by showing the driver energy-efficient relevant driving recommendations every day. However, with continuous feedback, the energy savings is 10%.

There are already driving systems and methods which try to reduce energy consumption or to improve road safety at the vehicle level, such as the driving systems of Khayyam, Nahavandi, and Davis (2012) and Milanes, Perez, Godoy, and Onieva (2012) and the method of Jo, Lee, Park, Kim, and Kim (2014). Furthermore, there are also driving systems, such as those of Fiat (2010), Cho (2008) or Lotan and Toledo (2006), which focus on improving the driving behavior

* Corresponding author. Tel.: +49 71212714063.
E-mail addresses: emre.yay@reutlingen-university.de, yayemre@aol.com (E. Yay), natividad.martinez@reutlingen-university.de (N. Martínez Madrid), jortega@us.es (J.A. Ortega Ramírez).

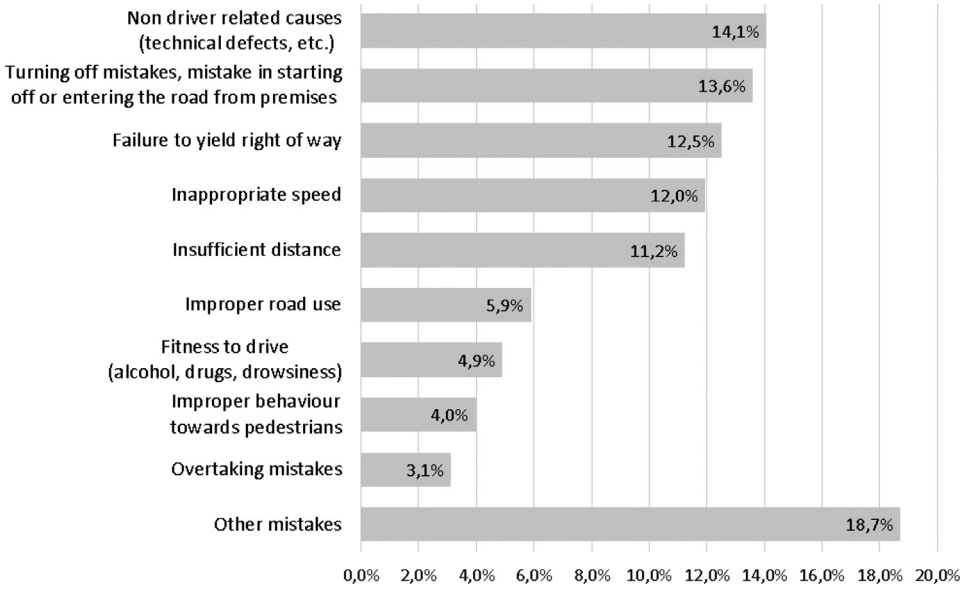

Fig. 1. Causes of accidents involving personal injury 2013 in road traffic, in German Statistical Office (2014).

in terms of energy-efficiency or safety. However, these driving systems only cover either the area of energy-efficiency or safety and provide insufficient feedback to the driver, for example by showing a green lamp when the driver is driving in an energy-efficient manner. Furthermore, they do not adapt to the individual driver, by considering the driver condition or the individual driving behavior. In contrast, the adaptive and rule-based driving system that is in development has the goal to improve the driving behavior in terms of energy-efficiency and safety by giving customized recommendations to the driver during a journey. Thus, according to Barkenbus (2009), this can lead to an increase in the energy-efficiency of up to 10%. The driving system adapts to the driver by customizing the recommendations on the basis of his or her reaction to the previously given recommendations and the driver condition. For example, the driving system creates no recommendation when the driver is under stress. Furthermore, it decreases the generation frequency of a recommendation when the driver has ignored a recommendation several times. The energy-efficiency and safe driving behavior is described by a set of rules. These rules are used to check if the driver is driving efficiently and safely. If the breaking of a driving rule is detected, an energy-efficient or safety-relevant recommendation is shown to the driver.

Rule matching algorithms, such as Rete (Forgy, 1982) or Treat (Miranker, 1987), are designed to check a certain data set against rules, making them suitable for checking driving rules against driving behavior in order to find an energy-inefficient or unsafe driving behavior. However, existing rule matching algorithms, like Rete or Treat, are not designed to handle frequently changing data like in a driving system, which is why the performance of these algorithms is not ideal for their use in driving systems. There are already improved versions of the Rete algorithm, like the algorithm of Li, Liu, Cao, Yin, and Yao (2016). They created an algorithm based on Rete that allows to subtask the task of rule matching to different computers in a distributed or parallel computing environment. Therefore, the algorithm decomposes rules to sub rules and distributes them to a distributed environment for parallel match, after which the algorithm merges the results through a reduce function. According to Li et al., this approach improves the match efficiency for massive rules matching. Another modified Rete algorithm is introduced in Liu, Huang, Zhang, and Du (2014), who try to improve the efficiency of the Rete algorithm for the area of social insurance audit using the active hash method. The active hash method allows to improve the search time for a node within the network by managing the nodes and the corresponding facts within a hash table. The experimental results showed that the active hash method improves the efficiency of the Rete algorithm. Rete+ is another improved Rete algorithm introduced by Gao, Qiu, and He (2013) that has the focus on matching rules in the context of Web of Things environments, like smart homes. According to Gao et al. the Rete algorithm executes the same rules in the environment of Web of Things, why the Rete+ algorithm stores the index of the alpha nodes for the last executed rules. Each fact that is put into the working memory, it will be matched first with the alpha nodes whose index were stored. In case of an successful match, the beta nodes of the corresponding alpha nodes are updated. However, when the matching fails, the algorithm goes on to match the facts like in the original Rete algorithm. The experimental evaluation showed an higher performance of the Rete+ algorithm in comparison to the Rete algorithm in the context of smart home environment.

However, the presented improved Rete algorithms are designed for specific environments, like Web of Things, social insurance and so on. They are using parallel computing, active hash methods or storing the last used alpha nodes to optimize the Rete algorithm. However, these optimizations are not sufficient for the environment of the driving system, as the optimizations are fitted for a specific environment that is different from the environment of the driving system. Furthermore, the driving system has to match driving rules against data that is updated in near time. The goal of the rule matching algorithm introduced in this paper is to match driving rules to a data set with a higher performance than existing rule matching algorithm in the context of the driving system, in order to allow the driving system to show recommendations to the driver in time. An early stage of the introduced rule matching algorithm is explained in Yay, Martńez Madrid, and Ortega Ramírez (2014).

This article presents an adaptive driving system for energy-efficient and safe driving and focuses on the development of a rule matching algorithm that checks driving rules against the driving behavior to find an energy-inefficient or unsafe driving behavior. As the rule matching algorithm is in the context of a driving system, its architecture is explained in Section 2. Available rule matching algorithms, like Rete, Treat or Leaps, are described in Section 3. Section 4 introduces the concepts and the algorithm of the developed rule matching algorithm. The evaluation of the developed rule matching

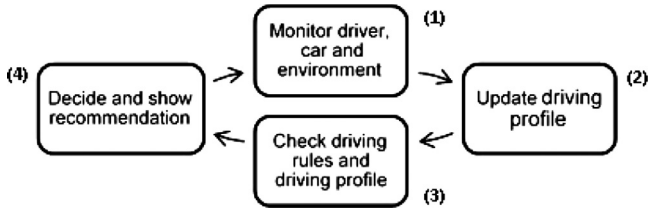

Fig. 2. Cycle of the driving system.

algorithm, on the basis of 15 journeys, is described in Section 5. The corresponding results of the evaluation are presented in Section 6. Finally, the conclusion and further work based on this proposal will be discussed in Section 7.

2. Adaptive and rule-based driving system

This section starts with an explanation of the driving system cycle, which shows the steps for creating a recommendation. The driving rules are explained in Section 2.2. Finally, an overview of the driving system architecture is given, including the explanation of the layers and module that are relevant for the matching of the driving rules to the driving behavior.

2.1. Adaptive and rule-based driving system cycle

The adaptive and rule-based driving system has four main tasks (Fig. 2). The driving system first monitors the driver, the car, and the environment (1) using vital sensors, in-vehicle sensors, and other additionally attached sensors, for example to get information about the weather. Furthermore, it can be extended, for example to receive location-based information using the internet or GPS. On the basis of the collected information, a driving profile is generated or updated, describing the typical driving behavior of the driver (2). After updating the driving profile, the collected information is used to check whether an energy-efficient and safety relevant driving rule is broken (3). Furthermore, the driving profile is compared to the current driving behavior as well, to indicate if the current driving behavior deviates significantly from the typical driving behavior. Upon recognition of any breaking of the driving rules or deviations from the typical driving behavior, the driving system decides whether or not to show a recommendation on the in-vehicle display unit (4). This decision is dependent on the individual driving behavior and the condition of the driver. For example, if the driver is under stress or ignores a recommendation repeatedly, the driving system will not show the corresponding recommendation to the driver. Thus, it prevents the driver from becoming mentally overstimulated, as this can cause distraction and lead to accidents (Brookhuis & de Waard, 2010). Furthermore, by suppressing repeatedly ignored recommendations, the driving system is more likely to be accepted by the driver.

2.2. Driving rules

The energy-efficient and safe driving behavior is described by a set of driving rules that are the basis of the adaptive and rule-based driving system. According to Barkenbus (2009), energy-efficient driving behavior involves such things as smooth acceleration between 2000 and 2500 revolutions, anticipating the traffic flow and signals, avoiding sudden starts and stops, driving below the speed limit, maintaining an even driving pace and eliminating excessive idling. van Mierlo et al. (2004) evaluated three energy-efficient relevant driving rules using 24 drivers who had to practice the energy-efficient driving rules. The driving rules used in the evaluation were to (1) shift as soon as possible at a maximum of 2500 revolutions, to (2) press the throttle quickly to keep up with traffic and to (3) shift down as late as possible to a lower gear to keep the car rolling in the highest gear without disengaging the clutch. The result of the evaluation showed that the correct interpretation of the driving rules decreased the energy-consumption of the vehicles by between 5% and 25%. However, most drivers had problems applying the driving rules (1) and (2), as these driving rules were contradictory to them. Thus, the drivers ignored the driving rule (2). According to the findings of Van Mierlo, the optimum in energy-efficiency can be achieved using the driving rules (1) and (3). Furthermore, adherence to the driving rules leads to a reduced driving speed and thus to an improvement in road safety (Haworth & Symmons, 2001).

Table 1
An excerpt of the derived driving rules.

Relevance	Driving rule
Safety relevant	Keep enough distance to the preceding car (minimum distance is equivalent to distance traveled by a vehicle in two seconds or the half of the speed in meters)
Safety relevant	Adapt your speed to the given situation and do not exceed the speed limit
Safety relevant	Avoid any distractions (i.e. do not use the mobile phone during the journey)
Energy-efficiency relevant	Shift as soon as possible at a maximum of 2500 r/min (2000 r/min for a diesel)
Energy-efficiency relevant	Drive at a steady speed using the highest possible gear to keep the engine speed down
Energy-efficiency relevant	Turn off the engine when the engine idles longer than one minute

Besides the energy-efficient driving rules, there are also safety relevant driving rules that prevent aggressive driving behavior and therefore improve road safety. According to UNECE – United Nations Economic Commission for Europe (2004), aggressive driving behavior includes amongst others speeding or driving too close to the car in front. Furthermore, the German Statistical Office (2014) showed that about 12% of the accidents happened because of speeding or inadequate speed in different driving situations, and about 11% of the accidents were caused by insufficient distance from the car in front. Additionally, the New Zealand Transport Agency (2007) defined driving rules to prevent aggressive driving behavior and to improve road safety by avoiding speeding, distraction, and fatigue. On the basis of these facts, the above driving rules were derived and used in the adaptive and rule-based driving system to analyze the driving behavior and to generate recommendations in terms of energy-efficiency and safety. Table 1 shows an excerpt of the driving rules.

2.3. Architecture

The adaptive and rule-based driving system is separated into three layers (Fig. 3): the data layer, processing layer, and graphical layer. The data layer gathers the data from the car, the driver, and the environment using the interface module, which passes the collected data to the data aggregation and the profile update module. The aggregated data is then passed from the data aggregation module to the profile update module, as well. The profile update module updates the driving profile of the driver using the collected data from the interface module and the aggregated data. The driving profile is stored in the mid-term knowledge base and represents the driver's typical driving behavior. It stores, for example, information about the average driving speed or the average stress level of the driver. The long-term knowledge base, which is also placed in the data layer, consists of energy-efficient driving rules and information about the car. The information about the car is used, for example, to show the driver the current energy consumption and the consumption when the driver adheres the recommendations. Finally, when the data is processed within the modules data aggregation and profile update, it is stored in the working data of the short-term knowledge base.

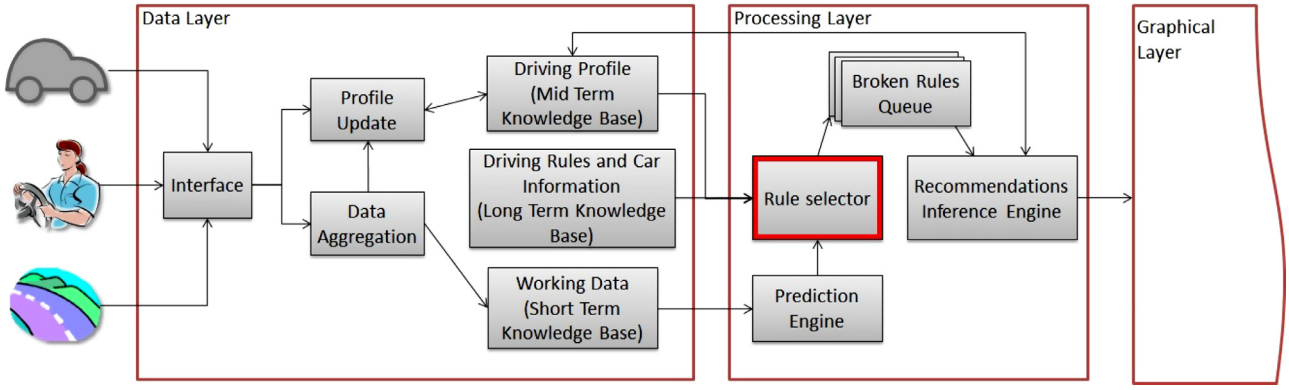

Fig. 3. The architecture of the driving system including the modules of the data and processing layer. The rule selector module (marked red) is responsible for detecting broken driving rules. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article).

As the main focus of this paper is on the detection of the broken driving rules using a rule matching algorithm within the rule selector module (marked red in Fig. 3), the processing layer of the adaptive and rule-based driving system will be briefly described in the following sections.

2.3.1. Processing layer

The processing layer (Fig. 4) is responsible for analyzing the current driving behavior in terms of energy-efficiency and safety and giving customized recommendations on the basis of the driving rules and the typical driving behavior of the driver. The first step of the processing layer is done in the prediction engine module that gets the data from the working memory, which is placed in the short-term knowledge base of the data layer. On the basis of the gathered data, the prediction engine module starts to predict the state of the car using the autoregressive moving average algorithm (Yay & Martńez Madrid, 2013). The prediction of the car's state allows the driving system to generate recommendations before a driving rule is broken. The predicted information is passed along with the data from the working memory to the rule selector module.

On the basis of the data passed from the prediction engine module, the rule selector module (Fig. 3) detects if a driving rule is broken or if the current driving behavior differs from the typical driving behavior. The developed algorithm is described in Section 4. The detected broken driving rule or deviation from the typical driving behavior is pushed into the broken rules queue, which is processed by the inference engine module according to the first in, first out principle. However, if no broken driving rule or deviation from the typical driving behavior is found, the processing layer gets new data from the working memory. The recommendations inference engine module decides whether to show a recommendation to the driver, on the basis of the drivers condition and reaction to previously given recommendations. The recommendations inference engine module will also reduce the frequency of a recommendation, when the driver repeatedly ignores a driving rule. This makes the driver more likely to accept the driving system, he or she will not be bothered by recommendations that her or she does not find relevant. Recommendations that are not suppressed are passed to the Graphical Layer, which displays them, for example, on the in-vehicle display unit.

3. Rule matching algorithms

The rule selector module of the processing layer (Fig. 3) detects broken driving rules and deviations from the typical driving behavior using a rule matching algorithm. Rule matching algorithms are pattern matching algorithms that match rules to a data set. They are often used in production systems to determine which of the production system rules have to be fired on the basis of the data stored in the working memory of the production system. The rule matching algorithms receive information about the changes made to the working memory of the production system and determine the changes that have to be made in the conflict set. The conflict set contains all rules that have to be fired. As there is a conflict, as to which rules should be fired first, the conflict set has to be solved using a conflict set resolution strategy, such as first come, first serve or the prioritization of the rules. Regarding the introduced driving system, the conflict set is the broken rules queue module of the processing layer (Fig. 3), which is solved by the recommendations inference engine using the first come, first serve principle. The rule matching algorithms compare productions (rules) against data tuples (facts). In the presented driving system, the rules are represented by the driving rules and the facts are stored in the working memory of the short-term knowledge base. The rules are described by conditions and consequences. For example, a condition of a rule is Rpm $>$ 2500 and a consequence is to show the recommendation to shift the gear. The rule is passed to the conflict set when the facts match the conditions of that rule. According to the conflict set resolution strategy, the consequence of the rule is fired when the conflict set is solved. There are several rule matching algorithms such as Rete, Treat, or Leaps, which are briefly described in the following section.

The Rete (Latin for net) algorithm (Forgy, 1982) uses a tree structured network, also called rete network, to represent the rules, see Fig. 5, whereby every rule has its own rete network. A rete network contains alpha and beta nodes, where each alpha node represents one condition of a rule and stores the fact that matched its condition. The beta nodes are used to store partial matches when different facts are joined from the nodes. Upon every update of a fact stored in the working memory, the old fact stored in the corresponding alpha or beta node memory is deleted. The updated fact is then pushed into the Rete network, which passes the fact to the corresponding alpha node. The alpha node then checks whether the new fact satisfies the node condition. The fact that satisfies the condition is stored in the alpha memory and is passed to the beta node. The beta node represents the joining of the alpha nodes and checks whether the joining between the alpha nodes is satisfied on the basis of the newly received fact. If the beta node is satisfied, the rule is put to the conflict set. The conflict set resolution strategy of the Rete algorithm has to be defined. Fig. 5 illustrates the data flow of the Rete algorithm, in which the Rete network of the rule shift as soon as possible ($Rule_1$) with the conditions RPM $>$ 2500 and Gear $<$ 6 is shown. The initial working memory, rete network, and conflict set are empty. However, the facts Rpm 3000 and Gear 3 are added to the working memory in the next cycle. The working memory passes the new facts to the rete network using an add operation. The root node of the rete network hands the incoming facts over to the corresponding alpha nodes that check if the facts satisfy their conditions and then stores them in their

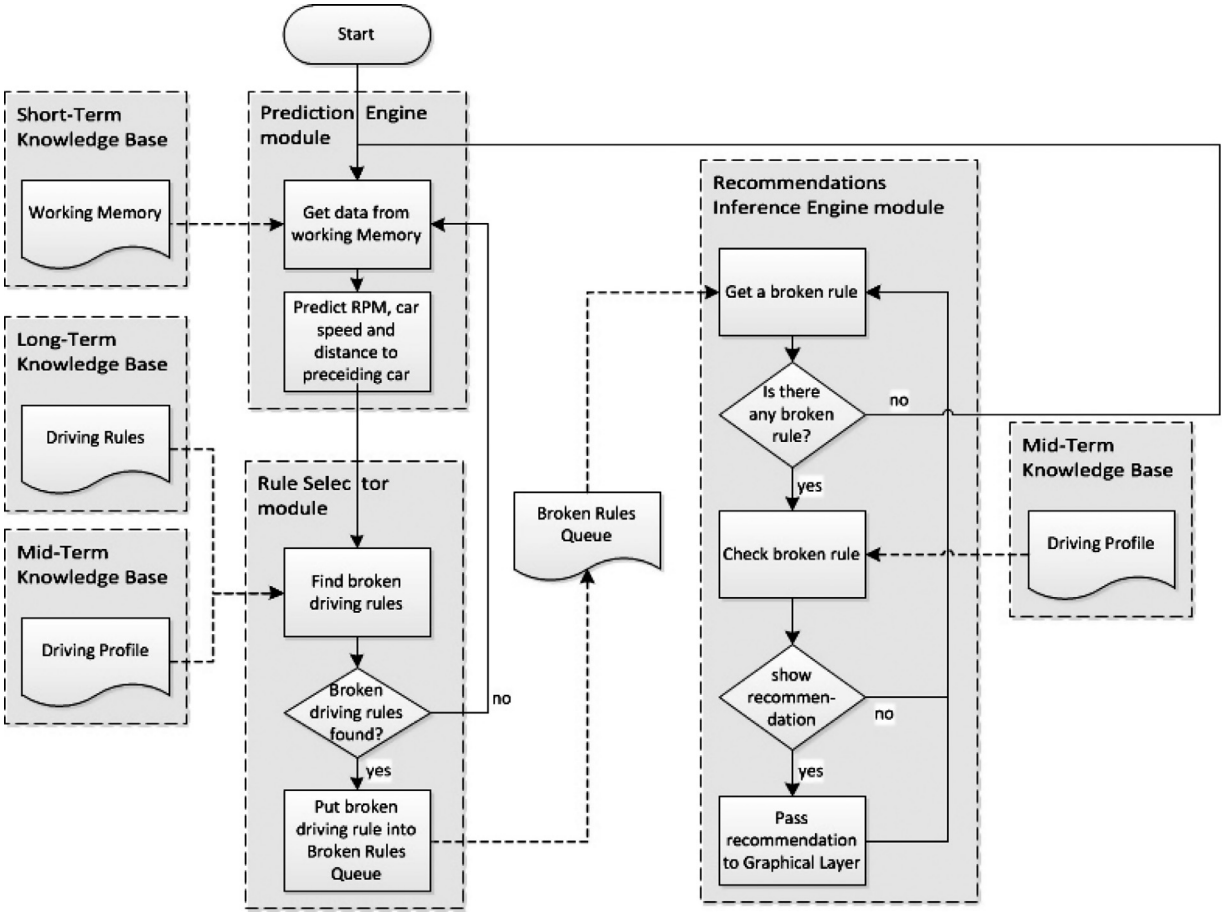

Fig. 4. Data flow of the processing layer.

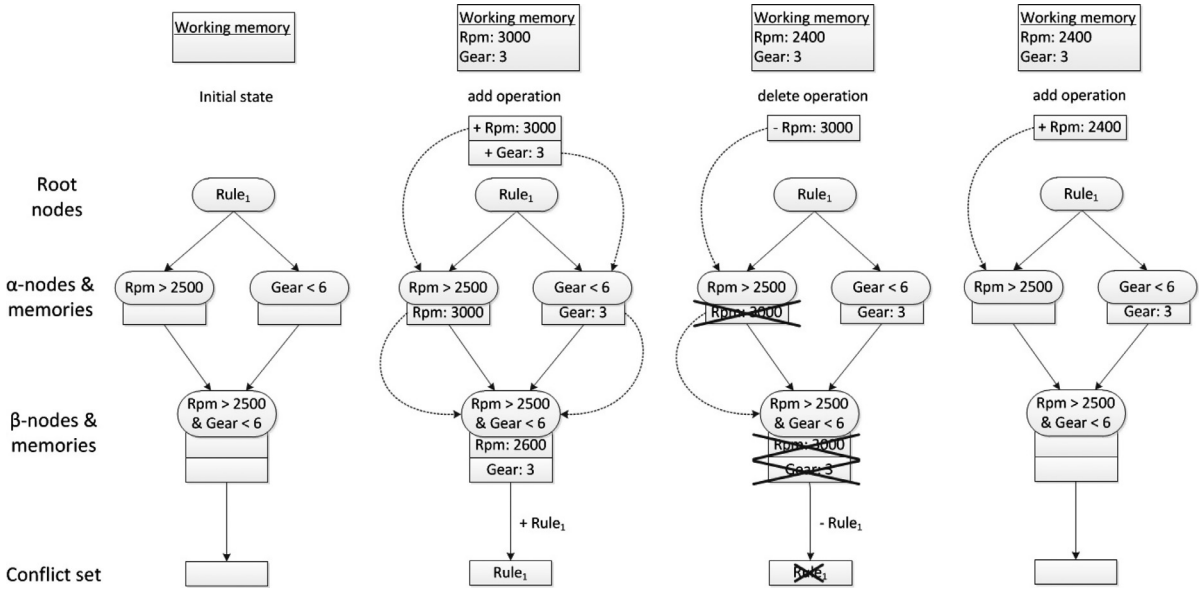

Fig. 5. Illustration of the Rete algorithm with the operations add and delete.

alpha memory. The facts are then passed to the beta node that checks if they satisfy the joining of the two alpha nodes. If they do, the facts are stored in the beta memory and the $Rule_1$ is put in the conflict set. In the next cycle of Fig. 5, the fact Rpm 3000 is updated to Rpm 2400. This update causes a delete operation of the old fact Rpm 3000 in the alpha memory and a removal of the facts in the beta node memory. An add operation passes the new fact Rpm 2400 to the Rete network. As the Rpm 2400 does not satisfy the condition Rpm $>$ 2500 of the alpha node, it is not stored in the alpha memory and is not passed to the beta node. Thus, the $Rule_1$ is not put into the conflict set again.

According to Wright and Marshall (2003) the worst-case space complexity of Rete is $O(W^{n-1})$, where W is the number of working

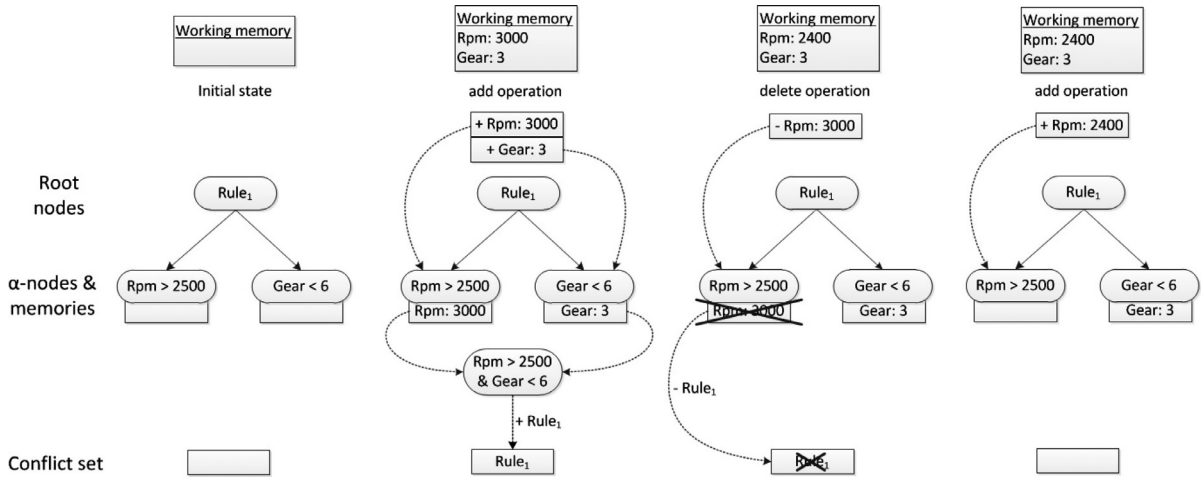

Fig. 6. Illustration of the Treat algorithm with the operations add and delete.

memory elements and n is the number of rule conditions. The matching of a rule in Rete comprises the operations add and delete. The add operation is used when a new fact is put into the working memory. The computational costs of an addition in Rete is represented by the equation $R(n)$, that is according to Wright and Marshall (2003)

$$R(n) = \frac{1}{n+1}\sum_{i=0}^{n} R(i,n)$$

where i is the alpha node that is entered by the fact in a network with $n+1$ alpha nodes. $R(m, n)$ is defined as

$$R(m,n) = \begin{cases} K(n) & m \leq 1 \\ \sum_{i=m}^{n} \frac{a_0 a_i}{a_m} \prod_{j=1}^{i-1} a_j pj & otherwise \end{cases}$$

where $K(n)$ represents the full computation of the Rete network including the comparisons of the facts against the alpha nodes and the computation of the intermediate relations of the alpha nodes when the Rete network is entered at the alpha node a_0 or a_1. Otherwise the cached state of the Rete network is used. a_0 is the first alpha node, a_i is the ith alpha node and a_m the entering alpha node. a_j represents the jth alpha node, whereas the matching probability of the fact that is stored in the beta node is p_j. $K(n)$ is

$$K(n) = \begin{cases} a_1 & n = 1 \\ \sum_{i=1}^{n-1} a_j + 1 \prod_{j=0}^{i-1} a_j + 1 p_j + 1 + a_1 & otherwise \end{cases}$$

where $n > 0$ and $p_0 = 0$. $p_j + 1$ is the constant matching probability of a fact stored in the jth beta node. The delete operation of the Rete algorithm is represented by the equation $R(n) + S(n)$ as a delete operation causes a recomputation of the alpha and beta nodes in the Rete network. The cost of the additional search of a beta node for deleting a fact is represented by $S(n)$ that is

$$S(n) = \frac{1}{n+1}\sum_{i=0}^{n} S(i,n)$$

where i is the alpha node that is entered by the fact in a network with $n+1$ alpha nodes. $S(m, n)$ represents the search of a beta node within an entry point m.

$$S(m,n) = \sum_{i=m}^{n} a_0 \prod_{j=1}^{i} a_i p_j$$

As the update of a working memory element causes a delete and an add operation in Rete, an update of an working memory element can be described in the Rete algorithm as $2S(n) + R(n)$.

The Treat algorithm (Miranker, 1987) is based on the Rete algorithm. It improves Rete, for example, in the memory usage, as it does not use beta nodes to join the alpha nodes and does not use a beta memory to store the facts that satisfied the joining of the alpha nodes. Instead, the satisfaction of the joining is checked when required. When an incoming fact satisfies an alpha node, the fact is stored in the alpha memory and the joining of the alpha node is recomputed. If the result of the computation is positive, which means that the facts satisfied the conditions of the alpha nodes, the rule is added into the conflict set. Fig. 6 illustrates the process on the basis of the driving rule shift as soon as possible $Rule_1$ with the conditions RPM > 2500 and Gear < 6. The driving rule is represented in the Treat algorithm also by a Rete network, however, without the beta-nodes. The initial working memory contains the facts 3000 rpm and 3rd gear, in which the revolutions per minute is updated to 2600 rpm. This causes the Treat algorithm to delete the old fact stored in the alpha memory and also to delete $Rule_1$ directly from the conflict set. During the add operation the new fact is passed to the corresponding alpha node, which checks if the fact satisfies the alpha node condition. On satisfaction of the condition, the intermediate relations of the alpha nodes are recomputed, which causes, in our example, an insertion of $Rule_1$ into the conflict set.

The worst-case space complexity of the Treat algorithm is $O(W)$, where W is the number of elements in the working memory (Wright & Marshall, 2003). In contrast, to Rete the add operation in Treat is defined as $K(n)$, as Treat does not use beta nodes to store the intermediate relations of the alpha nodes. Thus, the intermediate relations of the alpha nodes are computed when needed. The delete operation within the Treat algorithm requires the deletion of the fact from its corresponding alpha node. According to Wright and Marshall (2003), the delete operation is represented by $D(n)$ that is

$$D(n) = \frac{1}{n+1}\sum_{i=0}^{n} a_i$$

An update of a working memory element in Treat is similar to Rete, it causes a delete and an add operation within the Treat algorithm. Thus, an update of a working memory element can be described as $K(n) + D(n)$.

The conflict set resolution strategy of the Treat algorithm must be defined, similarly to the conflict set resolution strategy in the Rete algorithm. According to the results of Miranker (1987), the Treat algorithm is more effective than the Rete algorithm, as it needed fewer comparisons until it bound the facts to the corresponding nodes and needed less memory due to the missing beta nodes and beta memories. Moreover, during a deletion of a fact, Treat manipulates the

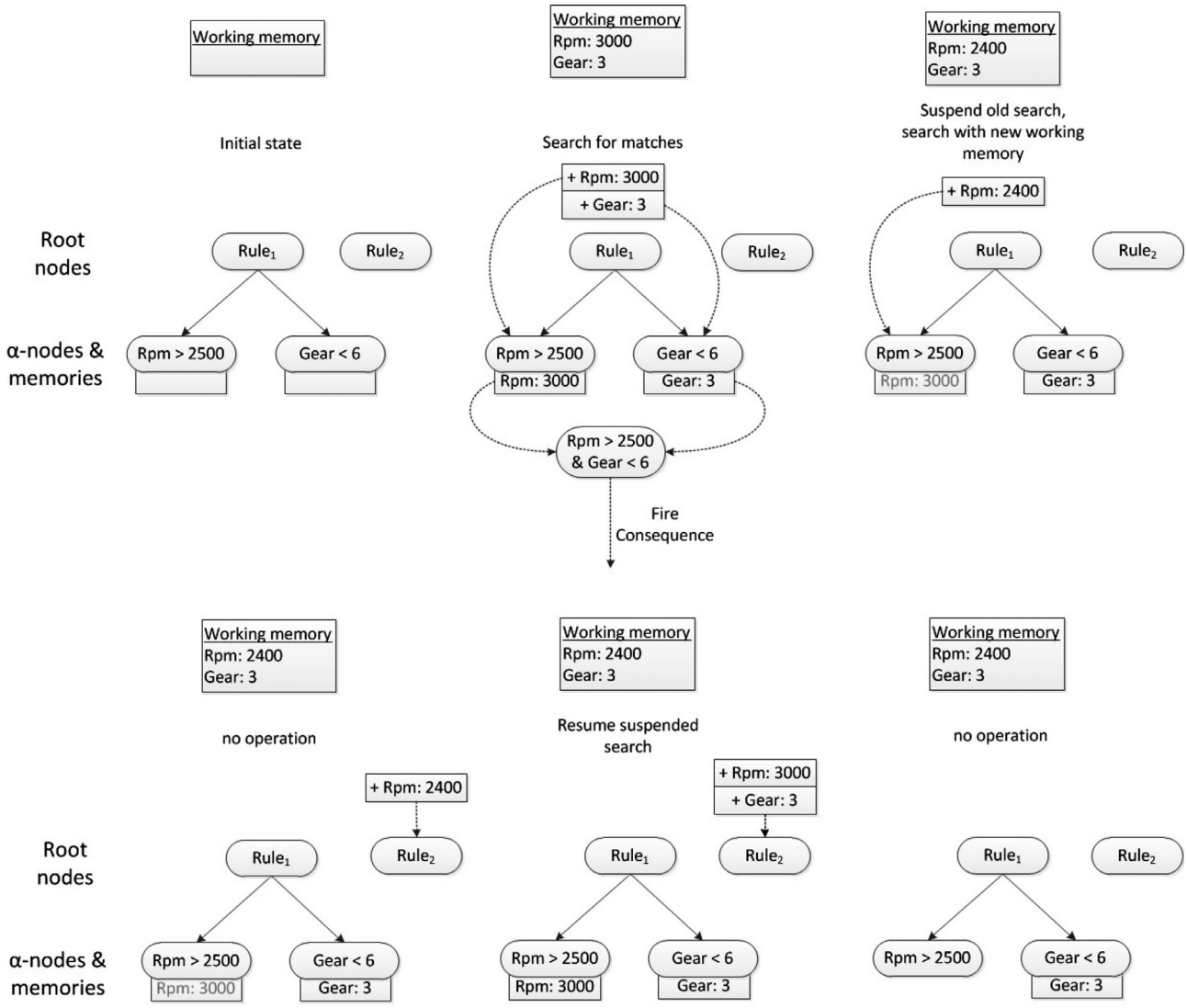

Fig. 7. Illustration of the Leaps algorithm when a new fact is added.

alpha nodes and the conflict set directly, instead of recomputing the joinings of the alpha nodes. In contrast, Rete has to recompute the joinings of the alpha nodes when a fact of the working memory is deleted, to keep the beta nodes up to date. However, Nayak, Gupta, and Rosenbloom (1993) showed in their evaluation that the Rete algorithm outperforms the Treat algorithm, especially when it is used in static structures, as the Rete algorithm joins the alpha nodes in static structures once, instead of recomputing the joinings every time. Nayak et al. define a structure as static when a single fact is not removed from the working memory, thus Rete has to compute every joining only when the whole structure is removed or added. According to Nayak et al. the results of their evaluation differ from the results of Miranker (1987), because Miranker counted only the number of comparisons, which may not reflect the intrinsic differences between the match algorithms (Nayak et al., 1993).

The Leaps (Lazy Evaluation Algorithm for Production Systems) algorithm (Miranker, 1990) is a rule matching algorithm based on the Treat algorithm. Its space complexity is according to Miranker et al. $O(max(ts) * c)$. The Leaps algorithm uses alpha nodes to store the facts and calculates joinings of the alpha nodes when they are needed. In contrast to the presented rule matching algorithms Rete and Treat, Leaps also considers the conflict set resolution strategy by combining the conflict set resolution strategy with the search for rule activations. Thus, Leaps does not have a conflict set. Instead, Leaps fires the consequence of a rule immediately when the rule conditions are satisfied. Leaps is using lazy evaluation to find rules whose alpha node conditions are satisfied by inspecting the facts of that rule one by one. If a rule is found whose alpha node conditions are satisfied, it pauses the current search and fires the rule consequence. Fig. 7 illustrates one cycle of the Leaps algorithm with two rules. The driving rule "shift as soon as possible" ($Rule_1$) with the conditions RPM $>$ 2500 and Gear $<$ 6 is used along with the rudimentary $Rule_2$, which is illustrated only for demonstrating the processing of Leaps.

The driving rules are represented in the Leaps algorithm like the Rete network without the beta nodes, as beta nodes are not used in Leaps. The initial working memory is empty, which is why the alpha memories of $Rule_1$ are empty. The working memory is then updated with the facts 3000 rpm and 3rd gear. Leaps now starts to search for the rule by checking the conditions of $Rule_1$. As the conditions as well as the joining of the alpha nodes are satisfied by the facts, the consequence of $Rule_1$ is fired immediately. After the firing of $Rule_1$, the working memory is updated with the fact revolution per minute 2400 rpm. As the search with the fact 3000 rpm caused a firing of a rule, the search is suspended and Leaps starts a new search using the updated fact of 2400 rpm. However, if the old search did not fire a rule, it would not be suspended. Instead, the updated fact would be pushed

onto a stack until the old search is finished. After the finished search, the updated fact would be popped from the stack and Leaps would start the search with the new fact. As the old search is suspended, the facts stored in the alpha memories are not deleted, but instead ignored in the new search. Leaps now carries on to search the rules $Rule_1$ and $Rule_2$ using the updated working memory. When the search is finished, Leaps resumes the suspended search with the old fact until all rules are checked. Finally, when all rules are checked with the old fact that is not in the working memory anymore, the old fact is deleted from the alpha memories of all rules. According to the results of Miranker (1990), the lazy evaluation of the rules allows Leaps to increase the rule firing rates and decrease the execution time. This can be achieved because Leaps avoids the computation of all rules in each cycle like Rete or Treat does. Instead, Leaps suspends the current search when a rule is fired and carries on to search the rules with the new fact.

The Treat algorithm tries to improve Rete by omitting the beta memories. Thus, it does not have to recompute the joinings of the alpha nodes on every change of the working memory. Instead, it recomputes the joinings only when necessary. However, according to Nayak et al. (1993) the Rete algorithm outperforms the Treat algorithm, especially in static structures. As the facts within the working memory of the introduced driving system are updated instead of deleted or added, the working memory of the introduced driving system has a static structure. However, updating the working memory is not considered in the Treat and Rete algorithms, which is why an update of the working memory results in deleting and adding the facts within these algorithms. Due to this fact, the Treat and Rete algorithms are not the ideal rule matching algorithms for the driving system. The evolution of the Treat is the Leaps algorithm, which combines the lazy rule matching with the solving of the conflict set. This allows the Leaps algorithm a faster firing of the rule consequences. However, this also makes the Leaps algorithm inflexible, as the conflict set resolution strategy cannot be changed. The conflict set of the introduced driving system, called the broken rules queue, is solved by the recommendations inference engine, which processes the rules within the broken rules queue according to the first in, first out principle. Each rule is checked by the recommendations inference engine for whether the firing of the consequence has to be suppressed, as the driver may be under stress. Due to this fact, the Leaps algorithm is also not the ideal solution for matching the rules against the facts in the driving system, because it fires the rules instantly without the option to suppress the recommendations. On the basis of the elaborated findings, a rule matching algorithm for the usage within the driving system will be created. The Rete algorithm will be the basis for the improved rule matching algorithm because it is designed to work in static structures. In the next section, the improved rule matching algorithm will be explained in detail.

4. Improved rule matching algorithm

The Rete algorithm is developed for environments with static structures. According to Nayak et al. (1993) static structures are defined as facts in a working memory which are not removed. Thus, the introduced driving system has a static structure, as the facts (the measured values from the car) stored in the working memory are not removed. Instead, they are updated with a frequency of 100Hz. However, an update operation using the Rete or Treat algorithms consists of deleting the old fact from the Rete network and adding the new fact. The delete and add operations are shown in Figs. 5 and 6. The usage of the Rete or Treat algorithms in the driving system would cause massive delete and write operations within the used network, as they would have to update the facts with a frequency of 100 Hz. To avoid the write and delete operations upon every update of the facts and therefore, to improve the performance, a rule matching algorithm has been created on the basis of the Rete algorithm, as it is designed to work in static structures. Instead of storing the fact within the node memories, the improved rule matching algorithm stores pointers to the corresponding facts within the working memory. Thus, upon every update of the facts within the working memory, the improved Rete network does not have to be updated. Instead, the network is only triggered to check whether the updated facts satisfy the node conditions. Listing 1 contains an abstract description of the improved rule matching algorithm. Upon every update of the facts within the working memory, every improved rete network is triggered to check its alpha nodes with the updated facts. If the result of the check differs from the previous result stored within the corresponding node memory, the old result is deleted and the new result is put in the alpha memory. If there is a beta node, the result is passed to that beta node, which checks first if both parents are updated with the new facts. Then, the beta node starts to check the intermediate relation between the alpha nodes. If the result of the check differs from the stored value in the beta node memory, the new result is stored and the old result is deleted from the beta node memory. If the beta node has another beta node, the result is passed to the child beta node. However, if there is no beta node, the conflict set is updated according to the result of the check. If the result of the check was positive, which means that the rule is broken, the rule is put into the conflict set. If the rule is not broken and the conflict set contains it, the rule is removed from the conflict set. The explained approach allows a faster processing of the rule matching algorithm, because it does not have to update the memory of every alpha or beta node using the fact of the working memory. Furthermore, the improved rule matching algorithm avoids storing redundant information in the alpha nodes, as the facts of the working memory are not stored within the node memories.

An example in Fig. 8 shows the improved Rete network and the update process of the improved rule matching algorithm using the driving rule shift as soon as possible ($Rule_1$) with the conditions RPM $>$ 2500 and Gear $<$ 6. The initial working memory in the example is empty as well as the memory of the alpha and beta nodes. The beta node points to the memory of the alpha nodes, which is needed to check whether the intermediate relation between the alpha nodes is satisfied. During the initial add operation, pointers to the facts in the working memory are passed to the corresponding alpha nodes. After the initialization of the pointers, the improved Rete network is triggered to check the conditions against the facts in the working memory according to the abstract algorithm in Listing 1. First, the alpha nodes check their condition against the fact. The result of the check is stored in the corresponding alpha node using a logical value. In the example, the facts stored in the working memory satisfy the condition of the alpha nodes, which then trigger the beta node to check the intermediate node relation. As the beta node checks the logical values of the parent nodes, which are both true, the conflict set is manipulated by adding $Rule_1$ into the conflict set. After the update of the fact RPM to 2400 rpm, the improved Rete network is triggered again to check the conditions within the network against the facts, instead of deleting the old and adding the new fact within the alpha node memories, like the Rete or Treat algorithm does. Thus, the alpha node with the condition RPM $>$ 2500 checks its condition against the corresponding fact in the working memory and stores the result, which is in the example false, in its node memory. Afterwards, it triggers the beta node to check if the intermediate relation of the alpha nodes is satisfied, using the values stored in the alpha node memories. In our example, the beta node memory consists of the value false after the update, as the intermediate relation of the alpha nodes is not satisfied. Thus, the beta node removes $Rule_1$ from the conflict set, as $Rule_1$ was in the conflict set before.

The improved rule matching algorithm is based on the Rete algorithm and uses therefore alpha and beta nodes for the rule matching. Thus, the space complexity of the improved rule matching algorithm is similar to the Rete algorithm: $O(W^{n-1})$, where W is the number of

```
For each rule do;
    For each alpha node in Rete Network do;
        Check condition of alpha node using the pointer to the fact stored in the working memory;
        If result of the checking differs from the stored logical value then;
            Store result in alpha node memory;
            If alpha node has a beta node then;
                Trigger child beta node to check the intermediate relation of the alpha nodes;
                If beta node has been triggered from both parents then;
                    Check the updated values of both parents using the stored pointers;
                    If result is true and the result differ from
                    the stored logical values then;
                        Store result in beta node memory;
                        If beta node has children then;
                            Trigger child beta node to check the intermediate relation in the same way
                            as current beta node;
                        Else;
                            Put rule into the conflict set;
                        End if;
                    Else if result of both values are false and the  result differs
                    from the stored logical values then;
                        Store result in beta node memory;
                        If beta node has children then;
                            Trigger child beta node to check the intermediate relation in the same way
                            as current beta node;
                        Else;
                            Remove rule from conflict set;
                        End if;
                    End if
                End if
            Else if alpha node condition is satisfied and rule is not in conflict set then;
                Put rule into the conflict set;
            End if;
            Else if alpha node condition is not satisfied and rule is in conflict set then;
                Remove rule from conflict set;
            End if;
        End if;
    End for;
End for;
```

Listing 1. Abstract improved rule matching algorithm, which shows the matching of the rule using the facts stored in the working memory.

working memory elements and n is the number of rule conditions. Table 2 shows the computational costs of the operations in the different rule matching algorithms. The computational cost of an add and a delete operation in the improved rule matching algorithm is the same as in the Rete algorithm, as the pointers have to be added or deleted from their corresponding nodes similarly to the Rete algorithm. In Rete and Treat, there is no update operation available, however, an update of a working memory element causes a delete and an add operation in Rete and Treat, why an update in Rete is $2R(n) + S(n)$ and in Treat $K(n) + D(n)$. The improved rule matching algorithm provides an update operation that is $R(n)$, as an update of an element in the working memory causes only the recomputation of the alpha and beta nodes in the improved rule matching algorithm. Thus, the performance of the improved rule matching algorithm should be higher than Rete and Treat in the driving system, as the data within the driving system is updated frequently by the sensors of the car, the driver and the environment.

The basis for the improved rule matching algorithm is the driving rule file (DRR), which contains the rules used for matching the facts within the improved rule matching algorithm. The antecedent and consequence of the rules are based on the driving rules listed in Table 1. On the basis of the description within the DRR files, the improved Rete network for the described rules is generated. The driving rule to shift the gear, when the rpm is higher than 2500 and the gear is not the maximum gear, which is 6 in the example, is defined by the following schema in the DRR file: #rule ecoRPM #when Rpm>2500 & Gear<6. The start of a rule in the DRR file is indicated by the tag #rule, which is followed by the name of the rule between quotation marks, ecoRPM in the example. The conditions of the rule are defined after the keyword #when and are separated by the character &. The conditions can be defined using terms, in which the facts of the working memory can be used by defining their names in the terms. In the example, the terms Rpm $>$ 2500 and Gear $<$ 6 are defined as the conditions of the rule, whereby the facts Rpm and Gear are used. The defined conditions are followed by the consequence of the rule, which is indicated by the key word #then. The consequence of the rule is defined in the Long Term Knowledge Base, which is placed in the Data Layer, and consists of the recommendation, which will be shown to the driver. It is possible to define multiple consequences in order to show the driver multiple recommendations.

On the basis of the defined rule, the improved rule matching algorithm creates the corresponding improved Rete network. The abstract algorithm for generating the improved Rete network is shown in Listing 2. For each rule in the DRR file, a root node for the rule is created. Then the algorithm parses the DRR file to find defined conditions for that rule. For any found condition, the improved rule matching algorithm creates an alpha node and sets it as a child of the root node. If another condition is found, another alpha node is created as a child of the root node. Furthermore, a beta node is created, which represents the intermediate relation of the created alpha nodes. In addition, pointers to the alpha node memories are stored in the beta node. After the generation of the improved Rete network, the improved rule matching algorithm starts to listen for an initial add of a fact in the working memory in order to pass the pointer to the corresponding alpha nodes and to match the facts against rules.

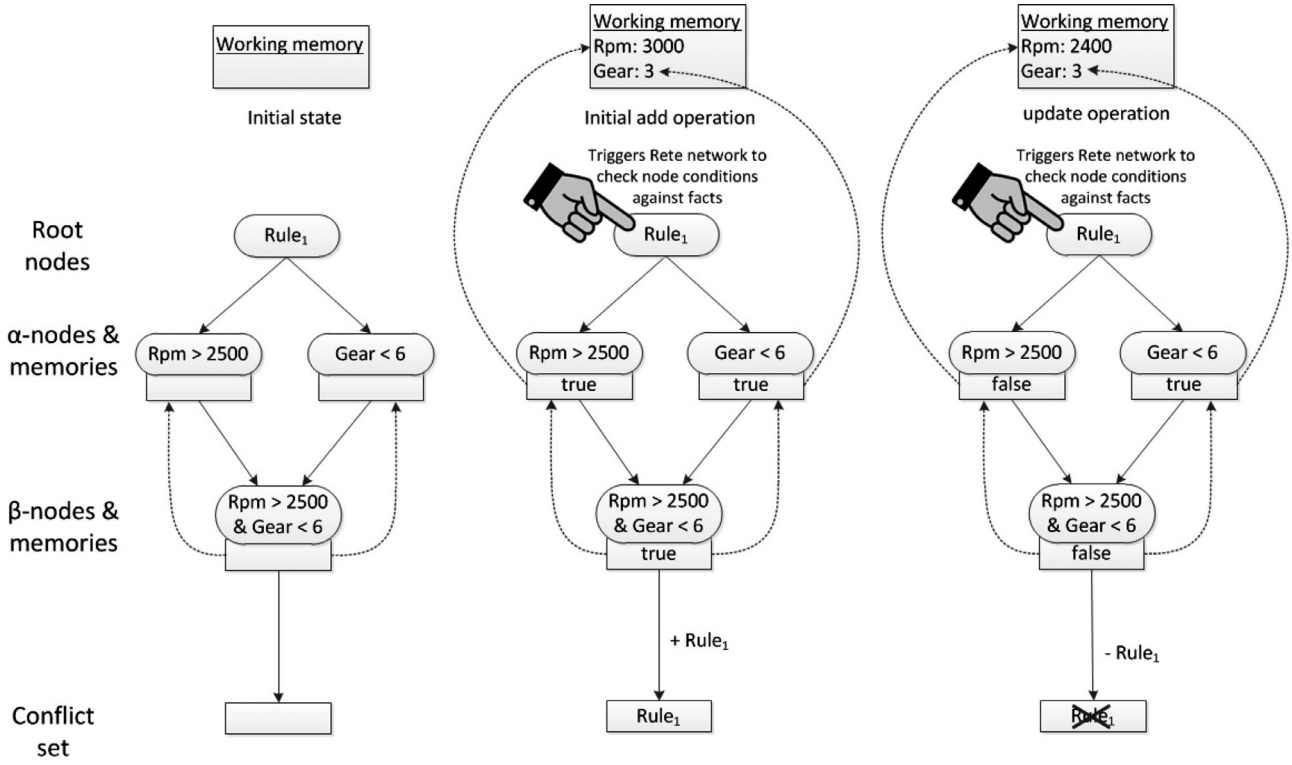

Fig. 8. The improved rule matching algorithm is using pointers to the facts and triggers the improved Rete network to check the conditions against the facts on every update of the working memory.

Table 2
The computation complexity of the operations add, delete and upate in the different rule matching algorithms.

Operation	Rete	Treat	Improved
Add	$R(n)$	$K(n)$	$R(n)$
Delete	$R(n) + S(n)$	$D(n)$	$R(n) + S(n)$
Update	$2R(n) + S(n)$	$K(n) + D(n)$	$R(n)$

An improved rule matching algorithm has been adapted on the basis of the Rete algorithm and optimized for the usage in the driving system, respectively in environments whose data is changing frequently. This has been achieved by adapting the alpha and beta nodes. Instead of storing all facts that satisfied the node condition within the node memories, the alpha nodes are pointing to the facts stored in the working memory and the beta nodes are pointing to the stored value in the alpha memory. The nodes store only the logical value that indicates if the node condition is satisfied. Furthermore, the improved Rete algorithm allows the definition of the rules within the DRR files whereby terms represent the conditions of the rule, which are used to pass the pointers to the corresponding nodes during the generation of the improved Rete network.

5. Evaluation

For the evaluation, the improved rule matching algorithm has been implemented in the Rule and Data Element Selector module of

```
1   For each rule in DRR file do;
2       Create a root node;
3       If there is a condition in driving rule then
4           Create an alpha node using the condition and set it as a child of the root node;
5           While there are further conditions in the driving rule do;
6               Create an alpha node using the condition and set it as a child of the root node;
7               Create a beta node;
8               Set beta node as a child of the new created alpha node;
9               Pass beta node a pointer to the memory of the new alpha node;
10              If there is a former created beta node;
11                  Set new beta node as a child of the former created beta node;
12                  Pass beta node a pointer to the memory of the former created beta node;
13              Else
14                  Set new beta node as a child of the former crated alpha node;
15                  Pass beta node a pointer to the memory of the former created alpha node;
16              End if;
17          End while;
18      End if;
19  End for;
```

Listing 2. Abstract improved matching algorithm, which illustrates the generation of the improved Rete network based on the rules defined in the DRR file.

Fig. 9. Driving system during the evaluation of the driving rule "shift as soon as possible".

the driving system. Besides the improved rule matching algorithm, the Rete and Treat algorithm has been implemented as well, as the results of its evaluation show the differences in the performance of the algorithms. Miranker (1987) and Nayak et al. (1993) used different metrics in their experiments to measure the performance of the rule matching algorithms. Therefore, a combination of the metrics was used in the evaluation of the improved rule matching algorithm. The following metrics have been used in the evaluation: (1) counting the comparisons of the facts against the node conditions; (2) counting the accesses to the node memories; (3) measuring the average execution time.

The improved rule matching algorithm and the original Rete algorithm were evaluated using the driving rules: (1) to switch the gear as soon as possible, (2) not to exceed the speed limit and (3) to keep enough distance from the car in front. These driving rules were described within the driving system with the rule conditions (1) Rpm $>$ 2500 & Gear $<$ 6, (2) Carspeed $<$Speedlimit and (3) DistanceToCar $<$Carspeed/2. On the basis of the three rules, the evaluations of the algorithms were done using 15 different journeys. During the different journeys, the rules (1)–(3) were used to initialize the algorithms, which created one Rete network for every rule. However, to have the same evaluation environment for every algorithm, the driving simulator shown in Fig. 9 was used to record the 15 journeys on a rural road. The recorded journeys consisted of information about the virtual car, like revolutions per minute, car speed, current gear and so on. The recorded journeys were played back on every run, which allowed the evaluation of the algorithms using the same conditions.

The recorded driving speeds of the journeys were compared against the average driving speeds on a rural road in the European Union, which is 77 km/h (André & Hammarström, 2000), using the one-tailed *t*-test. The one-tailed *t*-test is a statistical hypothesis test that allows to check whether the mean of samples are differing significantly from a particular value. Thus, using the *t*-test to check if the recorded data is able to represent the average driving behavior on a rural road. The result of the statistical hypothesis test was an *h*-value of 0 and a *p*-Value of 0.0515. Thus, the *t*-test does not reject the null hypothesis that the recorded data is able to represent the average driving behavior at a significance level of 5.15%.

6. Results and discussion

During the evaluation, the accesses to the node memories, the comparisons of the facts to the node conditions, and the execution time of the algorithm were measured. Table 2 shows the results of the evaluation. According to the results, it is clear that the improved rule matching algorithm outperforms the Rete and Treat algorithm in the environment of the driving system. The improved rule matching algorithm needed fewer comparisons of the facts to the node conditions, fewer accesses to the node memories, and took less average execution time. Thus, it can be assumed that passing a pointer to the nodes and storing the result of the comparison of the facts to the conditions is more efficient than passing the facts to the Rete network and storing the fact, which satisfies the node condition, within the node memories.

The results of the fact comparisons to the node conditions and accesses to the memory nodes are dependent on the duration of the journey. The longer the journey, the higher the comparisons to the node conditions and accesses to the node memories. However, in all 15 journeys, the improved Rete algorithm needed about 97% fewer accesses to the node memories than the Rete with a *p*-Value of 0.0000000004 or Treat algorithm with a *p*-Value of 0.0000000002 (Fig. 10). This is related to the logical value stored within the node memories and the pointer to the fact in the working memory. The stored pointers to the facts of the working memory allow the improved rule matching algorithm to avoid the deletion and addition of the facts within the node memories, like the Rete or Treat does. Another point is that the stored logical value is only altered when the result of the condition check differs from the stored logical value. This also saves accesses to node memories. Due to the missing beta nodes in the Treat algorithm, it needed fewer accesses to the memory nodes than the Rete algorithm (Table 3) .

Furthermore, the improved rule matching algorithm needed 35% fewer comparisons to the node conditions during all journeys than Rete with a *p*-Value of 0.000000000000004 and 15% fewer comparisons than Treat with a *p*-Value of 0.000000000001 (Fig. 10), because the improved rule matching algorithm triggers the improved Rete network to compare its node conditions to the facts after all facts are updated within the working memory. Therefore, the nodes have to

Table 3
The result of the evaluation based on 15 journeys.

		Journeys														
	Algorithms	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Accesses to node memories	*Improved*	219	203	215	317	175	159	247	207	131	291	283	191	182	171	166
	Rete	4331	8091	4821	5437	6873	11173	7549	7913	7981	6711	6437	9887	7148	7547	6510
	Treat	3437	6945	3997	4583	5991	8515	7325	7591	7603	5467	5531	7055	6890	7059	6688
Fact comparisons against node conditions	*Improved*	2860	3865	2960	3645	3640	3730	3695	3695	3595	3725	3835	3380	3450	3550	3540
	Rete	4423	5949	4512	5357	5482	5794	5841	5813	5649	5796	5463	5362	5344	5556	5308
	Treat	3462	4620	3516	4254	4314	4362	4464	4404	4248	4422	4146	4050	4110	4200	4182
Average execution time (in ms)	*Improved*	4	5	5	5	5	5	4	4	4	4	4	4	4	4	4
	Rete	11	12	11	11	12	12	12	11	12	11	11	11	12	12	11
	Treat	10	11	11	10	11	11	10	10	10	10	10	10	10	10	9

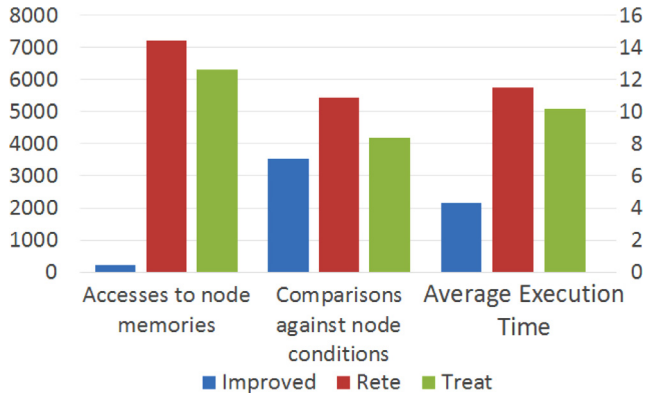

Fig. 10. The average results of 15 journeys.

check each condition against the facts once. In contrast, the Treat and Rete algorithms pass each fact from the working memory to their networks for the comparison to the node conditions. For example, when the facts speed and speed limit are updated within the working memory, the corresponding node in the improved rule matching algorithm has to compare the condition speed $<$ speed limit once to the working memory. In contrast, using the Rete or Treat algorithm, an update consists of deleting the old fact and adding the new fact. Thus, the facts speed and speed limit are first removed from the network. Then, the new facts are passed in succession to the network, which first has to check the fact speed against the condition and then the fact speed limit. Thus, an update with two facts in one condition causes in the Rete and Treat algorithm two comparisons instead of one comparison in the improved matching algorithm. Because the Treat algorithm has no beta nodes, whose node memories have to be updated on a deletion of a fact, it needs fewer comparisons than the Rete algorithm.

The evaluation showed that the improved rule matching algorithm was about 62% faster than the Rete with a p-Value of 0.0000000000000002 and 57% faster than Treat algorithm with a p-Value of 0.000000000000000001 (Fig. 10). Thus, passing a pointer to the facts in the working memory to the node memories and triggering the improved Rete network to check the node conditions against the updated facts needed less execution time than passing the facts to the network and storing them in the alpha and beta node memories as in Rete, or storing the facts in the alpha memories and the recomputation of the alpha node joins. However, according to the results of the evaluation, the Treat algorithm is faster than the Rete algorithm. Thus, the computation of the alpha node joins is more efficient in the area of the driving system than the recalculation of the beta nodes as the Rete algorithm does.

The results of the evaluated metrics showed an higher performance of the improved rule matching algorithm compared to the Rete and Treat algorithm. The main advantage of the improved rule matching algorithm is the ability to update a working memory fact without using a delete and an add operation, like Rete or Treat does. This allows a faster processing of facts that are changing frequently, like the car, driver or environmental information used in the introduced driving system. However, this also limits the introduced algorithm to the context of the driving system, as the improved rule matching algorithm is specific to the handling of frequently changing information.

7. Conclusion and further work

The goal of this research was to develop a rule matching algorithm for detecting broken driving rules within an adaptive and rule-based driving system that improves the performance of existing rule matching algorithms, as the performance of existing algorithms are not sufficient for matching rules against frequently changing data. It is clear, based on 15 journeys and three driving rules, that this has been achieved, as the improved rule matching algorithm outperforms existing rule matching algorithms like Rete and Treat.

In contrast to the existing algorithms, the improved rule matching algorithm stores pointers to the working memory facts, instead of the facts themselves. It also checks the node conditions after the update of all working memory facts, rather than checking the node conditions upon the change of a single working memory fact. Furthermore, an update of a fact causes in the improved rule matching algorithm a recomputation of the alpha and beta nodes, whereas Rete and Treat initiate first a removal of the old fact from the nodes, including a recomputation of the nodes to keep the network up-to-date. Then, the new fact is added to the network, followed by another recomputation of the alpha and beta nodes. This leads to a performance improvement of the improved rule matching algorithm during the update of the working memory facts, as an update in the introduced algorithm does not imply a delete and an add operation of a single working memory fact, like in Rete and Treat. Based on these findings, the introduced algorithm has a higher performance than the Rete or Treat algorithm, when used in the driving system to match frequently changing information against rules. Thus, it will be used in the energy-efficiency and safety relevant driving system for matching the data about the car, the driver, and the environment to the driving rules.

As the rule matching algorithm is tested in the environment of the driving simulator, further work will include testing it in the environment of a real car. Furthermore, as the rule matching algorithm is limited to the context of the driving system, it is planned to investigate the performance of the algorithm outside of the automotive area and in environments with less frequent changing data.

References

André, M., & Hammarström, U. (2000). Driving speeds in Europe for pollutant emissions estimation. *Transportation Research Part D: Transport and Environment, 5*(5), 321–335.

Barkenbus, J. N. (2009). Eco-driving: an overlooked climate change initiative. *Energy Policy, 38*(2), 762–769.

Bongard, A.-E. (2007). The environment and the driving instructor. Last visit: 07.10.2014 http://www.drivers.com/article/413/.

Brookhuis, K. A., & de Waard, D. (2010). Monitoring drivers mental workload in driving simulators using physiological measures. *Accident Analysis and Prevention, 42*(3), 898–903.

Burton, D., Delaney, A., Newstead, S., Logan, D. Fildes, B. (2004). Evaluation of anti-lock braking systems effectiveness. Royal Automobile Club of Victoria Ltd.

Chin, H.-C., & Quek, S.-T. (1997). Measurement of traffic conflicts. *Safety Science, 26*(3), 169–185.

Cho, H. J. (2008). Eco driving system. Last visit: 07.10.2014 http://kia-buzz.com/eco-driving-system/.

Fiat (2010). Eco-driving uncovered: the benefits and challenges of eco-driving, based on the first study using real journey data.

Forgy, C. L. (1982). Rete: a fast algorithm for the many pattern/many object pattern match problem. *Artificial Intelligence, 19*(1), 17–37.

Gao, T., Qiu, X., & He, L. (2013). Improved RETE algorithm in context reasoning for web of things environments. In *Proceedings of IEEE international conferance on green computing and communications and IoT and cyber, physical and social computing* (pp. 1044–1049).

German Statistical Office(2014). Verkehr-Verkehrsunfälle 2013.

Haworth, N., & Symmons, M. (2001). Driving to reduce fuel consumption and improve road safety. In *Proceedings of road safety research, policing and education conference: vol. 1* (p. 7).

Helms, H., Lambrecht, U., & Hanusch, J. (2010). Energieeffizienz im Verkehr. *Energieeffizienz, 1*, 309–329.

Jo, J., Lee, S. J., Park, K. R., Kim, I.-J., & Kim, J. (2014). Detecting driver drowsiness using feature-level fusion and user-specific classification. *Expert Systems with Applications, 41*, 1139–1152.

Khayyam, H., Nahavandi, S., & Davis, S. (2012). Adaptive cruise control look-ahead system for energy management of vehicles. *Expert Systems with Applications, 39*, 3874–3885.

Li, Y., Liu, W., Cao, B., Yin, J., & Yao, M. (2016). An efficient MapReduce-based rule matching method for production system. *Future Generation Computer Systems, 54*, 478–489.

Liu, G., Huang, S., Zhang, D., & Du, Y. (2014). A Rete rule reasoning algorithm based on the audit method ontology. *International Journal of Hybrid Information Technology, 7*(3), 211–224.

Lotan, T., & Toledo, T. (2006). *An in-vehicle data recorder for evaluation of driving behavior and safety: Paper No. 061607* (pp. 1–14). Transportation Research Board of the National Academies.

Milanes, V., Perez, J., Godoy, J., & Onieva, E. (2012). A fuzzy aid rear-end collision warning/avoidance system. *Expert Systems with Applications, 39*, 9097–9107.

Miranker, D. P. (1987). Treat: a better match algorithm for ai production systems. In *Proceedings of AAAI-87: vol. 1* (pp. 42–47).

Miranker, D. P. (1990). On the performance of lazy matching in production systems. In *Proceedings of AAAI-90: vol. 1* (pp. 685–692).

Nayak, P., Gupta, A., & Rosenbloom, P. (1993). Comparison of the RETE and TREAT production matchers for Soar (a summary). *The Soar Papers, 1*, 621–626.

New Zealand Transport Agency (2007). Your safe driving policy: helping you to manage work-related road safety and keep your employees and vehicles safe on the roads.

Patil, S. K. (2012). Regenerative braking system in automobiles. *International Journal of Research in Mechanical Engineering and Technology, 2*, 45–46.

Spencer, C. (2008). Ford tests show eco-driving can improve fuel economy by an average of 24 percent. Last visit: 07.10.2014 http://www.at.ford.com/news/cn/ArticleArchives/27527.aspx.

UNECE – United Nations Economic Commission for Europe (2004). Aggressive driving behaviour (background paper). Last visit: 07.10.2014 http://www.unece.org/trans/roadsafe/rs4aggr.html.

van Mierlo, J., Maggetto, G., van de Burgwal, E., & Gense, R. (2004). Driving style and traffic measures – influence on vehicle emissions and fuel consumption. *Proceedings of the Institution of Mechanical Engineers, Part D: Journal of Automobile Engineering, 218,*, 43–50.

Wessellink, B., Harmsen, R., & Eichhammer, W. (2010). *Energy savings 2020*. EcoFys and Fraunhofer Institute.

Wright, I., & Marshall, J. (2003). The execution kernel of RC++: RETE∗, a faster RETE with TREAT as a special case. *International Journal of Intelligent Games and Simulation, 2*(1), 36–48.

Xiaoqiu, F., Jinzhang, J., & Guoqiang, Z. (2011). Impact of driving behavior on the traffic safety of highway intersection. In *Proceedings of third international conference on measuring technology and mechatronics: vol. 2* (pp. 370–373).

Yay, E., & Martínez Madrid, N. (2013). SEEDrive – an adaptive and rule based driving system. In *Proceedings of the 9th international conference on intelligent environments, IE'13: 1* (pp. 262–265).

Yay, E., Martínez Madrid, N., & Ortega Ramírez, J. A. (2014). Using an improved rule match algorithm in an expert system to detect broken driving rules for an energy-efficiency and safety relevant driving system. *Procedia Computer Science, 35*, 127–136.