## 1.1. THE TRANSPORTATION PATHS ROUTING SEMANTIC DESCRIPTION

This chapter presents a semantic network formalism based on frame logic. Advanced techniques including HiLog extensions, transaction logic and dynamic module creation are applied to the problem of preferential routing and rerouteing. The deductive program is implemented in the FLORA-2 reasoning engine, which allows for scalable query execution. In the end possible extensions to the system and guidelines for future research are presented.

### 1.1.1. Introduction

Transportation networks are networks which show linkage relationships among numerous nodes [6]. Routing in transportation network usually deals with questions like [4]:

- What is the most efficient routing between two locations?
- What is a reasonable set of alternative routes if a segment of the transportation network is blocked?
- What are the comparison of costs for routing freight by highway, rail, or waterway?
- What additional travel times result from increasing traffic congestion along routes?
- What are the effects of changes in transportation costs, demands, or policies?

Route selection problem is present in various forms in transport networks. Bošnjak and Badanjak [1, p. 38] in their book on Basics of traffic engineering gave a general model for route, mode and time selection problem shown on figure 1.1.1.

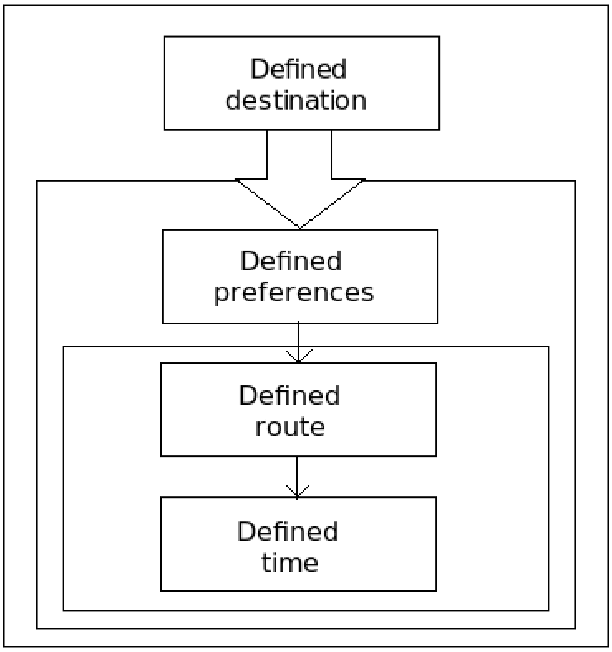

Fig.1.1.1 General model of route, preference and time selection problem

In general, to generate alternative routes for transportation many nodes and links which connect location for departure and destination have to be connected. Most common variables taken into consideration are departure location, destination, departure time, arrival time [6].

This paper takes the routing problem to the next level and adds more variables, for example, type of desired road, landscape type, stores on the way etc.

First part of the paper defines the syntax and semantics of frame logic, and defines semantic transport networks using graph theory. Next, the implementation in $\mathcal{F}$LORA-2 is shown, along with examples and directions for future research.

### 1.1.2. Frame logic

In the following we will define the syntax and semantics of frame logic.

**Definition 1** *(Frame logic alphabet)*

*The alphabet of an F-logic language $\mathcal{L}$ consists of the following* [5]*:*

- *a set of object constructors, $\mathcal{F}$;*
- *an infinite set of variables, $\mathcal{V}$;*
- *auxiliary symbols, such as, (, ),* [, ], $\rightarrow$, $\twoheadrightarrow$, $\bullet\!\!\rightarrow$, $\bullet\!\!\twoheadrightarrow$, $\Rightarrow$, $\Rightarrow\!\!\!\!\Rightarrow$, *etc.; and*
- *usual logical connectives and quantifiers,* $\vee$, $\wedge$, $\neg$, $\longleftarrow$, $\forall$, $\exists$.

Object constructors (the elements of $\mathcal{F}$) play the role of function symbols in F-logic whereby each function symbol has an arity. The arity is a non-negative integer that represents the number of arguments the symbol can take. A constant is a symbol with arity 0, and symbols with arity $\geq 1$ are used to construct larger terms out of simpler ones. An id-term is a usual first-order term composed of function symbols and variables, as in predicate calculus. The set of all variable free or ground id-terms is denoted by $U(\mathcal{F})$ and is commonly known as Herbrand Universe. Id-terms play the role of logical object identities in F-logic which is a logical abstraction of physical object identities.

A language in F-logic consists of a set of formulae constructed out of alphabet symbols. As in a lot of other logics, formulas are built out of simpler ones by using the usual logical connectives and quantifiers mentioned above. The most simple formulas in F-logic are called F-molecules.

**Definition 2** *(F-molecule)*

*A molecule in F-logic is one of the following statements:*

- *An is-a assertion of the form $C :: D$ ($C$ is a nonstrict subclass of $D$) or of the form $O : C$ ($O$ is a member of class $C$), where $C$, $D$ and $O$ are id-terms;*
- *An object molecule of the form O [ a ';' separated list of method expressions ] where $O$ is a id-term that denotes and object. A method expression can be either a non-inheritable data expression, an inheritable data expression, or a signature expression:*
    - *Non-inheritable data expressions can be in either of the following two forms:*
        - *A non-inheritable scalar expression $ScalMethod@Q_1, ..., Q_k \to T$,($k \geqslant 0$).*
        - *A non-inheritable set-valued expression*

          $SetMethod@R_1, ..., R_l \twoheadrightarrow \{S_1, ..., S_m\}$ ($l, m \geqslant 0$).
    - *Inheritable scalar and set-valued expression are equivalent to their non-inheritable counterparts except that $\to$ is replaced with $\bullet\!\!\to$, and $\twoheadrightarrow$ with $\bullet\!\!\twoheadrightarrow$*
    - *Signature expression can also take two different forms:*
        - *A scalar signature expression $ScalMethod@V_1, ..., V_n \Rightarrow (A_1, ..., A_r)$, ($n, r \geqslant 0$).*
        - *A set valued signature expression $SetMethod@W_1, ..., W_s \Rrightarrow (B_1, ..., B_t)$ ($s, t \geqslant 0$).*

*All methods' left hand sides (e. g. $Q_i$, $R_i$, $V_i$ and $W_i$) denote arguments, whilst the right hand sides (e. g. $T, S_i, A_i$ and $B_i$) denote method outputs. Single-headed arrows ($\to$, $\bullet\!\!\to$ and $\Rightarrow$) denote scalar methods and double-headed arrows ($\twoheadrightarrow$, $\bullet\!\!\twoheadrightarrow$ and $\Rrightarrow$) denote set-valued methods.*

Having the prerequisites defined we are now able to define F-formulae:

**Definition 3** *(F-formuale)*

*F-formulae are define recurively:*

- *F-molecules are F-formulae;*
- $\varphi \vee \psi$, $\varphi \wedge \psi$, *and* $\neg\varphi$, *are F-formulae if so are* $\varphi$ *and* $\psi$*;*
- $\forall X \varphi$ *and* $\exists Y \psi$ *are F-formulae, so are* $\varphi$ *and* $\psi$, *and* $X$ *and* $Y$ *are variables.*

For our purpose these definitions of F-logic are sufficient but the interested reader is advised to consult [5] for profound logical foundations of object-oriented and frame based languages.

### 1.1.3. Semantic transport networks

To define semantic transport networks we will first use graph theory [3, 9] and afterwards formalize the approach by using frame logic.

**Definition 4** *A graph* $\mathcal{G}$ *is the pair* $(\mathcal{N}, \mathcal{E})$ *whereby* $\mathcal{N}$ *represents the set of verticles or nodes, and* $E \subseteq N \times N$ *the set of edges connecting pairs from* $\mathcal{N}$*.*

The notion of directed- and valued directed graphs is of special importance to our study.

**Definition 5** *A directed graph or digraph* $\mathcal{G}$ *is the pair* $(\mathcal{N}, \mathcal{E})$*, whereby* $\mathcal{N}$ *represents the set of nodes, and* $\mathcal{E} \subseteq \mathcal{N} \times \mathcal{N}$ *the set of ordered pairs of elements from* $\mathcal{N}$ *that represent the set of graph arcs.*

**Definition 6** *A valued or weighted digraph* $\mathcal{G}$ *is the triple* $(\mathcal{N}, \mathcal{E}, \mathcal{V})$ *whereby* $\mathcal{N}$ *represents the set of nodes or verticles,* $\mathcal{E} \subseteq \mathcal{N} \times \mathcal{N}$ *the set of ordered pairs of elements from* $\mathcal{N}$ *that represent the set of graph arcs, and* $\mathcal{V} : \mathcal{E} \to \mathbb{R}$ *a function that attaches values or weights to arcs.*

A transport network can now be defined as a valued digraph in which nodes represent crossings between routes, arcs represent routes between nodes and weight represent the distance between nodes covered by arcs. The network has to be directed since there are one-way routes. Now, consider the following definition of a semantic transport network.

**Definition 7** *Let* $\mathcal{G} = (\mathcal{N}, \mathcal{E}, \mathcal{V})$ *be a valued digraph. Let further* $A = \{a_1, a_2, \ldots, a_n, \ldots\}$ *be an extensible set of attributes, and* $V = \{v_1, v_2, \ldots, v_n, \ldots\}$ *an extensible setof values, and let* $\nu : A \times V \longrightarrow \mathcal{N}$ *and* $\varepsilon : A \times V \longrightarrow \mathcal{E}$ *be two mappings which map attribute-value pairs to nodes and arcs respectively. We define the tuple* $\mathcal{STN} = (\mathcal{N}, \mathcal{E}, \mathcal{V}, A, V, \nu, \varepsilon)$ *to be a semantic transport network.*

The definition adds semantics in form of attribute-value tuples to the transport network. Consider the following example:

$$
\begin{array}{rcl}
\mathcal{N} & = & \{a, b, c, d, e, f, g\} \\
\mathcal{E} & = & \{p_1(a,b), p_2(a,c), p_3(a,d), p_4(b,e), p_5(c,e), p_6(d,f), p_7(e,g), p_8(f,h), p_9(f,g)\} \\
A & = & \{\text{type}, \text{landscape}\} \\
V & = & \{\text{highway}, \text{country road}, \text{farmland}, \text{forest}\} \\
\mathcal{V} & = & \begin{array}{c|ccccccccc} \mathcal{A} & p_1 & p_2 & p_3 & p_4 & p_5 & p_6 & p_7 & p_8 & p_9 \\ \hline \mathcal{V}(\mathcal{A}) & 1 & 2 & 1 & 5 & 2 & 1 & 3 & 5 & 2 \end{array} \\
\nu & = & \begin{array}{c|cc} & A \times V & \mathcal{N} \\ \hline & \multicolumn{2}{c}{\varnothing} \end{array} \\
\varepsilon & = & \begin{array}{c|cc} & A \times V & \mathcal{E} \\ \hline
& \text{type}\rightarrow\text{highway} & p_1 \\
& \text{landscape}\rightarrow\text{farmland} & p_1 \\
& \text{type}\rightarrow\text{country road} & p_2 \\
& \text{landscape}\rightarrow\text{forrest} & p_2 \\
& \text{type}\rightarrow\text{highway} & p_3 \\
& \text{landscape}\rightarrow\text{farmland} & p_3 \\
& \text{type}\rightarrow\text{highway} & p_4 \\
& \text{landscape}\rightarrow\text{farmland} & p_4 \\
& \text{type}\rightarrow\text{country road} & p_5 \\
& \text{landscape}\rightarrow\text{forrest} & p_5 \\
& \text{type}\rightarrow\text{highway} & p_6 \\
& \text{landscape}\rightarrow\text{farmland} & p_6 \\
& \text{type}\rightarrow\text{highway} & p_7 \\
& \text{landscape}\rightarrow\text{farmland} & p_7 \\
& \text{type}\rightarrow\text{highway} & p_8 \\
& \text{landscape}\rightarrow\text{farmland} & p_8 \\
& \text{type}\rightarrow\text{highway} & p_9 \\
& \text{landscape}\rightarrow\text{farmland} & p_9
\end{array}
\end{array}
$$

The semantics of this network are depicted on figure 1.1.2. Note that the node annotation ($\nu$) has intentionally been left empty ($\varnothing$) for sake of simlicity. In the following we will use this example and implement it using $\mathcal{F}$LORA-*2* .

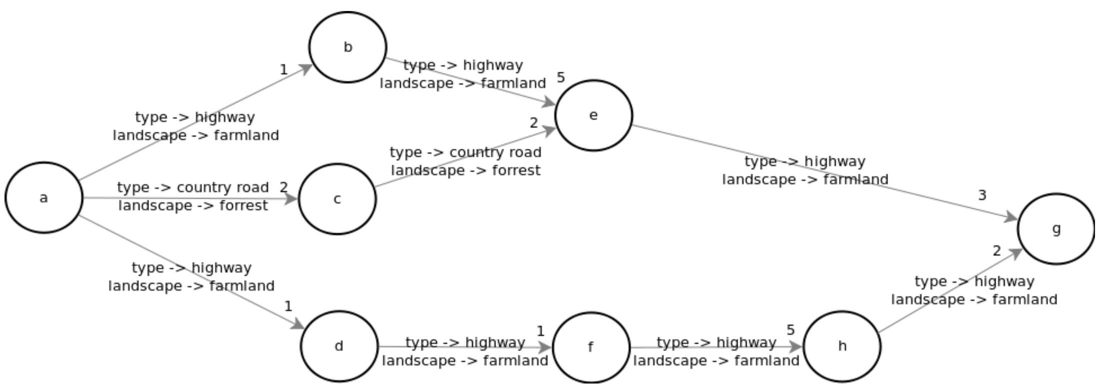


Fig.1.1.2 Semantic transport network example

Now consider the following problem (preferential routing): find the shortest path $p$ from node $n_1$ to node $n_2$ conforming to a set $P_{\mathcal{N}} \subseteq A \times V$ such that $\forall n_i \in p$ it holds that $\forall (a_i \rightarrow v_i) \in P_{\mathcal{N}} \Rightarrow [(a_i \rightarrow v_i) \longrightarrow n_i] \in \nu$, and conforming to a set $P_{\mathcal{E}} \subseteq A \times V$ such that $\forall p_i \in p$ it holds that $\forall (a_i \rightarrow v_i) \in P_{\mathcal{E}} \Rightarrow [(a_i \rightarrow v_i) \longrightarrow n_i] \in \varepsilon$.

### 1.1.4. Implementation

The implementation in $\mathcal{F}$LORA-*2* [11] follows an object-oriented approach. Two classes are defined: nodes and routes, whereby nodes are intersections between routes. In the following listing we define $\{a, b, c, d, e, f, g, h\}$ to be instances of class node. These instances would usually have additional attributes like lattitude and longitude.

```
a:node. b:node. c:node. d:node. e:node. f:node. g:node.
```

The following listing shows how routes (edges) are implemented. As one can see, three additional attributes where used to model source (attribute from), destination (attribute to) and route length (attribute length).

```
p1:route[
  from->a,
  to->b,
  length->1,
  type->highway,
  landscape->farmland ].
p2:route[
  from->a,
  to->c,
  length->2,
  type->country_road,
  landscape->forest ].
...
```

Having the basic classes defined, we are now able to define the specific methods. Prior to that we need to define $\mathcal{F}$LORA-*2* modules, transaction logic and HiLog extensions, which are specific for this logical platform. $\mathcal{F}$LORA-*2* modules are logical abstractions that allow us to split large programs into smaller instances and to facilitate reuse [10, p. 25]. A module consists formally of a name and a content. In a way, modules in $\mathcal{F}$LORA-*2* are similar to namespaces, that allow us to query only one part of a (potentially large) knowledge base. To call any literal (F-molecule or predicate) that is defined in some other module that the actual the following syntax is used: $\mathrm{literal@module}$

$\mathcal{F}$LORA-*2* also supports dynamic updates of the knowledge base [10, pp. 74 - 89]. The following syntax allows us to insert facts into the knowledge base at runtime: $\mathrm{insop}\{\mathrm{literals}[|\mathrm{query}]\}$

whereby insop can be any of insert, insertall, delete, deleteall, erase or eraseall. For our purpose we will use the insertall statement which inserts all literals that satisfy a given formula. Additionally, to create and erase modules on runtime the we will use the newmodule{modulename} and erasemodule{modulename} statements, respectively.

*F*LORA-*2* uses HiLog [2] as its default predicate representation. This in essence means that complex terms can appear wherever a function symbol is allowed [10, p. 42].

In order to reduce the search space and filter out only relevant routes, we will take the following strategy:

1. A new module is created on runtime
2. Facts (nodes and routes) that pass through the user supplied filters are inserted into the new module
3. A query is issued towards the new module to find the shortest path
4. The results are delivered and the module is erased

Since all queries will be issued against a newly created module, we need to find that module at runtime. This is why the module name will be a logic variable in all the following predicates. First we implement the path_to/2 method which allows us to query for paths to other nodes from a given one. The implementation is recursive, as usual:

```
?x:node[ path_to( ?y, ?mod ) -> [ ?x, ?y ] ] :-
  ?_:route[ from->?x, to->?y ]@?mod.

?x:node[ path_to( ?y, ?mod ) -> [ ?x, ?z | ?t ] ] :-
  ?_[ from->?x, to->?z ]@?mod,
  ?z[ path_to( ?y, ?mod ) -> [ ?z | ?t ] ].
```

With such a method defined we can now issue the query "show all paths from node d to node g" (main being the current module):

```
flora2 ?- d[ path_to( g, main )->?p ].

?p = [d, f, h, g]
```

In order to find the length of a path to a given node we implement the following method (path_length_to):

```
?x:node[ path_length_to( ?y, ?mod ) -> ?l ] :-
  ?x[ path_to( ?y, ?mod ) -> [ ?x, ?y ] ],
  ?_:route[ from->?x, to->?y, length->?l ]@?mod.

?x:node[ path_length_to( ?y, ?mod ) -> ?l ] :-
  ?x[ path_to( ?y, ?mod ) -> [ ?x, ?z | ?t ] ],
  ?_:route[ from->?x, to->?z, length->?l1 ]@?mod,
  ?z[ path_to( ?y, ?mod ) -> [ ?z | ?t ] ],
  ?z[ path_length_to( ?y, ?mod ) -> ?l2 ],
  ?l is ?l1 + ?l2.
```

Now we can ask the question "What are the lengths of all paths from d to g? " The following query yields the answer.

```
flora2 ?- d[ path_length_to( g, main )->?l ].

?l = 8
```

In order to be able to compare lengths of different paths, we implement the following auxiliary predicate (path_length).

```
path_length( [ ?x, ?y ], ?l, ?mod ) :-
  ?_:route[ from->?x, to->?y, length->?l ]@?mod.

path_length( [ ?x, ?y | ?t ], ?l, ?mod ) :-
  ?_:route[ from->?x, to->?y, length->?l1 ]@?mod,
  path_length( [ ?y | ?t ], ?l2, ?mod ),
  ?l is ?l1 + ?l2.
```

The following query shows the behavior of this predicate:

```
flora2 ?- path_length( [d, f, h, g], ?l, main ).

?l = 8
```

Now we are able to implement the minimal_path_to method which will allow us to find the minimal path between two nodes.

```
?x:node[ minimal_path_to( ?y, ?mod ) -> ?p ] :-
  ?m = min{ ?l | ?x[ path_length_to( ?y, ?mod )->?l ] },
  ?x[ path_to( ?y, ?mod ) -> ?p ],
  path_length( ?p, ?m, ?mod ).
```

As one can see the method makes use of the min aggragate function which finds the minimal path length. We can now ask the question “what is the minimal path from a to g? ” using the following query:

```
flora2 ?- a[ minimal_path_to( g, main ) -> ?p ].

?p = [a, c, e, g]
```

Now we need to implement a mechanism to filter out only those nodes/routes which conform to a set of user supplied preferences. A generic way to do this is to use HiLog as is done in the following predicate (filter).

```
filter( [], ?_ ).
filter( [ ?x->?y | ?t ], ?p ) :-
  ?p[ ?x -> ?y ],
  filter( ?t, ?p ).
```

The first parameter is a list of attributes with corresponding values, and the second is an object from the knowledge base. The predicate succedes iff the object has all attribute-value pairs which were supplied. Consider the following problem: “find all routes which are highway paths and go through farmland”. The following query solves the problem:

flora2 ? - filter( [ type->highway, landscape->farmland ], ? p ).

```
flora2 ?- filter( [ type->country_road, landscape->forrest ],
?p ).

?p = p2
?p = p5
```

The same query would apply to nodes if nodes were also annotated with additional semantics. In this way we could have filtered out only those nodes which are in some geographical area for example.

Now we can implement the preference_path predicate which will find a minimal path that conforms to user specified filters. First only those routes that conform to all filters are inserted in to a new module, and then a minimal path query is issued.

```
preference_path( ?from, ?to, ?filters, ?p ) :-
  insertall{
    ?p:route[ ?x->?y ]@pref |
    ?p:route[ ?x->?y ],
    filter( ?filters, ?p ) },
  ?from[ minimal_path_to( ?to, pref ) -> ?p ].
```

We can now ask the question: “What are the shortest paths from a to g that are highway paths and go through farmland? ” In order to issue such a query we first need to create a new module.

```
flora2 ?- newmodule{ pref },
preference_path( a, g, [type->highway,landscape->farmland], ?p
),
erasemodule{ pref }.

?p = [a, b, e, g]
?p = [a, d, f, h, g]
```

### 1.1.5. Conclusion

The paper presented semantic transport networks and their application to the preferential routing problem. The frame logic formalism was used to represent semantic transport networks and the *F*LORA-*2* reasoning engine was used for implementation. We demonstrated how advanced techniques like HiLog, transaction logic and dynamic modules can be used to avoid known obstacles.

One such obstacle is the combinatorial explosion that can lead to inefficient queries when solving routing problems with logic programming due to a multiple recursive procedure. By filtering out facts to a dynamic module, the fact base is dramatically reduced, and queries are faster and more efficient.

We believe that this approach can be taken even further by applying geographical filters as outlined already. Due to reasonable Python [7] and the Python Google API, the implemented system can be easily connected to Google Maps; and due to F-OWL [12] it can be connected to any OWL based ontology [8]. We leave the implementation of geographical and other filters to future research.

## BIBLIOGRAPHY

[1] Bošnjak, I., and Badanjak, D.: Osnove prometnog inženjerstva. Fakultet prometnih znanosti, Zagreb, Croatia, 2005.
[2] Chen, W., Kifer, M., and Warren, D. S.: Hilog: A foundation for higher-order logic programming. Journal of Logic Programming 15, 3 (Feb. 1993), 187–230.
[3] Divjak, B., and Lovrenčić, A.: Diskretna matematika s teorijom grafova. TIVA & Faculty of Organization and Informatics, 2005.
[4] Harrison, G.: Transportation network routing models. http://www.ornl.gov/sci/ees/etsd/cta/RCB_Transportation%20Network%20Rou%ting%20Models.pdf, 1.2.2011., 2011.
[5] Kifer, M., Lausen, G., and Wu, J.: Logical foundations of object-oriented and frame-based languages. Journal of the Association for Computing Machinery 42 (May 1995), 741–843.
[6] Park, B. J., Kang, M. H., Choi, H. R., and Kim, H. S.: Development of intellectual route selection system for intermodal transportation. In International Conference on Logistics, Shipping and Port Management (2007).
[7] Schatten, M.: Reasonable python or how to integrate f-logic into an object - oriented scripting language. In 11 th International Conference on Intelligent Engineering Systems Proceedings (2007), I. Rudas, Ed., IEEE, pp. 297–300.
[8] W3C.: Owl web ontology language overview - w3c recommendation, Feb. 2004.
[9] Wasserman, S., and Faust, K.: Social Network Analysis ; Methods and Applications. Structural analysis in the social sciences. Cambridge University Press, 1994.
[10] Yang, G., Kifer, M., Wan, H., and Zhao, C.: FLORA-2: User’s Manual Version 0.95 (Androcymbium), Apr. 2008.
[11] Yang, G., Kifer, M., and Zhao, C.: Flora-2: A rule-based knowledge representation and inference infrastructure for the semantic web. In Second International Conference on Ontologies, Databases and Applications of Semantics (ODBASE) Proceedings (Catania, Sicily, Italy, November 2003).
[12] Zou, Y., Finin, T., and Chen, H.: F-owl: An inference engine for semantic web. In Formal Approaches to Agent-Based Systems, M. Hinchey, J. Rash, W. Truszkowski, and C. Rouff, Eds., vol. 3228 of Lecture Notes in Computer Science. Springer Berlin / Heidelberg, 2005, pp. 238–248.

Authors:
**Markus Schatten**
Faculty of organization and informatics, Varaždin, Croatia
*markus.schatten@foi.hr*
**Miroslav Bača**
Faculty of organization and informatics, Varaždin, Croatia
*miroslav.baca@foi.hr*
**Petra Koruga**
Faculty of organization and informatics, Varaždin, Croatia
*petra.koruga@foi.hr*