

Первопорядковый логический вывод при решении задач распознавания с использованием результатов, полученных алгоритмически и нейросетевыми методами

Evgeny Cherkashin

Институт динамики систем и теории управления СО РАН,
Иркутск, Россия
bychkov@icc.ru

2024 октября, Новосибирск



Проблематика распознавания сценариев

1. Термин искусственный интеллект (ИИ) в настоящее время связывают с моделями, основанными на машинном обучении (МО):
 - ▶ нейронными сетями, включая этапы свертки,
 - ▶ генерирующими нейронными сетями,
 - ▶ моделями регрессии, таксономии, классификации на основе машинного обучения и т. п.
2. Известное ограничение применимости МО – невозможность интерпретации получаемых моделей в виде процедуры трансформации данных: нейронная сеть – это набор коэффициентов.
3. Другое ограничение МО – сложность построения моделей МО распознавания свойств (динамических систем), например, свойств процесса, представленного набором кадров видео. Необходимо уметь распознавать
 - ▶ набор допустимых состояний объектов, их классификация,
 - ▶ классы «недопустимых» и «целевых» состояний,
 - ▶ правила перехода объектов из состояния в состояние,
 - ▶ правила изменения свойств при выполнении перехода,
 - ▶ общий сценарий (модель) поведения объектов.

Проблематика распознавания сценариев

1. Применение МО требует большой объем данных для обучения, тестирования, верификации.
2. Предлагаемый вариант модели решения – построение иерархической системы моделей распознавания на основе существующих моделей МО «общего назначения»:
 - ▶ нижние уровни – результаты распознавания алгоритмами и моделями МО,
 - ▶ средние уровни – анализ статических свойств сцен,
 - ▶ более высокие уровни – анализ (распознавание) динамических свойств.

Результат анализа – классификация сценария (модели) и идентификация параметров.

3. Структуры данных результата далее интерпретируются, например, трансформируются в новую модель.
4. Вкладываем ресурсы не в создание новой модели МО с непредсказуемыми свойствами, а в построение моделей рассуждения.

Инструменты реализации

1. Алгоритмы распознавания (регулярные выражения, свертки).
2. Универсальные предобученные нейронные сети.
3. Хранилища разноформатных данных, накапливающие семантическую информацию об объектах.
4. Обеспечение доступа к внешним данным и знаниям.
5. Логическое программирование (ЛП): языки, реализации.
6. Методики представления моделей на языках ЛП:
 - ▶ моделирование сценариев,
 - ▶ процедур идентификации структур и параметров,
 - ▶ трансформации получаемых структур.

Экспериментальный инструментарий

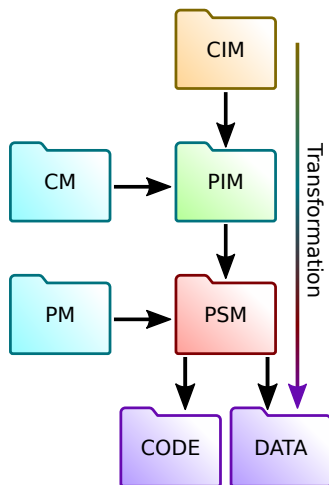
- ❑ Предобученная НС изображений (Segment Anything, 2024).
- ❑ Хранение данных в распределенных графах знаний. Linked Open Data (LOD), SPARQL.
- ❑ Язык логического ООП (ЛООП) Logtalk - макропакет над разными реализациями ISO-Prolog.
- ❑ Методики применения ЛООП при проектировании моделей распознавания и трансформации.

Выбран, т.к. имеет следующие свойства.

- ❑ наследует широко известный синтаксис и среду исполнения Prolog;
- ❑ реализован как макропакет, потери производительности составляют около 1.5-10%;
- ❑ имеет гибкую семантику: преобразования и ограничения определяются в рамках одного и того же синтаксиса;
- ❑ реализует объектно-ориентированное структурирование знаний (правил), инкапсуляцию и замену;
- ❑ из преобразований можно строить композиции;
- ❑ механизм включения ограничений перехватом сообщений объекта к объекту (событий);
- ❑ есть варианты для различных реализаций ISO Prolog.

HZ What

Интерпретация как трансформация. Model-Driven Architecture



- MDA Model-Driven Architecture
- CIM Computationally Independent Model
- CM Model of Computations
- PIM Platform Independent Model
- PM Platform Model
- PSM Platform-Specific Model
- CODE Генерируемый исходный код
- DATA Начальное состояние баз данных

Основная цель НИРОКР разработать технологии MDA, где модели CIM, PIM представлены в языках SysML, UML, BPMN, CMMN с использованием Семантического веба:

1. CIM представляется в UML, SysML, BPMN, CMMN, или как результат анализа текстов программ,
2. PIM, PSM представляется в UML и в RDF с использованием стандартных онтологий,
3. трансформации реализуются в языке Logtalk,
4. сервера LOD запрашиваются на предмет дополнительной информации,
5. порождение документов и интерфейсов пользователя, размеченных по требованиям LOD.

Технологии семантического веба – элемент моделей трансформации

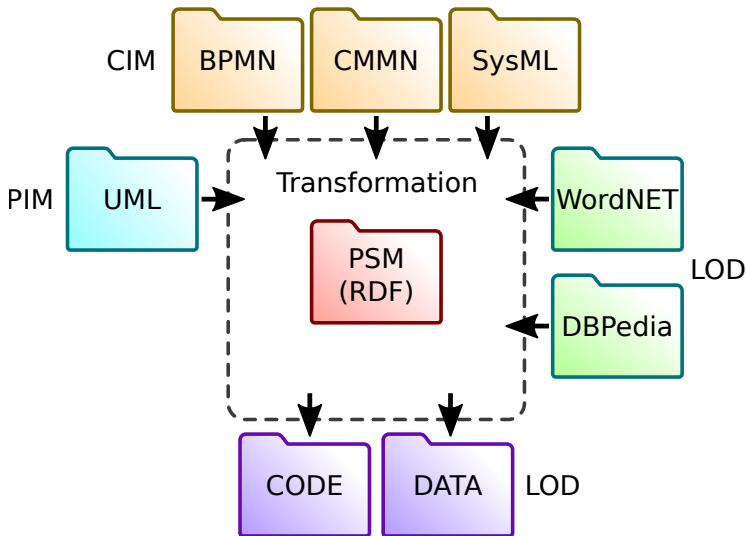
- ❑ Использование результатов исследований, формализации и стандартизации предметных областей
- ❑ Граф знаний задается множеством троек
- ❑ Онтологии описываются формально (`rdfs:domain`, `rdfs:range`);
- ❑ Поддержаны в большинстве систем программирования библиотеками, механизмами логического вывода, SPARQL
- ❑ Существует способ глобальной идентификации объектов в RDF: в разных программах можно идентифицировать один и тот же объект
- ❑ SWI-Prolog поддерживает механизмы прямых запросов к графу и интерпретацию некоторых отношений (`rdfs:label`, `dc:title`)
- ❑ Простая поддержка разделения доступа к информации (`rdfs:seeAlso`)
- ❑ Семантический веб и LOD – основа интеграции приложений

Связанные открытые данные, LOD

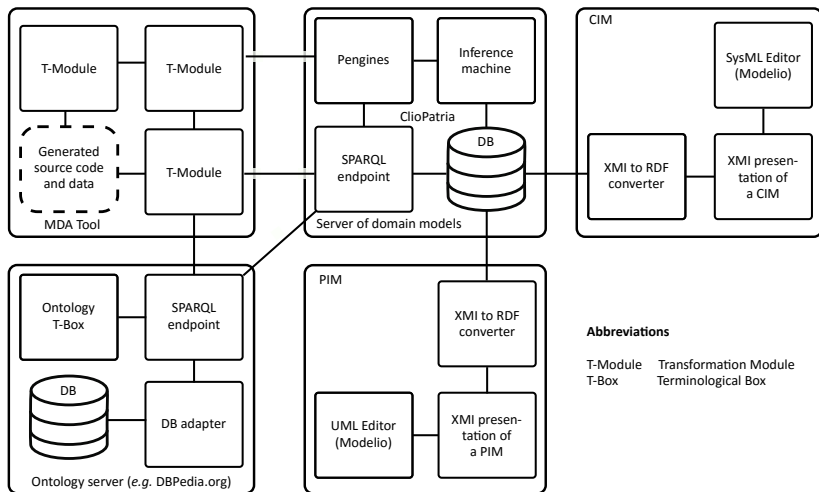
1. Информация публикуется в Интернете с лицензией открытого доступа
2. Он представлен в машиночитаемой форме, например, таблица Excel вместо растрового изображения
3. открытый формат, используемый, например, CSV вместо Excel
4. Формат основан на рекомендуемых стандартах W3C, что позволяет ссылаться на RDF и SPARQL
5. Опубликованные данные относятся к объектам, образующим контекст

Таким образом, приложения публикуют данные как отношения объектов (сущностей)

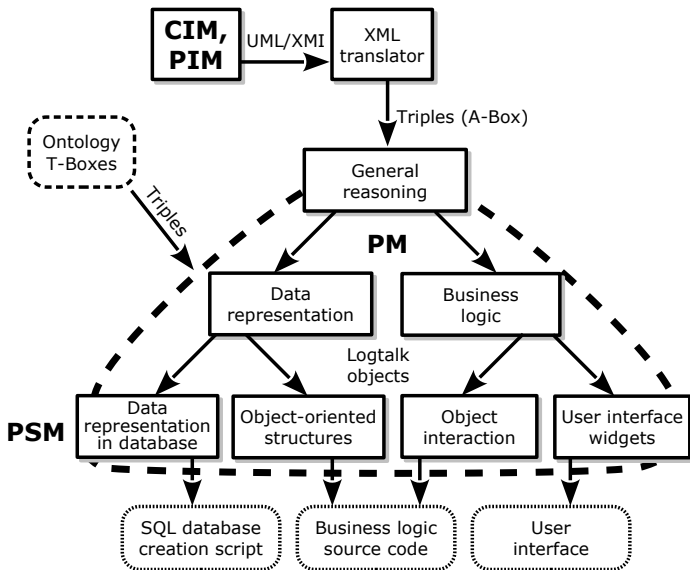
Model Driven Architecture and Linked Open Data



Инфраструктура MDA



Процесс трансформации



Сценарий синтеза PSM

```

:- object(direct(_Package,_LocalProf,_CodeProf)).
:- public([tr/4,tr/3]).
% . . . . .
tr(class, Class, ClassID):- ::package(Package),
    query(Package)::class(Name, ClassID),
    create_object(Class, % . . . . .
    create_object(Attributes, % . . . . .
    create_object(Methods, % . . . . .
    Class::name(Name),
    % Generate attributes of the class,
    % organizing them in a local database.
    % ...methods...
    Class::attributes(Attributes),
    Class::methods(Methods).

tr(attribute, Attribute, ClassID, AttributeID):-
    ::package(Package),
    query(Package)::attribute(Name,ClassID,AttrID),
    create_object(Attribute, % . . . . .
    Attribute::name(Name).

tr(method, Method, ClassID, MethodID):-
    ::package(Package),
    query(Package)::method(Name,ClassID,MethodID),
    create_object(Method, % . . . . .
    Method::name(Name).
:- end_object.

```

```

% Transformation driver object
% Public interface of a class synthesis scenario

```

```

% Synthesize a class
% Query package structure in RMI
% Create a «Class» object
% Create «Attributes» object
% ...«Methods».
% Name the class.

```

```

% Set the attributes for class.
% ...methods.

```

```

% Attribute transformations

```

```

% Name the attribute.

```

```

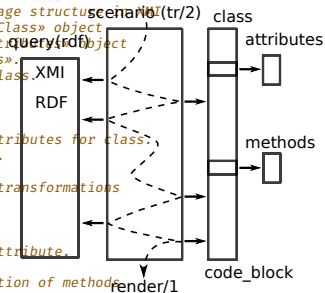
% Transformation of methods

```

```

% Name of the method

```

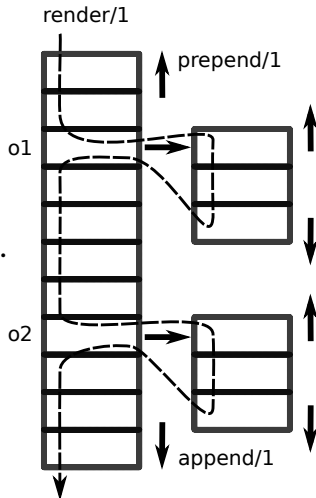


Обрамление модели PSM

```
:- object(query(_XMI)).
:- protected(xmi/1).
:- public([class/2, attribute/3, method/3]).
xmi(XMI) :- parameter(1, XMI).
class(Name, ID):-                                     % Recognition of Class
    ::xmi(XMI),
    XMI::rdf(ID,rdf:type,uml:'Class'),
    XMI::rdf(ID,rdfs:label, literal(Name)).
attribute(Name, ClassID, ID):-                       % ...attribute...
    ::xmi(XMI),
    XMI::rdf(ClassID, xmi:ownedAttribute, ID),
    XMI::rdf(ID, rdfs:label, literal(Name)).
method(Name, ClassID, ID):-                          % ...method...
    ::xmi(XMI),
    XMI::rdf(ClassID, xmi:ownedOperation, ID),
    XMI::rdf(ID, rdfs:label, literal(Name)).
% . . . . .
:- end_object.
```


Блок кода. Идея – llvmlite*

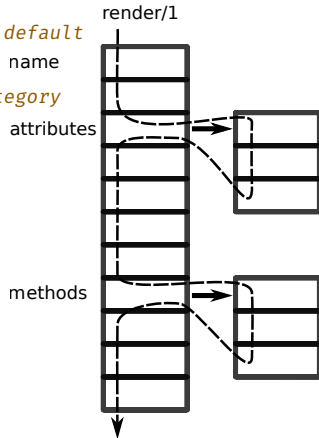
```
- object(code_block, specializes(root)).
% Public interface of the object
- public([append/1, prepend/1, clear/0,
  render/1, render_to/1, remove/1,
  item/1, items/1]).
% Code block items
- dynamic([item_/1]).
- private([item_/1]).
% Methods specialized during inheritance
- protected([renderitem/2, render_to/2]).
% . . . . .
% Delegate rendering to object itself
renderitem(Object, String):-
    current_object(Object, !,
    Object::render(String).
% Convert a literal to its string
% representation
renderitem(literal(Item), String):-!,
    atom_string(Item, String).
% Just print the item (debugging).
renderitem(Item, String):-
    root::iswritef(String, '%q', [Item]).
- end_object.
```



*) <https://github.com/numba/llvmlite>

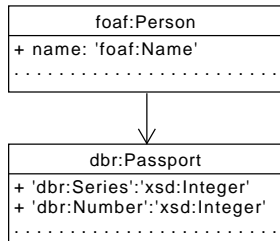
PSM-модель класса Python – специализация блока кода

```
:- object(class, specializes(code_block),
    imports([named])). % Category of named entities
:- public([classlist/1, methods/1, attributes/1]).
% . . . . .
renderitem(Item, Result):- % proceed with default
    ^^renderitem(Item, Result). % rendering name
render(Result):- % Source generator
    ^^render(Name), % implemented in a category
    ( ::item(classlist(List)) ->
        % . . . . .
        [Name]) ),
    ( ::item(attributes(Attributes))->
        % . . . . .
        [DefAttrList]),
    Attributes::items(InstanceAttrs),
    findall(S, ( % initialize attributes
        % . . . . .
        ), AttrAssigns),
    root::unindent,
    AttrList=[ConstructorDef|AttrAssigns];
    % . . . . .
    AttrList=[ConstructorDef, Pass] ),
    ( ::item(methods(Methods))-> % If any ...
        Methods::render(MethodList);
        MethodList=[] ),
    lists::append(AttrList,MethodList,StringList),
    root::unindent. Result=[Signature|StringList].
```

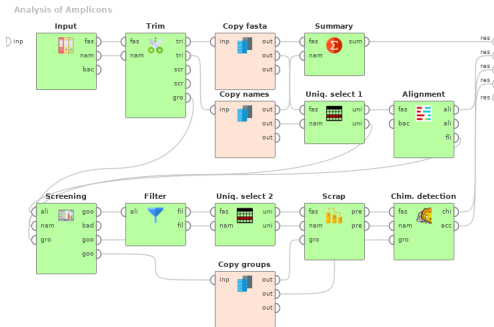


Доступ к данным LOD

```
:- category(sparql).  
:- public(query/2).  
query(Pattern, Parameters, Row):-  
    prepare(Pattern, Parameters, Query),  
    server(Host, Port, Path),  
    sparql_query(Query, Row,  
        [host(Host), port(Port), path(Path)]).  
:- protected(server/3). % must be implemented  
                                % by a subclass.  
:- protected(prepare/3). % prepares a query  
% . . . . . % string.  
:- end_category.  
  
:- object(dbpedia, extends(sparql)).  
:- protected(server/3).  
server('dbpedia.org', 80, '/sparql').  
:- public(entity_name/2).  
entity_name(Entity, Language, Name):-  
    query('select ?name where { '  
        ' %w rdfs:label ?name. '  
        'FILTER langMatches( lang(?label), '  
        ' "%w" )}'', [Entity, Language],  
        row(Name)).  
:- end_object.  
  
% ?- dbpedia::entity_name(dbr:'Passport', 'ru', Name).
```



Приложение: Диаграммы потока исполнения в NGS



Термин	Расшифровка
NGS	New Generation Sequencing
Amplicon	часть ДНК, РНК, скопированная много раз
Mothur	прикладной пакет для NGS
Rapidminer	визуальный инструмент для моделирования процессов Data Mining и их исполнения

Зеленые блоки — это модули Mothur. Остальные – модули Rapidminer.

Модуль Rapidminer

```
vector<string> AlignCommand::setParameters(){ // PART OF MODULE SOURCE
```

```
try {
```

```
    CommandParameter ptemplate("reference", "InputTypes", "", "", "none", "none", "none", "", false, true, true); p
```

```
    CommandParameter pcandidate("fasta", "InputTypes", "", "", "none", "none", "none", "fasta-alignreport-accnos
```

```
    CommandParameter psearch("search", "Multiple", "kmer-blast-suffix", "kmer", "", "", "", false, false, true)
```

```
    CommandParameter pksize("ksize", "Number", "", "8", "", "", "", false, false); parameters.push_back(pksize)
```

```
    CommandParameter pmatch("match", "Number", "", "1.0", "", "", "", false, false); parameters.push_back(pmat
```

```
// . . . . .
```

```
package com.rapidminer.ngs.operator; // GENERATED JAVA MODULE
```

```
// imports
```

```
class MothurChimeraCcodeOperator extends MothurGeneratedOperator {
```

```
    private InputPort fastaInPort = getInputPorts().createPort("fasta");
```

```
    private InputPort referenceInPort = getInputPorts().createPort("reference");
```

```
    private OutputPort chimeraOutPort = getOutputPorts().createPort("chimera");
```

```
    private OutputPort mapinfoOutPort = getOutputPorts().createPort("mapinfo");
```

```
    private OutputPort accnosOutPort = getOutputPorts().createPort("accnos");
```

```
    public MothurChimeraCcodeOperator (OperatorDescription description) {
```

```
        super(description);
```

```
    }
```

```
    @Override
```

```
    public void doWork() throws OperatorException {
```

```
        super();
```

```
        // . . . . .
```

```
    }
```

```
    @Override
```

```
    public List<ParameterType> getParameterTypes() {
```

```
        super();
```

```
        // . . . . .
```

```
    }
```

```
    @Override
```

```
    public String getOutputPattern(String type) {
```

```
        if (type=="chimera") return "[filename],[tag],ccode.chimeras-[filename],ccode.chimeras";
```

```
        if (type=="mapinfo") return "[filename],mapinfo";
```

```
        if (type=="accnos") return "[filename],[tag],ccode.accnos-[filename],ccode.accnos";
```

```
        return super.getOutputPattern(type);
```

```
    }
```

```
}
```

Исходный граф RDF (TTL) и его обрамляющий объект

```
@prefix xml: <http://www.w3.org/.../namespace> .
@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .
ngsp:spec a ngsp:Specification ;
    ngsp:module mothur:NoCommand,
        mothur:align-check,
        mothur:align-seqs,
# . . . . .
    mothur:align-check a ngsp:Module ;
        ngsp:outputPattern [ a cnt:Chars ;
            ngsp:parameterName "type" ;
            ngsp:pattern [ ngsp:patternString
                "[filename],align.check" ;
                dc:identifier "aligncheck" ] ;
            cnt:chars # . . . . .
# . . . . .
    mothur:align-check-idir-parameter
        a ngsp:Parameter ;
        ngsp:important false ;
        ngsp:multipleSelectionAllowed false ;
        ngsp:optionsDefault "" ;
        ngsp:required false ;
        ngsp:type mothur:String ;
        dc:title "inputdir" .

    mothur:align-check-map-parameter a ngsp:Parameter ;
        ngsp:important true ;
        ngsp:multipleSelectionAllowed false ;
        ngsp:optionsDefault "" ;
        ngsp:required true ;
        ngsp:type mothur:InputTypes ;
        dc:title "map" .

    mothur:align-check-name-parameter a ngsp:Parameter ;
        ngsp:chooseOnlyOneGroup "namecount" ;
        ngsp:important false ;
```

```
:- object(queryparam(_RDF,_Parameter),
    extends(ngsquerybase)).

:- public(type/1).
type(Type) :-
    ::attr(type, Type).
:- public(name/1).
name(Name) :- ::attr(dc:title, literal(Name)).
:- public(options/1).
options(Value):- ::attr(options, Value).
:- public(options_default/1).
options_default(Value):-
    ::attr(optionsDefault, Value).
% . . . . .
:- public(multiple_selection_allowed/0).
multiple_selection_allowed:-
    ::bool_attr(multipleSelectionAllowed).
:- public(required/0).
required:-
    ::bool_attr(required).
:- public(important/0).
important:-
    ::bool_attr(important).
:- protected(attr/2).
attr(NS:Name, Value):-
    ::ngs(RDF),
    ::second(Parameter),
    rdf_db::rdf_global_object(Value, V),
    RDF::rdf(Parameter, NS:Name, V).
attr(Name, Value):-
    \+ Name=_,!,
    ::ngs(RDF),
    ::second(Parameter),
    rdf_db::rdf_global_id(Value, V),
    RDF::rdf(Parameter, ngsp:Name, V).
```

Интересные положительные впечатления:

- ❑ Logtalk и RDF являются гибкими, универсальными и удобными средствами реализации MDA.
- ❑ Лучшие средства реализации – инкапсуляция в предикаты Prolog и объекты Logtalk;
- ❑ Не все свойства Logtalk исследованы: необходимо разработаны эффективные методики программирования, включая перехват сообщений.

Технические проблемы, затрудняющие использование Logtalk:

- ❑ Очень простые задачи требуют слишком много усилий, например, конвертация идентификатора в CamelCase.
- ❑ Много времени уходит на изучение свойств существующих онтологий, но это более продуктивно, чем разработка «с нуля»;
- ❑ Prolog не является популярным языком в MDA, как и Logtalk.

Еще польза для машинного обучения

Тенденции развития систем ИИ

- ❑ Переход от рассуждений (логического вывода) к реакции по шаблонам

Реализация тенденций

- ❑ Использовать маски Segment Anything и результаты распознавания в качестве данных для обучения

Conclusion

The following results have been obtained as for today:

- ❑ A technique for model representation has been developed and tested.
- ❑ A programming technique using object-oriented logical language Logtalk is devised.
- ❑ Prototypes of various transformation procedures are implemented.
- ❑ Transformation tools are tested in application areas and no significant technical problems were mentioned.

Further development directions are as follows:

- ❑ A technique for document automatic markup with vocabulary entities.
- ❑ A transformation implementation techniques, minimizing usage of dynamic objects, targeting on macro properties of Logtalk.
- ❑ Form a toolset out of existing prototypes obeying nowadays software development requirements.

The source codes are available at <https://github.com/isu-enterprise/icc.xmitransform>,
<https://github.com/eugeneai/icc.mothurpim>.

Conclusion (the final one)

- ❑ Classic knowledge-based systems are powerful AI tools for solving wide class of recognition problems and synthesis of various kind: source code, data objects, control
- ❑ Contemporary means combine classic and new approaches
- ❑ Less dependent on computational resources (as compared to machine learning)
- ❑ Allow justification of the produced solutions
- ❑ Cover a larger set of tasks
- ❑ Natural for math science, and require higher level of AI education

Спасибо за внимание!

