Logic Programming and Non-monotonic Reasoning from 1991 to 2019: a Personal Perspective.

Michael Gelfond

5 июня 2019 г.

The Beginning

FIRST LPNMR WORKSHOP

1991, Washington

Organized by :

A. NERODE, W. MAREK, and V.S. SUBRAHMANIAN

Related work before the workshop included the development of

- superclassical non-monotonic logics such as *circumscription*, *default logic*, *autoepistemic logics*, etc.,
- declarative semantics of logic programs.
- Prolog and Datalog inference engines, etc.

But *comparatively small number of people understood the connections between these areas of research.*

Preface to the Workshop Proceedings:

"Since 1988, there has been growing interest in the relationship between logic programming semantics and non-monotonic reasoning. It is now reasonably clear that there is ample scope for each of these areas to contribute to the other.

Well developed implementation techniques for logic programming may, in the near future, make pragmatic non-monotonic reasoning systems more realizable.

At the same time, non-monotonic logics may provide us with the variety of techniques for incorporating different modes of negation in logic programming."

My own Research Program:

The preface describes common agenda, but most of us came to the workshop with our programs.

I wanted to discover/invent logical tools to better understand:

- **fundamental concepts we use in thinking about the world: *beliefs*, *knowledge*, *defaults*, *causality*, *intentions*, *probability*, etc.,**
- **ways of using these notions to *build software components of agents* – entities which observe and act upon an environment and direct their activity towards achieving goals.**

My believe was that logics in this toolkit should be *non-monotonic* and based on *Answer Set Prolog*.

In 1991 we were struggling to find ways to represent

- ***Defaults*** **and their** ***exceptions*****:**

"*Birds normally fly, penguins normally do not, wounded birds may or may not fly.*"

- ***Inertia Axiom*****:**

"*Things normally stay as they are*".

- ***Inheritance Principle*****:**

"*More specific information is more important than the less specific one.*"

- **etc.**

Difficulties:

In addition to genuine technical and conceptual difficulties, the problem was caused by a large number of terminological and methodological confusions, which were not necessarily recognized by the researchers.

Here are some examples:

Terminological Confusions:

1. **Is "default" a**
   - ***communication agreement*,**
   - ***statistical statement*,**
   - ***property of a "typical object"*?**

2. **Are we studying how to reason**
   - ***with* defaults?**
   - ***about* defaults?**

I concentrated on reasoning *with defaults* understood as *communication agreements*. Other choices led to creations of other research areas.

Methodological Confusions:

- ***What answers are expected?***

Given:
(a) commonsense knowledge about birds, including their *class hierarchy* and a default "*birds normally fly*",

(b) a fact $is(tweety, bird)$

Many people answer queries:

Does Tweety fly?

Is Tweety a penguin?

by *Yes* to first and *No* to second.

What answers are expected?

Now suppose that we have a single query

Is Tweety a penguin?

If the answer is "*No*" then the same answer should be given to questions about Tweety's membership in *every* subclass of birds. But, *this is clearly wrong*.

It seems that according to our unspoken communication agreement we should use CWA for penguins in the context of *flying* but not necessarily in other contexts.

***Remedy*: Work on clarification of informal story, intuitive meaning of basic terms, and formalization of contexts.**

Methodological Confisions:

- **Is an unintended answer caused by**

  a *bad formalism* or a *bad formalization*?

***Remedy*: Develop methodologies of language design and language use.**

- ***"Efficiency now!"* versus search for *"adequate representational power"*.**

***Answer*: Representation comes first.**

Other Problems:

- **Multiplicity of the original formalisms and their complexity.**
- **Unknown or non-existent mathematics.**
- **Non-intersecting communities.**

LPNMR played a very substantial role in alleviating these problems, and bringing new insights in the field.

Two specific achievements:

I concentrate on two specific achievements:

- **Principles of Language Design.**
- **Answer Set Programming (ASP).**

The first is chosen since, despite large amount of work on languages and serious attempts on standardization, not much has been written on this topic.

The second gives me an opportunity to explain why I believe that the currently prevailing view of ASP is unnecessarily narrow and talk about other possible views.

Principles of Language Design:

Language description should come with

- **Syntax.**
- **Informal and formal semantics.**
- **Methodology of use.**
- **Basic mathematical theory.**

Design/discovery of a good KR language should be recognized as a substantial achievement both in Logic and in CS.

Efficient implementation may add huge practical value to it, but it may come much later and be done by different people.
(However, early naive implementation may be a very useful tool for language designers and teachers).

Desirable Features of a KR language:

- *Clarity*: the language vocabulary should be carefully selected and have a clear intuitive and formal meaning.
- *Elegance*: the corresponding mathematics should be simple and elegant.
- *Relevance*: a large number of interesting computational problems should be reducible to reasoning in the language.

I believe that *Answer Set Prolog receives reasonably high marks on these criteria.*

Expressiveness:

• *The language should suggest systematic and elaboration tolerant representations of a broad class of phenomena of natural language, including belief, knowledge, defaults, causality, probability, etc.* •

Answer Set Prolog allows representation of an important form of *beliefs*, *causal effects of actions*, *recursive definitions*, etc.

Extensions such as CR-Prolog, Plog, LP^{MLN} allow adequate representation of *defaults* and *probabilities*.

- ***Sets and Aggregates.***

  After 20 years of effort there is substantial progress but no consensus yet.

- ***Possibility of Beliefs*, i.e. statements like**

  If at least one of answer sets of a program contains successful_attack *then every answer set must contain* ¬safe*.*

  In the language of *Epistemic Specifications* (early 90s) this can be written as

$$\neg \mathrm{safe} \leftarrow \mathrm{M}\ \mathrm{successful\_attack}.$$

  Original attempt to design the language failed, but recent interest led to substantial progress.

Smaller groups worked on formalizing

- *intentions*,
- *obligations*,
- *permissions*,
- *causal reasons of events and responsibility for them.*

Again, substantial progress, but much more effort is needed to complete this work.

Elaboration Tolerance:

• *It should be possible to expand a language by new relevant constructs without substantial changes in its syntax and semantics.* •

This is an extension of McCarthy's idea of elaboration tolerance of a program to that of a language.

There are important generalizations of Answer Set Prolog which simply expand the syntax of the original language by a new construct and slightly modify or expand the notion of the program's reduct.

Examples:

This is true for some languages allowing *aggregates*, for CR-Prolog, Plog, etc.

The first allow expressions

$$f\{X : p(X)\} \odot k$$

(where $\odot$ is $=$, $>$, etc.) in the body of rules.

The second introduces *consistency-restoring* rules of the form

$$head :+ body$$

which say that "if body is believed then head *may be believed* too but only if the part of the program consisting of regular rules is inconsistent."

Default in CR-Prolog:

A default $d(X)$ may be represented by program D:

$$p(X) \leftarrow c(X), \textbf{not } ab(d(X)), \textbf{ not } \neg p(X)$$

$$ab(d(X)) :+$$

The last rule stops the application of default $d(X)$ if it leads to a contradiction. Clearly,

$$\mathbf{D} \cup \{\mathbf{c(0)}\} \cup \{\mathbf{q(X) \leftarrow p(X)}\} \models \mathbf{q(0)}.$$

But, if we add $\neg q(0)$ this conclusion will be withdrawn.

Examples:

In the above languages semantics are given by generalizations of standard notion of the reduct.

Semantics of P-log requires definition of probability function on program's possible worlds.

Other languages, such as Epistemic Specifications, seem to require substantially bigger shift in perspective.

Stability:

• *Informally equivalent transformations of a text should correspond to formally equivalent ones.* •

According to this principle programs

$$p(1) \leftarrow card\{X : p(X)\} \geq 0.$$

and

$$p(1) \leftarrow card\{X : p(X)\} = Y,\ Y \geq 0.$$

should be equivalent, which, to the best of my knowledge, is true in only one of the extensions of Answer Set Prolog by aggregates.

Naturalness and Parsimony:

• *Natural language constructs and their representations in a formal language should be close to each other.*

Whenever possible, we should make sure that each important type of informal statements we want to express in our formal language should corresponds to exactly one language construct. •

These are guidelines, not requirements, but I would not recommend abandoning them without a serious, and well articulated, reason.

Language Design (continued):

Of course, many people disagree with some or all of the above principles. Some also favor additional requirements:

- **Reasoning in the language should be efficient.**
- **Entailment relation should satisfy some natural properties, e.g. if $T \models F$ and $T \models G$ then $T \cup \{F\} \models G$.**
- **Every program written in the language should be, in some sense, consistent.**
- **Kr-languages should extend the first-order classical logic.**

Some research and open discussion of these features can be very useful.

Answer Set Programming (ASP)

Next I will talk about another important achievement of our community: development of the answer set programming paradigm.

The term is somewhat ambiguous, and my interpretation of it differ from the prevailing view.

I spent a considerable amount of time discussing the issue in private conversations and now decided to use this opportunity to do it publicly.

First View: Solving Combinatorial Problems

Wikipedia: "*Answer set programming is a form of declarative programming oriented towards difficult (primarily NP-hard) search problems.*

It is based on the stable model (answer set) semantics of logic programming.

In ASP, search problems are reduced to computing stable models, and answer set solvers — programs for generating stable models—are used to perform search."

First View (Continued):

Many remarkable achievements:

- **Language extensions implemented in existing systems: choice rules, weight constraints, weak constraints, non-recursive aggregates, etc.**
- **Mathematics: Connections with SAT, SMT, intuitionistic logic, strong equivalence, complexity, etc.**
- **Algorithms: grounding, SAT and SMT related algorithms, learning, etc.**
- **Remarkable progress in system design and implementation.**

Second View: solving knowledge intensive problems.

Answer set programming is a form of declarative programming with emphasis on *knowledge intensive problems*.

It is based on representing knowledge in Answer Set Prolog or its extensions and

reducing computational problems to computing *parts of* answer sets of the program, or values of *functions* defined on collections of such sets.

This approach *shifts the emphasis from the algorithms to knowledge representation.*

From this perspective, development and teaching of KR-methodology becomes slightly more important. One should also learn multiple reasoning methods.

If the problem is reduced to *computing answer sets* of the program one would use ASP solvers.

***To answer a query* one may often prefer Prolog, Datalog, XSB, CLP, or other systems which only compute *parts of answer sets*.**

Other solvers may deal with optimization problems or answering probabilistic queries by computing *values of functions defined on collections of answer sets of a program*.

This view brings tasks like

- query answering from formal KB and from natural language,
- probabilistic reasoning,
- reasoning with constraints,
- causal reasoning, etc.

into the domain of ASP, as long as the logic used to solve these tasks is based on answer sets.

From this perspective ASP becomes as closely connected to classical KR as it is to SAT.

Tasks such as:

- development of more powerful KR languages,
- formalization of basic KR notions such as causality, obligations, intentions, etc.,
- finding new ways of reducing problems to reasoning with answer sets and, of course,
- development of efficient reasoning algorithms and systems

should provide ASP researchers with a huge number of research challenges.

Next few slides give an example of a research program related to this view.

Question answering from natural language

Learn how to

- **create a library of micro-theories encoding common-sense knowledge,**
- **design methods to**
  - **analyze the text-question pair to determine parts of the library relevant to it and combine the parts into one theory T,**
  - **translate the pair into formal representations F of the text and Q of the query in the vocabulary of T,**
  - **check if Q is a consequence of $\mathsf{T} \cup \mathsf{F}$,**
- **build systems which use this approach to answer sophisticated queries from texts written in *controlled (but non-trivial)* fragments of natural language.**

Basic hypothesis

- Large parts of commonsense knowledge can be encoded by a comparatively *small number of carefully crafted axioms*.
- The axioms can be *distributed*, i.e. stored in a WordNet like dictionary.
- Initial emphasis should be on *action verbs* with axioms describing *effects of actions*, and on *domain dependent ontology* with *membership* and *subclass* relations.
- Axioms may be written in *modular action languages* (whose semantics is defined in terms of ASP or closely connected to it.)

Example:

Three students and a professor entered the empty room. Soon, one person left.

1. **How many people are still in the room?** 3
2. **How many students are still in the room?** 2 *or* 3
3. **Is the professor still in the room?** *unknown*

It is difficult to automate this type of reasoning. Doing this will bring us closer to answering Winograd schema challenge and other similar intelligence tests.

What is needed to succeed?

- **Further development and mathematical investigation of modular action languages.**
- **Selection and careful axiomatization of important parts of common-sense knowledge.**
- **Understanding of reasoning with modules, e.g. combining modules into a coherent whole, effects of module modification, etc.**
- **Establishing closer connections between people working in related areas including those in computational linguistics and logic programming.**

Third View: ASP in Dynamic World

ASP is a form of declarative programming *oriented towards building software components of agents* – entities which observe and act upon an environment and direct their activity towards achieving goals.

Clearly, the agent should have general knowledge about the world and it's own capabilities and goals together with a history of its recent activities.

This can be expressed in action languages or other similar formalisms, and used by an agent involved in some form of *agent loop*.

Agent Loop

while goal is not achieved do

- *interpret observations,*
- *find an action leading to the goal,*
- *attempt to perform this action and update history with the record of the attempt,*
- *observe the world and update history with the record of observations.*

In ASP approach, reasoning operations in the loop such as planning, diagnostics, query answering, etc., are performed by ASP algorithms.

Additional attention should be given to combining them with sensory operations and to doing knowledge updates.

But, more importantly, the emphasis shifts to design of *agent architecture*: high-level description of agent's components and their tasks, communication between these components, the world, and, possibly, other agents, etc.

This causes a *new set of problems*. Among other things, we need to better understand how to

- **represent knowledge at different levels of granularity and establish relationships between these representations,**
- **represent knowledge about other agents,**
- **communicate declarative information between different agents,**
- **deal with streams of continually arriving data, etc.**

We also need to learn more about methodology and properties of knowledge updates.

From my standpoint all this belongs to the domain of ASP.

Conclusion:

The talk was, to a large extant, about my personal views and its conclusion will also going to be personal.

First, I am very happy to be part of this community.

It has always been unusually friendly and stimulating, and provided a great environment for scientific research.

Thanks to it, I was able to learn (sometimes surprising) answers to many questions I always wanted to get answers to.

Let me mention some of these surprises.

- **Solutions of long standing KR problems (defaults, inertia, etc.) proved to be so simple and natural that it is difficult to explain why it took so long to find them.**

- **Efficient inference algorithms were developed, implemented, and applied to practical problems.**

  In 1991 I strongly believed that it will eventually happen, but my time estimate was off by about 40 years.

  I changed the estimate after learning Smodels around 1996, but did not anticipate rapid improvement in efficiency of SAT algorithms and its influence on ASP.

- **Since my university years I had a difficult relationship with probability. It was perfectly understandable as a part of measure theory, but its applications were often confusing and unconvincing.**

  Part of the problem lies in the difficulty of constructing probabilistic model of the phenomena, which is often a function of our knowledge.

  To my great surprise attempts to combine logical and probabilistic reasoning (as in P-log) led to solving many of my problems and proved to be much simpler (at least technically) than I expected.

• **Finally, I continue to be amazed and delighted by connections between my LPNMR work and earlier interests in foundations of mathematics and constructive (intuitionistic) logic.**

I never suspected, for instance, that there would be a relationship between stable models and constructive logic of here-and-there, or between so called Glivenko classes and the semantics of Clingo.

Discovery of this connection already led to many new insights, but I have a strong feeling that it is trying to tell us something about an important phenomena we do not yet even aware of.

I hope to continue to be part of this community, to learn more about research programs of people and groups which comprise it, and to experience more pleasant surprises.

And I like to feel even more confident answering questions like:

What are the fundamental problems in our field?

What are the most important advances in our field in the last 5-10 years?

What are the question individuals and groups in our community are trying to answer?

THANKS FOR LISTENING

Samples of papers from 1991 proceedings

- **Pereira et. al.:** "*define a semantics that extend WFS for programs with classical negation, and which avoids the absence of models...* "
- **Dung et. al.:** "*generalize WFS to handle classical negation*"
- **Marek, Truszczynski:** "*compute intersections of autoepistemic expansions*"
- **Dix:** "*first step towards a characterization of the various semantics for LP by means of general properties of non-monotonic inference* "
- **Gelfond, Przymusinska:** "*study conditions which guarantee that expansion of program* $\top$ *by a definition of a new symbol does not change theorems of* $\top$"