IST 2001-37004 WASP
Working Group on Answer Set Programming
Work Package 5
Deliverable 5.2 – Panel Reports

Thomas Eiter, Wolfgang Faber, Axel Polleres, and Stefan Woltran

TUWIEN

1 Introduction

This document contains reports on three panels about important issues for Answer Set Programming (ASP), namely

- on "Extensions of the Language of Logic Programming,";
- on "Methodology of Answer Set Programming"; and
- on "Potential Areas for the Use of ASP".

In each of the panels members from WASP actively participated, either as panelists and/or organizers. The first two panels have been held at the

- 7th International Conference on Logic Programming and Non-Monotonic Reasoning (LPNMR-7) in Fort Lauderdale, Florida, January 6-8, 2004.

The third panel took place at the

- 9th European Conference on Logics in Artificial Intelligence (JELIA'04), held in Lisbon, Portugal, September 27–30, 2004.

The two panels at LPNMR-7 can be seen as supplement of the panel report on potential use areas of ASP, which has been postponed in the project plan to the end of the second project year. The issues discussed in these two panels, however, are also of importance for assessing potential use areas of ASP, which in turn have an influence on ASP development. The panels gave WASP Members the opportunity to exchange views and opinions about these issues among each other and with world-wide experts in the field of non-monotonic logic programming outside Europe. In particular, most of the panelists were from the United States, and their views and opinions provide valuable input for the WASP activities.

Finally, the proposed panel on "Potential Areas for the Use of ASP" was held at the JELIA'04 conference. Having a WASP steering-committee meeting at JELIA as well, many WASP members attended this panel (either as panelists or in the audience) but also numerous other JELIA participants showed up at this panel, resulting in an audience of about 50 people. Therefore, this panel as well could successfully bring together ASP-experts and experts from other fields of AI in order to discuss application areas for ASP and in which ways the ASP-community should develop.

In what follows, the three panels are described in detail.

2 Panel "Extensions of the Language of Logic Programming"

The first panel at LPNMR-7, on "Extensions of the Language of Logic Programming" took place on January 6, 2004.

The panelists were:

- Gerd Brewka (WASP Member, University of Leipzig, Germany),
- Michael Gelfond (Texas Tech University, USA),
- Victor Marek (University of Kentucky at Lexington, USA), and
- Terrance Swift (State University of New York, Stony Brook, USA).

The moderator of the panel was Mirek Truszczyński (University of Kentucky at Lexington, USA).

2.1 Questions

The panel chair raised the following questions:

Q1: Are any general principles of the design of Logic Programming (LP)/ASP languages emerging?
Q2: What is the role of applications as advisory force for the design of LP/ASP languages?
Q3: What is the role of constraint modeling and solving as a driving force for the design of LP/ASP languages?
Q4: Are these two contradictory, complementary?
Q5: Adopting SQL (database query languages) for ASP. Is there any potential, are there any lessons to learn there?
Q6: Programming language issues for LP/ASP: APIs, I/O, integration with other programming languages, ...
Q7: The next ASP language – your vision, three to five essential features.
Q8: How to accomplish that?

2.2 Statements of the panelists

The panelists were asked for their in situ statements on the questions above and further comments.

Gerd Brewka started with a plead for true language extensions. As he pointed out, extensions can be of different nature:

- On the one hand, "syntactic sugar" in terms of macros or application-dependent frontends, which can be easily reduced to ordinary LP/ASP.
- On the other hand, there are extensions which are not (easily) reducible to ordinary LP/ASP, and which require modifications of the semantics.

This raises the question what ordinary LP/ASP is. In Brewka's opinion, this is difficult to answer since there is a set of core LP languages rather than a single one, in particular for ASP. As for having true language extensions, he reminds of John McCarthy's famous "Modalities, si, modal logics, no" statement, which he recast to the LP/ASP issue as "Expressiveness, si, language extensions, no." However, Brewka does not hold this statement, and expresses his belief that we can not and should not do without extensions of LP/ASP.

As for Q2, Brewka said that Knowledge Representation has a tremendous potential, but the AI community does not take much notice of it. This is a serious problem which should be worked upon. As a particular example of the potential, he presented preference handling, illustrating it on some examples. The need for preference handling emerges in areas like abduction, revision, inconsistency handling, planning, diagnosis, decision making, configuration etc etc. As for LP/ASP, the detailed preference handling showing a rich ontology of prioritized LP/ASP languages that are needed. As an important issue he pointed out modular versus non-modular preference handling. As for Q3, Brewka said that work on constraints is important.

Gerd Brewka's vision of ASP was that it provides facilities for qualitative optimization, that it shall be used in hybrid systems, and that ASP, following the example of the Description Logic community, will be one of the most successful computational logic streams in the future.

Michael Gelfond opened his statement with reminding the purpose of a programming language, quoting C.A.R. Hoare:

> "*The main purpose of a programming language is to help the programmer in the practise of his art.*"

and its influence, quoting E.W. Dijkstra:

> "*A … programming language … influences our thinking.*"

He then proposed some principles for the design of declarative (programming) languages:

1. Take epistemology serious.
2. A new language feature should be accompanied by an in depth discussion of the methodology of its use.
3. Such a method can only be developed on the basis of a solid mathematical theory.
4. Use principles of the design of procedural languages.

As for item 1, Gelfond reminded that we represent *knowledge* about different items, objects and relations, e.g., hierarchy of classes, causal effects of actions etc. Under this perspective, the epistemological assumptions of ASP have to be critically reviewed. Knowledge about relations between objects is given by facts and rules, which together with the semantics assumption elicit further relations.

As for item 2, Gelfond thinks that the language methodology should provide a unique translation of the important natural language constructs into their formal counterparts. As an example, he discussed different translations of the statement "A block can not occupy different locations."

Regarding the questions of the moderator, on Q2 Gelfond said that KR has a main role, but that design of the language and implementation should be separated. On Q3, he said that the design of inference engines is important, and on Q7, that the next ASP language should provide

- universal quantifiers, of the form $a \leftarrow (\forall x p(x))q(x)$,
- efficient reasoning with numbers,
- reasonably efficient and general probabilistic reasoning, and
- learning.

Victor Marek focused in his reply on linking ASP and SQL as well as programming languages (i.e., Q5 and Q6). As he pointed out, models of a logic program can be viewed as Boolean strings, which can be naturally stored in database tables, where atoms are viewed as columns and the entries are 0 or 1, depending on whether an atom is false or true, respectively.

Using this, embedding of ASP into SQL could be realized, following the scheme

```
select <atoms>
from stab(P) ...
where <condition>
at_most <k>
```

where `stab(P)` is the collection of stable models of a logic program P and `<k>` bounds the number of models considered.

Marek pointed out that a slightly subtler approach has been already suggested, in which relational structures are encoded by such tuples representing models, and that first order logic is used as the language for conditions. In this context, it is important that conditions can come from a language *not* related to ASP (e.g., Smodels has hooks for this).

For integrating imperative languages and ASP, Marek proposed that instead of building a declarative interface to ASP, classes or templates should be build which can be pulled into an imperative language. This has been tried before, and as examples for other declarative languages Marek mentioned Prolog solvers for JAVA, and a C++ library for CLP by ILog. Within ASP syntax, escapes to an imperative language from within the clauses have already been proposed in the ASP community.

Marek said that to pull either ASP into imperative languages (or conversely), suitable APIs are needed, and inspiration for that can be taken from ODBC and especially JDBC drivers, and other interfacing software may be the key.

Terrance Swift opened his statement with the following questions:

- To what extent is ASP programming?
- Is ASP a specification or an assembler language?

In order to answer these issues, Swift pointed out first some well-known model driven architectures and languages:

- Ontology-driven architectures and agents

    - OWL
    - Ontology-driven agents
  - UML-driven architectures
    - Java IDEs, Delphi.net
    - Business engines like Bold, Epic, J2EE
    - OCL expressions
  - Workflow-type models (e.g. WfMC)
    - Open Business Engine

Swift then raised and answered some more questions:

- What particular strengths does ASP bring to these problems?
  In his opinion, these are
  - Non-monotonic reasoning (and preferences)
  - Fixpoint operator
  - Impressively fast engines (usually)
- How should ASP be used in these BIG systems?
  - As a specification language?
  - As an assembly language? e.g. a reasoning co-processor?
- What are the biggest weaknesses of current ASP systems?
  - Nearly non-existent user interface (need to think beyond emacs!)
  - APIs inadequate for widespread use

Swift then pointed out that using APIs in ASP (e.g., the C-API of Smodels) is a "labor of love" for the following reasons:

- It is difficult to translate atoms to internal representation.
- It does not translate weight constraints.
- All constants must be known before commit.
- There is no incremental commit.
- All rules must be known before commit.
- There are no "background theories."

In comparing ASP with CLP, Swift pointed out the following observations.

- Good operational understanding of how constraint store can fit with Prolog, C++, etc.
- CLPQR is used in Sicstus, XSB, Yap, etc but not in ASP yet.
- Constraint handling rules allow programmers to model new constraint theories.

Finally, Swift pointed out some possibilities for improvements:

- Incremental commit of rules and programs
  - For compilation, and for "abductive" programs.
  - Would allow incremental addition of ASP rules on forward execution
  - Would allow retraction of rules on backtracking via trail hooks, backtrackable updates
  - Would allow "pruning" of search space, like CLP
- "Background Theories" e.g. inheritance
  - Allow guards in the bodies of rules
  - if the guards fire, a callback to an outside system is made, to incrementally ground call.
  - Semantically meaningful, programmer is only doing lazily what she would otherwise do eagerly.

2.3 Discussion and remarks from the audience

In the general discussion, Victor Marek expressed with regard to Swifts's questions his view that LP may be conceived as an assembly language, which is too complicated for the real programmer.

Michael Gelfond replied that in his view, LP and ASP in particular is both an assembly and a specification language. Furthermore, in his view it is not the case that ASP is not widely used because it is a complicated language, but rather because few people know about it at present.

Mirek Truszczyński adds to this that in his opinion, the distinction line between an assembly and a higher level programming language is by the handling of the programs through the engine.

With respect to using ASP for applications, an integration in hybrid systems, in extension to Victor Marek's comments the development of libraries is pointed out as an important issue. Michael Gelfond suggests to focus on two kinds of libraries:

– Nontrivial libraries which require a lot of development.
– Planning libraries.

He also adds that the development of ASP application is not well-accepted by the research community, and that it is difficult to publish such work. He sees this a major problem for the dissemination of ASP to applications. Gerd Brewka seconds this and raises the issue how to get credit for applications in the research community. There is some discussion but no real conclusion.

The views on applications are different. Chitta Baral points out Microsoft's initiative for next generation knowledge bases, and in particular the AP Chemistry Test as a knowledge-based systems challenge, and suggests that the ASP community works on or towards this application. An application like this could attract a lot of interest on ASP. In connection with this, John Schlipf asked whether a killer application might help, but remained reserved about this.

Mirek Truszczyński said that perhaps the ASP researchers should not do the development of applications themselves, but should see about that applications are done. This, however, leaves the question how knowledgeable and well-skilled persons can be found for the applications. According to Michael Gelfond suggestions, students could be for an initial step be the ideal candidates.

Finally, Victor Marek points out that, after a longer period of foundational research, now ASP systems are available, and expresses his confidence that applications will follow.

2.4 Summary and Impression

The panel on extensions of the language of Logic Programming has considered a number of important issues in this respect, and in particular discussed this issue for the Answer Set Programming Paradigm.

There seems to be general agreement that extensions to the language of LP/ASP are necessary, and that there is a variety of possibilities for such extensions. Some guiding principles have been pointed out for the design of such extensions, which might prove valuable. While applications and usage of LP/ASP might be a driving force for LP/ASP extensions, the knowledge representation and epistemological perspective of LP should not be forgotten about, and carefully respected.

Couplings of ASP with other programming languages, or database query languages, is an interesting issue, which at present is not solved and requires more work.

Overall, the panel and the discussion made a very optimistic impression about the further development of extensions to ASP and LP.

3 Panel "Methodology of Answer Set Programming"

The second panel at LPNMR-7, on "Methodology of Answer Set Programming," took place on January 8, 2004.

The panelists were

- Chitta Baral (University of Arizona, USA),
- Michael Gelfond (Texas Tech University, USA),
- Tomi Janhunen (WASP Member, Technical University of Helsinki, Finland), and
- Hudon Turner (University of Duluth, USA).

The moderator of the panel was Vladimir Lifschitz (University of Texas at Austin, USA).

3.1 Questions

In the beginning of the panel, Vladimir Lifschitz raised the following four questions on the methodology of Answer Set Programming:

Q1: How do we write provably correct answer set programs?
Q2: How do we optimize answer set programs?
Q3: When we start developing an answer set program: Is it necessary to decide beforehand which solver we use?
Q4: What is the relations between ASP and Prolog programming?

3.2 Statements of the panelists

The panelists were asked to comment on these questions and give their statements.

Chitta Baral raised in his comments one more question: In most approaches, ASP solutions to a problem are represented by answer sets. While doing answer set programming, is one still interested in entailment?

His answers to questions 1.-4. are:

Ad Q1. Although most languages/language extensions have a clean formal characteristics, we still lack of more useful building block results (like e.g. splitting sets) for proofs.

Ad Q4. In answer set programming solutions are answer sets, while in prolog, solutions are substitutions. If we deal with lists/sets of assignments, this makes a major difference.

An interesting point is when we deal with problems in P vs problems in NP. For problems in P there are nice efficient solutions in Prolog, while in ASP we have not so much control on the search and we have to effectively indirectly guess solutions. A motivating example is List concatenation:

```
conc([],L,L).
conc([H|L1],L2,[H|L3]) :- conc(L1,L2,L3).
```

While in Prolog this can be done efficiently, in ASP we can represent the data structures to some extent but guess all solutions, i.e. lists in some sense.

Chitta reminded at this point Piero Bonatti's approach for finitary programs and restricted use of function symbols in this context.

Michael Gelfond made the following statements:

Ad Q1. For correct programs, start with - empirical evidence - proofs are needed based on precise specifications, e.g. by

(a) transition diagrams
(b) electrical circuit viewed as a function from input to output
(c) inheritance hierarchies based on independently defined semantics

The problem which comes up is here complete vs. incomplete specification.

Ad Q2. First, one needs a good design. Second, small tricks, which preserve equivalence in some sense (strong, unif., weak). Michael shows two examples:

1.
```
 a :- -b.        s       a :- not b.
-b :- not b.   <===>    -b :- not b.
```

2.
```
:- a,b.        add
:- -a,c.       ===>    :- b,c.
```

Ad Q4. We miss function symbols.

T. Janhunen made the following comments:

Ad Q1. The correctness of solvers is crucial (can at least be checked on pseudocode-descriptions) - independent answer set verifiers can be introduced.

Ad Q2. Optimization is always a tradeoff between length of the program P (compact representation) and execution time T needed to solve the program.

Ad Q3. As for the compact representation: Suitable Lemmas might often help to solve a problem, so the most compact representation is not always the best wrt. T.
The relation of T and P, i.e. efficiency of different encodings might be highly dependent on different solvers.
Equivalence preserving transformations are and different notions of equivalence are an important point, effects to performance however, are not completely clear.
One possibility would be "blind" optimizations (e.g. by dropping rules), using equivalence verifiers as recently proposed.

Ad Q3. As for the proprietary syntax: It would help if suitable translations should be provided wherever possible.

- normal and disjunctive logic programs now form a safe subset to start with.
- problems might arise if different solvers treat recent syntactic extensions differently.
- Is a low-level format (like DIMACS for SAT-Solvers) achievable?

Ad Q4. ASP can be implemented in Prolog, but also the following issues might be considered:

- Remote ASP solver calls can be realized in Prolog
- Prolog has problems with negation as failure
- Order of subgoals matters in Prolog
- we should identify common subsets.

Hudson Turner commented as follows.

Ad Q1. He points out again that the generate and test paradigm is a nice general approach for this direction. Furthermore, Turner's reported about his own experience from using ASP in reasoning about actions.
Proving correctness means proving one to one correspondence between models and solutions.
Theorems like splitting set and generalizations are useful for proving correctness.
For devising translations maybe a few ideas are needed. These ideas should be used as *idioms*. One example is completeness:

```
 a :- not -a.
 -a :- not a.
intertia:
f1 :- f0, not -f1.
-f1 :- f0, not f1.
```

Are these intuitively correct? With these idioms in mind high-level languages work. Implementations of high-level languages are important to test bigger examples!

3.3 Discussion and remarks from the audience

After the panelists made their statements, the possibility to comment on the questions to the panelists and to raise questions was granted to all attendants.

The first who commented was Marc Denecker: He remarked that the different interpretations of "not" and ":-" still give him an unease feeling when actually formalizing problems.

Mirek Truszczyński made a general comment to Q1: What does it mean that a program is correct?

Alexander Bochman raises the controversial point that DLP is not a Logic at all. This led to some discussion.

Marc Denecker commented on Q4 that Prolog forces one to use different vocabulary than one would use for an intuitive declarative description of a program, because it disallows guesses and there is nothing such as constraints. This enforces "metaprogramming."

Another interesting point raised by Tomi Janhunen in personal discussion off-track concerning Q1: We have only a semantics in terms of the whole program, not in terms of single rules which makes correctness checks harder of course, because one always has to consider the whole program. Can we e.g. define the semantics of a rule separately wrt. to a given program? Does this make some sense?

3.4 Summary and Impression

In summary, the statements of the panelists and the general discussion have raised the following impressions and observations:

- Language extensions of ASP, and function symbols in particular, seem to be an issue, and should be further explored.
- There is an interest in formal building-block results like splitting set theorem, equality testing, etc which allow to "engineer" Answer Set Programs.
- Some researchers, including Michael Gelfond and Hudson Tuner, point out that problem encodings without complete/total information tend to be incorrect. This, of course, challenges common-sense reasoning, and has to be seen in connection with elaboration tolerance.
- The view of ASP as a "logic" and formalization of common-sense reasoning is a predominant issue for some researchers, while the full power of ASP as a problem solving paradigm (in the computational sense) is of lesser interest to them.
- Interaction between ASP-Solvers and Prolog systems might be promising, as well as interacting applications.

4 Panel "Potential Areas for the Use of ASP"

This panel took place at the 9th European Conference on Logics in Artificial Intelligence (JELIA'04), Lisbon, Portugal, on September 27, 2004. The panelists were

- José Júlio Alferes (Universidade Nova de Lisboa, Portugal),
- Nicola Leone (Universitá della Calabria, Italy, WASP member),
- Torsten Schaub (Universität Potsdam, Germany, WASP member),
- Ken Satoh (National Institute of Informatics, Japan),
- Yannis Dimopoulos (University of Cyprus, Cyprus, WASP member), and
- Tomi Janhunen (Helsinki University of Technology, Finland, WASP member).

The panel was organized by the WASP members

– Gerhard Brewka (Universität Leipzig, Germany),
– Jürgen Dix (Technische Universität Clausthal, Germany), and
– Thomas Eiter (Technische Universität Wien, Austria).

with grateful help from the local JELIA Organizing Committee.

4.1 Questions

The panel chair, Jürgen Dix, listed the questions posed to the panelists:

Q1: Killer Applications for ASP (should one care about?)
Q2: For which application areas is ASP well-suited, for which it is unsuitable?
Q3: In which ways should ASP be used?
Q4: What is the advantage of ASP over other approaches (e.g. LP techniques) for applications?
Q5: Which language extensions and features of ASP are missing to make it useful for practical applications?
Q6: Is ASP a hype that will soon be over?

4.2 Statements of the panelists

J.J. Alferes started with the following statements:

Ad Q1. People should care about killer applications. Since ASP is already application-oriented, a killer application is needed.
Ad Q2. ASP is used for several application areas, for which it is well-suited, while it is sometimes also used in areas where constraint solvers (CLP) are better suited (e.g. the 8-queens problem).
Ad Q3. ASP should be used in conjunction with other systems (e.g. smodels+XSB, DLV+DB, etc.) for full applications. Important issue: Localization (magic sets, reducts etc).
Ad Q5. Debugging, visualization. In general: Programming tools (he refers to the demo by Thomas Linke on the same day). This is a difficult, but important task.
Ad Q6. *"I don't hope so."*

Nicola Leone made the following comments:

Ad Q1. Agrees with J.J. Alferes to some extent: One should definitely care about killer applications. *"But is there a single killer application?"*
Ad Q2. Connected to Q1. Agrees with J.J. Alferes that often ASP is used improperly. But there are many areas where it is well-suited: Complex knowledge, incomplete knowledge and reason about it.
One concrete case is the data integration setting presented in the systems track: Interaction with a database is needed, "evolution" of a data has to be considered, reasoning by cases and solving a co-NP task is needed (so well-founded semantics is not appropriate), query answering is fundamental. Magic Set techniques make this task practical.

Another application area is reasoning about ontologies. Query answering over huge and complex amounts of data is necessary there.

Yet another area is the classification of documents (as presented in the system session before the panel), where constraint satisfaction is definitely not appropriate (since there is no query).

Ad Q3. ASP should usually be used by knowledge engineers and be encapsulated in a "computational kernel". Frontends should be built around this kernel in order to grant non-expert users easier access.

Ad Q5. Agrees with J.J. Alferes. The need for debugging tools is emphasized. Other minor issues are raised (e.g., real numbers).

Torsten Schaub answered the questions as follows:

Ad Q1. One should care about applications. If they are killer, that's fine. Candidates: Michael Gelfond etal's work for the Space Shuttle. Chitta Baral etal's work in Bioinformatics. This is the way to go: Try to move to application areas in other communities. The "Guess and Check" Paradigm can often be fruitfully applied there.

N-queens is useful for teaching. Emphasis on visualizations: E.g., Sokoban, Quake (as presented by Alessandro Provetti in the system session before the panel). Wrappers exist around ASP for visualization.

Ad Q3. there are two directions: "general high-level language" vs. "assembler language" (like SAT). The latter is used for encoding other formalisms.

Ad Q4. Shift from query oriented (Automated Theorem Proving view) to model oriented (Knowledge Representation/SAT view). Working on models is yet to be explored.

Ad Q5. The "assembler language" direction gives rise to language extensions. E.g., model checking, Linux configuration (refers to the work at Helsinki University of Technology).

Ad Q6. ASP is necessary. If it does not stay, it will be re-invented.

Ken Satoh first presented his general view on ASP. Its key issue is to provide declarative and concise specifications, but no (explicit) control. Therefore, we have:

+ people not wanting a search algorithm are happy;
– no direct influence and control.

Programming is different than for standard logic programming, but somewhat similar to linear/integer programming. Satoh's answers to the concrete questions are:

Ad Q1. A candidate for a killer application could be from the area of linear/integer programming, where modeling using numbers is hard.

Ad Q2. Not suitable for interactive systems.

Ad Q3. Use ASP as a batch system, combined with standard (procedural) languages. E.g., Quake as in the presentation by Alessandro Provetti in the JELIA'04 system-demonstration session.

Ad Q5. Missing: declarative control mechanism, combination methods with procedural methods.

Ad Q6. *"Work hard!"*

Yannis Dimopoulos pointed out his general view: ASP is a CLP language; in particular LP with SAT and minimality. Competitors are identified as SAT and constraint programming. The main weak points for ASP is the lack of debuggers and programming tools.

ASP needs to exploit the relation to deductive databases and to inductive logic programming (e.g., bioinformatics). All applications where minimality comes into play are important. Agreement with previous panelists that 8-queens is not well-suited as ASP application, because there is no minimality involved.

A bad point is that an "LP language" is not learned understood by many people. In contrast, CP networks are easy to understand with "standard education".

Tomi Janhunen concludes the round of panelists with the following comments:

ad Q1+Q2. Configuration. One issue is that specification is not always clear. Therefore revising the KR language is often necessary. Examples: Chemical engineering, chromatography. Problems: Inexact definitions, errors in the data.

ad Q3. Programmers often do not know how to characterize the solutions. ASP seems to be suitable for "trial and error" approaches.

ad Q4. Compared to Prolog, ASP is more efficient and truly declarative. Mathematical definitions exist.

ad Q5. Modularity is needed, e.g. subprograms, aggregates, and other abstract features. Software Engineering techniques and tools are needed. Combination with randomized computations (e.g., random assignments for student exercises) is desirable.

ad Q6. Engineers are already trained to use ASP, so chances are low that ASP is just a hype which will disappear soon. So the current state is promising.

J. Dix thanks the panelists and opens the discussion.

4.3 Discussion and remarks from the audience

L.M. Pereira shares views with J.J. Alferes. ASP should be integrated into "standard" LP systems to get the best out of the two worlds. He mentions problems with infinite grounding and that interpreters are missing and problematic to produce. Top-down is not feasible for ASP.

For abduction there is the problem that it is usually encoded in ASP as an even loop, therefore relevance is not guaranteed. Magic Sets can be used, but only to a certain degree. Finally he identifies a problem with odd loops, especially with respect to updates. Proposes an alternative semantics termed "revised stable models".

Marco Cadoli first points out three important features of Ilog solvers:

1. Not rule-based.
2. Rich syntax.
3. Declarative way to express procedural aspects.

Even with this, people are reluctant to buy these solvers. Along these features, ASP needs to:

1. Hide rules.
2. Enrich the syntax.
3. Provide a declarative way to "guide" search.

Yannis Dimopoulos asks in response to L.M. Pereira: *Why shift from "top-down" (goal-oriented, resolution) to "bottom-up" (model-oriented, Davis-Putnam)? The answer is efficiency.* The well-founded semantics is not sufficiently expressive.

F. Banti comments on the relation between ASP and Artificial Intelligence, in particular in the area of agents. He points out that ASP has something to say here, and ASP should be seen as a tool for agents, although ASP is not suitable for everything in an agent. However, one should aim at integrating ASP and goal-oriented approaches.

4.4 Summary and Impression

The panel on "Potential Areas for the Use of ASP" considered and deepened a number of issues, which already have become apparent in the two previously held panels. In particular, the question for "killer application" was once more central. On the other hand, it was addressed that sometimes ASP is used for problems where other programming languages are better suited. However, the suggestions how to use ASP still show a broad application range. As well, there was a certain focus on discussing which particular directions ASP should take in the future, in particular pointing out the need for debugging methods which lack both theoretical foundations and sophisticated practical realizations.

With an audience of about 50 people this panel definitely was a remarkable success, since many experts from other fields in AI attended the panel. Therefore this event significantly contributed to the Working Group's goal to make ASP well known in the scientific community.

5 Concluding Remarks

The three panels show that there is a vivid discussion on fundamental issues in ASP going on. This is good news since one has to bear in mind that ASP as a programming paradigm is still at an early stage and therefore a broad and intensive process of discussion is necessary in order to fine-tune ASP into a successful and widely accepted programming language. We summarize some of the fundamental questions raised in all panels:

- there are different opinions, which in direction ASP should evolve: (i) ASP seen as a "core", resp. "assembler"-language, with several front-ends providing dedicated application-specific interfaces to users; (ii) with focus to be used in connection with other languages, e.g. database query languages; or (iii) development of ASP into a "general high-level language" providing numerous libraries etc.
- Especially in connection with (iii), some shortcomings have been identified, in particular the lack of utilities which support the programmer. It was mentioned several times that debugging and visualization tools are issues which have to be solved (both theoretically and practically). As long as answer-set programming lags behind other programming paradigms in these particular aspects, it would be hard to turn ASP into an accepted and popular methodology for a broader class of users.

– Finally, there have been discussions which applications are well suited for ASP, and in which areas ASP could succeed compared to established competitor paradigms. It was mentioned that concrete promising realizations using ASP, including a diagnosis system for the space shuttle or in the area of bio-informatics should guide the direction. Further examples mentioned have been data integration and reasoning over complex knowledge, for instance, within ontologies. This application-oriented view also goes conform with comments on more theoretical comments on application areas, where the key words in the panelists' comments are combinatorial problems combined with a minimization problem.

To conclude, all the panels announced an optimistic view on the further development of ASP. This view was also supported by attending scientists which are not that tightly linked to the ASP-community. The overall impression about the outcome of these panels is therefore definitely positive and gives an optimistic opinion on the future of ASP.