Workshop on Nonmonotonic Reasoning, Answer Set Programming and Constraints

G. Brewka, I. Niemelä, T. Schaub, M. Truszczynski

November 7, 2002

Scientific highlights of the event

Answer set programming is an emerging programming/problem solving paradigm. The fundamental underlying idea is to describe a problem declaratively in such a way that models of the description provide solutions to problems. One particular instance of this paradigm are logic programs under stable model semantics (respectively answer set semantics if an extended class of logic programs is used). Tremendous progress has been made recently in this area concerning both the theoretical foundations of the approach and implementation issues. Several highly efficient systems are available now which make it possible to investigate some serious applications.

The talks of the workshop were centered around the following main research topics:

- Useful language extensions and their theoretical foundations, with a particular focus on cardinality, weight and other types of constraints.
- Preferences in answer set programming and their implementation, where the preferences considered are among rules, among literals, or among disjuncts in heads of rules.
- Implementation techniques for answer set solvers. Several new methods or improvements of existing methods were presented, some of them based on highly efficient existing satisfiability solvers.
- New attempts to handle programs with variables. Existing solvers produce the ground instantiation of a program before computing answer sets and disallow function symbols. More flexible and less space consuming techniques are needed for large applications.
- Applications of the answer set paradigm in planning, scheduling, linguistics etc.

In addition to the talks a system competition took place during the workshop. Five systems participated in the competition, namely *dlv* (TU Vienna/Univ.

Calabria), *Smodels* (Helsinki UT), *ASSat* (Univ. Honkong), *cmodels* (UT Austin) and *aspps* (University of Kentucky). In a meeting at the beginning of the seminar the participants agreed about the benchmark problems to be used in the competition. The problems were encoded and tested and results presented in a plenary session at the end of the week.

Another topic of interest was standardization. There was an panel on the subject followed by open discussion. A general feeling was that the matter of standardization is a topic that requires a thorough attention on the part of the community in the near future.

Training

Among the participants of the workshop were 11 young researchers, most of them PhD students. The students were allotted the same amount of time as everybody else for their talks to make sure they received enough attention from senior scientists. For many of the students it was the first time they presented their results/projects to an international audience. The students had a chance to discuss with world leading researchers in their area. This will certainly have an impact on their future work.

European added value

It is fair to say that in the field of answer set programming, and in particular in implementing advanced answer set solvers, Europe is currently on par with research in North America, if not leading. There is a number of European research groups active in this area. The EC just started to fund a Working Group on Answer Set Programming. The major goals of the Working group are the further advancement of the theoretical understanding of ASP (this includes the investigation of new potentially useful language constructs and their semantics), the further development of efficient advanced reasoning systems which make ASP techniques widely available (this includes the development of front ends for specific application problems), and the investigation of the applicability of ASP to areas such as planning, configuration, encryption, verification, knowledge extraction and others.

During the seminar the kickoff meeting of the working group took place, and the members had an excellent opportunity to get first hand information about current research developments in each group.

Given the numerous application areas for which promising answer set programming solutions already exist today, we expect tremendous economic benefit of this research. The seminar was important to keep Europe at the forefront of research in this area.

Public Outreach

Answer set programming is a new declarative programming methodology. The basic idea is that programmers, rather than having to specify how a computer should solve a problem, just describe what the problem is. Each model of the problem description then provides a possible solution to the problem. The exact notion of a model used here depends on the language used for describing problems, but in all cases the models (also called answer sets in this context) can be thought of as sets of facts representing what is true and what is false.

Although theoretical foundations have been laid and some highly efficient implemented systems are available, there are still numerous challenging scientific questions which need to be answered: improved implementation techniques, extensions of the declarative languages which facilitate the problem description, methods for applying these techniques to problems like planning, scheduling, configuration etc. Contributions to all of these topics were presented and discussed during the seminar.

Abstracts

Logic Programming with Set Constraints
V.W. Marek and J.B. Remmel

We generalize the set constraints of the form kXl present in the current implementation of smodels ASP solver to the situation where the constraints are arbitrary families of subsets of a given finite set.

It turns out that the Niemela-Simons-Soinanen construction of stable models smoothly generalizes to that context. The model theory of programs with set constraints requires studies of nondeterministic operators in complete lattices. Those operators, even when monotonic, do not need to possess fixpoints. We find conditions that ensure the existence of such fixpoints, and show that stable models are, indeed, fixpoints.

Logic Programs with Weight Constraints
Ilkka Niemelä

Abstract: A novel logic program like language, weight constraint rules, is introduced for answer set programming purposes. It generalizes normal logic programs by allowing weight constraints in place of literals to represent, e.g., cardinality and resource constraints and by providing optimization capabilities. It has a declarative semantics which extends the stable model semantics of normal programs. For ground programs the computational complexity of the language is shown to be similar to that of normal programs under the stable model semantics. A simple embedding of general weight constraint rules to a small subclass of the language called basic constraint rules is devised. The language has been implemented in the Smodels system (http://www.tcs.hut.fi/Software/smodels/) using a two level architecture. A front-end compiles general weight constraint

rules to basic constraint rules for which an efficient search procedure for computing stable models has been developed.

An Approach to Capture Stable Models with Classical Ones
Tomi Janhunen

In the talk, I address the relationship of two important ASP formalisms, namely normal logic programs (under stable model semantics) and sets of clauses (under classical models). It is easy to reduce the latter to the former, but translations in the other direction are more or less troublesome. In simple cases (e.g. tight programs addressed by Lifschitz), Clark's program completion does the job, but programs containing positive loops are not necessarily covered. There are also theoretical results indicating that removing positive loops cannot be done in a modular and faithful way. Despite of these difficulties, I present in this talk a non-modular and faithful translation. As distinctive features (compared to earlier approaches) we obtain a bijective correspondence between the models and the size of the translation grows linearly in the length of the input times the logarithm of the number of atoms is the input. I have also developed an implementation of the translation. The results from my preliminary experiments look promising, although the performance of the implementation, used together with a leading sat solver zchaff, is still behind smodels. To conclude, further optimizations are needed to really compete with smodels.

ASP with propositional schemata
Deborah East, Mirek Truszczynski

We show that propositional logic and its extensions can support answer-set programming in the same way stable logic programming and disjunctive logic programming do. To this end, we introduce a logic based on the logic of propositional schemata and on a version of the closed-world assumption. We call it the extended logic of propositional schemata with CWA (PS+, in symbols). An important feature of the logic PS+ is that it supports explicit modeling of constraints on cardinalities of sets. In the talk, we present properties of the logic PS+ and discuss its implementation, program we call aspps. that computes models of PS+ programs.

The SLP System: An Implementation of Super Logic Programs
Stefan Brass

Super logic programs were introduced by Brass, Dix, Przymusinski in a KR'96 paper. They consist of formulas that can use arbitrarily nested propositional connectives, plus default negation and variables with certain restrictions (default negation can only be used in negated context, i.e. in the "body", and variables must satisfy the usual allowedness condition). Super logic programs include disjunctive logic programs and integrity constraints (rules with empty heads). We showed that for this class of nonmonotonic theories, the definition of Przymusinki's static semantics can be significantly simplified and developed

a model-theoretic characterization that can be used for query evaluation under the static semantics.

The SLP system can compute disjunctive answers to queries under the static semantics using this characterization. SLP seems to be still the only implementation of this semantics. Currently, the disjunctive stable model semantics is being added to SLP. SLP can already compute stable models, but is still very slow compared to systems like DLV or smodels. However, further development of SLP will improve the efficiency.

SLP was developed in C++ and has currently 21.000 lines of code. The source code is available from [http://www.informatik.uni-giessen.de/staff/brass/slp/] and [http://purl.oclc.org/NET/slp/]. SLP can be tried via a web interface, so a local installation is not required.

In this talk, we explained the main algorithms used in SLP: The hyperresolution operator for conditional facts, the elimination of duplicate and subsumed disjunctions, the computation of the residual program, the computation of static interpretations for the remaining negations, the minimal model generator, and a simple generator for stable models using Clark's completion.

Reconstructing the Evolutionary History of Indo-European Languages using Answer Set Programming

Esra Erdem, Vladimir Lifschitz, Luay Nakhleh, Donald Ringe

The evolutionary history of languages can be modeled as a tree, called a phylogeny, where the leaves represent the extant languages, the internal vertices represent the ancestral languages, and the edges represent the genetic relations between the languages. Languages not only inherit characteristics from their ancestors but also sometimes borrow them from other languages. Such borrowings can be represented by additional non-tree edges. This paper addresses the problem of computing a small number of additional edges that turn a phylogeny into a "perfect phylogenetic network". To solve this problem, we use answer set programming, which represents a given computational problem as a logic program whose answer sets correspond to solutions. Using the answer set solver SMODELS, with some heuristics and optimization techniques, we have generated a few conjectures regarding the evolution of Indo-European languages.

Optimal models of disjunctive logic programs: semantics, complexity, and computation

Francesco Scarcello

Almost all semantics for logic programs with negation identify a *set*, $SEM(P)$, of models of program P, as the intended semantics of P, and any model M in this class is considered a possible meaning of P w.r.t. the semantics the user has in mind. Thus, for example, in the case of stable models, choice models, answer sets, etc., different possible models correspond to different ways of "completing" the incomplete information in the logic program. However, different end-users may have different ideas on which of these different models in $SEM(P)$ is a

reasonable one from their point of view. For instance, given $SEM(P)$, user U_1 may *prefer* model $M_1 \in SEM(P)$ to model $M_2 \in SEM(P)$ based on some evaluation criterion that she has. In this paper, we develop a logic program semantics based on *Optimal Models*. This semantics doesn't add yet another semantics to the logic programming arena – it takes as input, an existing semantics $SEM(P)$ *and* a user-specified objective function Obj, and yields a new semantics $optsem(P) \subseteq SEM(P)$ that realizes the objective function *within the framework of preferred models identified already by* $SEM(P)$. Thus, the user who may or may not know anything about logic programming has considerable flexibility in making the system reflect her own objectives by building "on top" of existing semantics known to the system. In addition to the declarative semantics, we provide a complete complexity analysis and algorithms to compute optimal models under varied conditions when $SEM(P)$ is the stable model semantics, the minimal models semantics, and the all-models semantics.

Benchmarking ASP-Systems
Gerald Pfeiffer

With the recent arrival of new Answer Set Programming (ASP) systems and improved versions of already existing systems, we have seen an increasing interest in comparing and benchmarking in our community.

In this presentation we will try to lay down some generally applicable (and acceptable) guidelines on how to perform benchmarking, as well as on the kind of information to publish alongside the actual results. In the second part of the presentation, we report results from extensive benchmarks we performed for the DLV, Smodels, and ASSAT systems. Benchmarks problems were taken from a variety of domains such as deductive databases, graph theory, planning, and optimization, which put a focus on various knowledge representation and implementation aspects of ASP systems.

Knowledge representation, reasoning and declarative problem solving with answer sets
Chitta Baral

In this talk we present some of the important results about logic programming with answer sets. We first start with a simplified nomenclature of various classes of logic programs, and refer to the general class as AnsProlog*, where 'Ans' comes from 'answer set' and 'Prolog' comes from 'Programming in logic'. We then present several simple modules for programming using AnsProlog*. These simple modules include, modules for 'choice', modules for quantified boolean formulas, modules for doing aggregation, modules for solving the frame problem, and modules for systematic removal of closed world assumption. We then present several building-block results that allow us to build up large AnsProlog* programs from smaller modules and also help us analyze large programs by breaking them to parts. We start with Marek and Subrahmanian's lemma relating rules of an AnsProlog program and atoms in its answer sets. We

then mention various subclasses of AnsProlog programs (such as, stratified, locally stratified, tight, signed, acyclic, call-consistent, order consistent, negation cycle free, and head cycle free) and various properties (such as categoricity, coherence, relation with completion, sub-class determination, filter-abducibility, language independence, language tolerance, and restricted monotonicity). In particular, we present the relationship between signed programs and restricted monotonicity, and quickly discuss the notion of splitting and its application, complexity and expressiveness, and relation with Pure Prolog. We then quickly mention some interesting application development using AnsProlog: scheduling, combinatorial auctions, planning with durative actions and active database verification. Finally, we conclude with the suggestion that perhaps AnsProlog* is the declarative language with the largest body of building block results and support structures making it the most suitable language for konowledge representation, reasoning and declarative problem solving. We then mention some future directions.

Answer Set Programming and Information Agents
Thomas Eiter

In the recent years, software agents have received increasing attention as building blocks in a new paradigm of building distributed, open information systems. These agents should operate autonomously and intelligently. Im particular, they should have the capability of making rational decision given possibly incomplete, inconsistent etc information on the basis of which an agent should act. In this talk, we look at some specific tasks of informations agents, and consider the potential of using Answer Set Programming (ASP) to solve them. We then zoom on the particular task of information site selection, and report about an ASP based method for this task and an example application that we have built. We talk about the lessons learned in this project, and about research issues on ASP that emerge.

CMODELS: a program for computing models of completion
Yuliya Babovich

Cmodels is a system that computes answer sets for tight logic programs. Programs used in answer set programming, including those related to planning and common sense reasoning, are often tight. For tight programs, the answer set semantics is equivalent to the completion semantics, so that the answer sets for such a program can be found by a SAT solver. Cmodels forms the completion of the given program, converts it to clausal form, and calls a SAT solver to find the models of the completion. In some cases, this method produces answer sets faster than the algorithms used in the general-purpose answer set solvers.

ASSAT: Computing Answer Sets of A Logic Program By SAT Solvers
Yuting Zhao

We propose a new translation from normal logic programs with constraints under the answer set semantics to propositional logic. Given a logic program, we show that by adding, for each loop in the program, a corresponding loop formula to the program's completion, we obtain a one-to-one correspondence between the answer sets of the program and the models of the resulting propositional theory. Based on this result, we propose an alternative implementation of answer set programming using SAT solvers. We test our system, called ASSAT(X), depending on the SAT solver X used, on a variety of benchmarks including the graph coloring, the blocks world planning, and Hamiltonian Circuit domains. The results are compared with those by smodels and dlv, and it shows a clear edge of ASSAT(X) over them in these domains.

The Frame Problem in Induction

Ramon Otero

The following problem will be considered: From examples on the behavior of a dynamic system induce a description of the system.

Additional requirements: - The examples given will be scattered –as usual in learning– both on some sequences, and on some situations inside each sequence. Other solutions require the given sequences to be complete at every situation, which can be considered an instance of the frame problem in induction. - The description induced must be concise and modular, in particular, the induced rules shall not manifest the frame problem.

For the description of the system to be modular and concise, action formalisms will be chosen, including a solution to the frame problem. Some candidates are Situation Calculus, STRIPS, Event Calculus or recent formalisms based on causality: Lin's, McCain-Turner, or Pertinence by the author.

We will use a generic action formalism that is widely applicable and representable in LP. The solution induced in this generic formalism can be easily translated or adapted to other action formalisms.

It is relatively simple to induce a description of a dynamic system that suffers from the frame problem. The known solutions to the frame problem require a non-monotonic formalism. Unfortunately induction under non-monotonic formalisms –e.g. normal logic programs– is not well understood yet.

We will present a method for induction under the restricted non-monotonic behavior needed to solve the frame problem. The method eventually relies on known ILP techniques, as Inverse Entailment for Horn programs.

The incompleteness of the set of examples, mainly inside a given sequence, introduces another problem in induction of dynamic systems because the incompleteness interacts with the solution to the frame problem. Again, it is relatively simple to deal with incomplete sequences if the solution does not solve the frame problem. It has been also proposed how to induce without the frame problem from complete sequences (join work with Lorenzo). Solving both issues is not easy, though this seems needed to deal with the 'regular' setting in induction: the examples do not have to be complete in any sense and, nevertheless, a solution to induction without the frame problem has to be provided.

From the previous method, an extension will be presented that is able to deal with the mentioned 'regular' setting in induction of dynamic systems.

Well-founded and Stable Semantics for Logic Programs with Aggregates
Nikolay Pelov

We investigate the problem of defining well-founded and stable model semantics for logic programs extended with arbitrary aggregate functions. Our work is based on Approximation Theory which is a unifying framework for the semantics of several major non-monotone reasoning systems. We also look at the problem of the complexity of computing the different semantics and in particular for which aggregate functions it stays the same as the complexity of standard logic programs.

Encoding definitions and axioms in answer set programming
David Gilis

ID-Logic and Answer Set Programming are two formalisms coping with incomplete knowledge. Answer Set Programming has already been intensively studied, resulting in good knowledge about pros and cons of this formalism and in an efficient implementation of a solver, the SMODELS system. ID-Logic is more recent. It is based on the concept of human knowledge. We present in this talk a transformation from an ID-Logic theory to a general logic program such that the models of the first theory correspond to the stable models of the program resulting from the transformation.

Extending the Applicability Range of ASP
P.A. Bonatti

Application needs are calling for more expressive ASP languages, and at the same time require ASP engines to cope with programs that have extremely large ground instantiation. Moreover, in order to build applications, ASP solvers should be embedded into a general purpose programming language.

We are exploring *finitary programs* as a means to tackle these aspects simultaneously. Finitary programs have the property that queries can be answered by reasoning with strict subsets of the ground instantiation (and of the stable models) of the given program. This property may reduce memory needs and computation time. Furthermore, this property makes it possible to accept programs with function symbols and proper recursion, thereby extending the expressiveness of the language. The existing core search engines need not be modified, while front-ends should be replaced with new front-ends able to generate only the relevant fragment of the ground program. This results in more powerful - potentially Turing complete - front-ends, and hence at this level we obtain a first integration between ASP and standard Logic Programming.

Computing preferred answer sets in answer set programming
Toshiko Wakaki

Sakama and Inoue proposed a framework of prioritized logic programs (or PLP for short). PLP is defined by a pair (P, Φ), which has a mechanism of explicit representation of priority knowledge as Φ by extending a general extended disjunctive logic program P. Then PLP enables us to reduce non-determinism under incomplete or conflicting knowledge and lead to an intended conclusion as well as it realizes various forms of non-monotonic reasoning such as abduction, default reasoning and prioritized circumscription. Such expressive power of PLP is based on preferred answer sets of PLP. However, as for prior works w.r.t. computing preferred answer sets, Sakama and Inoue's procedure is naive, or rather it may be said as definition itself. So, in this talk, we propose our method of computing preferred answer sets of PLP in answer sets programming. Roughly speaking, the basic idea of our method is to translate a PLP (P,Φ) and an answer set S of P under consideration into a single translated program Q whose answer sets (if they exist) yield strictly preferable to S. Hence, by checking the inconsistency of such a translated program Q constructed from P, Φ and S, we can decide whether each answer set is preferred or not, based on our method. Thus our method makes it possible to compute preferred answer sets of PLP easily by making use of efficient tools such as DLV, Smodels, and then can be considered as a useful application of answer set programming.

What's your preference? And how to express and implement it in logic programming!

Torsten Schaub

We introduce a methodology and framework for expressing general preference information in logic programming under the answer set semantics. At first, we are interested in semantical underpinnings for existing approaches to preference handling in extended logic programming. To begin with, we explore three different approaches that have been recently proposed in the literature. Because these approaches use rather different formal means, we furnish a uniform characterizations that allows us to gain insights into the relationships among these approaches.

We then draw on this study for furnishing implementation techniques. In the resulting framework, an ordered logic program is an extended logic program in which rules are named by unique terms, and in which preferences among rules are given by a set of atoms of the form $s \prec t$ where s and t are names. Such an ordered logic program is transformed into a second, regular, extended logic program wherein the preferences are respected, in that the answer sets obtained in the transformed program correspond with the preferred answer sets of the original program. Our approach allows the specification of dynamic orderings, in which preferences can appear arbitrarily within a program. Static orderings (in which preferences are external to a logic program) are a trivial restriction of the general dynamic case. We develop a specific approach to reasoning with prescriptive preferences, wherein the preference ordering specifies the order in which rules are to be applied.

Since the result of our translation is an extended logic program, we can make

use of existing implementations, such as dlv and smodels. To this end, we have developed the so-called plp compiler, available on the web at url
`http://www.cs.uni-potsdam.de/~torsten/plp/`,
as a front-end for these programming systems.

Using ASP for ontology management
Terry Swift

Ontologies are becoming an increasingly important paradigm for knowledge representation, particularly for the semantic web. Because of their prevalence, ontologies offer logic programming an excellent way to store knowldge about various domains and to share that knowledge with other systems. Just as importantly, logic programming and ASP may offer ontologies a theoretically sound way to handle default, quantificational and paraconsistent reasoning. As a result, an ontology management system, ColdDeadFish is being built in XSB to allow parts of logic programms to be represented as ontologies. In particular, a theorem prover for consistency of ColdDeadFish class definitions has been implemented using ASP, which shows promising performance and also indicates connections between logic programs and ontologies.

Implementing Logic Programs with Ordered Disjunction
G. Brewka (joint work with I. Niemelä and T. Syrjänen)

Logic programs with ordered disjunction (LPODs) add a new connective to logic programming. This connective allows us to represent alternative, ranked options for problem solutions in the heads of rules: A x B intuitively means: if possible A, but if A is not possible, then at least B. The semantics of logic programs with ordered disjunction is based on a preference relation on answer sets. In the talk we show how LPODs can be implemented using answer set solvers for normal programs. The implementation is based on a generator which produces candidate answer sets and a tester which checks whether a given candidate is maximally preferred and produces a better candidate if it is not. We also discuss potential applications to qualitative decision making.

Using Non-Monotonic Reasoning Techniques in the Semantic Web
Dietmar Seipel

The challenge of the Semantic Web is to provide a language that expresses both data and rules for reasoning about the data. Many practical applications of knowledge bases can be handled using extensions of Logic Programming (LP) and Non-Monotonic Reasoning (NMR). Advanced features of LPNMR that are frequently used are, e.g., disjunctive or uncertain information, aggregation, cardinality constraints, and optimization.

We have focussed on the special NMR-formalism of Revision Programs, and we have applied it to reconfiguration problems. For a given initial configuration I, a revision rule can specify which components should be added or removed.

The goal is to find a new configuration M that satisfies all revision rules in a certain way and at the same time minimizes the amount of changes made to I.

Marek and Truszczynski had defined P-justified revisions as a semantics for revision programs P; it is based on program transformations similiar to the Gelfond-Lifschitz transformation. Since P-justified revisions were not adequate for the reconfiguration problem that we wanted to solve, we defined an extension called P-grounded revisions, and we proved some properties of this new concept: e.g., every P-justified revision is also a P-grounded revision.

The application of NMR-techniques for reasoning in the Semantic Web requires the handling of complex, structured documents. Thus, we have developed a Prolog library for querying and transforming semi-structured data, such as XML-data, in LP.

Abstract analytical methods for computing 2-valued and multi-valued supported models
Howard A. Blair

Newton-like and gradient methods for computing supported models are presented. Such methods are based on derivatives of Boolean and multi-valued truth functions. There is a range of possibilities - among which are those commonly found in the literature - for what is to count as a truth-function derivative that is dependent on the underlying connectivity among truth-values. This dependendence is due to the fact that derivatives and differentials (even in conventional contexts involving real or complex numbers) are continuous solutions of certain kinds of contraints, and the notion of continuity is determined by the convergence structures on the spaces involved. The categories of reflexive digraphs and topological spaces are each full subcategories of the category of convergence spaces whose arrows are continuous functions. Since the arrows between reflexive digraphs are digraph homomorphisms, various notions and results from analysis and topology translate to properties of digraph-homomorphisms when dealing with discrete stuctures. We apply these ideas to constructing differentials of logic programs relative to partial orderings among truth values, and look at the affect on the behavior of common differential operators.

SBSAT: A State-Based, BDD-Based, Satisfiability Solver
John Schlipf

State-Based SAT (SBSAT) is a new solver – still under development – for propositional satisfiability problems. Thus this talk does not directly address the Answer Set Programming concerns of this workshop. However, there are huge overlaps in technique between ASP solvers and SAT solvers. Furthermore, some of the ASP solvers in this conference produce propositional satisfiability problems, which they pass to SAT solvers. Thus the technology is clearly relevant.

Whereas most SAT solvers take sets of clauses as inputs, SBSAT allows sets of arbitrary boolean formulas (with some size limits), expressed as BDDs. By

analyzing (essentially) all partial evaluations for each individual formulas, it can (a) often find forced inferences closer to the root of the search tree than can a SAT solver, and (b) base its heuristic upon complete information about the future states of each individual input function. SBSAT determines all this information in preprocessing and stores it in Mealy machines, effectively trading space for time,

SBSAT includes modified BDD-type tools to use in its front end, thus attempting to combine good features of both SAT and BDD approaches. (Full BDD tools, which are an alternative to SAT tools, typically cause space explosion on the problems we are interested in.) SBSAT also includes features of modern SAT engines, noticably lemma learning, to speed up its search.

More on noMoRe
Thomas Linke, Christian Anger, Kathrin Konczak

This talk focuses on the efficient computation of answer sets for normal logic programs. It concentrates on a recently proposed rule-based method (implemented in the noMoRe system) for computing answer sets. We show how noMoRe and its underlying method can be improved tremendously by extending the computation of deterministic consequences. With these changes noMoRe is able to deal with problem classes it could not handle so far.

Knowledge-based Planning and ASP
Wolfgang Faber (joint work with Thomas Eiter, Nicola Leone, Gerald Pfeifer, Axel Polleres)

We propose a new declarative planning language, called $\mathcal{K}$, which is based on principles and methods of logic programming. In this language, transitions between states of knowledge can be described, rather than transitions between completely described states of the world, which makes the language well-suited for planning under incomplete knowledge. Furthermore, our formalism enables the use of default principles in the planning process by supporting negation as failure. Furthermore, we present the language $\mathcal{K}^c$, which extends $\mathcal{K}$ by action costs and optimal plans that minimize overall action costs (cheapest plans). We give an overview of the planning system $\texttt{DLV}^{\mathcal{K}}$, which has been implemented as a front-end to the Answer Set Programming System `DLV`. Our experience is encouraging and supports the claim that answer set planning may be a valuable approach to advanced planning systems in which intricate planning tasks can be naturally specified and effectively solved.

Planning and Scheduling with ASPPS
Deborah East

I describe how the aspps system (answer set programming with propositional schemata) can be used for scheduling problems. Two problems are discussed. First, timetabling for determining classrooms, times and faculty assignments for classes. I present a general set of rules for class assignments. In addition to these

general rules, we show how requirements and restrictions for room and faculty assignments are modeled. The second problem we present is jobshop scheduling. I have tasks, machines and times with restrictions on the order in which tasks can be executed on the machines. The goal of the jobshop scheduling problem is to find an optimal time schedule for completing all tasks. Last, I will discuss the limitations of the aspps system and future work.

A distributed demand-driven algorithm for computing minimal models

Rachel Ben-Eliyahu-Zochary

The task of generating minimal models of a knowledge base is a significant computational problem in artificial intelligence. This task is at the computational heart of diagnosis systems like truth maintenance systems, and of nonmonotonic systems like autoepistemic logic, default logic, and disjunctive logic programs. Unfortunately, it is NP-hard. In this paper we present a hierarchy of classes of knowledge bases, $\Psi_1, \Psi_2, ...$, with the following properties: first, Ψ_1 is the class of all Horn knowledge bases; second, if a knowledge base T is in Ψ_k, then T has at most k minimal models, and all of them may be found in time $O(lnk)$, where l is the length of the knowledge base and n the number of atoms in T; third, for an arbitrary knowledge base T, we can find the minimum k such that T belongs to Ψ_k in time polynomial in the size of T; and, last, where $\mathcal{K}$ is the class of all knowledge bases, it is the case that $\bigcup_{i=1}^{\infty} \Psi_i = \mathcal{K}$, that is, every knowledge base belongs to some class in the hierarchy. The algorithm is demand-driven, that is, it is capable of generating one model at a time.

Answer Set Programming on a Minimal Model Generation Theorem Prover MM-MGTP

Ryuzo Hasegawa

Answer Set programming, that computes answer sets from extended logic programs (ELP) containing negation as failure (NAF) and classical negation, has been a focus of the attention not only in the logic programming field but also in application areas. In 1992, We proposed a method to transform any logic programs with NAF into a disjunctive logic program (DLP) without NAF, by introducing modal operators called K-literals. In this method, an extended logic program $A_1, ..., A_n, notB_1, ..., notB_i -> C_1 v ... v C_m$ is translated into a disjunctive logic program $A_1, ..., A_n -> (-KB_1, ..., -KB_i, C_1) v ... v (-KB_1, ..., -KB_i, C_m) v KB_1 v ... v KB_i$ where K is a modal operator. Intuitively, KB means the hypothesis that B must hold, and -KB means that B is assumed not to hold. MGTP can compute the stable models of a generic logic programs and answer sets of an ELP as the fixed point of model candidates. The naive implementation of this method, however, generates enormous combination of hypothes and redundant models that cannot be answer sets. To overcome these, we propose a new method based on a minimal model generation theorem prover MM-MGTP. MM-MGTP can generate only minimal models by using branching assumptions and lemmas. It

also incorporates some techniques to prune redundant branches, such as proof condensation and folding-up. In the new method, the above extended logic program is simply translated to $A_1, ..., A_n -> C_1 v ... v C_m v KB_1 v ... v KB_i$, because MM-MGTP employs a complement splitting. To support K-literals, unit refutation and unit simplification are extended. For example, if KB is in a model candidate M, a disjuction A v -B v C is simplified to A v C. In addition, if B is in M, then B subsumes A v KB v C. For further improvements on K-literals, we introduce: (1) extended conjunctive matching which derives KL from a clause C -¿ L and KC in the model candidate, and (2) suppressing case splitting for K-literals by using the modal disjunction buffer to retain them. Note that this buffer does not cause any splitting. The latter method (2) enables it to check the stability at the fixed point (T-condition) in a constant time. We have implemented the above method on a Java-version of MM-MGTP. Some experimental results with MM-MGTP on Quasigroup problems in finite algebara and the 3-coloring problems show that the new method eliminates the redundancy in the naive implementation, and can be the basis for answer set programming.

On implementing nested logic programs
Stefan Woltran

Nested logic programs have recently been introduced in order to allow for arbitrarily nested formulas in the heads and the bodies of logic program rules under the answer sets semantics. These programs generalise the well-known class of disjunctive logic programs providing a very flexible and compact framework for knowledge representation and reasoning.

Recent research provided several different approaches to implement such programs. One of the considered techniques deals with translations of nested logic programs into disjunctive logic programs, thus giving front-ends to dlv and smodels to compute the answer sets of nested programs. The other approaches follow a different strategy, albeit they also rely on a certain reduction technique. However, the outcome here is not a logic program, but a formula in quantified Boolean logic, and for some newly introduced subclasses, a formula in classical propositional logic. In the talk, we discuss these approaches.

A pair-wise compatibility heuristic for computing default extensions: Some experimental results
Robert E. Mercer and Vincent Risch

An extension-building heuristic is developed and a preliminary investigation of its heuristic properties is given. The ability of the new heuristic to reduce search over a range of problems is compared to the search reduction properties of DeReS which uses relaxed stratification, another extension-building heuristic.

The heuristic that we develop and study uses cliques of a graph representing pair-wise compatibility between default rules. This heuristic can discover structural properties of a default theory which may allow divide-and-conquer-like techniques to be applied on those problems which exhibit appropriate structural properties. The heuristic provides a separation between two levels of the

building of extensions. Computing the cliques is a meta-level search strategy for approximating extensions. These approximations are upper bounds of the extensions. After the approximation is made, an iterative low-level search in the classical sense is performed. The heuristic is incremental, which means that as information is discovered by the low-level search, the meta-level search can produce cliques that are better approximations of the extensions.

Our preliminary results show that the clique heuristic finds useful structure in default theories that is different than the structure found by relaxed stratification. Most importantly, the clique heuristic and relaxed stratification can be used together, resulting in the positive effects of both heuristics being observed.

Our preliminary investigations indicate future work in the two search levels: in the low-level search one can make the search more contextually dependent with dynamic stratification and dynamic ordering of defaults. In the meta-level search more information flow from the low-level search to meta-level search can reduce the production of cliques. What needs to be investigated is what information can be discovered in the search that can act as a filter in the meta-level search and what information can be memoized to be used in the low-level search of cliques that differ on few defaults. In addition to these search enhancements, cliques provide a natural parallelization of the search.

Kernelization of Logic Programs
Alessandro Provetti

Normal forms for logic programs under stable/answer set semantics are introduced. We argue that these forms may simplify the study of program properties, viz. consistency, the design of deduction algorithms and, potentially, efficient implementations.

The first normal form, called *kernel* of the program, is synthetic w.r.t. existence and number of answer sets. While the complexity of answer set computation over kernel programs does no= t decrease as far as the polynomial hierarchy is concerned, programs in kerne= l form tend to be a lot more compact, thus yielding better computation perfor= mance in general. We address computational issues concerning the reduction of a given program to its kernel (called *kernelization*). We address consistency by defining a second normal form, *3-kernel*, where, apart from kernel form, the length of rule bodies is limited to two.

Bottom-up Computation of Selector-generated Models
Sibylle Schwarz

Stable models were defined for normal logic programs by Gelfond and Lifschitz. This definition was extended to disjunctive programs by Przymusinski and to programs with nested expressions by Lifschitz, Tang and Turner. In all these approaches, stable models are defined as minimal models of transformed programs. An alternative way to generalize stable models to disjunctive and nested programs without program transformations was given by selector-generated models, an generalization of stable generated models defined by Herre

and Wagner. This approach depends neither on an fixed logic language nor a fixed set of truth values and is therefore more general than the first definition. Stable models of normal logic programs are in fact a special case of selector-generated models. The definition of selector generated models suggests a generate-and-test computation method similar to the computation of stable models. In general, each model of a program has to be tested for the properties given in the definition. This talk outlines a bottom-up-method for computation of selector-generated models without guessing that uses an idea similar than a fixed point characterization for stable models (and other classes of models) by Sakama and Inoue.

Reduction of nested expressions in logic programs -Tight logic programs revisited

Jia-Huai You

We present a polynomial time reduction that transforms a logic program with nested expressions in the body of a rule to a logic program without nested expressions. This makes it possible to use an answer set generator for normal or extended programs, such as Smodels and DLV, to compute answer sets for logic programs with nested expressions. We define a tightness condition which is weaker than that of Erdem and Lifschits so that any program is weakly tight on any of its answer sets. It is therefore a sufficient as well as a necessary condition for the equivalence between answer sets and models of completion.

Consistency and Coherence in Kernel Logic Programs: Some Preliminary Results

Stefania Costantini and Alessandro Provetti

The aim of this talk is that of introducing a Software Engineering perspective on Answer Set Programming, i.e., studying problems related to developing, restructuring and updating programs. We will argue that these problems are closely related to the problem of consistency, i.e., of the existence of answer set. Thus, the same conceptual methods and tools may be used for both checking consistency, and supporting step-by-step program development. After summarizing previously known notions related to program development under the answer set semantics, namely cumulativity, extended cumulativity and strong equivalence, we propose a new property, called coherence, that in our view should be satisfied when updating a program. In particular, coherence requires that after an update, the answer sets of the original program should be in some sense preserved, though possibly extended. We define a weak and a strong notion of coherence. The method and tools we propose for studying consistency and coherence are in particular: (i) kernelization of programs, i.e., reduction to a normal form where we get rid of irrelevant literals and rules; (ii) representation of programs by means of the Cycle Graphs, where both the cycles the program is composed of and their interrelations are made explicit. We show that, based on Cycle Graphs of kernel programs, necessary and sufficient conditions for consistency and useful sufficient conditions for coherence can be defined.

Intelligent Agents and Answer Set Programming
Michael Gelfond

In this talk I will briefly describe an architecture of intelligent agents capable of reasoning and acting in a changing environment. I'll discuss the use of the language of logic programming under the answer set semantics for representing the agent's knowledge about its domain, history, abilities and goals and demonstrate how various reasoning tasks such as planning, diagnostics of unexpected observations, etc. can be reduced to computing answer sets of various logic programs. Part of the talk will be devoted to discussion of open problems which need to be solved to efficiently implement the architecture.

CSP consistency as stable model computation
Gerard Ferrand and Arnaud Lallouet

The stable model semantics for normal logic programs provides a very declarative way of specifying problems since clauses are considered as constraints for the acceptability of a model. In particular, a finite domain constraint satisfaction problem (CSP) can be represented by a logic program which allows to select from a family of finite models those which represent the solutions of the CSP. The feasability of representing CSP as stable logic programs (SLP) has been shown. We moreover have shown that not only the CSP is encodable in SLP but also the approximation computed by the consistency. The different search states are represented by 3-valued stable models, the first of all being the well-founded semantics. And the CSP's solutions are related to a category of stable models we call singletonic. We provide a little translator which converts a CSP expressed in a simple syntax to a normal logic program intended to feed the smodels system. We moreover discuss applications of this result to the debugging of CSP and relate this issue to the debugging of answer set programs.