From LP to ASP and TN

P.A. Bonatti

Pisa, October 23, 2009
Tentative goal

- Tracing some unlikely, unexpected influences
- Of the ideas floating in Giorgio's group in early 90s
- On apparently unrelated areas
  - ASP (Answer Set Programming)
  - TN (Trust Negotiation)

“after you learn LP, you can't help using it while solving new problems”
(V.S. Subrahmanian, personal communication)
Some evidence about unrelatedness

Claim 1

- Not a single joint paper since 1987

[Guinness Book of Records, submitted]
Giorgio has been my advisor

- Master thesis (accidentally)
- PhD thesis (deliberately)
From LP to ASP and TN

- From Levi in Pisa to A Satisfactory Profession
From LP to ASP and TN

- From **Levi** in **Pisa** to **A Satisfactory Profession**
- And ... ?
  - Technical **Nonchalance** ?
  - Tendentious **Nonmonotonicity** ?
  - Terùn in **Naples** ?

- Send suggestions to bonatti@na.infn.it
From LP to ASP and TN

- From Levi in Pisa to A Satisfactory Profession
- And ... ??
  - Technical Nonchalance ?
  - Tendentious Nonmonotonicity ?
  - Terùn in Naples ?

- Send suggestions to bonatti@na.infn.it

- Traino a Nuoto...

This is about energy
Giorgio has been my advisor

- Master thesis (unintentionally)
- PhD thesis (deliberately)
  - Back from IRST with many ideas on nonmonotonic reasoning
    - Empty intersection with G.'s background and TO-DO list
  - Visit to VS Subrahmanian @ Univ. of Maryland
  - Unexpected bonus: first steps in security
Giorgio has been my advisor

- Master thesis (unintentionally)
- PhD thesis (deliberately)
  - Back from IRST with many ideas on nonmonotonic reasoning
    - Empty intersection with G.'s TO-DO list
  - Visit to VS Subrahmanian @ Univ. of Maryland
  - Unexpected bonus: first steps in security
- Exposure to the group's cultural environment
  - Constraint logic programming
  - Meta-interpreters
  - Partial evaluation
  - Static LP analysis

Now for the inspiration...
Applications to Trust Negotiation (TN)
What is Trust Negotiation

- Client and server exchange pieces of information
  - Servers need to know user properties for access control
    - age, membership to companies, subscriptions, nationality
  - Clients ask servers for certifications (privacy guarantees)
    - seal programs: eTrust, BBB (Better Business Bureau)
  - Encoded as X.509 digital credentials and unsigned declarations

- Automatic negotiation
  - For usability (reduce burden on users)
  - And stronger guarantees (credential requests may be forced)
An example (I)

- **Server policy**
  - Public resources can be downloaded by everybody
  - **Authenticated users** can download the resources they subscribed
  - Any resource can be downloaded by providing
    - An **ID** (passport, driving licence, student ID, ...)
    - An accepted **credit card** (VISA, Mastercard, American Express, ...)

- **Client policy**
  - Credit cards are disclosed only to **members** of eTrust or BBB
An example (II)

Step 1: Alice requests a service from Bob

Step 2: Bob discloses his policy for the service

Step 3: Alice discloses her policy for VISA

Step 4: Bob discloses his BBB credential

Step 5: Alice discloses her VISA card credential

Step 6: Bob grants access to the service
How to request information

- One specific set of credentials?
  - (among all possible alternatives)
  - Blind backtracking!
- All possible sets of credentials?
  - Huge messages!
- The policy applying to the requested resource
  - More efficient (less and smaller messages)
  - Privacy-enhancing (users can choose the best option)
However, policies may be sensitive

- For example:
  - Only my best friends can see these pictures
  - Company X's employees can access confidential data
  - The list of accepted credit cards can be disclosed
  - The list of correct user-password pairs cannot

- How to protect the sensitive parts of a policy?
Policy filtering

- Partial evaluation! In PROTUNE:

1. Filter non-applicable and irrelevant rules
2. Compile applicable non-public rules
3. Partial evaluation of public rules
4. Execute immediate actions
5. Evaluate local provisional literals
6. Blur deferred state conditions
7. Filter irrelevant policies due to blurring
8. Replace provisional state predicates with actions
9. Anonymize abbreviation predicates
Adapted results

- Equivalence of original and filtered policies w.r.t. a given goal (authorization)
  - Equivalence up to *blurring*
    - Partial or total removal of a predicate's definition
  - In general there is loss of information
  - Soundness is guaranteed
  - Completeness relative to non-blurred information

- Not mentioned here: metainterpreters for credential selection and explanations
Now for something completely different (ASP)
What is answer set programming

- A declarative problem solving paradigm
- Syntax: Normal logic programs
  \[ A \leftarrow B_1, \ldots, B_m, \text{not } B_{m+1}, \ldots, \text{not } B_n \]
- Semantics: Stable model semantics
  - Gelfond-Lifschitz reduct \( P^M \) (partial evaluation again...)
    - Remove all rules with a not \( B \) such that \( B \in M \)
    - Remove all negative literals from the surviving rules
  - \( M \) is a stable model of \( P \) iff
    \[ M = \text{least model of } P^M \]
- Negation is an amalgamated unprovability modal operator
How to solve problems in ASP

- A normal program may have 0, 1, or multiple stable models
- Write $P$ so that its stable models are in 1-1 correspondence with problem solutions
- Complete for NP (Datalog case)
How to implement ASP

\[ P \rightarrow \text{ground instantiation} \rightarrow \text{Ground}(P) \rightarrow \text{core reasoner} \rightarrow \text{Stable models} \]
How to implement ASP

$P \rightarrow \text{ground instantiation} \rightarrow \text{Ground}(P) \rightarrow \text{core reasoner} \rightarrow \text{Stable models}$

very large, despite clever simplifications
How to implement ASP

1. $P$
2. ground instantiation
3. $\text{Ground}(P)$
4. core reasoner
5. Stable models

- very large, despite clever simplifications
- a serious limiting factor: conspicuous literature on this problem
An obviously appealing solution

- **S-semantics**!
- Explored by VS Subrahmanian
- Some difficulties with disunifiers
Our approach

- **ASP + Constraint Logic Programming**
  - Joint work with S. Baselice and M. Gelfond

- Rigid partitioning of program rules
  - ASP, Constraints, Bridge
  - Strong syntactic restrictions, but
  - Amazing speedups in a planning setting
    - USAdvisor (planning operations for the Space Shuttle)

- Only ASP rules need to be grounded
  - On a small domain
  - Constraints handle time (much larger!)
  - Granularity up to minutes for weeks-long plans
  - (awarded at the ASP workshop 2005)
Infinite domains
an (almost) all italian contribution to ASP
Extending ASP

- Originally ASP was restricted to Datalog with negation
  - Reasoning with function symbols: $\Pi_1^1$-complete
- However function symbols are important for:
  - Encapsulation
  - Recursive data structures
    - Including HTML, XML
- We introduced the first computationally well-behaved ASP fragment with functions [IJCAI'01]
  - **Finitary programs**
  - Ground queries: decidable
  - Nonground queries: r.e.-complete
A very expressive class

- It contains
  - All the standard list/tree manipulation programs
  - Many planning programs
  - SAT and QBF metaintepreters
  - A simulator of Turing machines with bounded tape
- Much more expressive than Datalog ASP
Checking whether a normal LP is finitary is **undecidable**

Let's see why
Finitary programs: definition

- Finitary =
  1. Finitely recursive
  2. Finitely many odd-cycles

- Based on the atomic dependency graph:
  - Nodes: ground atoms
  - Edges: (A,B) such that A=head(r) and B occurs in body(r) for some r in $\text{Ground}(P)$

- Finitely recursive = each A depends on finitely many B
  - $A \text{ depends on } B$ iff $\exists$ a path from A to B

- Odd-cycle = with an odd number of negative edges
  - Negative edge: B occurs in the scope of negation
More on undecidability

- Turing machines can be simulated with binary clauses
- Computations = paths in the dependency graph
- Undecidability proof by reduction from termination
First idea: use *norms* ("size" of a term)
- $|t| = \#\text{symbols in } t$

Check that the norms of some predicate arguments are decreasing during recursion
Rescued by static analysis

- First idea: use norms ("size" of a term)
  - $|t| = \#\text{symbols in } t$
- Check that the norms of some predicate arguments are decreasing during recursion
- More problems due to local variables (occurring only in the body)
  - They cause infinite branching in the graph
Rescued by static analysis

- First idea: use norms ("size" of a term)
  - $|t| = \#\text{symbols in } t$
- Check that the norms of some predicate arguments are decreasing during recursion
- More problems due to local variables (occurring only in the body)
  - They cause infinite branching in the graph
- This requires proving that
  - For each ground instance of the head
  - The answer substitutions for the body are finitely many
  - By subterm analysis
    - e.g. member(X,L) where X local and L occurs in the head
What happened next

- Prototype static analyzer presented at LPNMR'01
What happened next

- Prototype static analyzer presented at LPNMR'01
- Syntactic decidable subclasses of finitary programs developed by Nicola Leone's group (2007-2009)
- Relaxed by Lierler & Lifschitz (2008)
  - Only finite models, though!
What happened next

- Prototype static analyzer presented at LPNMR'01
- Syntactic decidable subclasses of finitary programs developed by Nicola Leone's group (2007-2009)
- Relaxed by Lifschitz (2008)
  - Only finite models, though!
- Rigid subclass of finitely recursive programs introduced by Simkus and Eiter: \textit{FDNC} programs
  - To capture description logics
  - Tree-model property instead of odd-cycle restrictions
What happened next

- Prototype static analyzer presented at LPNMR'01
- Syntactic decidable subclasses of finitary programs developed by Nicola Leone's group (2007-2009)
- Relaxed by Lifschitz (2008)
  - Only finite models, though!
- Rigid subclass of finitely recursive programs introduced by Simkus and Eiter: \textit{FDNC} programs
  - To capture description logics
  - Tree-model property instead of odd-cycle restrictions
- Thorough analysis of finitely recursive programs
  - That generalize all of the above
  - Best paper at IJCAI'07
What will happen next

- Syntactic, decidable class based on norms
  - Without ruling out infinite models
- So far, restrictions are somewhat *ad hoc*
- Plan: use *abstract interpretations* to prove that the class is “optimal”
  - Best exploitation of the adopted approximation
That's almost all, folks
Final remarks

- Unexpected long-term influence of the cultural environment of Giorgio's group on TN and ASP
Final remarks

- Unexpected long-term influence of the cultural environment of Giorgio's group on TN and ASP
- A final mention to his rich vision of a professor's role, beyond mere scientific influence
Thank you!