Theory and Practice of Answer Set Programming

Esra Erdem[1], Joohyung Lee[2], and Yuliya Lierler[3]

[1]Sabanci University, Turkey

[2]Arizona State University, USA

[3]University of Nebraska at Omaha, USA

AAAI 2012 Tutorial

Objective

The tutorial will introduce the current state of the art in declarative problem solving via answer set programming. The audience will walk away with an understanding of the mathematical foundation of ASP, algorithms and systems for computing answer sets, recent applications of ASP including biomedical query answering and cognitive robotics.

The slides are available online at

`http://peace.eas.asu.edu/aaai12tutorial`

Disclaimer: the coverage of ASP is not extensive, and may reflect our own biased view.

Contents

General Introduction

Answer Set Programming (ASP)

- Declarative programming paradigm.
- Theoretical basis: answer set semantics (Gelfond & Lifschitz, 1988).
- Expressive representation language: Defaults, recursive definitions, aggregates, preferences, etc.
- ASP solvers:
  - SMODELS (Helsinki University of Technology, 1996)
  - DLV (Vienna University of Technology, 1997)
  - CMODELS (University of Texas at Austin, 2002)
  - PBMODELS (University of Kentucky, 2005)
  - CLASP (University of Potsdam, 2006) – winning first places at ASP'07/09/11/12, PB'09/11/12, and SAT'09/11/12

Applications of ASP in AI

- planning ([Lif02a], [DEF+03], [SPS09], [TSGM11], [GKS12])
- theory update/revision ([IS95], [FGP07], [OC07], [EW08], [ZCRO10], [Del10])
- preferences ([SW01], [Bre07], [BNT08a])
- diagnosis ([EFLP99], [BG03], [EBDT+09a])
- learning ([Sak01], [Sak05], [SI09], [CSIR11])
- description logics and semantic web ([EGRH06], [CEO09], [Sim09], [PHE10], [SW11], [EKSX12])
- probabilistic reasoning ([BH07], [BGR09a])
- data integration and question answering ([AFL10], [LGI+05])
- multi-agent systems ([VCP+05], [SPS09], [SS09], [BGSP10], [Sak11], [PSBG12])
- multi-context systems ([EBDT+09a], [BEF11], [EFS11], [BEFW11], [DFS12])
- natural language processing/understanding ([BDS08], [BGG12], [LS12])
- argumentation ([EGW08], [WCG09], [EGW10], [Gag10])

Applications of ASP in Other Areas

- product configuration ([SN98], [TSNS03])
- Linux package configuration ([Syr00], [GKS11])
- wire routing ([ELW00], [ET01])
- combinatorial auctions ([BU01])
- game theory ([VV02], [VV04])
- decision support systems ([NBG+01])
- logic puzzles ([FMT02], [BD12])
- bioinformatics ([BCD+08], [EY09], [EEB10], [EEEO11])
- phylogenetics ([ELR06], [BEE+07], [Erd09], [EEEF09], [CEE11], [Erd11])
- haplotype inference ([EET09], [TE08])
- systems biology ([TB04], [GGI+10], [ST09], [TAL+10], [GSTV11])
- automatic music composition ([BBVF09],[BBVF11])
- assisted living ([MMB08], [MMB09], [MSMB11])
- team building ([RGA+12])
- robotics ([CHO+09a], [EHP+11], [AEEP11a], [EHPU12], [APE12])
- software engineering ([EIO+11])
- bounded model checking ([HN03], [TT07])
- verification of cryptographic protocols ([DGH09])
- e-tourism ([RDG+10])

Applications of ASP in Other Areas

- product configuration ([SN98], [TSNS03]): used by Variantum Oy
- Linux package configuration ([Syr00], [GKS11])
- wire routing ([ELW00], [ET01])
- combinatorial auctions ([BU01])
- game theory ([VV02], [VV04])
- decision support systems ([NBG+01]): used by United Space Alliance
- logic puzzles ([FMT02], [BD12])
- bioinformatics ([BCD+08], [EY09], [EEB10], [EEEO11])
- phylogenetics ([ELR06], [BEE+07], [Erd09], [EEEF09], [CEE11], [Erd11])
- haplotype inference ([EET09], [TE08])
- systems biology ([TB04], [GGI+10], [ST09], [TAL+10], [GSTV11])
- automatic music composition ([BBVF09],[BBVF11])
- assisted living ([MMB08], [MMB09], [MSMB11])
- team building ([RGA+12]): used by Gioia Tauro seaport
- robotics ([CHO+09a], [EHP+11], [AEEP11a], [EHPU12], [APE12])
- software engineering ([EIO+11])
- bounded model checking ([HN03], [TT07])
- verification of cryptographic protocols ([DGH09])
- e-tourism ([RDG+10])

Workforce Management at Gioia Tauro Seaport

- The Gioia Tauro seaport:
  - the largest transshipment terminal of the Mediterranean
  - recently become an automobile hub
- Automobile Logistics by ICO BLG:
  - several ships of different size shore the port every day
  - transported vehicles are handled, warehoused, technically processed and then delivered to their final destination.
- Crucial management task: to build teams of employees to handle incoming ships subject to many constraints (e.g., skills, fairness, legal workload regulations).

ASP-Based Workforce Management at Gioia Tauro Seaport

In cooperation with Exeura Srl, a University of Calabria (UNiCaL) spin-off, and ICO BLG, an Italian logistics company, Nicola Leone's group at UNiCaL has developed an ASP-based system for team building based on the DLV solver.

- ASP rules describe the requirements that should be fulfilled regarding: necessary skills of team members; availability of employees; fairness of workload distribution; and distribution of "heavy" or "risky" tasks.
- Since in practice not all requirements can be satisfied, the system has an implicit conflict handling strategy that gives higher priority to more important criteria.
- The system, which has been adopted by ICO BLG for workforce management, can generate shift plans for 130 employees within a few minutes. In addition, the plan quality turned out to be considerably better and overtime was decreased by 20%.

Inferring Phylogenetic Trees

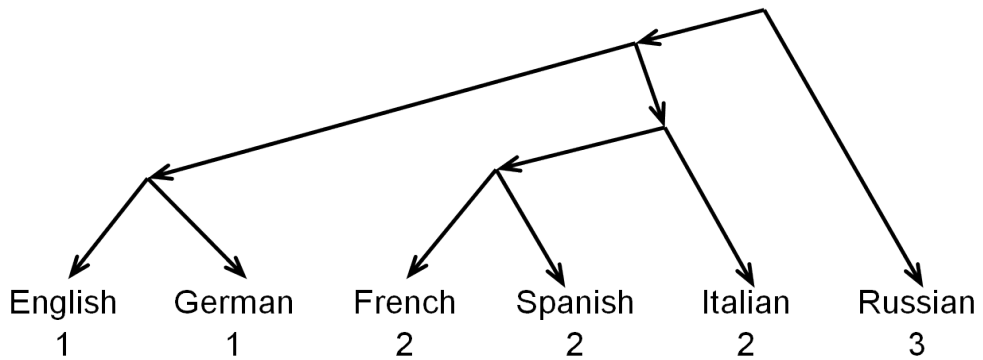
- Phylogenetic trees of individual languages
  - help historical linguists to infer principles of language change; and
  - are also of interest to archaeologists, human geneticists, physical anthropologists (e.g., evolutionary history of certain languages can help us answer questions about human migrations).
- Phylogenetic trees of parasites
  - give us information on where they come from and when they first started infecting their hosts;
  - help understanding the changing dietary habits of a host species and the structure and the history of ecosystems, and identifying the history of animal and human diseases; and
  - allow identification of regions of evolutionary “hot spots”, and thus can be useful to assess the importance of specific habitats, geographic regions, areas of genealogical and ecological diversity.

Inferring Phylogenetic Trees

- After describing each taxonomic unit with a set of characters, and determining the character states...

English	German	French	Spanish	Italian	Russian
hand	*Hand*	*main*	*mano*	*mano*	*ruká*
1	1	2	2	2	3

- the goal is to reconstruct a phylogeny with the maximum number of "compatible" characters.



- Challenges: reachability checks, aggregates, constraints, weights, etc.

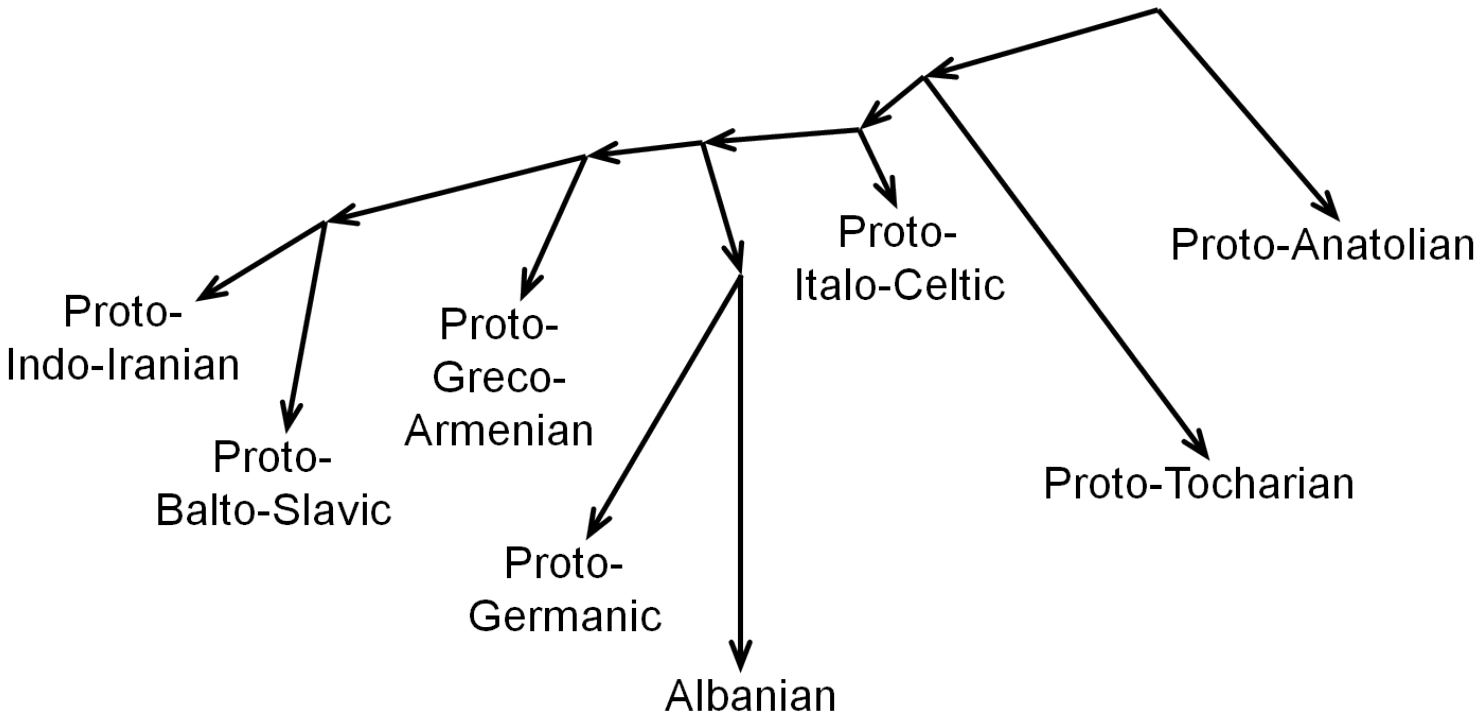
PHYLO-ASP: Phylogenetic Systematics using ASP

We have developed an ASP-based phylogenetic system PHYLO-ASP that not only infers (weighted) phylogenetic trees but also helps the experts analyze and compare them (e.g., by generating similar/diverse phylogenetic trees).

In collaboration with zoologist Dan Brooks (U. of Toronto), historical linguists Don Ringe (UPENN) and Feng Wang (Peking U.), and language engineer James Minett (Chinese U. of Hong-Kong), we have reconstructed plausible phylogenies for *Alcataenia* species (a tapeworm genus), Indo-European languages, and Chinese dialects using PHYLO-ASP.

`http://krr.sabanciuniv.edu/projects/Phylo-ASP/`

The Most Plausible Phylogeny for Indo-European Languages



ANTON: An ASP-based Music Composition System

ANTON is an automatic composition tool that can compose melodic and harmonic music in the style of the “Palestrina Rules” for Renaissance music.

It uses an answer set solver as its core computational engine, CSOUND for synthesis and can optionally output to LILYPOND.

Declarative Problem Solving using ASP

The basic idea is

- to represent the given problem by a set of rules,
- to find answer sets for the program using an ASP solver, and
- to extract the solutions from the answer sets.

Programs

Programs consist of rules of the form

$$A_0 \leftarrow A_1, \ldots, A_m, \mathit{not}\ A_{m+1}, \ldots, \mathit{not}\ A_n$$

where each A_i is a propositional atom.

Intuitive meaning of a rule:

> If you have generated $A_1, \ldots, A_m$, and
>
> it is impossible to generate any of $A_{m+1}, \ldots, A_n$,
>
> then you may derive A_0.

Answer Sets (or Stable Models)

Program	Answer sets
$p \leftarrow \ not\ q$	$\{p\}$
$p \leftarrow \ not\ q$ $q \leftarrow \ not\ p$	$\{p\}, \{q\}$
$p \leftarrow \ not\ q$ $q \leftarrow \ not\ p$ $r \leftarrow \ p$ $r \leftarrow \ q$	$\{p, r\}, \{q, r\}$

Answer Sets vs. Models

$p \leftarrow s, \mathit{not}\ q$ $q \leftarrow s, \mathit{not}\ r$ $s \leftarrow \mathit{not}\ p$	$s \wedge \neg q \rightarrow p$ $s \wedge \neg r \rightarrow q$ $\neg p \rightarrow s$
answer set: $\{q, s\}$	models: $\{p\}$, $\{p, q\}$, $\{p, r\}$, $\{q, s\}$, $\{p, q, r\}$, $\{p, q, s\}$, $\{p, r, s\}$, $\{q, r, s\}$, $\{p, q, r, s\}$

Answer Sets and Prolog

$$p \leftarrow not\ q$$
$$q \leftarrow not\ p$$

Prolog does not terminate on query `p` or `q`.

```
?- p.
ERROR: Out of local stack
   Exception: (729,178)
```

SMODELS returns

```
Answer: 1
Stable Model: p
Answer: 2
Stable Model: q
```

Finite ASP programs are guaranteed to terminate.

More General Programs

Program	Answer sets
$1 \leq \{p, q, r\} \leq 2 \leftarrow$	$\{p\}, \{q\}, \{r\}, \{p, q\}, \{p, r\}, \{q, r\}$
$1 \leq \{p, q, r\} \leq 2 \leftarrow$ $\leftarrow p$	$\{q\}, \{r\}, \{q, r\}$

ASP Programs presented to CLASP

The ASP program

$$1 \leq \{p, q\} \leq 1$$
$$r \leftarrow p$$
$$r \leftarrow q$$

is presented to CLASP as follows:

```
1 {p, q} 1.
r :- p.
r :- q.
```

The program

$$p_i \leftarrow \mathit{not}\ p_{i+1} \qquad (1 \leq i \leq 7).$$

is presented to CLASP as follows:

```
index(1..7).
p(I) :- not p(I+1), index(I).
```

ASP Example: Clique Problem

- Given an undirected graph $G = (V, E)$ and a positive integer c, decide whether a set of c vertices that are pairwise adjacent exists.
- Generate a subset of V that have c vertices

```
c{clique(V) : vertex(V)}c.
```

Eliminate the subsets in which two vertices are not adjacent.

```
:- clique(V1), clique(V2), not edge(V1,V2),
   V1 != V2.
```

- A solution is computed using an ASP solver:

```
{clique(2), clique(7), . . .}
```

Contents

Finding Answers and Generating Explanations for Complex Biomedical Queries

Motivation

- Biomedical data is stored in various structured forms and at different locations.
- With the current Web technologies, reasoning over these data is limited to answering simple queries by keyword search and by some direction of humans.
- Vital research, like drug discovery, requires high-level reasoning.

A Simple Query

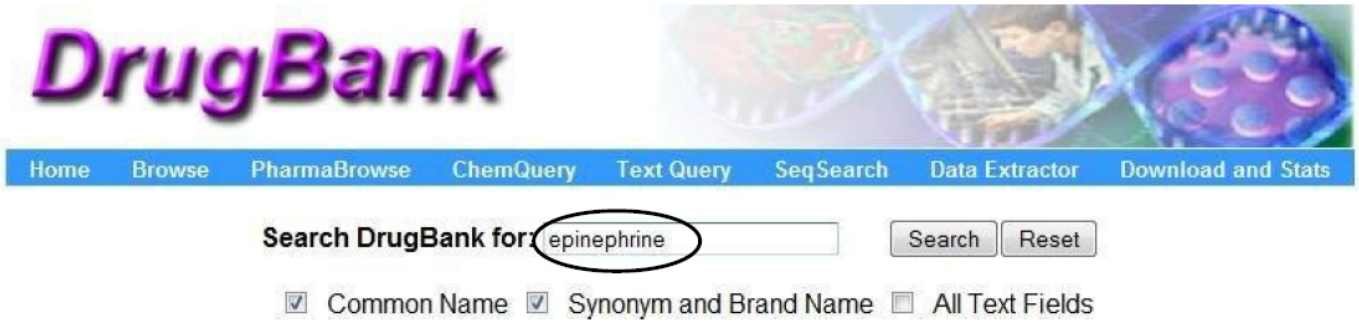
What are the genes that are targeted by the drug Epinephrine?

A Simple Query

What are the genes that are targeted by the drug Epinephrine?

A Simple Query

What are the genes that are targeted by the drug Epinephrine?



Drug Target 1	
Drug Target 1 Name	Alpha-1A adrenergic receptor
Drug Target 1 Synonyms	1. Alpha 1A-adrenoceptor 2. Alpha 1A- adrenoreceptor 3. Alpha-1C adrenergic receptor 4. Alpha adrenergic receptor 1c
Drug Target 1 Gene Name	ADRA1A

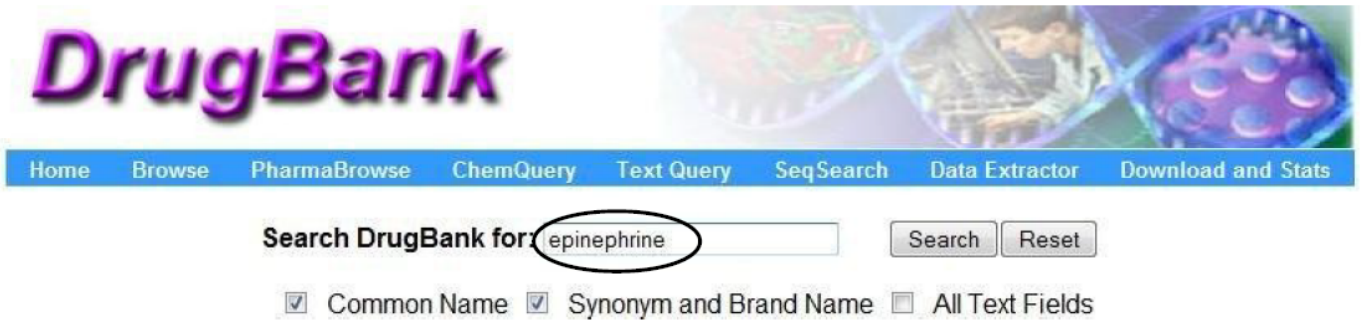
A Simple Query

What are the genes that are targeted by the drug Epinephrine?

Drug Target 2	
Drug Target 2 Name	Beta-1 adrenergic receptor
Drug Target 2 Synonyms	1. Beta-1 adrenoceptor 2. Beta-1 adrenoreceptor
Drug Target 2 Gene Name	ADRB1

A Simple Query

What are the genes that are targeted by the drug Epinephrine?



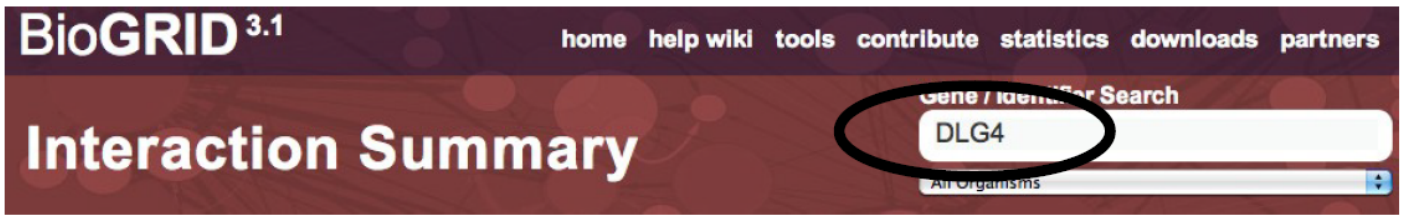
Drug Target 3	
Drug Target 3 Name	Beta-2 adrenergic receptor
Drug Target 3 Synonyms	1. Beta-2 adrenoceptor 2. Beta-2 adrenoreceptor
Drug Target 3 Gene Name	ADRB2

Another Simple Query

What are the genes that interact with the gene DLG4?

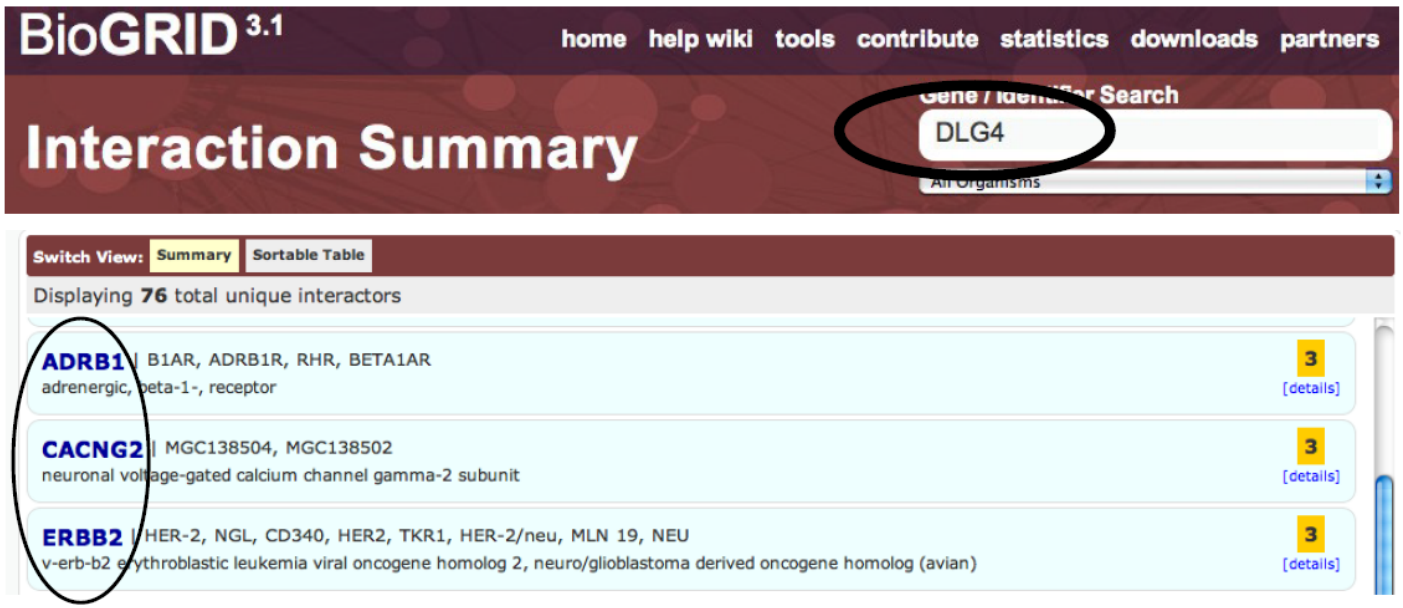
Another Simple Query

What are the genes that interact with the gene DLG4?



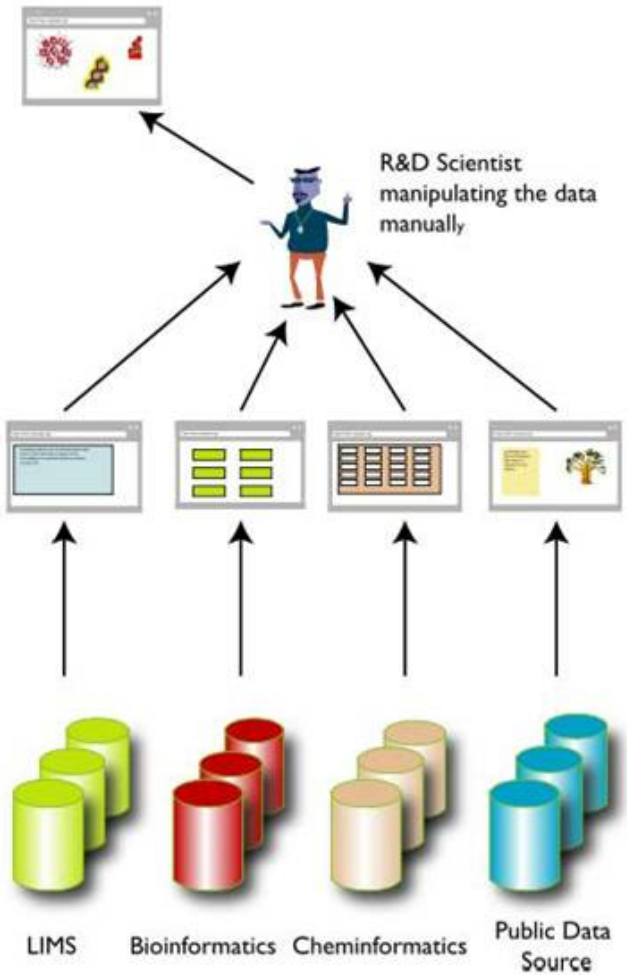
Another Simple Query

What are the genes that interact with the gene DLG4?



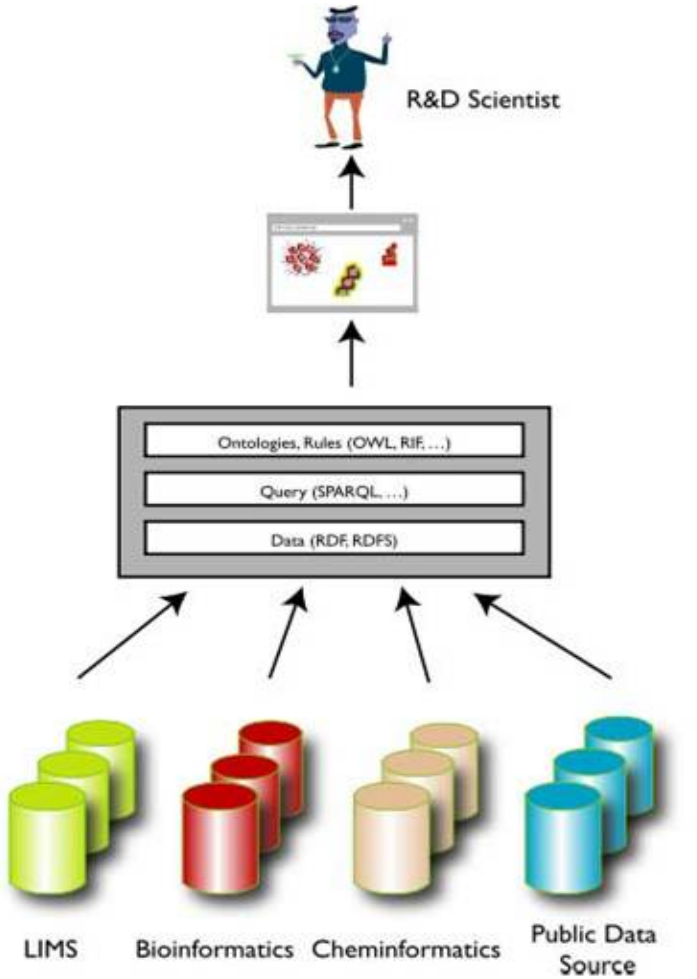
Complex Queries

What are the genes that are targeted by the drug Epinephrine and that interact with the gene DLG4?



Our goal is...

... to extract relevant parts of the knowledge resources, integrate them, answer the queries efficiently, and generate explanations.



Complex Queries

- Q1 What are the genes that are targeted by the drug Epinephrine and that interact with the gene DLG4?
- Q2 What are the genes that are targeted by all the drugs that belong to the category Hmg-coa reductase inhibitors?
- Q3 What are the cliques of 5 genes, that contain the gene DLG4?
- Q4 What are the genes that are related to the gene ADRB1 via a gene-gene relation chain of length at most 3?
- Q5 What are the most similar 3 genes that are targeted by the drug Epinephrine?

Challenges

1. It is hard to represent a query in a formal language.

Challenges

1. It is hard to represent a query in a formal language.
   - Represent queries in a controlled natural language – BIOQUERY-CNL* [EY09, EEO11].

Challenges

1. It is hard to represent a query in a formal language.
   - Represent queries in a controlled natural language – BioQuery-CNL* [EY09, EEO11].
2. Databases/ontologies are in different formats/locations.

Challenges

1. It is hard to represent a query in a formal language.
   - Represent queries in a controlled natural language – BioQuery-CNL* [EY09, EEO11].
2. Databases/ontologies are in different formats/locations.
   - Integration of knowledge via a rule layer in ASP [BCD+08, EEO11].

Challenges

1. It is hard to represent a query in a formal language.
   - Represent queries in a controlled natural language – BioQuery-CNL* [EY09, EEO11].
2. Databases/ontologies are in different formats/locations.
   - Integration of knowledge via a rule layer in ASP [BCD+08, EEO11].
3. Complex queries require recursive definitions, aggregates, etc..

Challenges

1. It is hard to represent a query in a formal language.
   - Represent queries in a controlled natural language – BIOQUERY-CNL* [EY09, EEO11].
2. Databases/ontologies are in different formats/locations.
   - Integration of knowledge via a rule layer in ASP [BCD+08, EEO11].
3. Complex queries require recursive definitions, aggregates, etc..
   - Represent queries as ASP programs [BCD+08, EEEO11].

Challenges

1. It is hard to represent a query in a formal language.
   - Represent queries in a controlled natural language – BioQuery-CNL* [EY09, EEO11].
2. Databases/ontologies are in different formats/locations.
   - Integration of knowledge via a rule layer in ASP [BCD+08, EEO11].
3. Complex queries require recursive definitions, aggregates, etc..
   - Represent queries as ASP programs [BCD+08, EEEO11].
4. Databases/ontologies are large.

Challenges

1. It is hard to represent a query in a formal language.
   - Represent queries in a controlled natural language – BioQuery-CNL* [EY09, EEO11].
2. Databases/ontologies are in different formats/locations.
   - Integration of knowledge via a rule layer in ASP [BCD+08, EEO11].
3. Complex queries require recursive definitions, aggregates, etc..
   - Represent queries as ASP programs [BCD+08, EEEO11].
4. Databases/ontologies are large.
   - Extract the relevant part for faster reasoning [EEEO11].

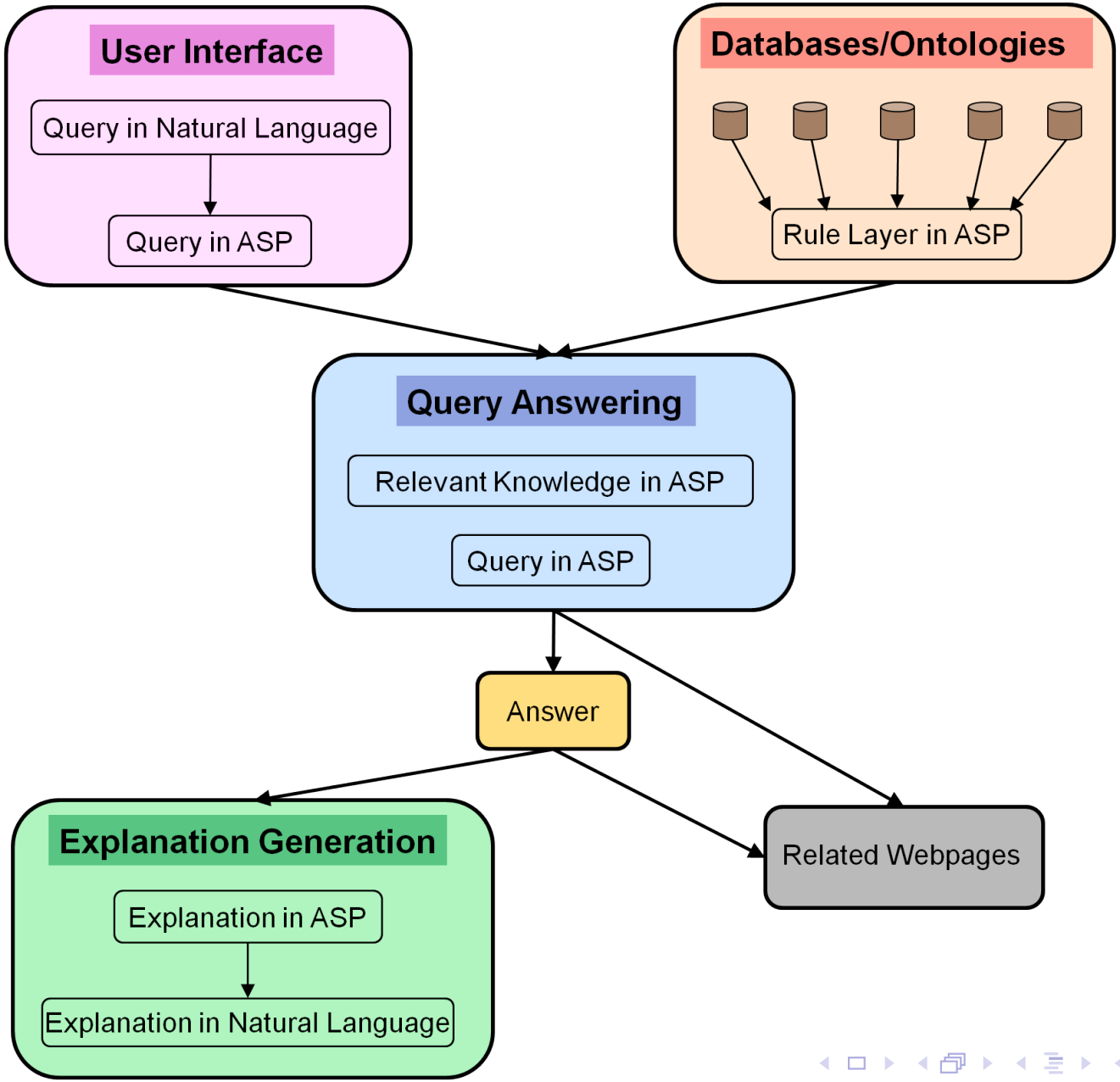
Challenges

1. It is hard to represent a query in a formal language.
   - Represent queries in a controlled natural language – BioQuery-CNL* [EY09, EEO11].
2. Databases/ontologies are in different formats/locations.
   - Integration of knowledge via a rule layer in ASP [BCD+08, EEO11].
3. Complex queries require recursive definitions, aggregates, etc..
   - Represent queries as ASP programs [BCD+08, EEEO11].
4. Databases/ontologies are large.
   - Extract the relevant part for faster reasoning [EEEO11].
5. Experts may ask for further explanations.

Challenges

1. It is hard to represent a query in a formal language.
   - Represent queries in a controlled natural language – BioQuery-CNL* [EY09, EEO11].
2. Databases/ontologies are in different formats/locations.
   - Integration of knowledge via a rule layer in ASP [BCD+08, EEO11].
3. Complex queries require recursive definitions, aggregates, etc..
   - Represent queries as ASP programs [BCD+08, EEEO11].
4. Databases/ontologies are large.
   - Extract the relevant part for faster reasoning [EEEO11].
5. Experts may ask for further explanations.
   - Algorithm for generating shortest explanations [EEEO11].

BioQuery-ASP: System Overview



Representing Queries in ASP

Query Q2 in BIOQUERY-CNL*: What are the genes that are targeted by all the drugs that belong to the category Hmg-coa reductase inhibitors?

Query Q2 in ASP:

$$notcommon(gn_1) \leftarrow not\ drug\_gene(d_2, gn_1), condition_1(d_2)$$
$$condition_1(d) \leftarrow drug\_category(d, \text{“}Hmg - coa\ reductase\ inhibitors\text{”})$$

$$what\_be\_genes(gn_1) \leftarrow not\ notcommon(gn_1), notcommon\_exists$$
$$notcommon\_exists \leftarrow notcommon(x)$$

$$answer\_exists \leftarrow what\_be\_genes(gn)$$

Extraction and Integration of Knowledge using ASP

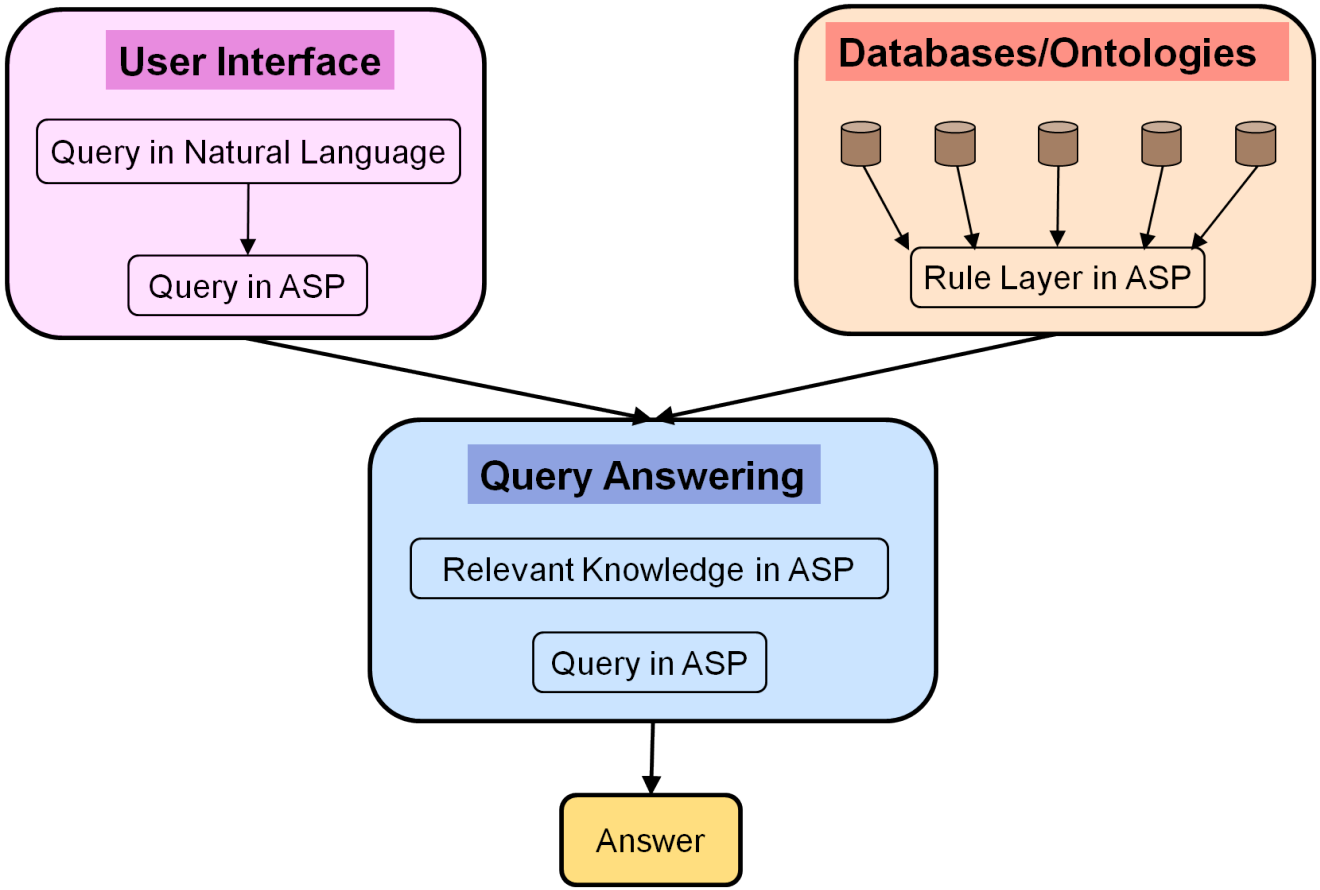
Knowledge from RDF(S)/OWL ontologies can be extracted using "external predicates" supported by the ASP solver DLVHEX [EGRH06]:

$$triple\_gene(x, y, z) \leftarrow \&rdf[\text{“}URIforGeneOntology\text{”}](x, y, z)$$
$$gene\_gene(g_1, g_2) \leftarrow triple\_gene(x, \text{“}geneproperties : name\text{”}, g_1),$$
$$triple\_gene(x, \text{“}geneproperties : related\_genes\text{”}, b), \ldots$$

ASP rules integrate the extracted knowledge, or define new concepts:

$$gene\_reachable\_from(x, 1) \leftarrow gene\_gene(x, y), start\_gene(y)$$
$$gene\_reachable\_from(x, n+1) \leftarrow gene\_gene(x, z),$$
$$gene\_reachable\_from(z, n), max\_chain\_length(l) \quad (0 < n, n < l)$$

Query Answering in ASP



- Generally, only a small part of the underlying databases and the rule layer is related to the given query.
- We introduce a method to identify the relevant part of the ASP program for more efficient query answering.

Relevant Part of a Program

Underlying databases as facts:

$gene\_gene(G1, G2) \leftarrow$ $\quad$ $gene\_gene(G2, G3) \leftarrow$

$drug\_drug(D1, D2) \leftarrow$ $\quad$ $drug\_drug(D2, D3) \leftarrow$

Rule layer:

$gene\_gene(g_1, g_2) \leftarrow gene\_gene(g_2, g_1)$

$gene\_related\_gene(g_1, g_2) \leftarrow gene\_gene(g_1, g_2)$

$gene\_related\_gene(g_1, g_3) \leftarrow gene\_related\_gene(g_1, g_2), gene\_gene(g_2, g_3)$

$drug\_drug(d_1, g_2) \leftarrow drug\_drug(d_2, d_1)$

$drug\_related\_drug(g_1, g_2) \leftarrow drug\_drug(d_1, d_2)$

$drug\_related\_drug(g_1, g_3) \leftarrow drug\_related\_drug(d_1, d_2), drug\_drug(d_2, d_3)$

Query: What are the genes that are related to gene $G1$?

$what\_be\_genes(g) \leftarrow gene\_related\_gene(g, G1)$

Relevant Part of a Program

Underlying databases as facts:

$gene\_gene(G1, G2) \leftarrow$ $gene\_gene(G2, G3) \leftarrow$

$drug\_drug(D1, D2) \leftarrow$ $drug\_drug(D2, D3) \leftarrow$

Rule layer:

$gene\_gene(g_1, g_2) \leftarrow gene\_gene(g_2, g_1)$

$gene\_related\_gene(g_1, g_2) \leftarrow gene\_gene(g_1, g_2)$

$gene\_related\_gene(g_1, g_3) \leftarrow gene\_related\_gene(g_1, g_2), gene\_gene(g_2, g_3)$

$drug\_drug(d_1, g_2) \leftarrow drug\_drug(d_2, d_1)$

$drug\_related\_drug(g_1, g_2) \leftarrow drug\_drug(d_1, d_2)$

$drug\_related\_drug(g_1, g_3) \leftarrow drug\_related\_drug(d_1, d_2), drug\_drug(d_2, d_3)$

Query: What are the genes that are related to gene $G1$?

$what\_be\_genes(g) \leftarrow gene\_related\_gene(g, G1)$

* Identifying the relevant part improves the computational time up to 7 times.

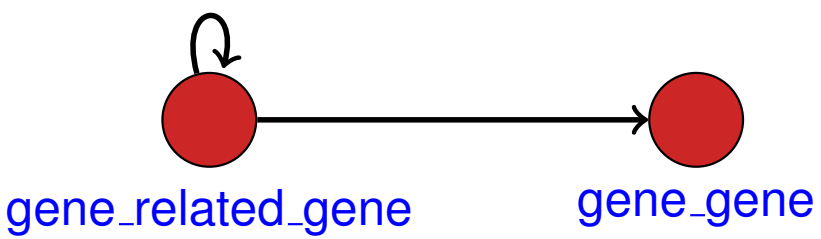
Identifying the Relevant Part of a Program

Predicate Dependency Graph

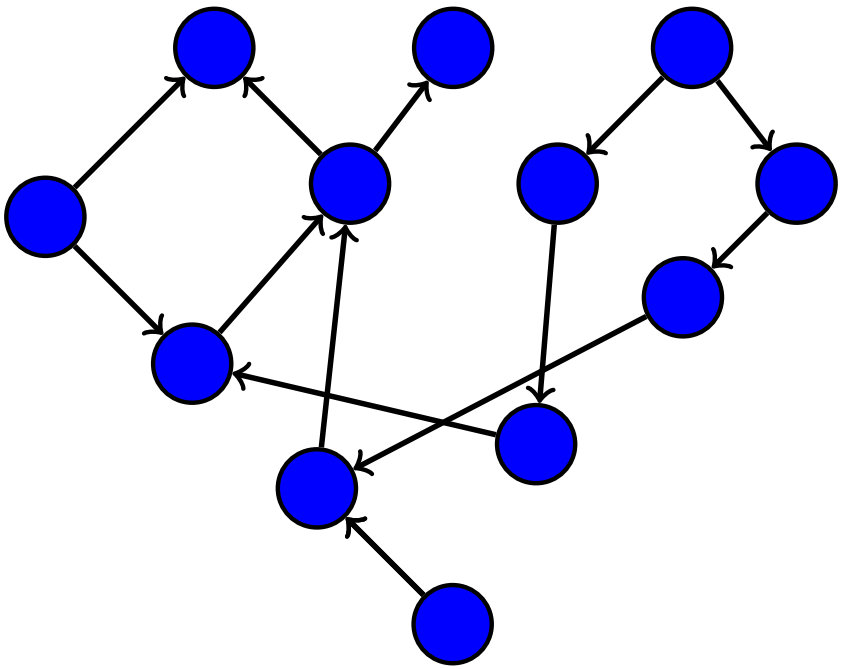
It is a directed graph where the vertices represent the predicate symbols and the edges $\langle p_i, p_j \rangle$ denote the existence of a rule r, such that $p_i \in HP(r)$ and $p_j \in BP(r)$.

Example:

$$gene\_related\_gene(g1, g3) \leftarrow gene\_related\_gene(g1, g2), \\ gene\_gene(g2, g3)$$



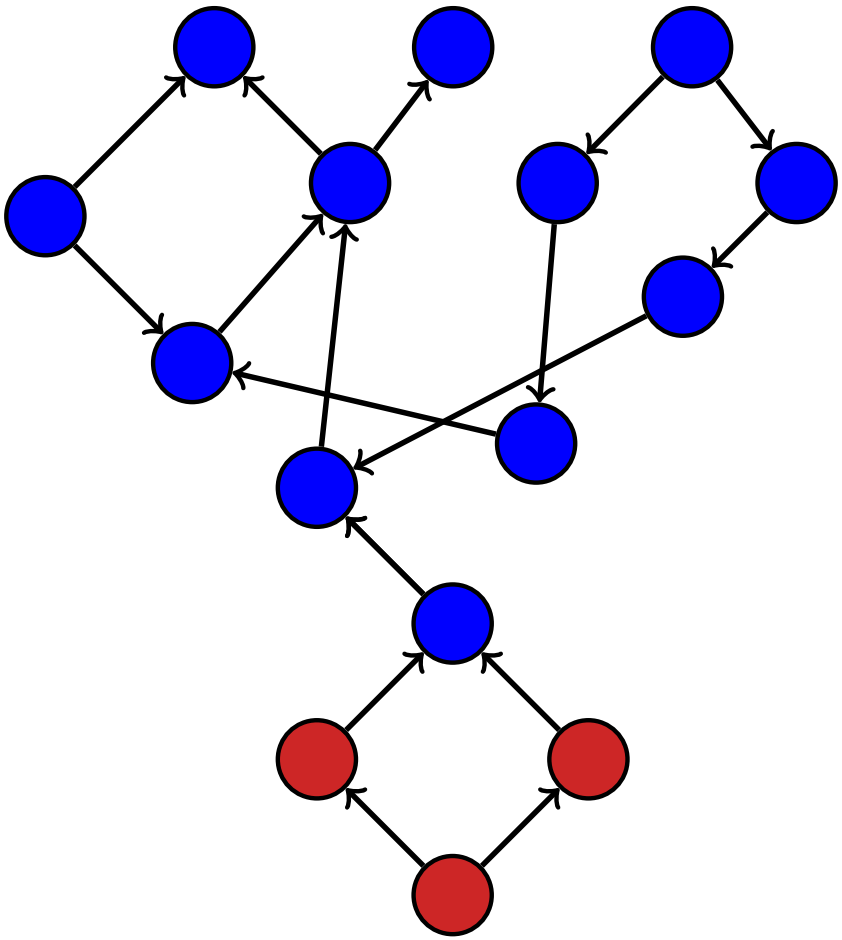
Identifying the Relevant Part of a Program



% Databases and Ontologies:
fact 1.
fact 2.
fact 3.
:
:

% Rule Layer:
rule 1.
rule 2.
rule 3.
:
:

Identifying the Relevant Part of a Program

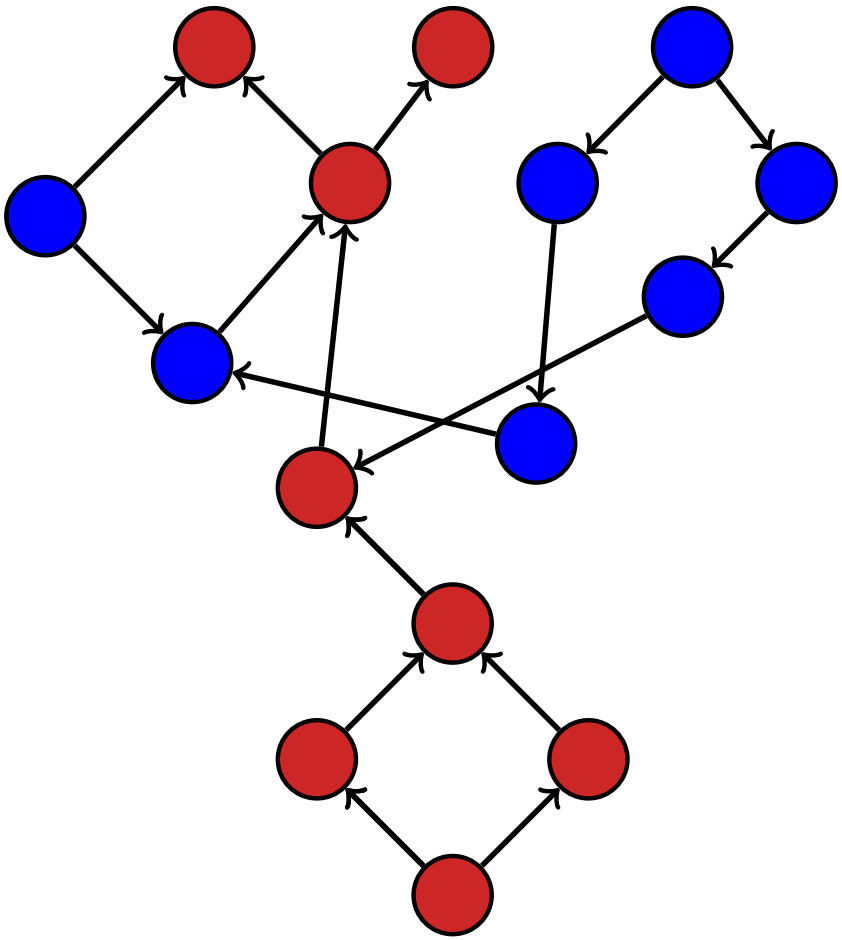


% Databases and Ontologies:
fact 1.
fact 2.
fact 3.
⋮

% Rule Layer:
rule 1.
rule 2.
rule 3.
⋮

% Query:
rule 1.
rule 2.
⋮

Identifying the Relevant Part of a Program



% Databases and Ontologies:
fact 1.
fact 2.
fact 3.
⋮

% Rule Layer:
rule 1.
rule 2.
rule 3.
⋮

% Query:
rule 1.
rule 2.
⋮

Identifying the Relevant Part of a Program

Theorem 1

Let Π be a stratified normal program, Q be a general program. Then $Rel_{\Pi,Q}$ is the relevant part of Π with respect to Q.

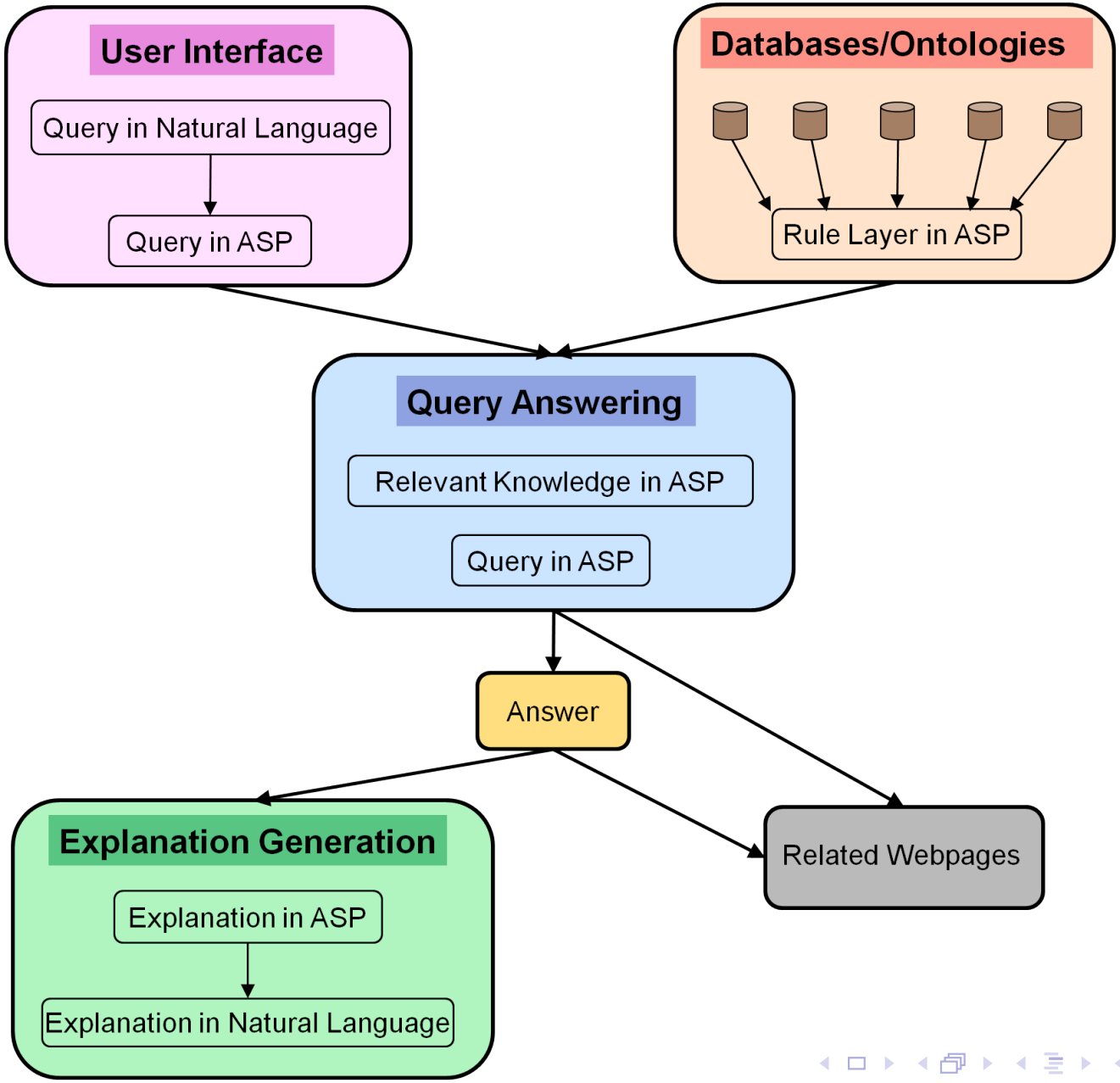
Experimental Results: Databases & Ontologies

Source	Relation (number of ASP facts)
BIOGRID	gene-gene (372.293)
DRUGBANK	drug-drug (21.756) drug-category (4.743)
SIDER	drug-sideeffect (61.102)
PHARMGKB	drug-disease (3.740) drug-gene (15.805) disease-gene (9.417)
CTD	drug-disease (704.590) drug-gene (259.048) disease-gene (8.909.071)
	Total : 10.3 M

Experimental Results

Query	Complete	Relevant
Q1	271.39 Rules: 21059323	**13.08** Rules: 1961789
Q2	266.06 Rules: 21059909	**14.34** Rules: 2084579
Q3	266.62 Rules: 21059248	**9.85** Rules: 1567401
Q4	**273.93** Rules: 21059353	321.11 Rules: 19450525
Q5	265.91 Rules: 21061727	**9.93** Rules: 1460831
Q6	**269.69** Rules: 21111842	320.56 Rules: 19512500
Q7	270.05 Rules: 21062006	**6.07** Rules: 1023061
Q8	275.19 Rules: 21079275	**7.02** Rules: 1040406
Q9	272.48 Rules: 21059597	**3.48** Rules: 547545
Q10	266.37 Rules: 21077252	**11.25** Rules: 1594891

BioQuery-ASP: System Overview



Explanations

ASP program Π:

$$a \leftarrow b, c$$
$$a \leftarrow d$$
$$d \leftarrow$$
$$b \leftarrow c$$
$$c \leftarrow$$

An answer set X for Π: $\{a, b, c, d\}$

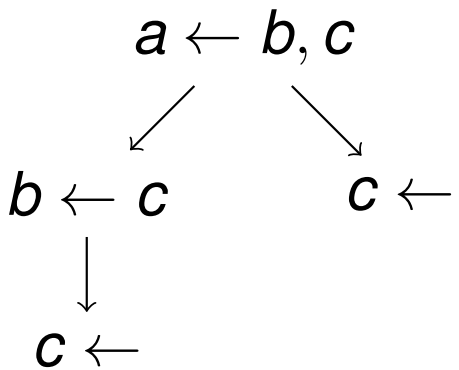
Explanations

ASP program Π:

$$a \leftarrow b, c$$
$$a \leftarrow d$$
$$d \leftarrow$$
$$b \leftarrow c$$
$$c \leftarrow$$

An answer set X for Π: $\{a, b, c, d\}$

An explanation for a wrt Π and X:



Explanations

ASP program Π:

$$a \leftarrow b, c$$
$$a \leftarrow d$$
$$d \leftarrow$$
$$b \leftarrow c$$
$$c \leftarrow$$

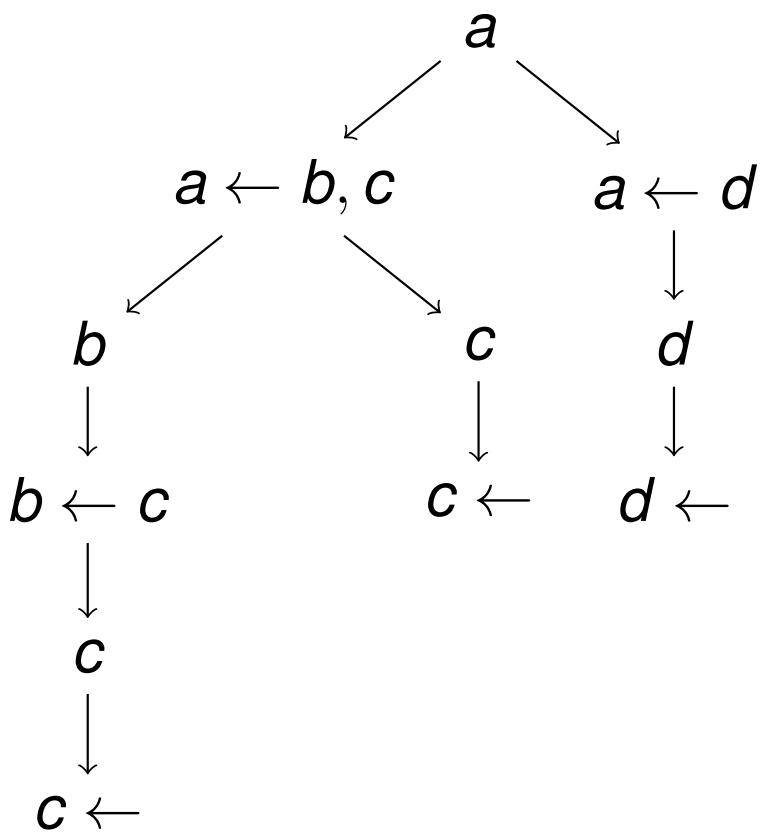
An answer set X for Π: $\{a, b, c, d\}$

Another explanation for a wrt Π and X:

$$a \leftarrow d$$
$$\downarrow$$
$$d \leftarrow$$

Finding Explanations

The and-or explanation tree for atom a with respect to Π and X:



Π :

$a \leftarrow b, c$

$a \leftarrow d$

$d \leftarrow$

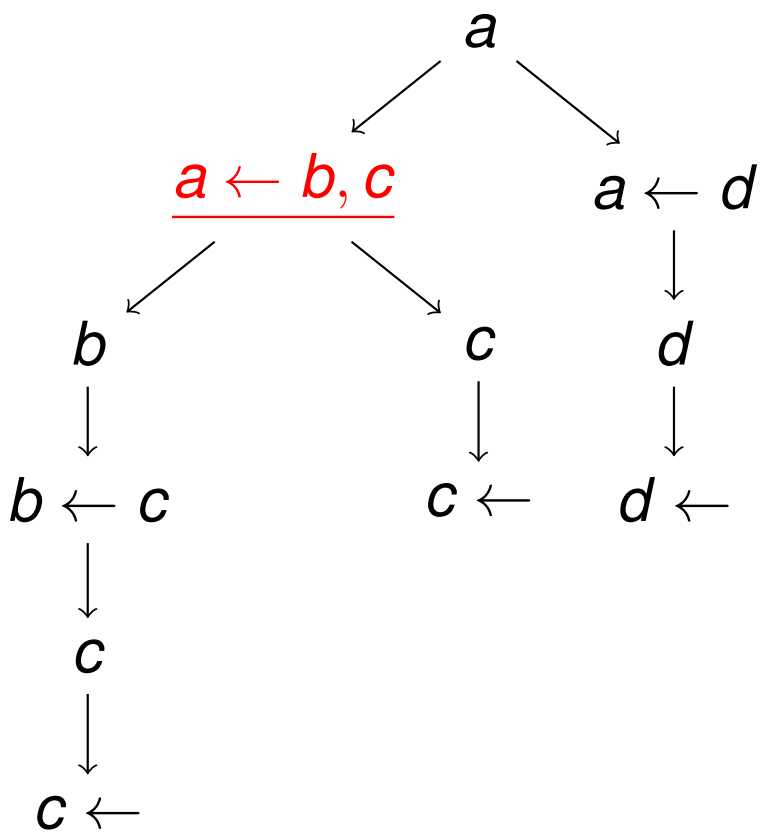
$b \leftarrow c$

$c \leftarrow$

$X = \{a, b, c, d\}$

Finding Explanations

The and-or explanation tree for atom a with respect to Π and X:



Π :

$a \leftarrow b, c$

$a \leftarrow d$

$d \leftarrow$

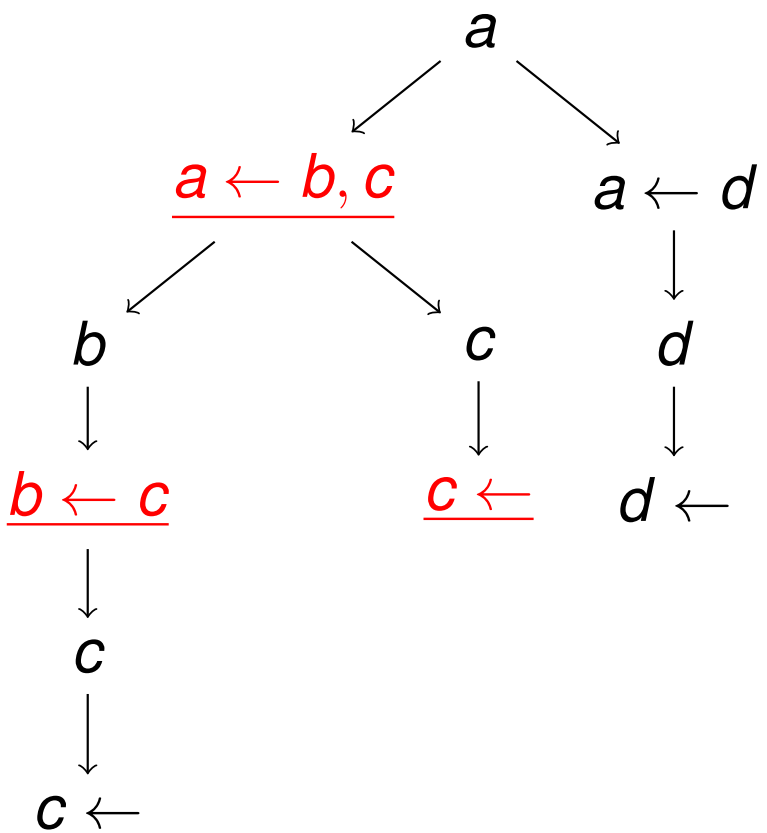
$b \leftarrow c$

$c \leftarrow$

$X = \{a, b, c, d\}$

Finding Explanations

The and-or explanation tree for atom a with respect to Π and X:



Π :

$a \leftarrow b, c$

$a \leftarrow d$

$d \leftarrow$

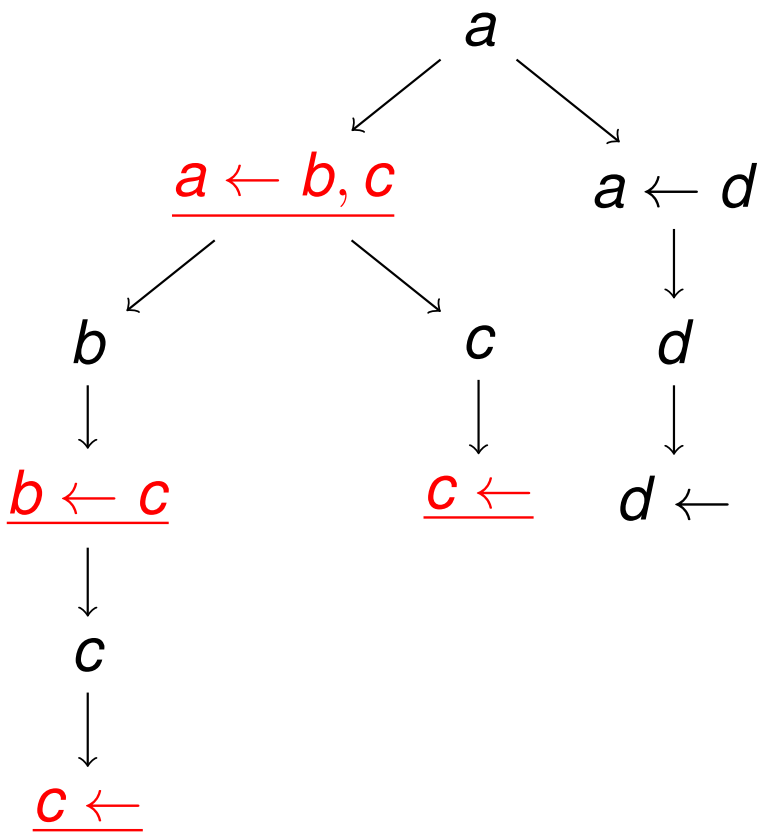
$b \leftarrow c$

$c \leftarrow$

$X = \{a, b, c, d\}$

Finding Explanations

The and-or explanation tree for atom a with respect to Π and X:



Π :

$a \leftarrow b, c$

$a \leftarrow d$

$d \leftarrow$

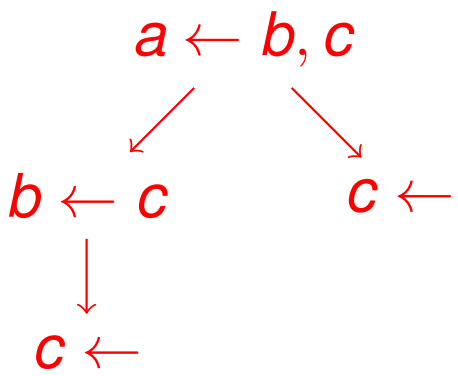
$b \leftarrow c$

$c \leftarrow$

$X = \{a, b, c, d\}$

Finding Explanations

An explanation for atom a with respect to Π and X:



Π :

$a \leftarrow b, c$

$a \leftarrow d$

$d \leftarrow$

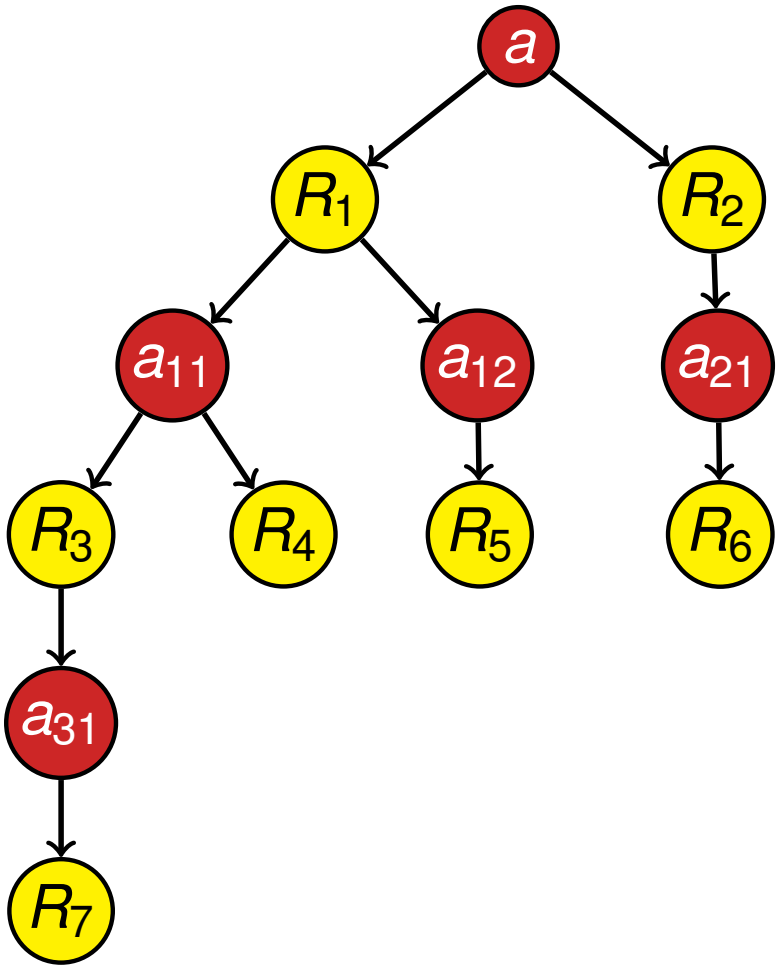
$b \leftarrow c$

$c \leftarrow$

$X = \{a, b, c, d\}$

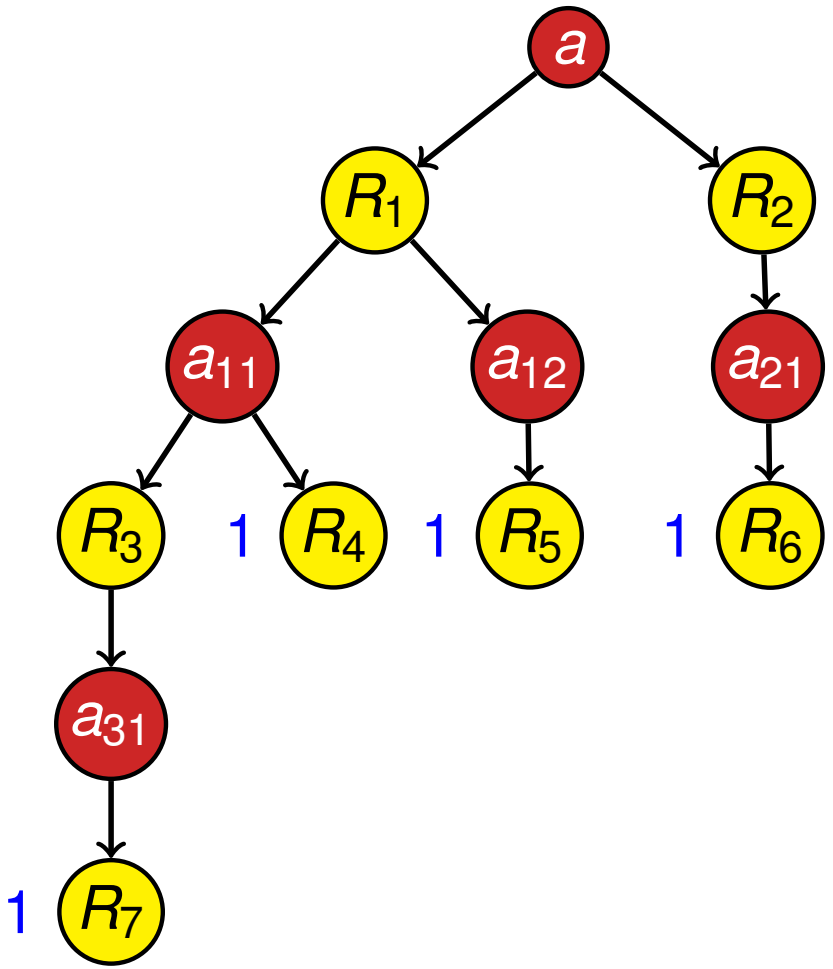
Shortest Explanations

- $W(a) = min_{c \in child(a)}(W(c))$
- $W(r) = \sum_{c \in child(r)} W(c) + 1$



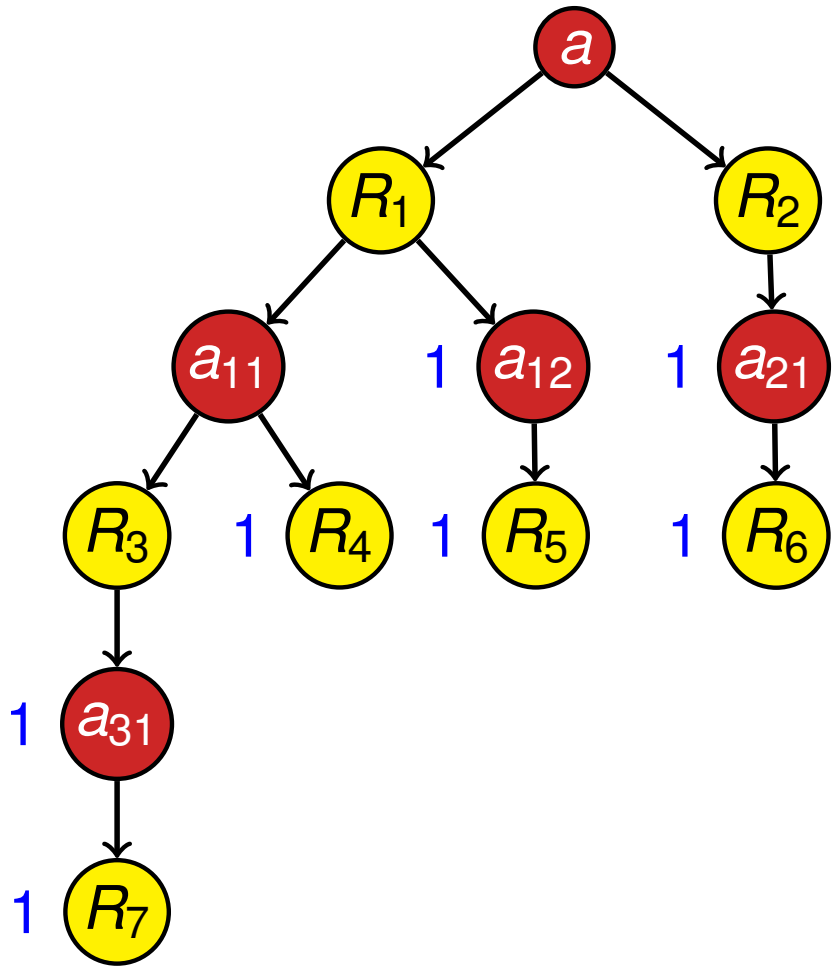
Shortest Explanations

- $W(a) = min_{c \in child(a)}(W(c))$
- $W(r) = \sum_{c \in child(r)} W(c) + 1$



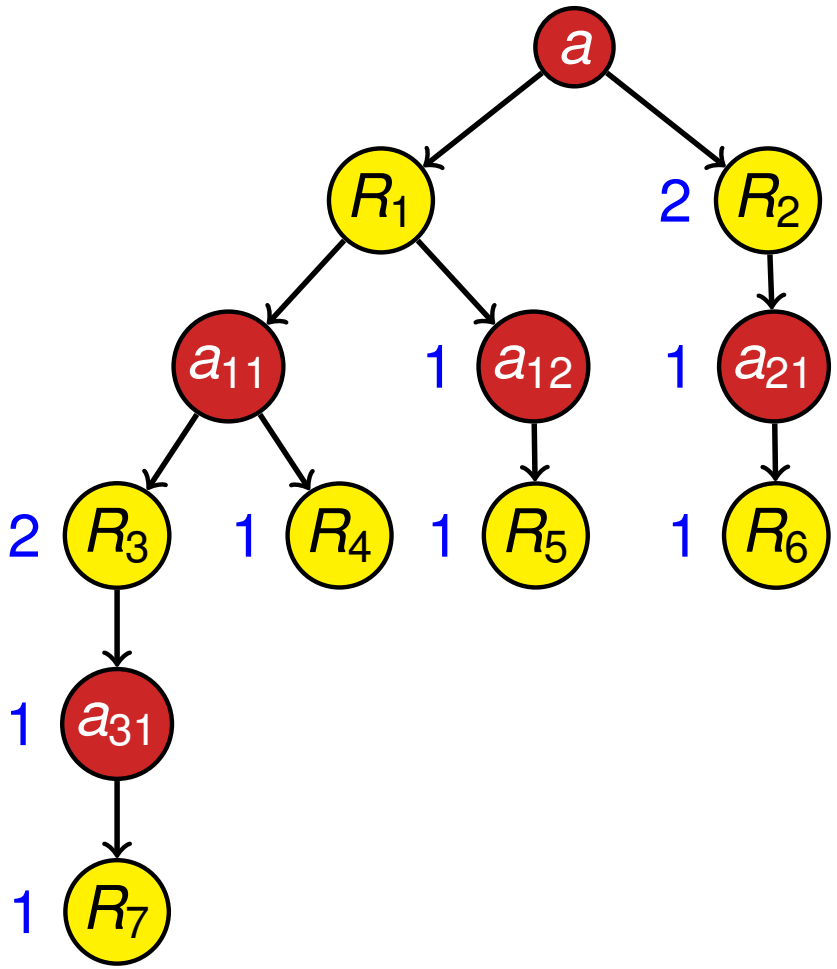
Shortest Explanations

- $W(a) = min_{c \in child(a)}(W(c))$
- $W(r) = \sum_{c \in child(r)} W(c) + 1$



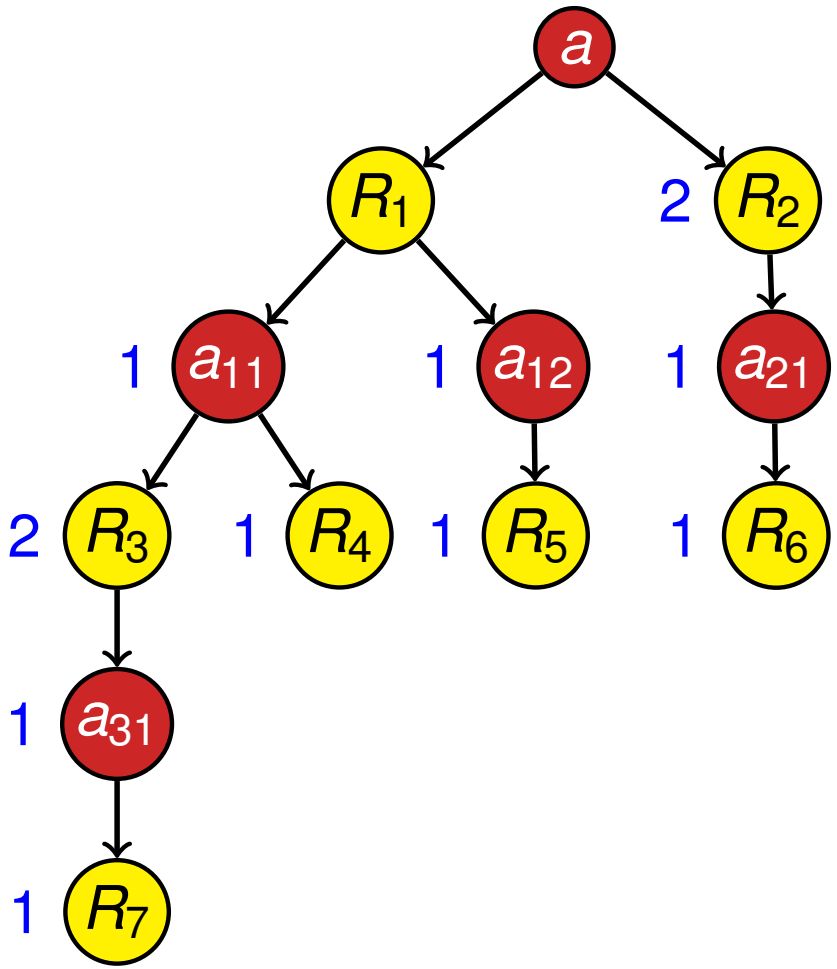
Shortest Explanations

- $W(a) = min_{c \in child(a)}(W(c))$
- $W(r) = \sum_{c \in child(r)} W(c) + 1$



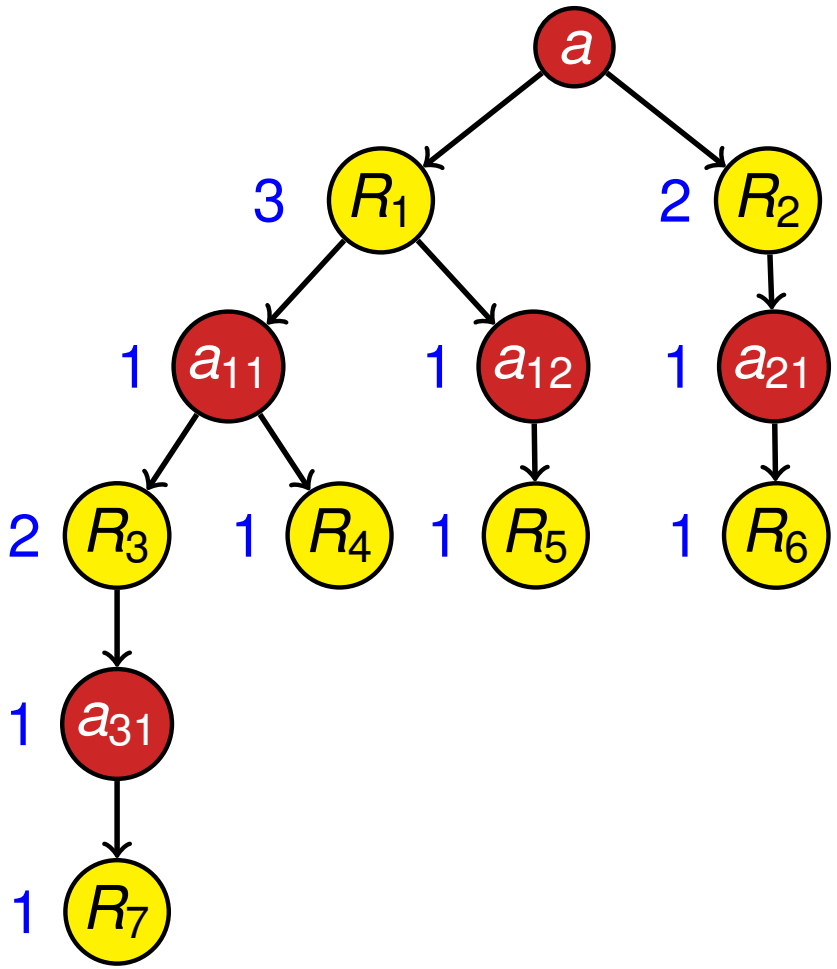
Shortest Explanations

- $W(a) = min_{c \in child(a)}(W(c))$
- $W(r) = \sum_{c \in child(r)} W(c) + 1$



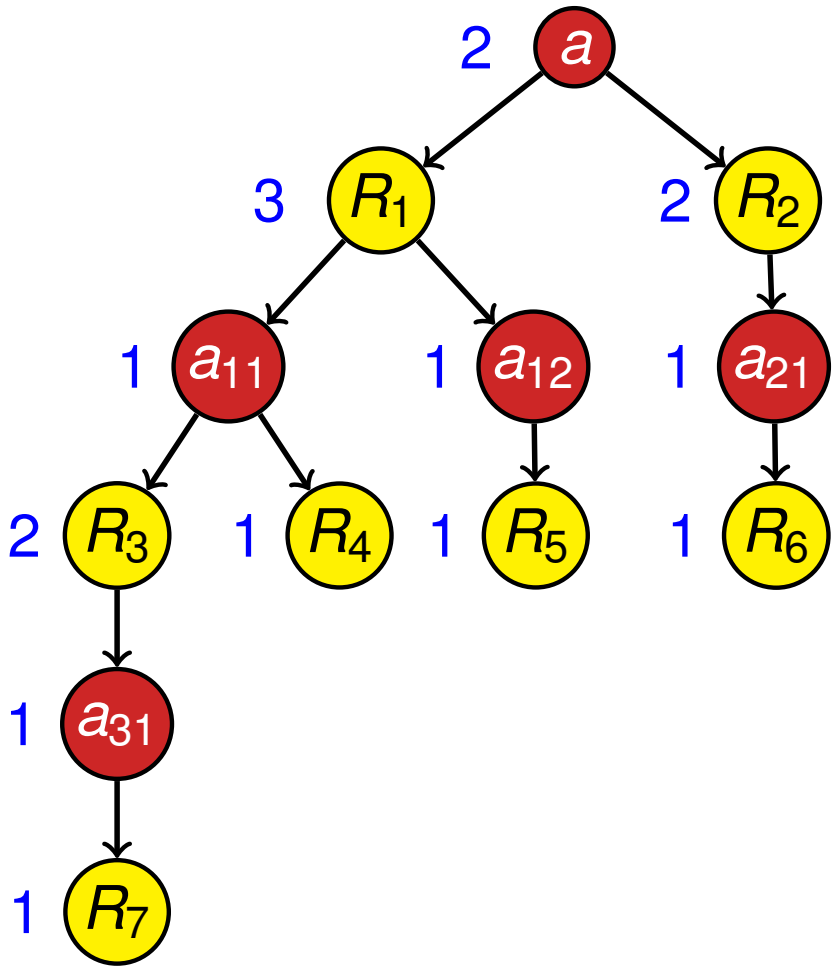
Shortest Explanations

- $W(a) = min_{c \in child(a)}(W(c))$
- $W(r) = \sum_{c \in child(r)} W(c) + 1$



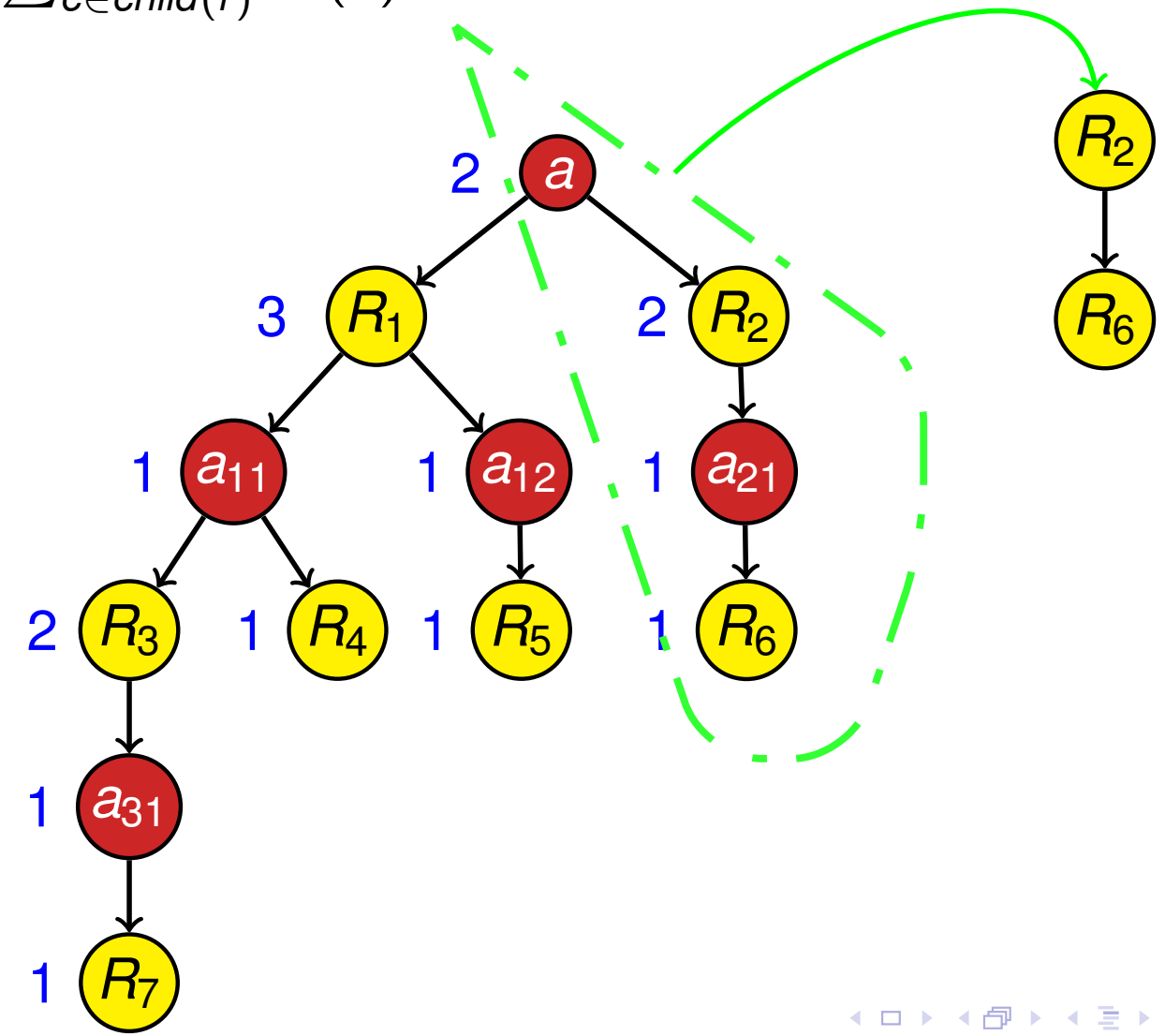
Shortest Explanations

- $W(a) = min_{c \in child(a)}(W(c))$
- $W(r) = \sum_{c \in child(r)} W(c) + 1$



Shortest Explanations

- $W(a) = min_{c \in child(a)}(W(c))$
- $W(r) = \sum_{c \in child(r)} W(c) + 1$



Explanation Generation

Theorem 2

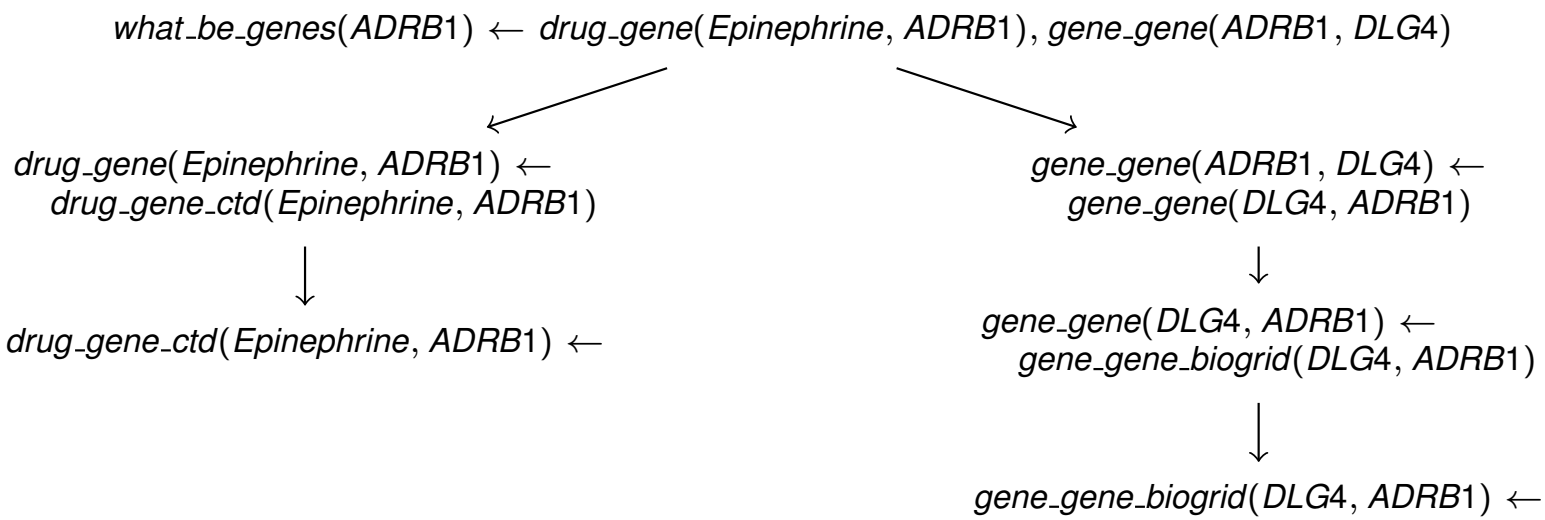
Let Π be a normal ASP program, X be an answer set for Π and p be an atom in X. Our algorithm generates a shortest explanation for p with respect to Π and X.

Example: Explanation Generation

Query in BioQuery-CNL*: What are the genes that are targeted by the drug Epinephrine and that interact with the gene DLG4?

An Answer: `ADRB1`

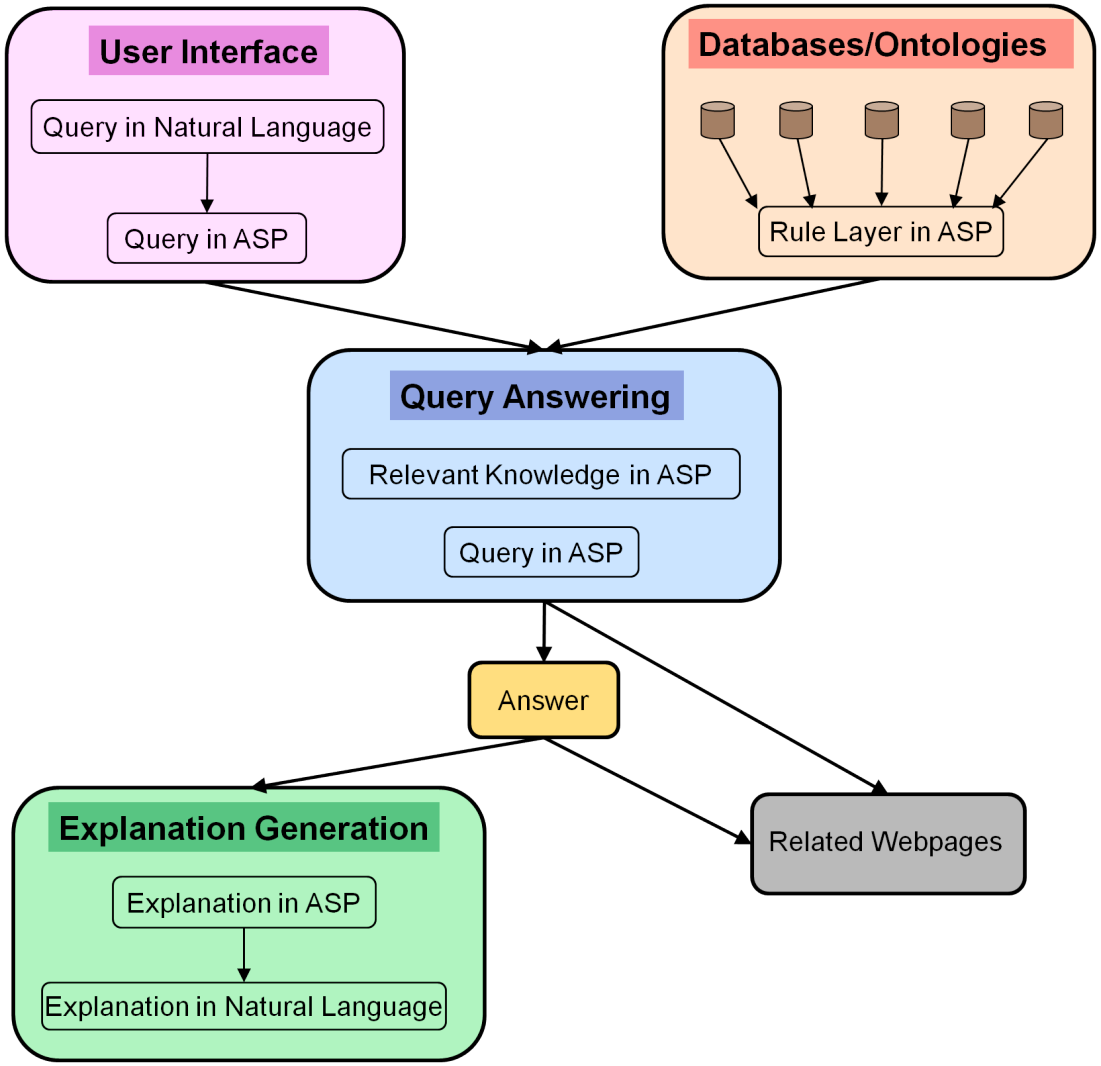
Shortest Explanation in ASP:



Explanation in Natural Language:

```
The drug Epinephrine targets the gene ADRB1 according to CTD.
The gene DLG4 interacts with the gene ADRB1 according to BioGrid.
```

BioQuery-ASP



http://krr.sabanciuniv.edu/projects/BioQuery-ASP/

Contents

Positive Programs: Syntax

positive rule:

$$A_0 \leftarrow A_1 \wedge \cdots \wedge A_m$$

where $A_0, \ldots, A_m$ are propositional atoms. We identify a positive rule with an implication

$$A_1 \wedge \cdots \wedge A_m \rightarrow A_0 \ .$$

Example

$$p$$
$$r \leftarrow p \wedge q$$

is a positive program, which can be identified with

$$p \wedge (p \wedge q \rightarrow r).$$

Stable Models of a Positive Program

We identify an interpretation with the set of atoms that are true in it.

- An interpretation I of signature $\{p, q\}$ such that $I(p) = \mathbf{f}$ and $I(q) = \mathbf{t}$ is identified with $\{q\}$.

$$p$$
$$r \leftarrow p \wedge q$$

has three models: $\{p\}$, $\{p, r\}$, $\{p, q, r\}$.

Every positive program has a unique minimal model. That model is called the stable model (a.k.a. answer set) of the program.

Normal Logic Program

(Normal) rule with negation:

$$A_0 \leftarrow A_1 \wedge \cdots \wedge A_m \wedge \neg A_{m+1} \wedge \cdots \wedge \neg A_n$$

(often written as $A_0 \leftarrow A_1, \ldots, A_m,\ \mathit{not}\ A_{m+1}, \ldots, \mathit{not}\ A_n$)
Informally,

> If you have generated $A_1, \ldots, A_m$, and
>
> it is impossible to generate any of $A_{m+1}, \ldots, A_n$,
>
> then you may generate A_0.

A stable model of Π is a set of atoms that can be generated from Π.

How do we know it is impossible to generate negated atoms?

$$p \leftarrow \neg q \qquad\qquad p \leftarrow \neg q$$
$$q \leftarrow \neg r \qquad\qquad q \leftarrow \neg p$$

Fixpoint Definition

The difficulty is overcome by employing a “fixpoint construct” called reduct [GL88].

To find a set of atoms that can be generated from Π:

- Guess a set X that you suspect to be the set of atoms that can be generated from Π.
- Transform Π into a positive program Π^X (reduct of Π relative to X) by assuming that only atoms in X can be generated.
- If the set of atoms that can be generated from Π^X is identical to X, then X is a good “guess.”

$$\Pi: \quad p \leftarrow \neg q \qquad\qquad \Pi^{\{q\}}: \quad$$
$$q \leftarrow \neg r \qquad\qquad q \leftarrow$$

Stable Models of a Normal Logic Program [GL88]

Let Π be a normal logic program and X a set of atoms. The reduct Π^X is obtained from Π by replacing every occurrence of the form $\neg A$ by

- $\top$ if $X \models \neg A$ (i.e., $A \notin X$), and
- $\bot$ otherwise.

$$\Pi: \begin{array}{l} p \leftarrow \neg q \\ q \leftarrow \neg r \end{array} \qquad \Pi^{\{q\}}: \begin{array}{l} p \leftarrow \bot \\ q \leftarrow \top \end{array}$$

X is a stable model (a.k.a. answer set) of Π if X is the minimal model of Π^X.

To find a stable model of Π:

1. Guess X and form Π^X.
2. Find the minimal model Y of Π^X.
3. If $Y = X$, X is a stable model of Π.

Examples

$$\Pi: \begin{array}{l} p \leftarrow \neg q \\ q \leftarrow \neg r \end{array} \qquad \Pi^{\{q\}}: \begin{array}{l} p \leftarrow \bot \\ q \leftarrow \top \end{array} \qquad \Pi^{\{p\}}: \begin{array}{l} p \leftarrow \top \\ q \leftarrow \top \end{array}$$

$$\Pi^{\{p,q\}}: \begin{array}{l} p \leftarrow \bot \\ q \leftarrow \top \end{array} \qquad \Pi^{\emptyset}: \begin{array}{l} p \leftarrow \top \\ q \leftarrow \top \end{array}$$

$\{q\}$ is the only stable model of Π.

Theorem

If X is a stable model of Π, then every element of X appears in the head of a rule in Π.

Examples

A normal logic program may have none, one, or multiple stable models.

Program	Stable models
$p \leftarrow \neg q$ $q \leftarrow \neg p$	{p}, {q}
$p \leftarrow \neg p$	none

General Rule [LTT99, Fer05]

$$F \leftarrow G$$

where F and G are formulas that contain no connectives other than $\{\bot, \top, \wedge, \vee, \neg\}$.

The reduct Π^X is obtained from Π by replacing all maximal subformulas of the form $\neg H$ by

- $\top$ if $X \models \neg H$, and
- $\bot$ otherwise.

We say that X is a stable model of Π if X is a minimal model of Π^X.

(Π^X may have none, one, or multiple minimal models.)

Examples

General program	Stable models
$p \vee q$	$\{p\}, \{q\}$
$p \vee q$ $p \leftarrow q$ $q \leftarrow p$	$\{p, q\}$
p	{p}
$\neg\neg p$	none
$p \vee \neg p$	$\emptyset, \{p\}$
$p \leftarrow \neg\neg p$	$\emptyset, \{p\}$

Propositional logic is monotonic: if X satisfies $F \wedge G$ then X satisfies F.
Stable model semantics is nonmonotonic.

ASP Idioms

- Choice rules
- Constraints
- Cardinality expressions

Choice Rules

By $\{p, q, r\}^c$ we denote the rule

$$(p \vee \neg p) \wedge (q \vee \neg q) \wedge (r \vee \neg r)$$

It has 8 answer sets, each of which is a subset of $\{p, q, r\}$.

In general, if Z consists of n atoms then Z^c has 2^n answer sets.

Under the stable model semantics, Z^c says: for every element of Z, choose arbitrarily whether to include it in the answer set.

Constraints

Constraint: A rule with the head $\bot$.

Theorem

X is a stable model of $\Pi \cup \{\leftarrow F\}$ iff X is a stable model of Π that does not satisfy F.

Example

$$p \vee q$$
$$\leftarrow p$$

has only one answer set $\{q\}$.

Embedding Propositional Logic in SM

By combining choice rules and constraints.

Proposition

For any propositional formula F and any set X of atoms occurring in F, X is a model of F iff X is a stable model of $Z^c \wedge (\leftarrow \neg F)$, where Z is the set of all atoms occurring in F.

Propositional formula	General program
$\neg p \vee q$	$\{p, q\}^c$ $\leftarrow \neg(\neg p \vee q)$
Model: $\emptyset$, $\{q\}$, $\{p, q\}$	Stable Models: $\emptyset$, $\{q\}$, $\{p, q\}$

The theorem on strong equivalence tells us that $Z^c \wedge (\leftarrow \neg F)$ can be replaced with $Z^c \wedge F$.

Cardinality Expressions

- $2\{p, q, r\}$ stands for

$$(p \wedge q) \vee (p \wedge r) \vee (q \wedge r).$$

  X satisfies $2\{p, q, r\}$ iff $|X \cap \{p, q, r\}| \geq 2$.

- $\{p, q, r\}2$ stands for $\neg 3\{p, q, r\}$.
- $2\{p, q, r\}2$ stands for $2\{p, q, r\} \wedge \{p, q, r\}2$.

ASP Programs with Variables

Variables in ASP are understood in terms of grounding. In other words, a rule with variables is understood as a shorthand for the set of its ground instantiations over the Herbrand Universe of the program.

$$
\begin{aligned}
&p(a)\\
&q(b)\\
&r(x) \leftarrow p(x) \wedge \neg q(x)
\end{aligned}
$$

is shorthand for the formula

$$
\begin{aligned}
&p(a)\\
&q(b)\\
&r(a) \leftarrow p(a) \wedge \neg q(a)\\
&r(b) \leftarrow p(b) \wedge \neg q(b).
\end{aligned}
$$

In the Input Language of CLINGO

```
index(1..3).

{q(I,J) : index(J)} :- index(I).
:- {q(I,J) : index(J)} 0, index(I).
:-  2 {q(I,J) : index(J)}, index(I).
```

Here, `I` is a "global" variable and `J` is a "local" variable.

When `I` $=$ 1, the second rule is grounded as

```
{q(1,1), q(1,2), q(1,3)}.
```

The program can be equivalently written as

```
index(1..3).
#domain index(I).

1 {q(I,J) : index(J)} 1.
```

which has 27 answer sets.

System F2LP[LP09]

The input languages of ASP solvers do not allow complex formulas.

F2LP is a front-end to ASP solvers that turns first-order formulas into logic program syntax.

- `f2lp [input-program] | clingo`

$p \leftarrow \neg\neg p$ can be encoded in the language of F2LP as

```
p <- not not p.
```

The F2LP rule

```
t(X) <- v(X) & not ?[Y]:e(X,Y)
```

describes the set t of terminal vertices (the symbol `?` represents the existential quantifier).

Various Extensions

- Strong negation [GLR91]
- Arbitrary aggregates
  [SNS02, FLP04, Fer05, PDB07, LM09, FL10], . . .
- Preferences [BNT08b]
- Integration with CSP [Bal09a, GOS09a]
- Integration with SMT [JLN11]
- Integration with Description Logics [EIL$^+$08, LP11]
- Stable Model Semantics of formuals with generalized quantifiers [LM12]
- Probabilistic answer sets [BGR09b]
- Intensional functions [Cab11, Lif12, BL12]
- . . .

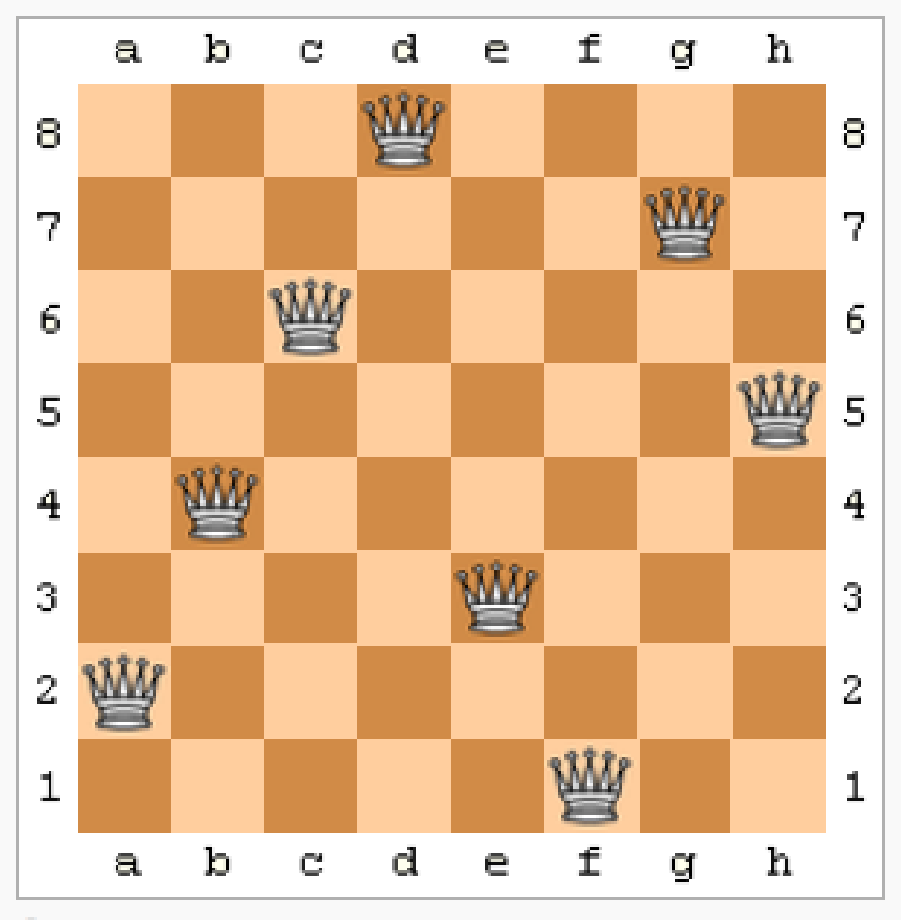
Contents

GENERATE-(DEFINE)-TEST

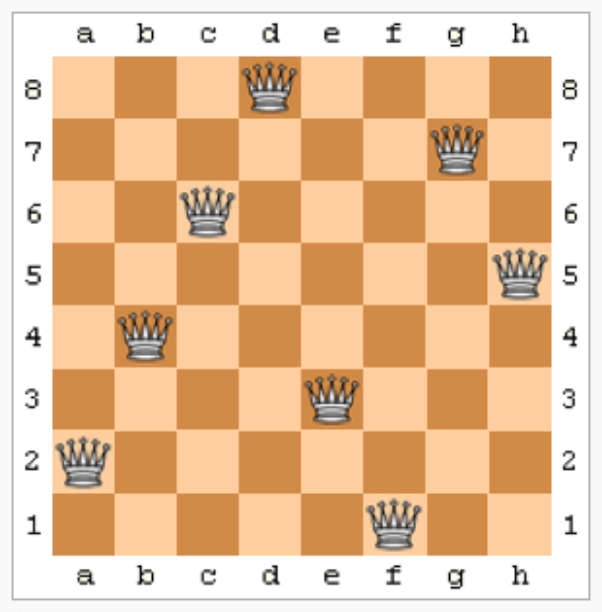
A way to organize rules.

- GENERATE part: generates a “search space” – a set of potential solutions.
- DEFINE part: defines new atoms in terms of other atoms.
- TEST part: weed out the elements of the search space that do not represent solutions.

N-Queens Puzzle



8-Queens Puzzle



"Each row has exactly one queen"

$$1 \leq \{q_{i,1}, \ldots, q_{i,8}\} \leq 1 \quad (1 \leq i \leq 8).$$

"Two queens cannot stay on the same column"

$$\bot \leftarrow q_{i,j} \wedge q_{i',j} \quad (1 \leq i < i' \leq 8; 1 \leq j \leq 8).$$

"Two queens cannot stay on the same diagonal"

$$\bot \leftarrow q_{i,j} \wedge q_{i',j'} \quad (1 \leq i < i' \leq 8; 1 \leq j, j' \leq 8; i' - i = |j' - j|).$$

N-Queens Puzzle in ASP

In the language of GRINGO:

```
num(1..n).

1 {q(I,J): num(J)} 1 :- num(I).
:- q(I,J), q(I1,J), I<I1.
:- q(I,J), q(I1,J1), I<I1, I1-I==#abs(J1-J).
```

Finding One Solution for the 8-Queens Puzzle

With command line

```
% gringo -c n=8 queens | clasp
```

we get the following output:

```
Solving...
Answer: 1
num(1) num(2) num(3) num(4) num(5) num(6) num(7) num(8)
q(8,4) q(7,2) q(6,8) q(5,5) q(4,7) q(3,1) q(2,3) q(1,6)
SATISFIABLE

Models       : 1+
Time         : 0.002s (Solving: 0.00s 1st Model: 0.00s Unsat: 0.00s)
CPU Time     : 0.000s
```

Finding All Solutions for the 8-Queens Puzzle

We can specify the number of solutions to return.

With the same program, but with the following command line

```
% gringo -c n=8 queens | clasp 0
```

CLASP computes and shows all 92 valid queen arrangements. For instance, the last part is

```
Answer: 92
num(1) num(2) num(3) num(4) num(5) num(6) num(7) num(8)
q(8,1) q(7,7) q(6,4) q(5,6) q(4,8) q(3,2) q(2,5) q(1,3)
SATISFIABLE

Models       : 92
Time         : 0.695s (Solving: 0.69s 1st Model: 0.00s Unsat: 0.05
CPU Time     : 0.000s
```

Sudoku

9				8		3		
			2	5		7		
	2		3					4
	9	4						
			7	3		5	6	
7		5		6		4		
		7	8		3	9		
		1						3
3								2

9	7	6	4	8	1	3	2	5
1	4	3	2	5	9	7	8	6
5	2	8	3	7	6	1	9	4
6	9	4	5	1	8	2	3	7
8	1	2	7	3	4	5	6	9
7	3	5	9	6	2	4	1	8
4	6	7	8	2	3	9	5	1
2	5	1	6	9	7	8	4	3
3	8	9	1	4	5	6	7	2

(Pictures from `http://www.cross-plus-a.com/sudoku.htm`)

Sudoku in ASP

Rules:

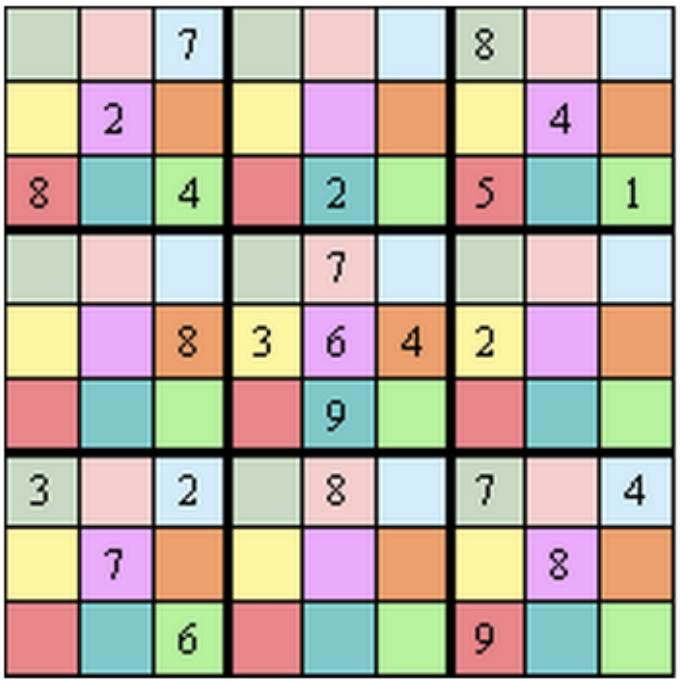
```
num(1..9).
border(1;4;7).

1 {a(R,C,N) : num(N)} 1 :- num(R;C).
1 {a(R,C,N) : num(R)} 1 :- num(C;N).
1 {a(R,C,N) : num(C)} 1 :- num(R;N).
1 {a(R,C,N) : num(R;C): X<=R: R<=X+2: Y<=C: C<=Y+2} 1
    :- num(N), border(X;Y).
```

Instance:

```
a(1,1,9).   a(1.5,8).   a(1,7,3).
  ....
```

Offset Sudoku

A region is represented by the same color. In addition to the requirement of Sudoku, every region must contain all the digits 1 through 9.



		7				8		
	2						4	
8		4		2		5		1
				7				
		8	3	6	4	2		
				9				
3		2		8		7		4
	7						8	
		6				9		

1	5	7	6	4	3	8	2	9
9	2	3	8	5	1	6	4	7
8	6	4	7	2	9	5	3	1
2	3	1	5	7	8	4	9	6
7	9	8	3	6	4	2	1	5
6	4	5	1	9	2	3	7	8
3	1	2	9	8	5	7	6	4
5	7	9	4	3	6	1	8	2
4	8	6	2	1	7	9	5	3

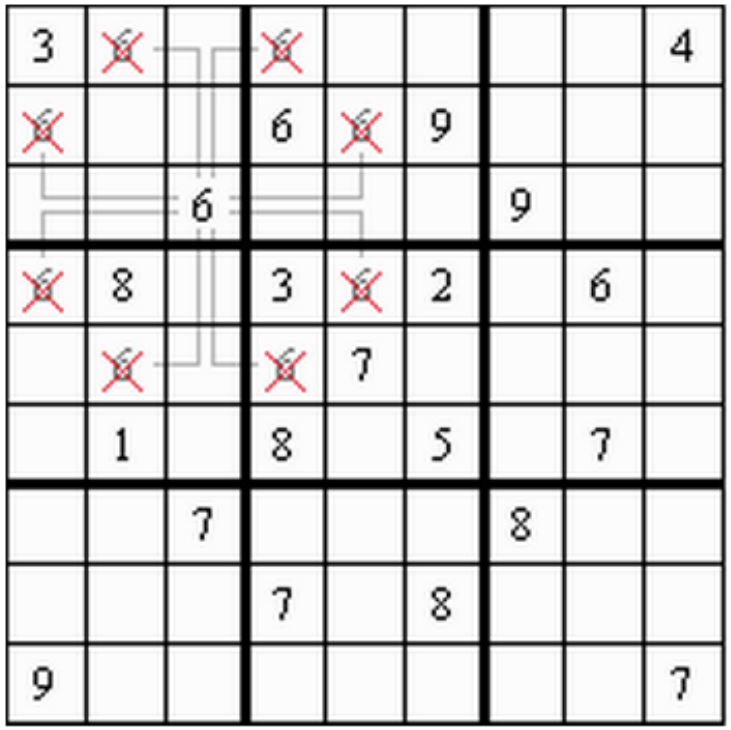
Offset Sudoku in ASP

Add to the basic program:

```
:- a(R,C,N), a(R1,C1,N),
   R #mod 3 == R1 #mod 3, C #mod 3 == C1 #mod 3,
   R != R1, C != C1.
```

Anti-Knight Sudoku

Cells that are a chess knight's move away from each other cannot hold equal values:



3	9	1	5	2	7	6	8	4
4	2	5	6	8	9	7	3	1
8	7	6	4	3	1	9	5	2
7	8	9	3	1	2	4	6	5
6	5	3	9	7	4	2	1	8
2	1	4	8	6	5	3	7	9
1	4	7	2	5	3	8	9	6
5	6	2	7	9	8	1	4	3
9	3	8	1	4	6	5	2	7

Anti-Knight Sudoku in ASP

Add to the basic program

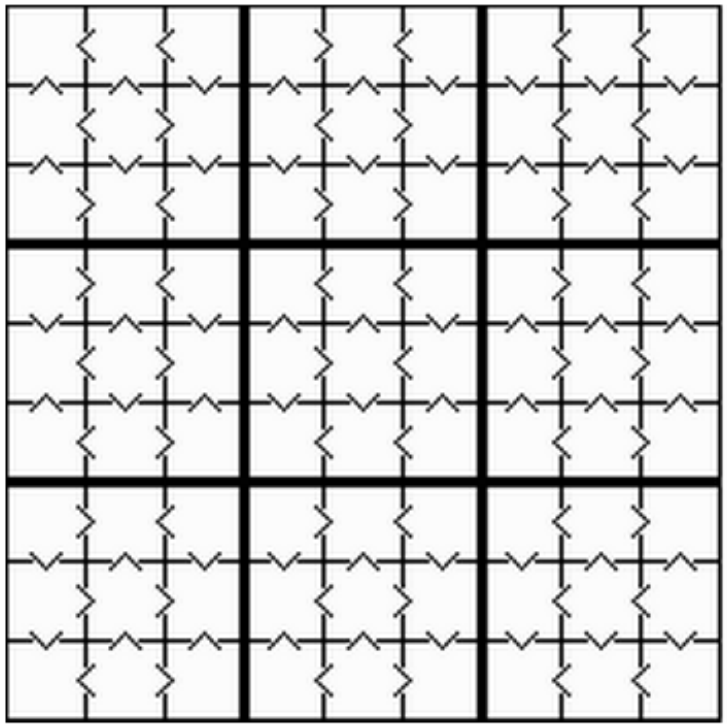
```
:- a(R,C,N), a(R-2,C-1,N).
:- a(R,C,N), a(R-2,C+1,N).
:- a(R,C,N), a(R-1,C-2,N).
:- a(R,C,N), a(R-1,C+2,N).
:- a(R,C,N), a(R+1,C-2,N).
:- a(R,C,N), a(R+1,C+2,N).
:- a(R,C,N), a(R+2,C-1,N).
:- a(R,C,N), a(R+2,C+1,N).
```

or simply,

```
:- a(R,C,N), a(R1,C1,N), #abs(RR-R) + #abs(CC-C) == 3
```

Greater-Than Sudoku

No numerical clues; Only greater-than relationships



2	3	9	5	1	4	6	7	8
4	7	6	8	9	3	1	2	5
8	1	5	7	6	2	9	3	4
9	6	7	3	4	8	5	1	2
1	8	2	9	5	6	7	4	3
3	5	4	1	2	7	8	9	6
7	2	8	6	3	9	4	5	1
6	4	1	2	7	5	3	8	9
5	9	3	4	8	1	2	6	7

Greater-Than Sudoku in ASP

Let `gt(R,C,R1,C1)` represent that the number in `(R,C)` is greater than the number in `(R1,C1)`.

Add to the basic program

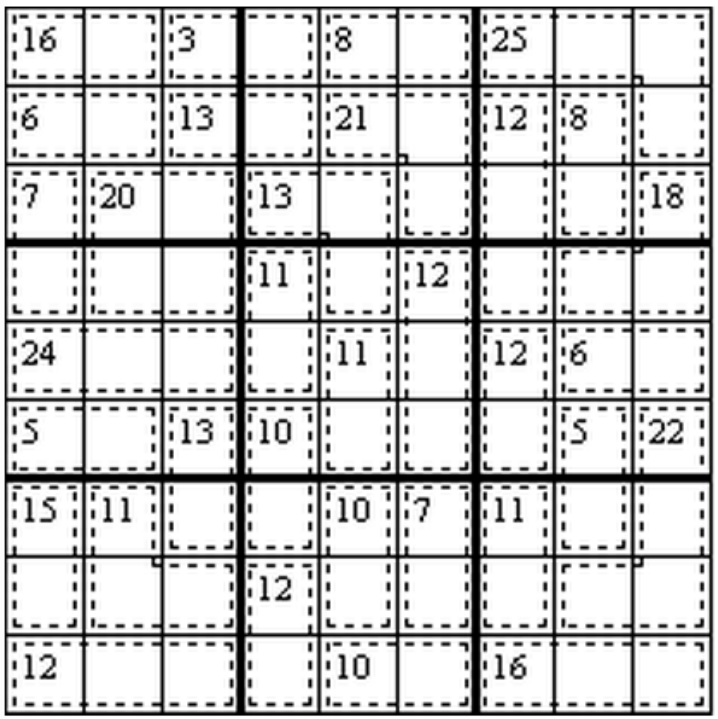
```
:- a(R,C,N), a(R1,C1,N1), gt(R,C,R1,C1), N <= N1.
```

together with input data:

```
gt(1,2,1,1).    gt(1,3,1,2).  gt(2,1,1,1).
 ...
```

Killer Sudoku

Each cage (“dotted area”) is associated with a number. The sum of the cells in a cage must be equal to the number given for the cage. Each digit in the cage must be unique.



7	9	2	1	3	5	4	8	6
1	5	4	9	8	6	2	3	7
3	6	8	2	4	7	1	5	9
4	1	5	8	7	2	9	6	3
8	7	9	3	6	1	5	2	4
2	3	6	4	5	9	7	1	8
9	2	7	6	1	3	8	4	5
6	8	1	5	9	4	3	7	2
5	4	3	7	2	8	6	9	1

Killer Sudoku in ASP

Add to the basic program

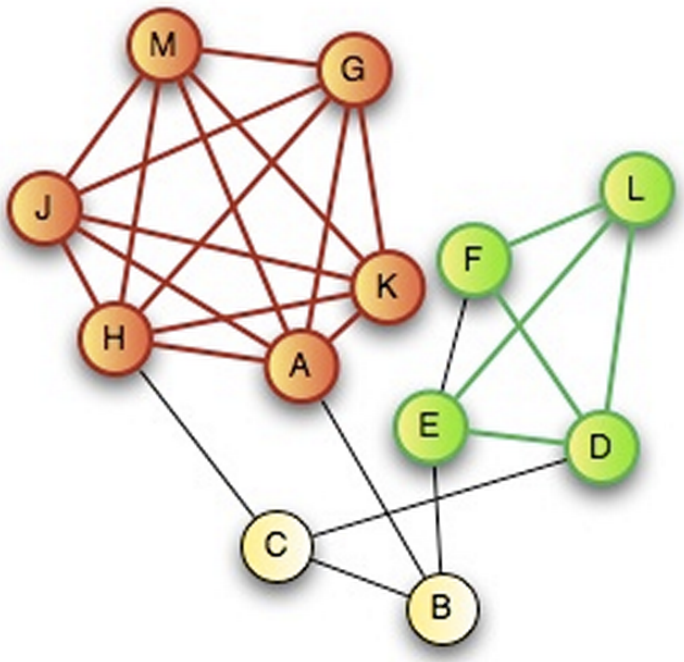
```
% Get values in each cage
cage_values(CA,N) :- a(R,C,N), cell_cage(R,C,CA).

% The values in the cage must add to the value of the cage
:- N1 = #sum[a(R,C,N)=N: cell_cage(R,C,CA): num(N)],
   cage_sum(CA,N2), N1 != N2.

% There can only be one of each value per cage
:- a(R,C,N), a(R1,C1,N),
   cell_cage(R,C,CA), cell_cage(R1,C1,CA),
   R!=R1, C!=C1.
```

together with input data:

```
cage_sum(1,16).    cage_sum(2,3).   cage(3,8).   ...
cell_cage(1,1,1). cell_cage(2,1,1).
cell_cage(3,1,2). cell_cage(4,1,2).  ...
```

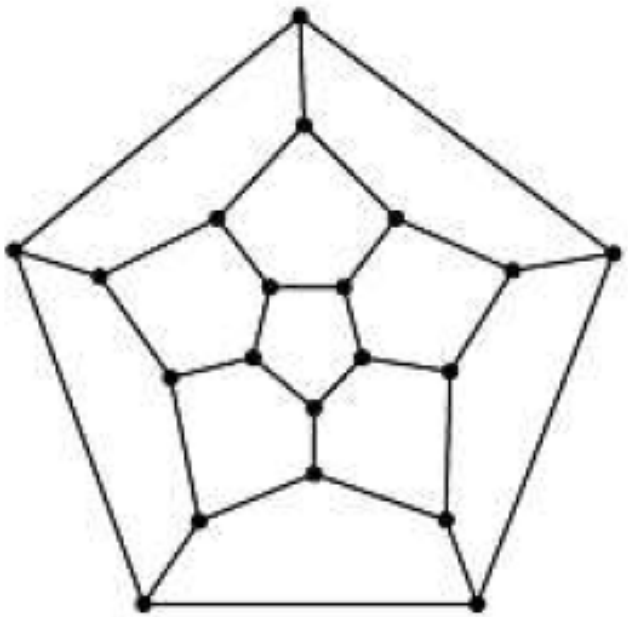
Clique

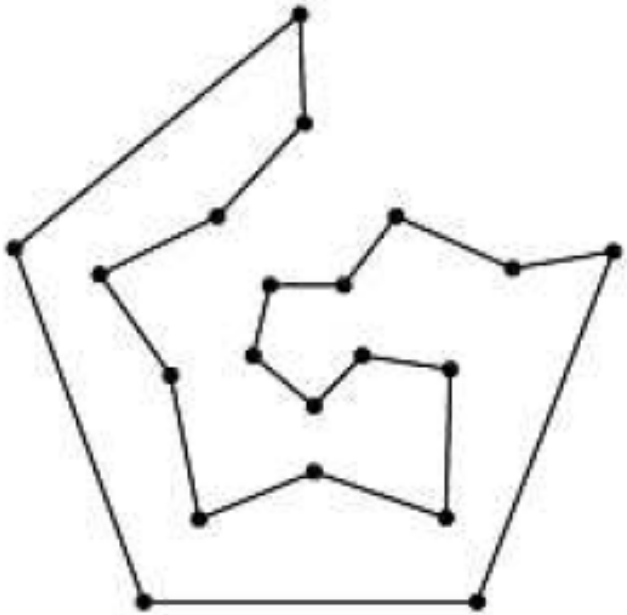
A clique in a graph *G* is a subset of its vertices in which every two elements are adjacent.




```
% File 'clique': Find a clique of size >= n

n {in(X) : v(X)}.
:- in(X), in(Y), v(X), v(Y), X!=Y, not e(X,Y), not e(Y,X).
```

Hamiltonian Cycles

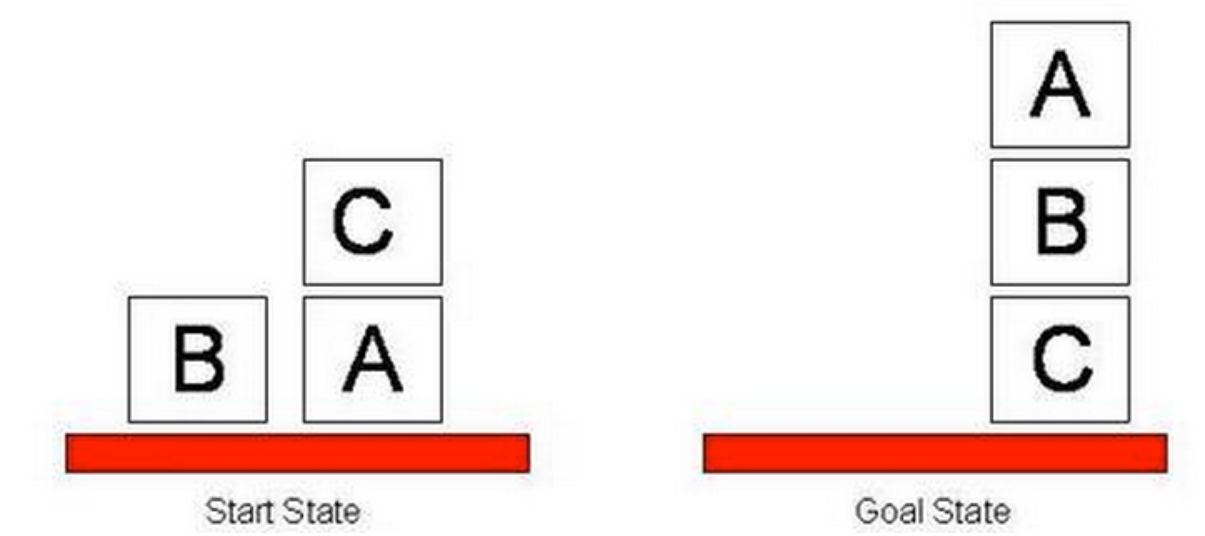




Hamiltonian Cycles in ASP

```
{in(U,V)} :- e(U,V).

:- in(U,V), in(U,W), V!=W.
:- in(U,W), in(V,W), U!=V.

reachable(U) :- in(v0,U).
reachable(V) :- reachable(U), in(U,V).

:- not reachable(U), v(U).
```

Answer Set Planning [Lif02b]



Encode a planning problem as a logic program whose answer sets correspond to solutions. Run ASP solvers to find the solutions.

Can be viewed as enhanced SAT planning [KS92].

- easier to represent properties of actions
- indirect effects
- defeasible rules

Blocks World in ASP: Declarations

```
step(0..maxstep).
astep(0..maxstep-1) :- maxstep > 0.

#domain step(ST).          #domain astep(T).
#domain block(B).          #domain block(B1).
#domain location(L).       #domain location(L1).

% every block is a location
location(B) :- block(B).

% the table is a location
location(table).
```

Blocks World in ASP: GENERATE, DEFINE

```
%% GENERATE
{on(B,L,0)}.

{move(B,L,T)}.

{on(B,L,T+1)} :- on(B,L,T).

%% DEFINE
% effect of moving a block
on(B,L,T+1) :- move(B,L,T).
```

Blocks World in ASP: TEST

```
%% TEST
% uniqueness constraint: no blocks are on two locations
:- 2{on(B,LL,ST): location(LL)}.

% existence constraint: every block has a location
:- {on(B,LL,ST): location(LL)}0.

% two blocks can't be on top of the same block
:- 2{on(BB,B,ST): block(BB)}.

% a block can't be moved unless it is clear
:- move(B,L,T), on(B1,B,T).

% a block can't be moved onto a block that is being moved also
:- move(B,B1,T), move(B1,L,T).
```

A Solution to the Frame Problem in ASP

The frame problem: how to formalize that by default fluents do not change their values.

```
{on(B,L,T+1)} :- on(B,L,T).

:- 2{on(B,LL,ST): location(LL)}.
:- {on(B,LL,ST): location(LL)}0.

on(B,L,T+1) :- move(B,L,T).
```

- If `on(B,L,T)` then decide arbitrarily whether to assert `on(B,L,T+1)`.
- In the absence of additional information asserting `on(B,L,T+1)` is the only option, in view of the existence constraint.
- If we are given conflicting information about the location of `B` at time `T+1`, then not asserting `on(B,L,T+1)` is the only option, in view of the uniqueness constraint.

Action Languages

Transition systems can be more succinctly described in a high level language on top of ASP.

Action languages are formal models of parts of natural language for representing and reasoning about transition systems.

$$\mathit{Move}(b, l) \textbf{ causes } \mathit{Loc}(b) = l$$

Action language $\mathcal{C}+$ [GLL$^+$04] is an expressive formalism that can represent actions with conditional and indirect effects, nondeterministic actions, and concurrently executed actions.

The semantics of $\mathcal{C}+$ can be defined by a modular translation into ASP.

Blocks World in C+

Causal laws:

constraint $\neg(On(b_1) = b \wedge On(b_2) = b)$ for $b_1 \neq b_2$

$Move(b, l)$ **causes** $On(b) = l$
nonexecutable $Move(b, l)$ **if** $On(b_1) = b$
nonexecutable $Move(b, b_1) \wedge Move(b_1, l)$

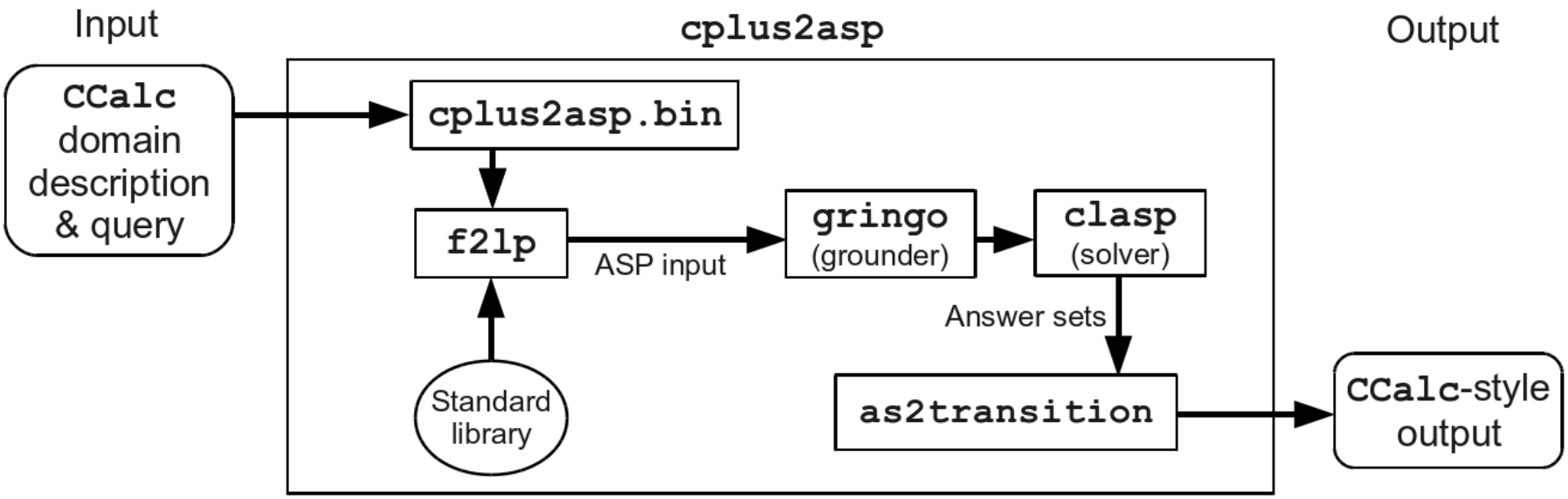
exogenous $Move(b, l)$

inertial $On(b)$

Translating $\mathcal{C}+$ to ASP [CL11]

$\mathcal{C}+$	ASP
$Move(b, l)$ **causes** $On(b) = l$	$\{On(b, l, t+1)\} \leftarrow Move(b, l, t)$
nonexecutable $Move(b, l)$ **if** $On(b_1) = b$	$\leftarrow Move(b, l, t) \wedge On(b_1, b, t)$
exogenous $Move(b, l)$	$\{Move(b, l, t)\}$
inertial $On(b)$	$\{On(b, l, t+1)\} \leftarrow On(b, l, t)$
constraint $\neg(On(b_1) = b \wedge On(b_2) = b)$	$\leftarrow (On(b_1, b, t) \wedge On(b_2, b, t))$

System CPLUS2ASP

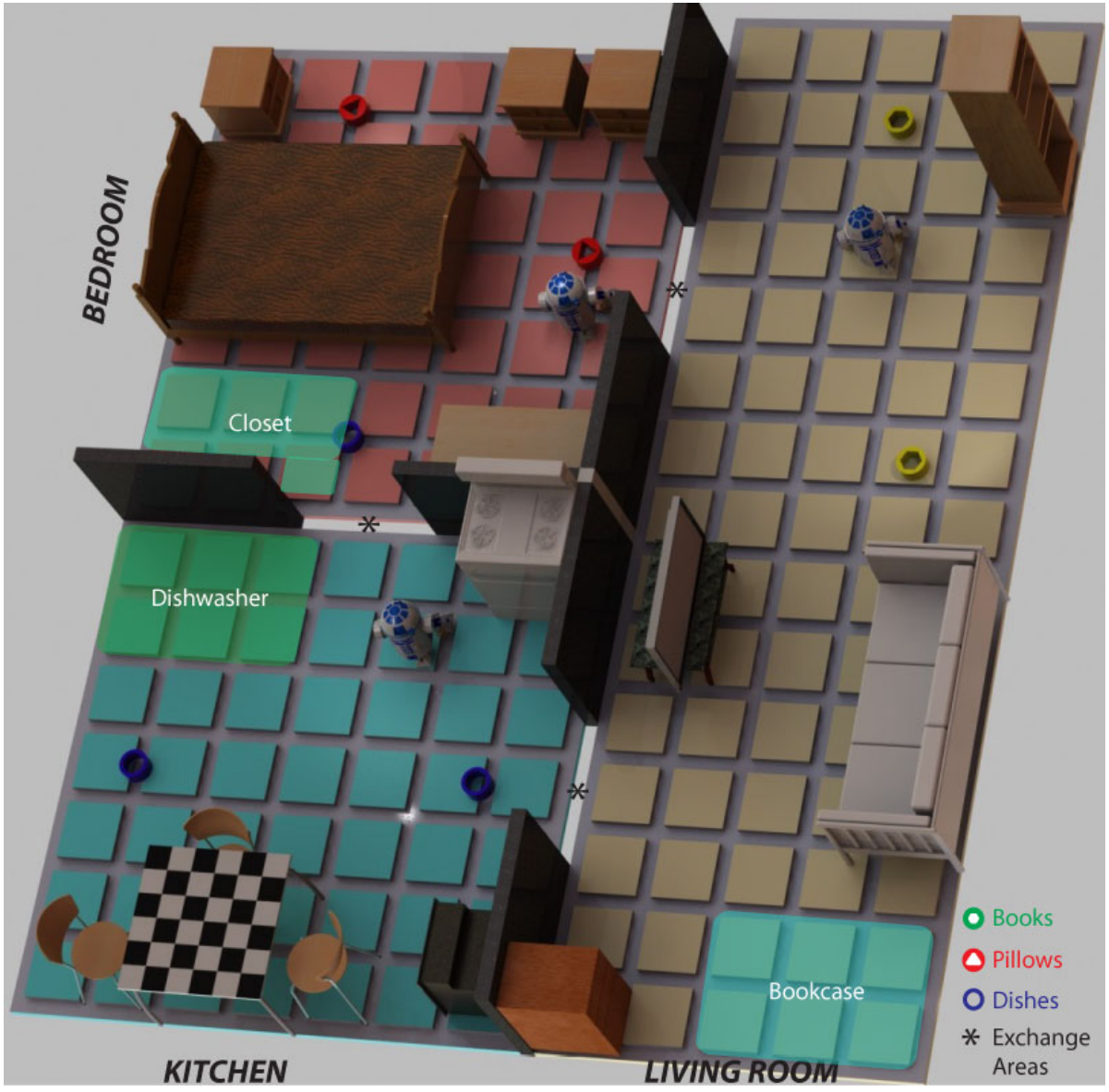


http://reasoning.eas.asu.edu/cplus2asp/

Contents

Housekeeping with Multiple Autonomous Robots: Representation, Reasoning and Execution

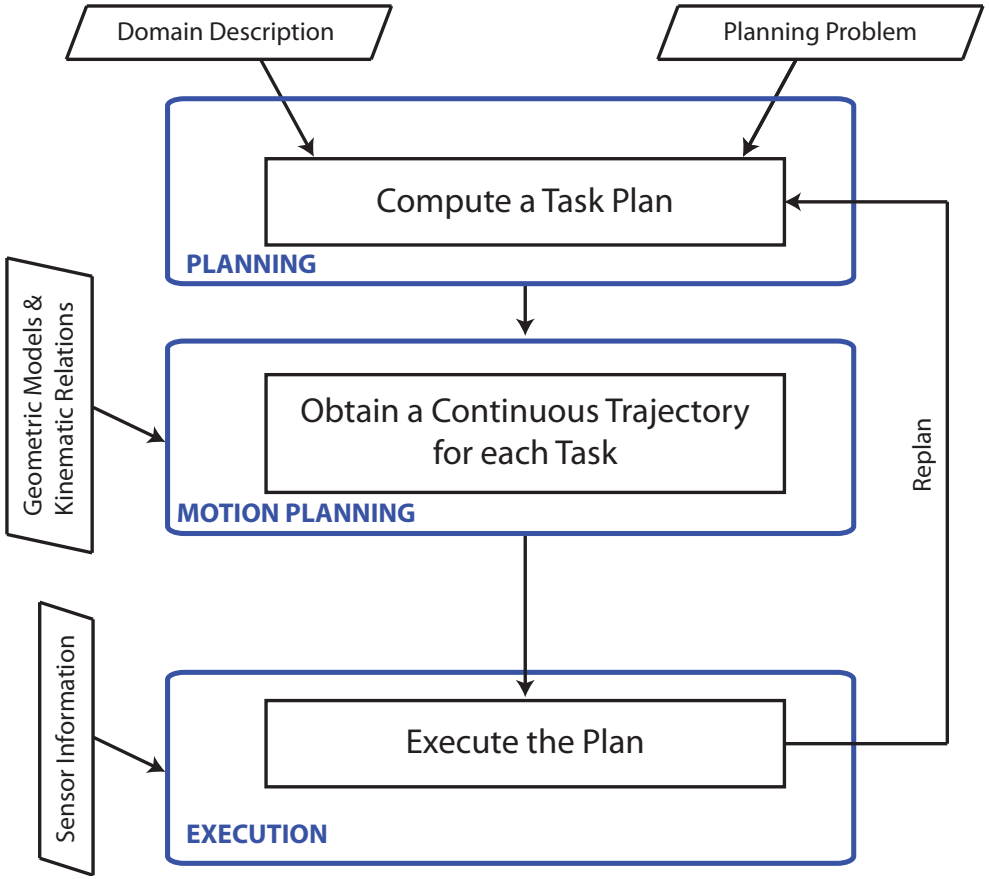
Housekeeping Domain



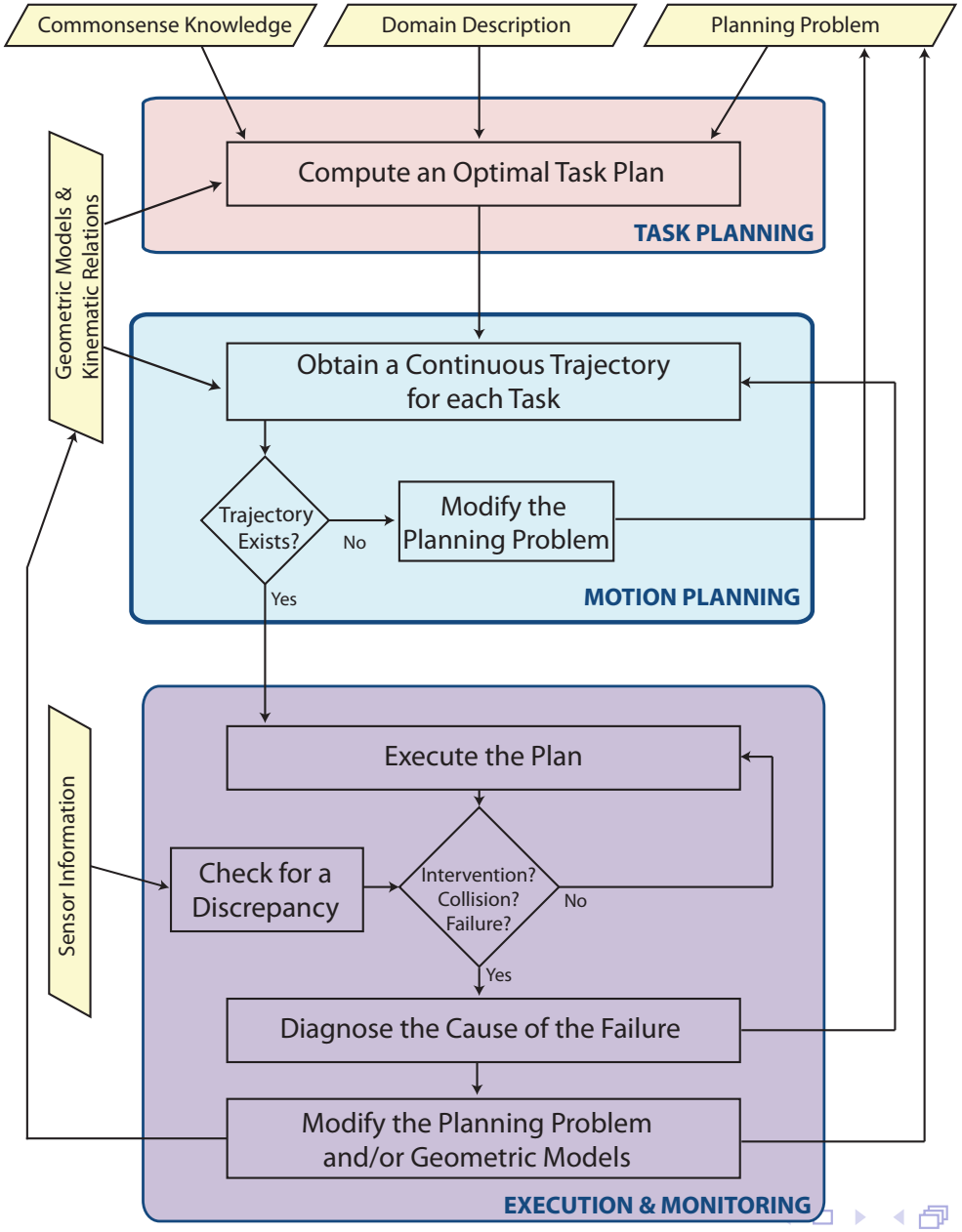
Challenges

- Commonsense knowledge (e.g., expected locations of objects in the house) is required for intelligent behavior of robots
- Geometric constraints are required to find feasible plans (e.g., to avoid collisions)
- In case of a plan execution failure (e.g., due to heavy objects that cannot be lifted by a single robot), recovery is required depending on the cause of failure
- Collaboration of robots is required to complete some tasks (e.g., carrying heavy objects)
- Temporal constraints (e.g., on the total duration of execution, or the order of actions) are required to complete all the tasks by a given time, and for intelligent replanning

Classical Approach [ACF+98, BBE+07]



Our Approach



Our Approach

We implement the proposed framework by utilizing

- the expressive action description language $\mathcal{C}+$ and the automated causal reasoner CCALC, and
- the expressive knowledge representation formalism and efficient solvers of ASP,

like in [CHO+09a, CHO+09b, HPU+10, EHP+11, AEEP11b, AEEP11a, AEEP11c, EP12, EHPU12, APE12].

http://cogrobo.sabanciuniv.edu/

Representing Housekeeping Domain in ASP

Fluents and Actions

- We view each room as a grid.
- Fluents:
  - $at(th, x, y, t)$
  - $connected(r, ep, t)$
- Actions:
  - $goto(r, x, y, t)$
  - $detach(r, t)$
  - $attach(r, t)$ with an attribute $attach\_point(r, ep, t)$

Representing Housekeeping Domain in ASP

Actions and Change

- Direct effects:

$$at(r, x, y, t+1) \leftarrow goto(r, x, y, t)$$

- Preconditions:

$$\leftarrow goto(r, x, y, t), at(r, x, y, t)$$

- Ramifications:

$$at(ep, x, y, t) \leftarrow connected(r, ep, t), at(r, x, y, t)$$

- Constraints:

$$\leftarrow at(ep, x, y, t), at(ep1, x, y, t) \qquad (ep \neq ep1)$$

- Inertia:

$$at(ep, x, y, t+1) \leftarrow at(ep, x, y, t), not \sim at(ep, x, y, t+1)$$
$$\sim at(ep, x, y, t+1) \leftarrow \sim at(ep, x, y, t), not\ at(ep, x, y, t+1)$$

- The robots need to know that books are expected to be in the bookcase, dirty dishes in the dishwasher, and pillows in the closet.
- Moreover, a bookcase is normally in the living-room, dishwasher in the kitchen, and the closet in the bedroom.
- In addition, the robots should have an understanding of a “tidy” house to be able to clean a house autonomously: tidying a house means that the objects are at their desired locations.
- Also, while cleaning a house, robots should pay more attention while carrying fragile objects; for that they should have an understanding of what a fragile object is.

Such commonsense knowledge is formally represented already in commonsense knowledge bases, such as `ConceptNet` [LS04].

Embedding Commonsense Knowledge in Planning

Expected Locations of Objects

The object *ep* is at its desired location if it is at some "appropriate" position (x, y) in the right room.

$$at\_desired\_location(ep, t) \leftarrow at(ep, x, y, t), in\_place(ep, x, y)$$

Normally the movable objects in an untidy house are not at their desired locations.

$$\sim at\_desired\_location(ep, t) \leftarrow not\ at\_desired\_location(ep, t)$$

Here *in_place* is not a fluent, but an external predicate (defined in Prolog) that acquires the related commonsense knowledge about expected locations of objects from `ConceptNet` via its Python API.

```
in_place(EP, X, Y) :- part_of(EP, Obj), type_of(Obj, Type),
expected_location(Type, Room), expected_area(Room, X, Y).
```

The house is tidy normally.

$$tidy(t) \leftarrow not\ {\sim}tidy(t)$$

When an object is not at its expected location, the house is untidy.

$${\sim}tidy(t) \leftarrow {\sim}at\_desired\_location(ep, t)$$

We embed continuous geometric reasoning (e.g., to avoid robot-robot, robot-stationary object and robot-moveable object collisions) in the high-level discrete representation of robots actions in ASP, utilizing external predicates.

The robot r cannot go from $(x1, y1)$ to (x, y) if there is no collision-free path between them:

$$\leftarrow goto(r, x, y, t), at(r, x1, y1, t), @path\_exists(x1, y1, x, y) == 0.$$

Here the external predicate $path\_exists(x, y, x1, y1)$ (implemented in C++ utilizing Rapidly exploring Random Trees [Lav98]) returns 1 (resp. 0) if there is a (resp. there is no) collision-free path between (x, y) and $(x1, y1)$.

The default value for the duration of an action is 0:

$$robot\_time(r, 0, t) \leftarrow not \sim robot\_time(r, 0, t)$$

The duration of attaching to an object can be defined as 3 units of time:

$$robot\_time(r, 3, t + 1) \leftarrow attach(r, t)$$

The duration of moving from an initial position $(x1, y1)$ to a final position $(x2, y2)$ can be estimated utilizing a motion planner via an external function $time\_estimate(x1, y1, x2, y2)$:

$$robot\_time(r, @time\_estimate(x1, y1, x2, y2), t + 1) \leftarrow goto(r, x1, y1, t), at(r, x2, y2, t)$$

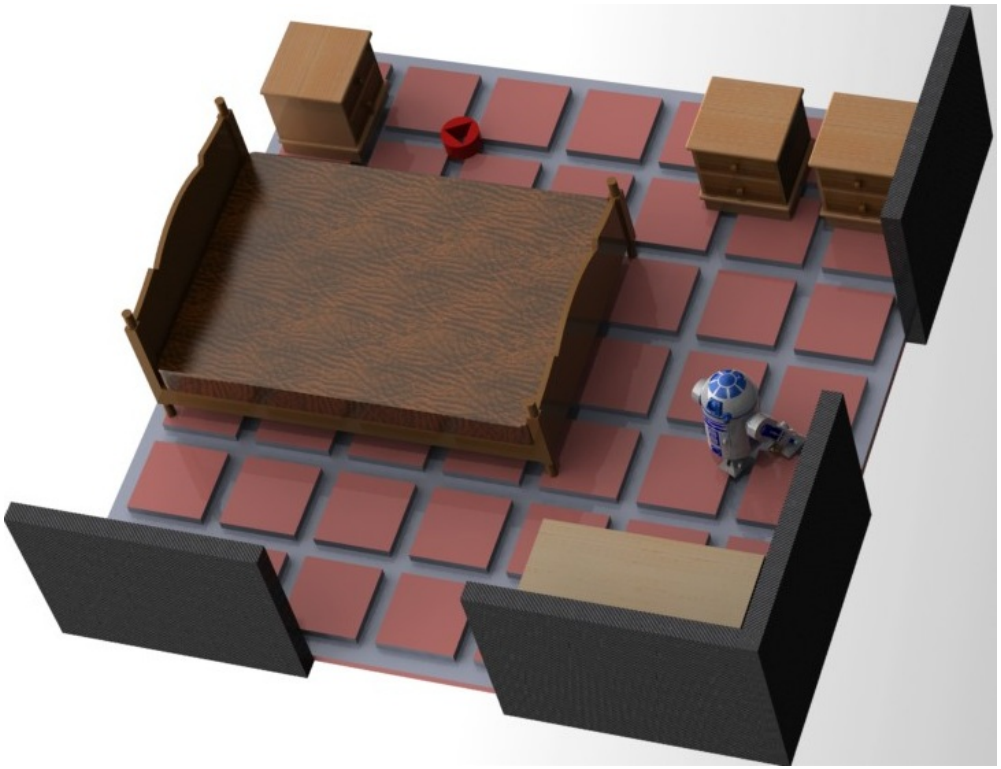
Hybrid Planning in ASP

Once the housekeeping domain is represented, and both the background/commonsense knowledge and geometric/temporal reasoning are embedded in the high-level representation, we can solve planning problems using ASP.

Since geometric reasoning and temporal reasoning (via a motion planner) are embedded in the computation, the calculated plans can be seen as hybrid plans integrating discrete ASP planning and continuous motion planning.

Hybrid plans help computation of plans that are geometrically feasible as well as temporally feasible.

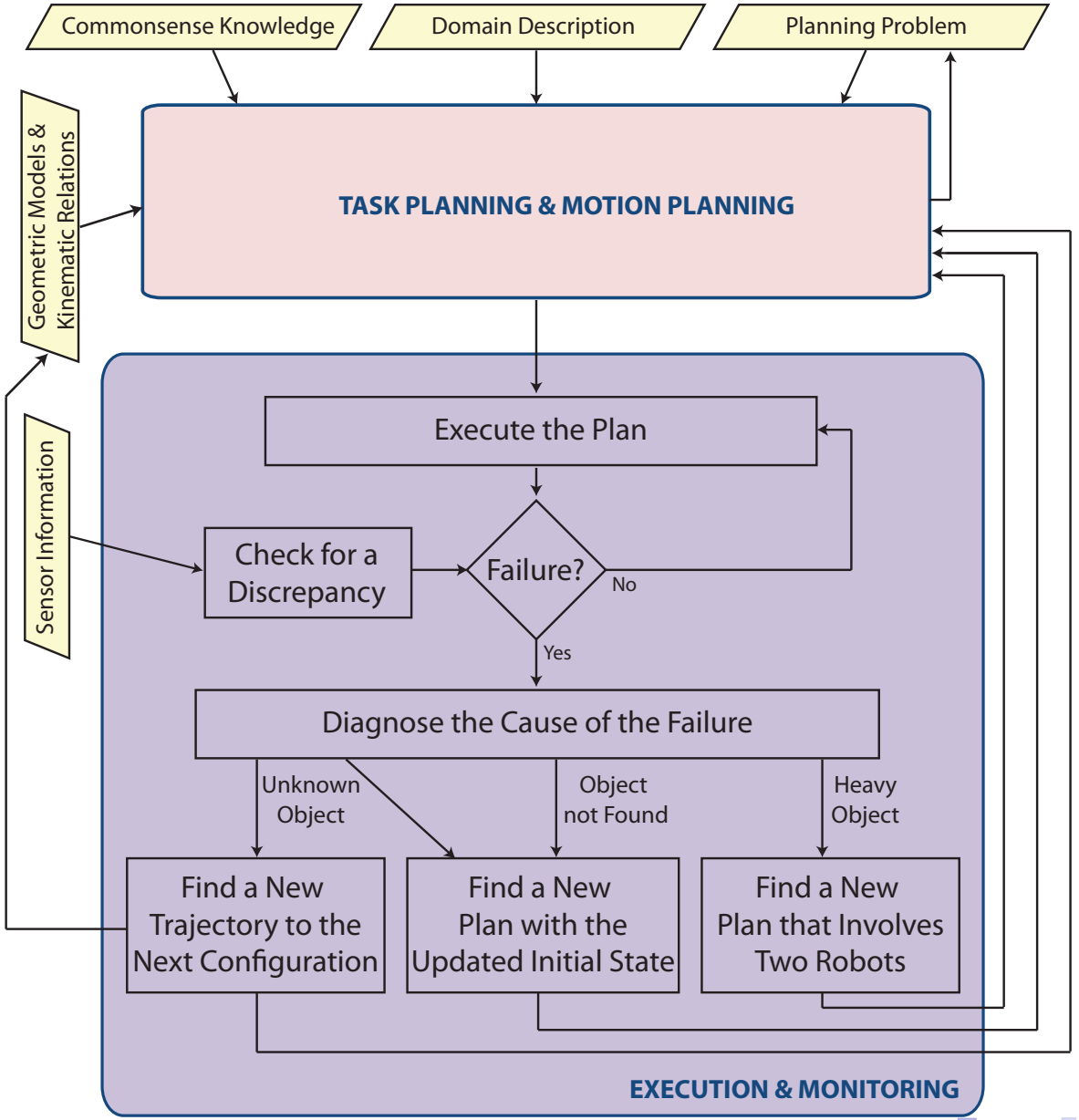
Example: Hybrid Planning in ASP



Without geometric feasibility checks, the ASP solver `iClingo` [GKK+08a] computes the following geometrically infeasible plan:

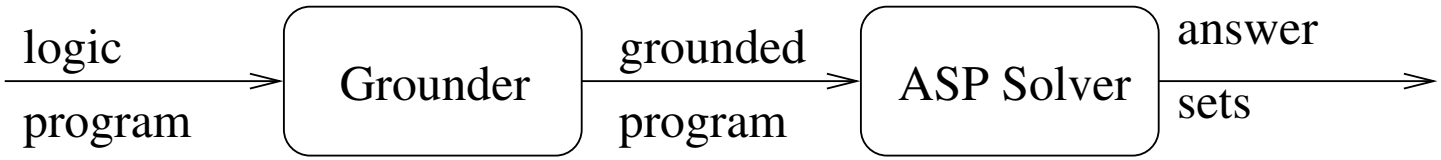
$$\langle goto(R1, 1, 2), attach(R1), goto(R1, 5, 1), detach(R1)\rangle$$

Execution Monitoring Algorithm



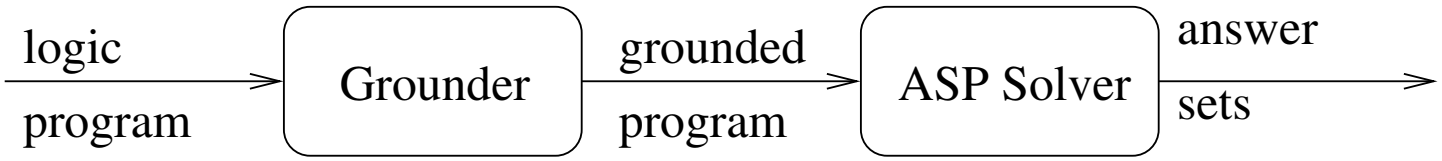
Contents

ASP Systems



- Grounding — task of instantiating variables
  - GRINGO (Potsdam), DLV (Calabria), LPARSE (Helsinki)
- Answer set solving – task of finding answer sets (decision problem is NP-complete):
  - SMODELS (Helsinki),
  - DLV (Vienna, Calabria),
  - CMODELS (Austin),
  - CLASP (Potsdam)
  - LP2X (Helsinki) . . .

Grounding



Grounding — task of instantiating variables

Π	*ground*(Π)	*intelligent instantiation*(Π)
$\{a(1), a(2), a(3)\}$	$\{a(1), a(2), a(3)\}$	$\{a(1), a(2), a(3)\}$
$\{b(1)\}$	$\{b(1)\}$	$\{b(1)\}$
$c(X) \leftarrow a(X), b(X)$	$c(1) \leftarrow a(1), b(1)$	$c(1) \leftarrow a(1), b(1)$
	$c(2) \leftarrow a(2), b(2)$	
	$c(3) \leftarrow a(3), b(3)$	

Issues in Grounding

- Ground programs maybe huge (even infinite)
- Intelligent grounding [CCIL08]
  - database techniques
- Function symbols lead to infinite programs
  - classes of programs whose intelligent instantiation is finite
- Intermixing solving and grounding
- Development of mixed languages that delegate solving over large domains to other computational methods [MGZ08], [GOS09b], [Bal09b]

Solving Backtrack Search, SAT

- ASP Solving: backtrack search through exponential size search space
- Propositional Satisfiability (SAT) — task of finding satisfying truth assignments for propositional formulas
- Classic backtrack search SAT algorithm: Davis-Putnam-Logemann-Loveland (DPLL)
- SAT solvers: MINISAT, SATZILLA, PLINGELING ...
  - performance boost $\Rightarrow$ success story in automated reasoning

SAT: Basics

$$a \vee b$$
$$c$$

- 8 interpretations: consistent and complete sets of literals over a, b, c
- 3 satisfying interpretations – models: An interpretation satisfies a clause if at least one of its literals is True in it

$\{a, b, c\}$ $\quad$ $\{\neg a, \neg b, \dots\}$
$\{\neg a, b, c\}$ $\quad$ $\{\neg c, \dots\}$
$\{a, \neg b, c\}$

SAT Solving: Basics

- Enumerate all interpretations and Test if satisfying
- DPLL: classic backtrack search approach
  - UnitPropagate

$$\begin{array}{l} a \\ \neg a \vee b \\ \neg b \vee c \vee d \\ \neg c \vee d \end{array} \Longrightarrow \begin{array}{l} b \\ \neg b \vee c \vee d \\ \neg c \vee d \end{array} \Longrightarrow \begin{array}{l} c \vee d \\ \neg c \vee d \end{array}$$

  - Decide
    - *pick arbitrary the value of c or d:* $\neg d$

$$\begin{array}{l} c \vee d \\ \neg c \vee d \end{array} \Longrightarrow \begin{array}{l} c \\ \neg c \end{array} \Longrightarrow \emptyset$$

  - Backtrack

Davis-Putnam-Logemann-Loveland

```
DPLL(F, ρ)
begin
    (F, ρ) ← UnitPropagate(F, ρ)
    if F contains the empty clause then return UNSAT
    if F has no clauses left then
        Output ρ
        return SAT
    l ← a literal such that its atom occurs in F
    if DPLL(F|_l, ρ ∪ {l}) = SAT then return SAT
    return DPLL(F|_l̄, ρ ∪ {l̄})
```

Modern SAT Solvers

MODERNSAT:

- Lookahead techniques
- Intelligent backtracking – backjumping
- Clause learning
- Clever restarts
- Watched literals

Answer Sets: Basics

$$a \leftarrow c,\ not\ b$$
$$c$$

- 8 interpretations
- 1 answer set $\{a, c\}$ identified with interpretation $\{a, c, \neg b\}$

Π	Π^{cl}
$a \leftarrow c,\ not\ b$	$a \vee \neg c \vee b$
c	c

X is an answer set of Π iff

1. X is a model of Π^{cl} and
2. X is *unfounded-free on* Π

$\{a, c, b\}$ is a model of Π^{cl} but *not* unfounded-free

ASP Solving: Basics

$$\Pi: \quad a \leftarrow c,\ not\ b \qquad c$$

- Backtrack search approach similar to DPLL
  - UnfoundedPropagate: $\neg b$
  - UnitPropagate on Π^{cl} using $\neg b$

$$a \vee \neg c \vee b \qquad c \qquad \Longrightarrow \qquad a \qquad \Longrightarrow \qquad \emptyset$$

    $\Longrightarrow$ Π is solved by propagation
  - Decide
  - Backtrack

DPLL-ASP

```
DPLL-ASP (Π, Π^cl, ρ)
begin
    while ρ changes do
        (Π^cl, ρ) ← UnitPropagate (Π^cl, ρ)
        ρ ← UnfoundedPropagate (Π, ρ)
    if inconsistency detected then return UNSAT
    if ρ is complete then
        Output ρ
        return SAT
    l ← a literal such that its atom occurs in Π
    if DPLL-ASP (Π, Π^cl|_l, ρ ∪ {l}) = SAT then return SAT
    return DPLL-ASP (Π, Π^cl|_l̄, ρ ∪ {l̄})
```

Answer set solvers SMODELS and DLV implement DPLL-ASP extended with additional ASP-specific propagates

Modern SAT Solving Methods for ASP

- Modern SAT solvers offer great improvements over DPLL
- From DPLL-ASP to Modern SAT technology for ASP
  - SAT-based answer set solvers
    - Translation-based: LP2SAT, LP2DIFFZ3 ...
    - *loop-formula* based
    - employ (incremental) SAT solvers iteratively
    - ASSAT, CMODELS
  - MODERNSAT-ASP: CLASP

Completion

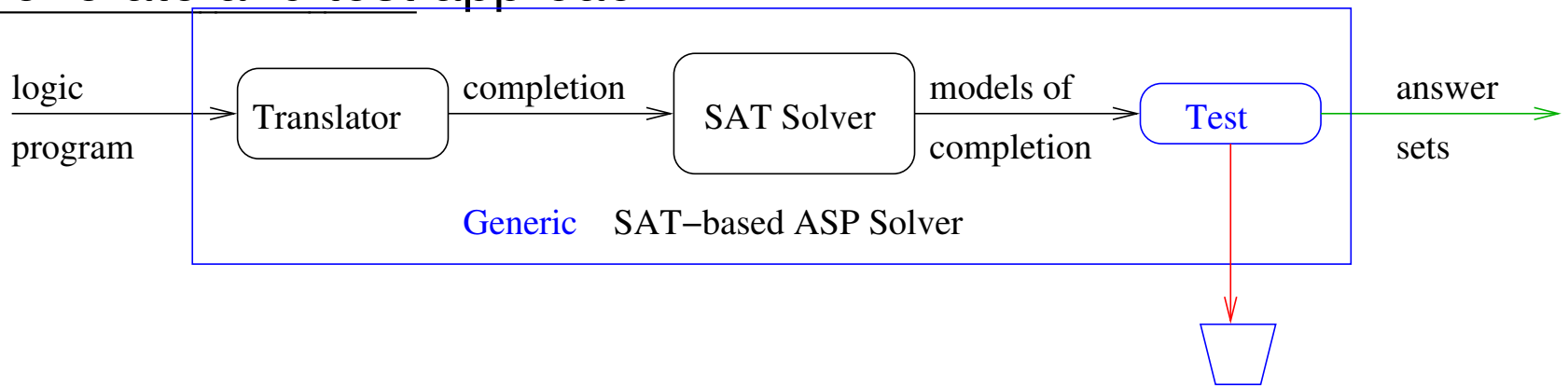
- Any logic program corresponds to propositional formula — completion [Cla78]
- Any answer set $\rightarrow$ a model of completion

Π	$Comp(\Pi)$
$a \leftarrow c,\ not\ b$	$a \equiv (c \wedge \neg b) \vee b$
$a \leftarrow b$	$b \equiv \bot$
c	$c \equiv \top$
$\{a, c\}$	$\{a, c, \neg b\}$

- Special class of tight programs:
  any model of completion $\rightarrow$ an answer set
- For tight programs, any SAT solver can be used as is on program's completion

Generic “Completion”-based SAT Approach

- Generate and test approach:



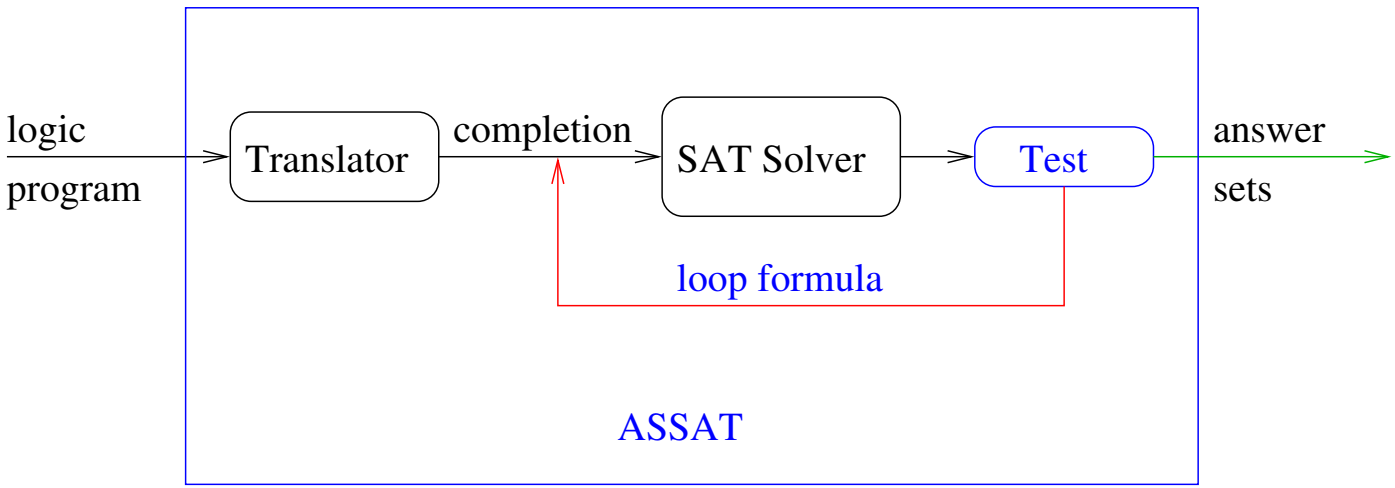
- Possibly exponential number of models and 0 answer sets.

Completion + Loop Formulas

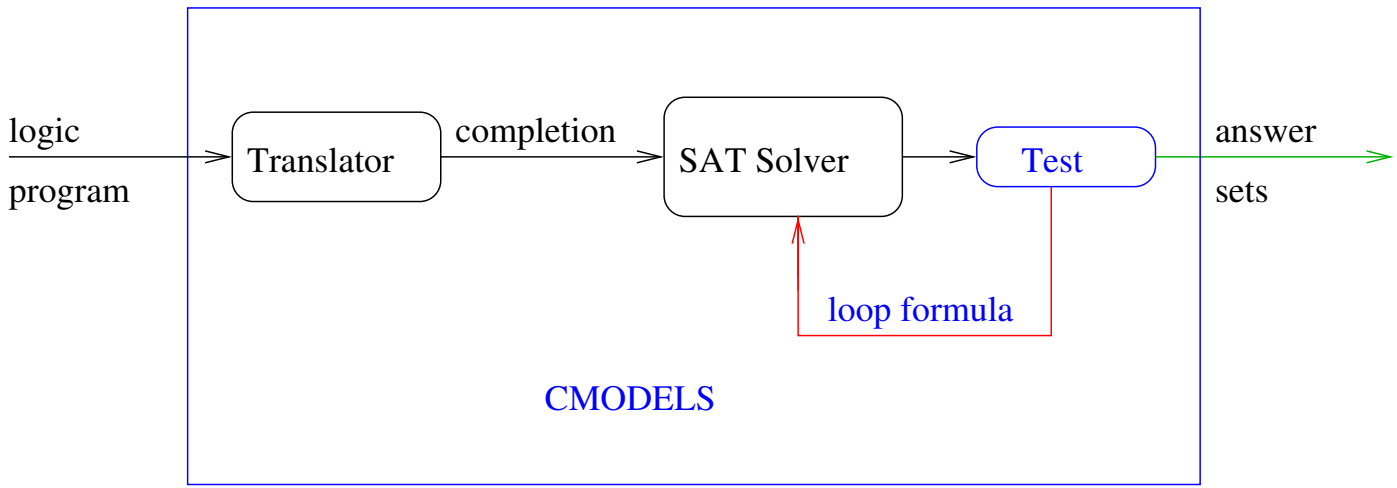
- For a nontight program, its answer sets coincide with models of completion *extended with* loop formulas [LZ02]
- Intuition: loop formulas allow to capture UnfoundedPropagate via UnitPropagate
- Possibly exponential number of loop formulas

ASSAT

- ASSAT [LZ02] implements lazy approach to utilizing loop formulas in using SAT for ASP
  - enumerates loop formulas on demand
  - invokes a SAT solver over and over



- Clause learning
  - add *conflict* clauses to original set of clauses
  - help a solver to disregard the irrelevant search tree branches
  - may exponentially improve performance
- CMODELS incorporates learning into generate and test approach
  - Test component [GLM04] is extended with *loop formulas-based* conflict clause computation
  - incremental SAT solving for adding these conflict clause



CLASP

- [GKNS07] computes completion of a program
  - UnitPropagate – native DPLL propagate on the completion
  - UnfoundedPropagate – ASP-based propagate on the program
- implements backjumping, clause learning, restarts, watched literals

ASP Competitions

Second ASP System Competition, 2009: 16 systems

Place	Decision Problem	Decision Problem in NP
1	CLASP-FOLIO	CLASP-FOLIO
2	CMODELS(MINISAT)	CMODELS(MINISAT)
3	DLV	IDP

Third ASP System Competition, 2011: 10 systems

Place	Decision Problems in NP
1	CLASP-FOLIO
2	CLASP
3	IDP

Questions to Raise

- As search procedures behind systems become more complex:
  - How to analyze their correctness?
  - How to compare and relate systems?

Abstract DPLL

Usually, pseudocode is used to describe algorithms including backtrack search DPLL-like algorithms.

Nieuwenhuis, Oliveras, and Tinelli [NOT06] described the DPLL and its enhancements using transitions systems — graphs — instead of pseudocode.

The DPLL Graph

DPLL_F graph for a set of clauses F:

- Its nodes – states of computation:
  - ordered sets of literals with some members annotated as decision literals, e.g.,

    $$a\ \neg b\ c^{\Delta}$$

    where value true is assigned to literals a, $\neg b$ and tentatively to c
  - *FailState*
- Its edges are described by *transition rules . . .*

Transition Rules of DPLL_F

Notation: $\bar{l}$ is the complement of l; $\quad \overline{C} = \{\bar{l} : l \in C\}$.

UnitPropagate: $\quad M \Longrightarrow M\ l \qquad$ if $\quad C \vee l \in F$ and $\overline{C} \subseteq M$

Decide: $\quad M \Longrightarrow M\ l^{\Delta} \qquad$ if $\quad l$ is unassigned by M

Fail: $\quad M \Longrightarrow \mathit{FailState} \qquad$ if $\begin{cases} M \text{ is inconsistent, and} \\ M \text{ contains no decision literals} \end{cases}$

Backtrack: $\quad P\ l^{\Delta}\ Q \Longrightarrow P\ \bar{l} \qquad$ if $\begin{cases} P\ l^{\Delta}\ Q \text{ is inconsistent, and} \\ Q \text{ contains no decision literals} \end{cases}$

Properties of DPLL_F

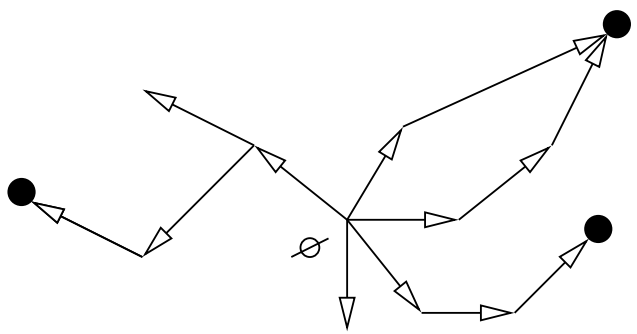
Proposition on DPLL_F For any F,

(a) graph DPLL_F is finite and acyclic,

(b) any terminal state of DPLL_F other than *FailState* is a model of F,

(c) *FailState* is reachable from $\emptyset$ in DPLL_F iff F is unsatisfiable.

- DPLL_F can be used for deciding whether F has a model by constructing a path from $\emptyset$ to a terminal node:



- Termination (a), Correctness (b), (c)
- DPLL_F — an abstract description of a class of DPLL-like algorithms.

DPLL$_F$: Example

$$\textit{UnitPropagate}: \quad M \Longrightarrow M\ l \quad \text{if } C \vee l \in F \text{ and } \overline{C} \subseteq M$$

$$\textit{Decide}: \quad M \Longrightarrow M\ l^{\Delta} \quad \text{if } l \text{ is unassigned by } M$$

F is a CNF formula $\quad a \vee b \quad \wedge \quad \neg a \vee c.$

A path in DPLL$_F$:

$$\begin{array}{lll} \emptyset & \Longrightarrow & (\textit{Decide}) \\ a^{\Delta} & \Longrightarrow & (\textit{UnitPropagate}) \\ a^{\Delta}\ c & \Longrightarrow & (\textit{Decide}) \\ a^{\Delta}\ c\ b^{\Delta} & & \end{array}$$

The state $a^{\Delta}\ c\ b^{\Delta}$ is terminal $\Rightarrow \{a, c, b\}$ is a model of F

This path corresponds to an execution of DPLL.

DPLL by means of $\mathtt{DPLL}_F$

$\mathtt{DPLL}_F$ captures DPLL by assigning priorities to its transition rules:

$$\mathit{UnitPropagate} >> \mathit{Backtrack}, \mathit{Fail} >> \mathit{Decide}$$

```
DPLL (F, ρ)
begin
    (F, ρ) ← UnitPropagate (F, ρ)
    if F contains the empty clause then return UNSAT
    if F has no clauses left then
        Output ρ
        return SAT
    l ← a literal such that its atom occurs in F
    if DPLL (F|_l, ρ ∪ {l}) = SAT then return SAT
    return DPLL (F|_l̄, ρ ∪ {l̄})
```

Abstract SMODELS

Classic ASP solver SMODELS [SNS02]

SMODELS_Π [Lie08]:

Decide, *Backtrack*, *Fail*
$\mathit{UnitPropagate}_\Pi$, $\mathit{BackchainTrue}_\Pi$,
$\mathit{BackchainFalse}_\Pi$, $\mathit{AllRulesCancelled}_\Pi$
$\mathit{Unfounded}_\Pi$

- Proposition on SMODELS_Π for correctness and termination
- Priorities of SMODELS:

$$\mathit{UnitPropagate}_\Pi, \mathit{BackchainTrue}_\Pi >>$$
$$\mathit{BackchainFalse}_\Pi, \mathit{AllRulesCancelled}_\Pi >>$$
$$\mathit{Unfounded}_\Pi >>$$
$$\mathit{Backtrack}, \mathit{Fail} >> \mathit{Decide}$$

Relation: DPLL and SMODELS

Recall:

- Logic program — propositional formula completion
- For tight programs answer sets correspond to the models of completion

Comp(Π) — *"Straightforward Clausified"* Completion

For tight programs:

$$\mathrm{DPLL}_{Comp(\Pi)} = \mathrm{SMODELS}_{\Pi}$$

DPLL_F	$\mathrm{SMODELS}_\Pi$
$UnitPropagate_F$	$UnitPropagate_\Pi$, $BackchainTrue_\Pi$ $Unfounded_\Pi$, $BackchainFalse_\Pi$, $AllRulesCancelled_\Pi$
Decide, *Backtrack*, *Fail*	

Abstract ASP-SAT with Learning: CMODELS, CLASP

$CC(\Pi)$ — *Clausified* Completion

MODERNSAT-ASP$_\Pi$ [Lie11]:

UnitPropagate$_{CC(\Pi)}$, *Unfounded*$_\Pi$, *Decide*, *Fail*,
Backjump, *Learn*$_{\Pi,CC(\Pi)}$, *Forget*

CMODELS and CLASP differ by priorities

- CMODELS:
  UnitPropagate$_{CC(\Pi)}$ $>>$ *Backjump*, *Fail* $>>$ *Decide* $>>$ *Unfounded*$_\Pi$
- CLASP:
  UnitPropagate$_{CC(\Pi)}$ $>>$ *Backjump*, *Fail* $>>$ *Unfounded*$_\Pi$ $>>$ *Decide*
- IDP: implements the same rules and priorities as CLASP

Summary on Abstract Transition Systems Framework

Transition systems

- provide birds eye view on the DPLL-like algorithms and methodology for proving their correctness
- promote development of new solvers
- clear picture on the relation between the systems
  - major ASP solvers: SMODELS, SMODELS$_{cc}$, SUP, SAG, CMODELS, CLASP, IDP

Disjunctive Answer Set Programming

- Allows programs with a disjunction of atoms in the head
- Deciding whether a disjunctive logic program has an answer set is Σ_2^P-complete
- DLV, CMODELS, CLASPD

Incremental Answer Set Programming

- CMODELS allows adding constraints on the fly via API
  - similar to how clauses may be added in incremental SAT
- ICLINGO implements elaborate approach to incremental ASP [GKK+08b]
  - extends the ASP language to describe 3 parts of the program
    - base, cumulative, volatile
  - parameter k
  - base – independent of k,
  - cumulative and volatile – k specific
  - well-suited for domains such as planning or model checking

Constraint Answer Set Programming

- Similar Direction to Satisfiability Modulo Theories (SMT)
- Development of mixed declarative languages that delegate parts of solving to other *specialized* computational methods
  - takes advantage of different automated reasoning tools *under one roof*
- Integration of ASP and Constraint Logic Programming/Constraint Satisfaction Processing [MGZ08], [GOS09b], [Bal09b]
- CLINGCON, EZCSP

"On the Relation of Constraint Answer Set Programming Languages and Algorithms" (Tuesday 2:25-2:45 PM)

Answer Set Programming and Multi-Context Systems

- HEX-programs [EBDT$^+$09b] extend logic programs under answer set semantics towards integration of external computation sources via *external atoms*
- Combining distributed knowledge based systems under one semantics (multi-context systems)
- System DLVHEX allows defining plug-ins for inference on external atoms

Integrated Development Environments for ASP

ASP processing tools are under continuous development for large scale practical applications.

- ASPIDE [FRR11] (https://www.mat.unical.it/ricca/aspide/), SEALION [OPT11] (http://sourceforge.net/projects/mmdasp/): Integrated development environments that support debugging, testing and annotating ASP programs.
- JASP [FLGR12]: A hybrid language that supports interaction between ASP and JAVA.

Resources

ASP Solvers

- LPARSE, SMODELS: http://www.tcs.hut.fi/Software/smodels/
- CMODELS: http://www.cs.utexas.edu/users/tag/cmodels
- GRINGO, CLASP, CLASP+FOLIO, ICLINGO, CLINGCON: http://potassco.sourceforge.net/
- DLV: http://www.dbai.tuwien.ac.at/proj/dlv/
- IDP: http://dtai.cs.kuleuven.be/krr/software/idp
- EZCSP: http://marcy.cjb.net/ezcsp/index.html
- DLVHEX: http://www.kr.tuwien.ac.at/research/systems/dlvhex/
- F2LP: http://reasoning.eas.asu.edu/f2lp/

Contents

- Loop Formulas
- First-Order stable model semantics
- Relation to Other KR Formalisms.

Embedding ASP in Propositional Logic

Embedding propositional logic into the stable model semantics is straightforward.

The other direction is a bit more involved.

- Completion: for tight programs only.
- Loop formulas: general case

Stable Models vs. Models

$p \leftarrow s, not\ q$ $q \leftarrow s, not\ r$ $s \leftarrow not\ p$	$s \wedge \neg q \rightarrow p$ $s \wedge \neg r \rightarrow q$ $\neg p \rightarrow s$
stable model: $\{q, s\}$	models: $\{p\}$, $\{p, q\}$, $\{p, r\}$, $\{q, s\}$, $\{p, q, r\}$, $\{p, q, s\}$, $\{p, r, s\}$, $\{q, r, s\}$, $\{p, q, r, s\}$

$p \leftarrow q$ $q \leftarrow p$ $p \leftarrow not\ r$ $r \leftarrow not\ p$	$q \rightarrow p$ $p \rightarrow q$ $\neg r \rightarrow p$ $\neg p \rightarrow r$
stable models: $\{p, q\}, \{r\}$	models: $\{p, q\}$, $\{r\}$, $\{p, q, r\}$

Theorem on Loop Formulas [LZ04]

Some answer set solvers (e.g., ASSAT, CMODELS, SAG) use SAT solvers as search engines, based on the theorem on loop formulas.

The theorem shows how to turn answer set programs into propositional logic by means of loop formulas.

Allows to combine an expressive representation language (ASP language) with efficient reasoning engines (SAT solvers).

External Support Formula

Consider a program whose rules have the form:

$$a \leftarrow \underbrace{a_1, \ldots, a_m}_{P}, \underbrace{not\ a_{m+1}, \ldots, not\ a_n}_{N}.$$

Given a program Π, for any set Y of atoms, the external support formula for Y, ES_Y, is the disjunction of

$$P \wedge N$$

for all rules $a \leftarrow P, N$ of Π such that $a \in Y$ and $P \cap Y = \emptyset$.

Π_1:

$p \leftarrow q$
$q \leftarrow p$
$p \leftarrow not\ r$
$r \leftarrow not\ p$

$ES_{\{p\}} = q \vee \neg r$
$ES_{\{q\}} = p$
$ES_{\{p,q\}} = \neg r$

Theorem on Loop Formulas

A model X of Π is stable iff its every nonempty subset of X is "externally supported."

Loop formula of Y :

$$LF(Y) = \left(\bigwedge_{a \in Y} Y \right) \rightarrow ES(Y)$$

Theorem : A model X of Π is stable iff it satisfies $LF(Y)$ for all nonempty sets Y of atoms occurring in Π.

Example

Theorem : A model X of Π is stable iff it satisfies $LF(Y)$ for all nonempty sets Y of atoms occurring in Π.

Π	$\Pi \cup \{LF(Y) : Y \text{ is a set of atoms}\}$
$p \leftarrow q$	$q \rightarrow p$
$q \leftarrow p$	$p \rightarrow q$
$p \leftarrow not\ r$	$\neg r \rightarrow p$
$r \leftarrow not\ p$	$\neg p \rightarrow r$
	$p \rightarrow (q \vee \neg r)$
	$q \rightarrow p$
	$r \rightarrow \neg p$
	$(p \wedge q) \rightarrow \neg r$
	$(p \wedge r) \rightarrow (q \vee \neg r \vee \neg p)$
	$(q \wedge r) \rightarrow (p \vee \neg p)$
	$(p \wedge q \wedge r) \rightarrow (\neg r \vee \neg p)$

	stable	not stable
models	$\{p, q\}, \{r\}$	$\{p, q, r\}$

We can reduce the number of loop formulas by considering loops.

Positive Dependency Graph

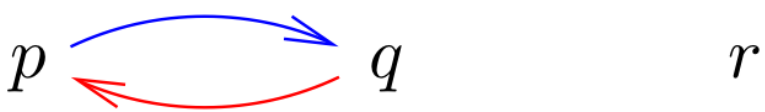
Consider a normal logic program

$$a \leftarrow \underbrace{a_1, \ldots, a_m}_{P}, \underbrace{not\ a_{m+1}, \ldots, not\ a_n}_{N}.$$

The positive dependency graph of Π is the directed graph such that

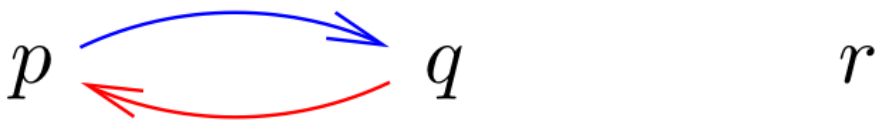
- its vertices are the atoms occurring in Π, and
- for each $a \leftarrow P, N$ in Π, its edges go from a to each atom in P.

Π_1 :
$p \leftarrow q$
$q \leftarrow p$
$p \leftarrow not\ r$
$r \leftarrow not\ p$



Definition of a Loop

A nonempty set L of atoms is called a loop of Π if, for every pair a, b of atoms in L, there exists a path from a to b in the positive dependency graph of Π such that all vertices in this path belong to L.



Π_1 has four loops. $\{p\}, \{q\}, \{r\}, \{p, q\}$.

Theorem on Loop Formulas

Theorem

Theorem on Loop Formulas A model X of Π is stable iff it satisfies $LF(Y)$ for all loops Y of Π.

Π	$\Pi \cup \{LF(Y) : Y \text{ is a loop of } \Pi\}$	
$p \leftarrow q$	$q \rightarrow p$	$p \rightarrow (q \vee \neg r)$
$q \leftarrow p$	$p \rightarrow q$	$q \rightarrow p$
$p \leftarrow not\ r$	$\neg r \rightarrow p$	$r \rightarrow \neg p$
$r \leftarrow not\ p$	$\neg p \rightarrow r$	$(p \wedge q) \rightarrow \neg r$
		~~$(p \wedge r) \rightarrow (q \vee \neg r \vee \neg p)$~~
		~~$(q \wedge r) \rightarrow (p \vee \neg p)$~~
		~~$(p \wedge q \wedge r) \rightarrow (\neg r \vee \neg p)$~~

	stable	not stable
models	$\{p, q\}, \{r\}$	$\{p, q, r\}$

We still get exponentially many loop formulas in the worst case.

Why Are There So Many Loop Formulas?

Theorem

Any equivalent translation from logic programs to propositional formulas involves a significant increase in size assuming a plausible conjecture ($P \not\subseteq NC^1/poly$) [LR06] ("Why are there so many loop formulas?")

> *How succinctly can the formalism express the set of models that it can? . . . [W]e consider formalism A to be stronger than formalism B if and only if any knowledge base in B has an equivalent knowledge base in A that is only polynomially longer, while there is a knowledge base in A that can be translated to B only with an exponential blowup. [GKPS95]*

More Work on Loop Formulas

- Loop formulas for disjunctive logic programs [LL03]
- Loop formulas for circumscription [LL04, LL06]
- Loop formulas for nonmonotonic causal logic [Lee04]
- Completion is a special case of loop formulas [Lee05]
- Generalization to arbitrary propositional formulas under the stable model semantics [FLL06]
- Refinement of loops [GS05, GLL06, GLL11]
- Led to First-Order Stable Model Semantics [FLL07, FLL11]

First-Order Stable Model Semantics [FLL07, FLL11]

Generalizes Gelfond and Lifschitz's 1988 definition of a stable model to first order sentences.

- Does not refer to grounding; not restricted to Herbrand models.
- Does not refer to reduct.
- Defined by a translation into second-order classical logic.

Idea 1: Treat logic programs as alternative notation for first-order formulas.

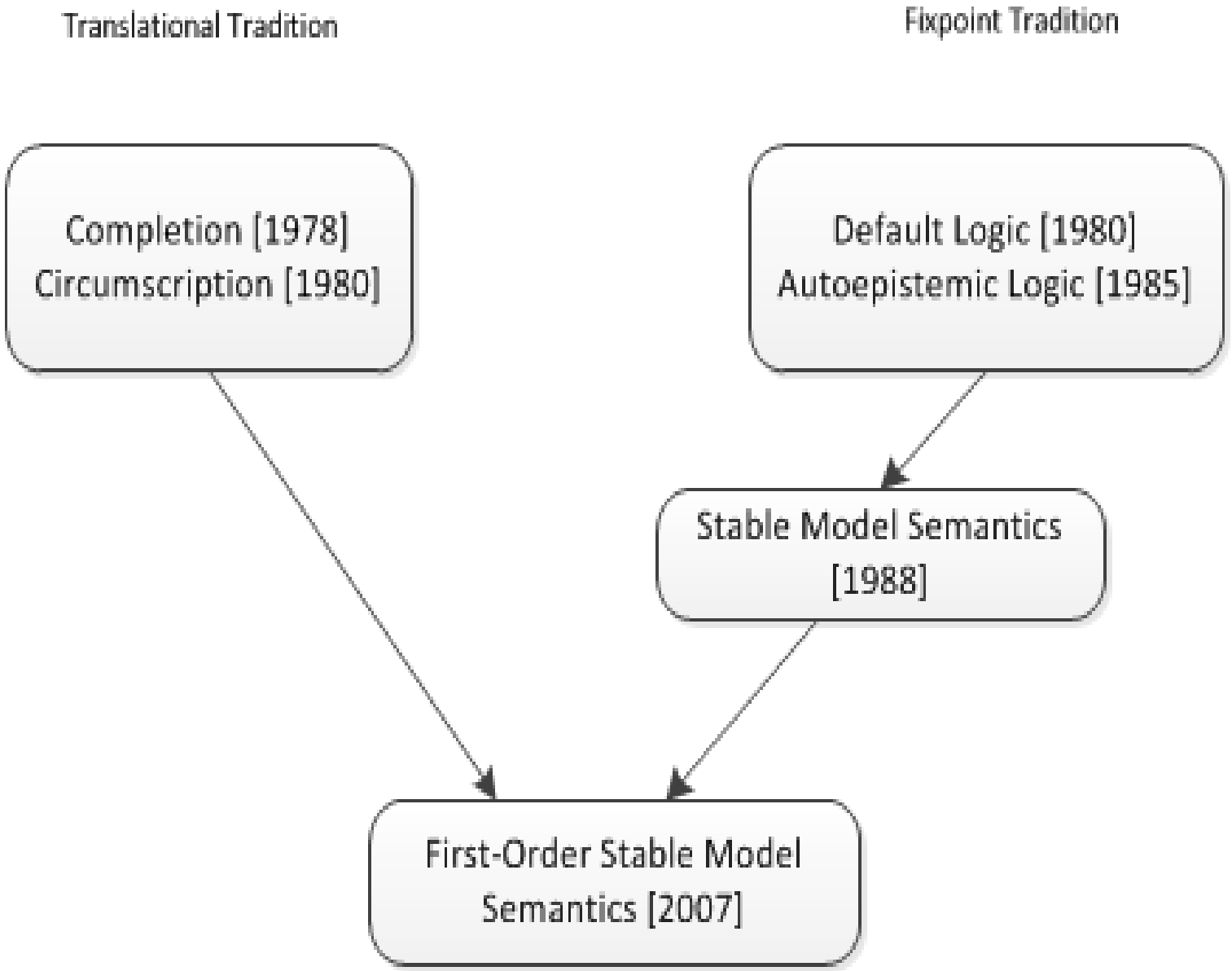
Logic program	FOL-representation
$p(a)$	$p(a)$
$q(x) \leftarrow p(x), \mathit{not}\ r(x)$	$\wedge\ \forall x(p(x) \wedge \neg r(x) \rightarrow q(x))$

Idea 2: Define the stable models of F as the models of

$$\mathrm{SM}[F; \mathbf{p}] = F \wedge \text{(2nd-order formula that enforces } \mathbf{p} \text{ to be } \textit{stable}\text{)}$$

Similar to circumscription. (c.f. stability vs. minimality)

Translation vs. Fixpoint Traditions



Circumscription

The models of $\mathrm{CIRC}[F; \mathbf{p}]$ are the models of F that are minimal on $\mathbf{p}$. Formally,

$$\mathrm{CIRC}[F; \mathbf{p}] = F \wedge \neg \exists \mathbf{u}(\mathbf{u} < \mathbf{p} \wedge F(\mathbf{u}))$$

- $\mathbf{u}$: a list of distinct predicate variables similar to $\mathbf{p}$;
- $\mathbf{u} < \mathbf{p}$: a formula that expresses that $\mathbf{u}$ is strictly stronger than $\mathbf{p}$;
- $F(\mathbf{u})$ is obtained from F by replacing all occurrences of $\mathbf{p}$ with $\mathbf{u}$.

First-Order Stable Model Semantics

The stable models of a first-order sentence F relative to a list $\mathbf{p}$ of predicate constants are the models of the second-order formula

$$\mathrm{SM}[F; \mathbf{p}] = F \wedge \neg\exists \mathbf{u}(\mathbf{u} < \mathbf{p} \wedge F^*(\mathbf{u}))$$

- $F^*(\mathbf{u})$ is defined as:
  - $p_i(\mathbf{t})^* = u_i(\mathbf{t})$ if $p_i \in \mathbf{p}$
  - for other atomic formula F, $F^* = F$
  - $(G \odot H)^* = (G^* \odot H^*) \qquad (\odot \in \{\wedge, \vee\})$
  - $(G \to H)^* = (G^* \to H^*) \wedge (G \to H)$
  - $(QxG)^* = QxG^* \qquad (Q \in \{\forall, \exists\})$

  ($\neg F$ is shorthand for $F \to \bot$.)

If we drop "$\wedge(G \to H)$" then it becomes the definition of circumscription [McC80].

Relation to Reduct-Based Semantics

Proposition

The stable models of a logic program Π *according to the 1988 definition are precisely the Herbrand models of* $\mathrm{SM}[\Pi; pr(\Pi)]$.

Example

$\{p(a), q(a)\}$ is the unique

- stable model of $\begin{cases} p(a) \\ q(x) \leftarrow p(x), not\ r(x) \end{cases}$ under the 1988 definition
- Herbrand model of $\mathrm{SM}[p(a) \wedge \forall x(p(x) \wedge \neg r(x) \rightarrow q(x)); p, q, r]$.

Reductive Semantics [LLP08]

- A simple, alternative approach to understanding the meaning of counting and choice in answer set programming by reducing them to first order formulas.

  The syntax of RASPL-1 (Reductive Answer Set Programming Language - Version 1) extends the syntax of disjunctive logic programs by allowing constructs for counting and choice.

  The semantics is defined by turning RASPL-1 programs to first-order sentences under the stable model semantics.

$$\{q(x)\} \leftarrow p(x) \quad \Rightarrow \quad \forall x(p(x) \rightarrow (q(x) \vee \neg q(x)))$$

$$r \leftarrow \#count\{x : p(x)\} \geq 2$$

$$\Rightarrow \quad (\exists xy(p(x) \wedge p(y) \wedge \neg(x = y))) \rightarrow r$$

Stable Models and Circumscription

Neither is stronger than the other.

- $$\begin{aligned} \mathrm{CIRC}[\forall x(p(x) \vee \neg p(x));\ p] &\Leftrightarrow \forall x \neg p(x) \\ \mathrm{SM}[\forall x(p(x) \vee \neg p(x));\ p] &\Leftrightarrow \top \end{aligned}$$
- $$\begin{aligned} \mathrm{CIRC}[\forall x(\neg p(x) \rightarrow q(x));\ p, q] &\Leftrightarrow \forall x(\neg p(x) \leftrightarrow q(x)) \\ \mathrm{SM}[\forall x(\neg p(x) \rightarrow q(x));\ p, q] &\Leftrightarrow \forall x(\neg p(x) \wedge q(x)) \end{aligned}$$

Stable Models and Circumscription

Theorem

The stable model semantics and circumscription coincide on the class of "canonical" formulas [LP12b].

In other words, minimal models and stable models coincide on canonical formulas.

The theorem allows us to reformulate the Event Calculus, Situation Calculus, and Temporal Action Logics in ASP, and use ASP solvers to compute them [KLP09, LP10, LP12b, LP12a]

"Reformulating Temporal Action Logics in Answer Set Programming", (Tuesday 2:45-3:05 PM)

Turning Event Calculus Description to ASP

$$(HoldsAt(f,t) \wedge \neg ReleasedAt(f,t+1) \wedge \\ \neg \exists e(Happens(e,t) \wedge Terminates(e,f,t))) \rightarrow HoldsAt(f,t+1).$$

is turned into the conjunction of

$$(HoldsAt(f,t) \wedge \neg ReleasedAt(f,t+1) \wedge \\ \neg q(f,t)) \rightarrow HoldsAt(f,t+1)$$
$$Happens(e,t) \wedge Terminates(e,f,t) \rightarrow q(f,t)$$

and then turned into rules

$$HoldsAt(f,t+1) \leftarrow HoldsAt(f,t), not\ ReleasedAt(f,t+1), \\ not\ q(f,t))$$
$$q(f,t) \leftarrow Happens(e,t),\ Terminates(e,f,t)$$

ECASP vs. DEC reasoner

http://reasoning.eas.asu.edu/ecasp

http://decreasoner.sourceforge.net/csr/ecas/

ASP-based vs. SAT-based Approach

- DEC reasoner is based on the reduction of circumscription to completion. Able to solve 11 out of 14 benchmark problems.
- ECASP can handle the *full* version of the event calculus (modulo grounding). Able to solve all 14 problems.
- For example, the following axiom cannot be handled by the DEC reasoner, but can be done by the ASP approach.

$$\begin{aligned} &\mathit{HoldsAt}(\mathit{HasBananas}, t) \\ &\wedge \mathit{Initiates}(e, \mathit{At}(\mathit{Monkey}, l), t) \rightarrow \mathit{Initiates}(e, \mathit{At}(\mathit{Bananas}, l), t) \end{aligned}$$

- ECASP computes faster.

Experiments (I)

Problem (max. time)	DEC reasoner	ECASP w/ LPARSE + CMODELS	ECASP w/ GRINGO + CLASP	ECASP W/ CLINGO
BusRide (15)	—	0.48 (0.42+0.06) A:156/R:7899/C:188	0.04 (0.03+0.01) A:733/R:3428	—
Commuter (15)	—	498.11 (447.50+50.61) A:4913/R:7383943/C:4952	44.42 (37.86 + 6.56) A:24698/R:5381620	28.79
Kitchen Sink (25)	71.10 (70.70+0.40) A:1014/C:12109	43.17 (37.17+6.00) A:123452/R:482018/C:0	2.47 (1.72+0.75) A:114968/R:179195	2.03
Thielscher Circuit (20)	13.9 (13.6+0.3) A:5138/C:16122	0.42 (0.38+0.04) A:3160/R:9131/C:0	0.07 (0.05+0.02) A:1686/R:6510	0.05
Walking Turkey (15)	—	0.05 (0.04+0.01) A:556/R:701/C:0	0.04 (0.01+0.03) A:364/R:503	0.01

A: number of atoms, C: number of clauses, R: number of ground rules

DEC reasoner and CMODELS used the same SAT solver RELSAT.

Experiments (II)

Problem (max. time)	DEC reasoner	ECASP w/ LPARSE + CMODELS	ECASP w/ GRINGO + CLASP	ECASP W/ CLINGO
Falling w/ AntiTraj (15)	270.2 (269.3+0.9) A:416/C:3056	0.74 (0.66+0.08) A:5757/R:10480/C:0	0.10 (0.08+0.02) A:4121/R:7820	0.08
Falling w/ Events (25)	107.70 (107.50+0.20) A:1092/C:12351	34.77 (30.99+3.78) A:1197/R:390319/C:1393	2.90 (2.01+0.89) A:139995/R:208282	2.32
HotAir Balloon (15)	61.10 (61.10+0.00) A:288/C:1163	0.19 (0.16+0.03) A:489/R:2958/C:678	0.04 (0.03+0.01) A:1137/R:1909	0.03
Telephone1 (40)	18.00 (17.50+0.50) A:5419/C:41750	1.70 (1.51+0.19) A:23978/R:30005/C:0	0.31 (0.26+0.05) A:21333/R:27201	0.25

A: number of atoms, C: number of clauses, R: number of ground rules

Summary

- Answer Set Programming is a declarative programming paradigm oriented towards knowledge intensive and combinatorial search problems.
- ASP processing tools are under continuous development for large scale practical applications.
- ASP = LP+KR+SAT+DB

Pointers are available at

`http://peace.eas.asu.edu/aaai12tutorial`

You're welcome to contact us for more questions.

Acknowledgements

We are grateful to *Michael Bartholomew, Vladimir Lifschitz, Max Ostrowski, Volkan Patoglu, Peter Schueller, Mirek Truszczynski* for providing valuable comments on these slides, and to *Nicola Leone, Torsten Schaub* for providing us with material on relevant ASP applications.

We acknowledge the funding support: National Science Foundation under Grant IIS-0916116 and by the South Korea IT R&D program MKE/KIAT 2010-TD-300404-001.

Bibliography

R. Alami, R. Chatila, S. Fleury, M. Ghallab, and F. Ingrand.

An architecture for autonomy.
International Journal of Robotics Research, 17:315–337, 1998.

Erdi Aker, Ahmetcan Erdogan, Esra Erdem, and Volkan Patoglu.

Causal reasoning for planning and coordination of multiple housekeeping robots.
In Proc. of LPNMR, pages 311–316, 2011.

Erdi Aker, Ahmetcan Erdogan, Esra Erdem, and Volkan Patoglu.

Housekeeping with multiple autonomous robots: Knowledge representation and automated reasoning for a tightly integrated robot control architecture.
In IROS 2011 Workshop: Knowledge Representation for Autonomous Robots, 2011.

Erdi Aker, Ahmetcan Erdogan, Esra Erdem, and Volkan Patoglu.

Housekeeping with multiple autonomous robots: Representation, reasoning and execution.
In Proc. of Commonsense, 2011.

M. Alviano, W. Faber, and N. Leone.

Disjunctive asp with functions: Decidable queries and effective computation.
TPLP, 10(4-6):497–512, 2010.

Erdi Aker, Volkan Patoglu, and Esra Erdem.

Answer set programming for reasoning with semantic knowledge in collaborative housekeeping robotics.
In Proc. of the 10'th IFAC Symposium on Robot Control (SYROCO), 2012.

Marcello Balduccini.

Representing constraint satisfaction problems in answer set programming.
In Working Notes of the Workshop on Answer Set Programming and Other Computing Paradigms (ASPOCP), 2009.

Marcello Balduccini.

Representing constraint satisfaction problems in answer set programming.
In Proceedings of ICLP'09 Workshop on Answer Set Programming and Other Computing Paradigms (ASPOCP'09), 2009.


C. Belta, A. Bicchi, M. Egerstedt, E. Frazzoli, E. Klavins, and G.J. Pappas.
Symbolic planning and control of robot motion [grand challenges of robotics].
Robotics Automation Magazine, IEEE, 14(1):61–70, 2007.


G. Boenn, M. Brain, M. D. Vos, and J. Fitch.
Anton: Composing logic and logic composing.
In Proc. of LPNMR, pages 542–547, 2009.


Georg Boenn, Martin Brain, Marina De Vos, and John Fitch.
Automatic music composition using answer set programming.
TPLP, 11(2–3):397–427, 2011.


Olivier Bodenreider, Zeynep H. Coban, Mahir C. Doganay, Esra Erdem, and Hilal Kosucu.
A preliminary report on answering complex queries related to drug discovery using answer set programming.
In Proc. of the 3rd Int'l Workshop on Applications of Logic Prog. to the Semantic Web and Web Services (ALPSWS), 2008.


Chitta Baral and Juraj Dzifcak.
Solving puzzles described in english by automated translation to answer set programming and learning how to do that translation.
In Proc. of KR, 2012.


Chitta Baral, Juraj Dzifcak, and Tran Cao Son.
Using answer set programming and lambda calculus to characterize natural language sentences with normatives and exceptions.
In Proc. of AAAI, pages 818–823, 2008.


D. R. Brooks, E. Erdem, S. T. Erdogan, J. W. Minett, and D. Ringe.
Inferring phylogenetic trees using answer set programming.
J. Autom. Reasoning, 39(4):471–511, 2007.


Gerhard Brewka, Thomas Eiter, and Michael Fink.
Nonmonotonic multi-context systems: A flexible approach for integrating heterogeneous knowledge sources.
In Proc. of LPNMR, pages 233–258, 2011.


Gerhard Brewka, Thomas Eiter, Michael Fink, and Antonius Weinzierl.

Managed multi-context systems.
In Proc. of IJCAI, pages 786–791, 2011.

M. Balduccini and M. Gelfond.

Diagnostic reasoning with a-prolog.
CoRR, cs.AI/0312040, 2003.

Chitta Baral, Marcos Alvarez Gonzalez, and Aaron Gottesman.

The inverse lambda calculus algorithm for typed first order logic lambda calculus and its application to translating english to fol.
In Correct Reasoning, pages 40–56, 2012.

Chitta Baral, Michael Gelfond, and J. Nelson Rushton.

Probabilistic reasoning with answer sets.
TPLP, 9(1):57–144, 2009.

Chitta Baral, Michael Gelfond, and J. Nelson Rushton.

Probabilistic reasoning with answer sets.
TPLP, 9(1):57–144, 2009.

Chitta Baral, Gregory Gelfond, Tran Cao Son, and Enrico Pontelli.

Using answer set programming to model multi-agent scenarios involving agents' knowledge about other's knowledge.
In Proc. of AAMAS, pages 259–266, 2010.

Chitta Baral and Matt Hunsaker.

Using the probabilistic logic programming language p-log for causal and counterfactual reasoning and non-naive conditioning.
In Proc. of IJCAI, pages 243–249, 2007.

Michael Bartholomew and Joohyung Lee.

Stable models of formulas with intensional functions.
In Proceedings of International Conference on Principles of Knowledge Representation and Reasoning (KR), 2012.
To appear.

Gerhard Brewka, Ilkka Niemelä, and Miroslaw Truszczynski.

Preferences and nonmonotonic reasoning.
AI Magazine, 29(4):69–78, 2008.

Gerhard Brewka, Ilkka Niemelä, and Miroslaw Truszczynski.

Preferences and nonmonotonic reasoning.
AI Magazine, 29(4):69–78, 2008.

G. Brewka.

Preferences, contexts and answer sets.
In Proc. of ICLP, page 22, 2007.

C. Baral and C. Uyan.

Declarative specification and solution of combinatorial auctions using logic programming.
In Proc. of LPNMR, pages 186–199, 2001.

Pedro Cabalar.

Functional answer set programming.
TPLP, 11(2-3):203–233, 2011.

Francesco Calimeri, Susanna Cozza, Giovambattista Ianni, and Nicola Leone.

Computable functions in ASP: theory and implementation.
In Proceedings of International Conference on Logic Programming (ICLP), pages 407–424, 2008.

D. Cakmak, E. Erdem, and H. Erdogan.

Computing weighted solutions in ASP: Representation-based method vs. search-based method.
Ann. Math. Artif. Intell., 62(3–4):219–258, 2011.

D. Calvanese, T. Eiter, and M. Ortiz.

Regular path queries in expressive description logics with nominals.
In Proc. of IJCAI, pages 714–720, 2009.

Ozan Caldiran, Kadir Haspalamutgil, Abdullah Ok, Can Palaz, Esra Erdem, and Volkan Patoglu.

Bridging the gap between high-level reasoning and low-level control.
In Proc. of LPNMR, pages 342–354, 2009.


Ozan Caldiran, Kadir Haspalamutgil, Abdullah Ok, Can Palaz, Esra Erdem, and Volkan Patoglu.
From discrete task plans to continuous trajectories.
In Proc. of the ICAPS 2009 Workshop on Bridging The Gap Between Task And Motion Planning (BTAMP), 2009.


Michael Casolary and Joohyung Lee.
Representing the language of the causal calculator in answer set programming.
In ICLP (Technical Communications), pages 51–61, 2011.


Keith Clark.
Negation as failure.
In Herve Gallaire and Jack Minker, editors, Logic and Data Bases, pages 293–322. Plenum Press, New York, 1978.


Domenico Corapi, Daniel Sykes, Katsumi Inoue, and Alessandra Russo.
Probabilistic rule learning in nonmonotonic domains.
In Proc. of CLIMA, pages 243–258, 2011.


J. Dix, T. Eiter, M. Fink, A. Polleres, and Y. Zhang.
Monitoring agents using declarative planning.
Fundam. Inform., 57(2-4):345–370, 2003.


James P. Delgrande.
A program-level approach to revising logic programs under the answer set semantics.
TPLP, 10(4-6):565–580, 2010.


Jürgen Dix, Wolfgang Faber, and V. S. Subrahmanian.
Privacy preservation using multi-context systems and default logic.
In Correct Reasoning, pages 195–210, 2012.


J. P. Delgrande, T. Grote, and A. Hunter.
A general approach to the verification of cryptographic protocols using answer set programming.
In Proc. of LPNMR, pages 355–367, 2009.


T. Eiter, G. Brewka, M. Dao-Tran, M. Fink, G. Ianni, and T. Krennwallner.
Combining nonmonotonic knowledge bases with external sources.

In Proc. of FroCos, pages 18–42, 2009.

Thomas Eiter, Gerhard Brewka, Minh Dao-Tran, Michael Fink, Giovambattista Ianni, and Thomas Krennwallner.

Combining Nonmonotonic Knowledge Bases with External Sources.
In Silvio Ghilardi and Roberto Sebastiani, editors, 7th International Symposium on Frontiers of Combining Systems (FroCos 2009), Trento, Italy, September 16–18, 2009, volume 5749 of LNAI, pages 18–42. Springer, September 2009.

Halit Erdogan, Esra Erdem, and Olivier Bodenreider.

Exploiting umls semantics for checking semantic consistency among umls concepts.
In Proc. of MedInfo, 2010.

T. Eiter, E. Erdem, H. Erdogan, and M. Fink.

Finding similar or diverse solutions in answer set programming.
In Proc. of ICLP, pages 342–356, 2009.

Esra Erdem, Yelda Erdem, Halit Erdogan, and Umut Oztok.

Finding answers and generating explanations for complex biomedical queries.
In Proc. of AAAI, pages 785–790, 2011.

Esra Erdem, Halit Erdogan, and Umut Oztok.

Bioquery-asp: Querying biomedical ontologies using answer set programming.
In Proc. of RuleML2011@BRF Challenge, 2011.

E. Erdem, O. Erdem, and F. Türe.

Haplo-asp: Haplotype inference using answer set programming.
In Proc. of LPNMR, pages 573–578, 2009.

T. Eiter, W. Faber, N. Leone, and G. Pfeifer.

The diagnosis frontend of the dlv system.
AI Communications, 12(1-2):99–111, 1999.

Thomas Eiter, Michael Fink, and Peter Schüller.

Approximations for explanations of inconsistency in partially known multi-context systems.
In Proc. of LPNMR, pages 107–119, 2011.


Thomas Eiter, G.Ianni, R.Schindlauer, and H.Tompits.

Effective integration of declarative rules with external evaluations for Semantic-Web reasoning.
In Proc. of ESWC, pages 273–287, 2006.


Uwe Egly, Sarah Alice Gaggl, and Stefan Woltran.

Aspartix: Implementing argumentation frameworks using answer-set programming.
In Proc. of ICLP, pages 734–738, 2008.


Uwe Egly, Sarah Alice Gaggl, and Stefan Woltran.

Answer-set programming encodings for argumentation frameworks.
Argument and Computation, 1(2):147–177, 2010.


Esra Erdem, Kadir Haspalamutgil, Can Palaz, Volkan Patoglu, and Tansel Uras.

Combining high-level causal reasoning with low-level geometric reasoning and motion planning for robotic manipulation.
In Proc. of the 2011 IEEE International Conference on Robotics and Automation (ICRA 2011), pages 4575–4581, 2011.


Esra Erdem, Kadir Haspalamutgil, Volkan Patoglu, and Tansel Uras.

Causality-based planning and diagnostic reasoning for cognitive factories.
In Proc. of the 17'th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA), 2012.


Thomas Eiter, Giovambattista Ianni, Thomas Lukasiewicz, Roman Schindlauer, and Hans Tompits.

Combining answer set programming with description logics for the semantic web.
Artificial Intelligence, 172(12-13):1495–1539, 2008.


Esra Erdem, Katsumi Inoue, Johannes Oetsch, Joerg Puehrer, Hans Tompits, and Cemal Yilmaz.

Answer-set programming as a new approach to event-sequence testing.
In Proc. of the 3rd International Conference on Advances in System Testing and Validation Lifecycle (VALID'11), 2011.


Thomas Eiter, Thomas Krennwallner, Patrik Schneider, and Guohui Xiao.

Uniform evaluation of nonmonotonic dl-programs.
In Proc. of FoIKS, pages 1–22, 2012.


E. Erdem, V. Lifschitz, and D. Ringe.

Temporal phylogenetic networks and logic programming.

Theory and Practice of Logic Programming, 6(5):539–558, 2006.

E. Erdem, V. Lifschitz, and M. F. Wong.

Wire routing and satisfiability planning.
In In Proceedings CL-2000, pages 822–836. Springer-Verlag. LNCS, 2000.

Esra Erdem and Volkan Patoglu.

Applications of action languages in cognitive robotics.
In Correct Reasoning, pages 229–246, 2012.

E. Erdem.

Phylo-asp: Phylogenetic systematics with answer set programming.
In Proc. of LPNMR, pages 567–572, 2009.

Esra Erdem.

Applications of answer set programming in phylogenetic systematics.
In Logic Programming, Knowledge Representation, and Nonmonotonic Reasoning, pages 415–431, 2011.

D. East and M. Truszczynski.

More on wire routing with asp.
In Proc. of ASP, 2001.

Thomas Eiter and Kewen Wang.

Semantic forgetting in answer set programming.
Artif. Intell., 172(14):1644–1672, 2008.

Esra Erdem and Reyyan Yeniterzi.

Transforming controlled natural language biomedical queries into answer set programs.
In Proc. of BioNLP, pages 117–124, 2009.

Paolo Ferraris.

Answer sets for propositional theories.
In Proceedings of International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR), pages 119–131, 2005.


Fernando Zacarias Flores, Mauricio Javier Osorio Galindo, and Edgar Fernandez Plascencia.
Updates under pstable.
Engineering Letters, 15(2):311–315, 2007.


Paolo Ferraris and Vladimir Lifschitz.
On the stable model semantics of first-order formulas with aggregates.
In Proceedings of the 2010 Workshop on Nonmonotonic Reasoning, 2010.


Onofrio Febbraro, Nicola Leone, Giovanni Grasso, and Francesco Ricca.
Jasp: A framework for integrating answer set programming with java.
In Proceedings of International Conference on Principles of Knowledge Representation and Reasoning (KR), 2012.


Paolo Ferraris, Joohyung Lee, and Vladimir Lifschitz.
A generalization of the Lin-Zhao theorem.
Annals of Mathematics and Artificial Intelligence, 47:79–101, 2006.


Paolo Ferraris, Joohyung Lee, and Vladimir Lifschitz.
A new perspective on stable models.
In Proceedings of International Joint Conference on Artificial Intelligence (IJCAI), pages 372–379. AAAI Press, 2007.


Paolo Ferraris, Joohyung Lee, and Vladimir Lifschitz.
Stable models and circumscription.
Artificial Intelligence, 175:236–263, 2011.


Wolfgang Faber, Nicola Leone, and Gerald Pfeifer.
Recursive aggregates in disjunctive logic programs: Semantics and complexity.
In Proceedings of European Conference on Logics in Artificial Intelligence (JELIA), 2004.


R. Finkel, V. W. Marek, and M. Truszczynski.
Constraint lingo: A program for solving logic puzzles and other tabular constraint problems, 2002.


Onofrio Febbraro, Kristian Reale, and Francesco Ricca.
Aspide: Integrated development environment for answer set programming.

In Procedings of International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR), pages 317–330, 2011.

Sarah Alice Gaggl.

Towards a general argumentation system based on answer-set programming.
In ICLP (Technical Communications), pages 265–269, 2010.

M. Gebser, C. Guziolowski, M. Ivanchev, T. Schaub, A. Siegel, S. Thiele, and P. Veber.

Repair and prediction (under inconsistency) in large biological networks with answer set programming.
In Proc. of KR, 2010.

Martin Gebser, Roland Kaminski, Benjamin Kaufmann, Max Ostrowski, Torsten Schaub, and Sven Thiele.

Engineering an incremental asp solver.
In Proc. of ICLP, pages 190–205, 2008.

Martin Gebser, Roland Kaminski, Benjamin Kaufmann, Max Ostrowski, Torsten Schaub, and Sven Thiele.

Engineering an incremental asp solver.
In ICLP, pages 190–205, 2008.

Martin Gebser, Benjamin Kaufmann, Andre Neumann, and Torsten Schaub.

Conflict-driven answer set solving.
In Proceedings of 20th International Joint Conference on Artificial Intelligence (IJCAI'07), pages 386–392. MIT Press, 2007.

Goran Gogic, Henry Kautz, Christos Papadimitriou, and Bart Selman.

The comparative linguistics of knowledge representation.
In Proceedings of International Joint Conference on Artificial Intelligence (IJCAI), pages 862–869, 1995.

M. Gebser, R. Kaminski, and T. Schaub.

aspcud: A Linux package configuration tool based on answer set programming.
In Proc. of LoCoCo, pages 12–25, 2011.

Martin Gebser, Roland Kaufmann, and Torsten Schaub.

Gearing up for effective asp planning.
In Correct Reasoning, pages 296–310, 2012.


Michael Gelfond and Vladimir Lifschitz.
The stable model semantics for logic programming.
In Robert Kowalski and Kenneth Bowen, editors, Proceedings of International Logic Programming Conference and Symposium, pages 1070–1080. MIT Press, 1988.


Enrico Giunchiglia, Joohyung Lee, Vladimir Lifschitz, Norman McCain, and Hudson Turner.
Nonmonotonic causal theories.
Artificial Intelligence, 153(1–2):49–104, 2004.


Martin Gebser, Joohyung Lee, and Yuliya Lierler.
Elementary sets for logic programs.
In Proceedings of National Conference on Artificial Intelligence (AAAI), 2006.


Martin Gebser, Joohyung Lee, and Yuliya Lierler.
On elementary loops of logic programs.
Theory and Practice of Logic Programming, 11(6):953–988, 2011.


Enrico Giunchiglia, Yuliya Lierler, and Marco Maratea.
SAT-based answer set programming.
In Proceedings of National Conference on Artificial Intelligence (AAAI), pages 61–66, 2004.


Michael Gelfond, Vladimir Lifschitz, and Arkady Rabinov.
What are the limitations of the situation calculus?
In Robert Boyer, editor, Automated Reasoning: Essays in Honor of Woody Bledsoe, pages 167–179. Kluwer, 1991.


M. Gebser, M. Ostrowski, and T. Schaub.
Constraint answer set solving.
In Proceedings of International Conference on Logic Programming (ICLP), pages 235–249, 2009.


Martin Gebser, Max Ostrowski, and Torsten Schaub.
Constraint answer set solving.
In Proceedings of 25th International Conference on Logic Programming (ICLP), pages 235–249. Springer, 2009.


Martin Gebser and Torsten Schaub.

Loops: Relevant or redundant?
In Proceedings of the Eighth International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR'05), pages 53–65, 2005.

M. Gebser, T. Schaub, S. Thiele, and P. Veber.

Detecting inconsistencies in large biological networks with answer set programming.
Theory and Practice of Logic Programming, 11(2):1–38, 2011.

K. Heljanko and I. Niemela.

Bounded LTL model checking with stable models.
In Proc. of LPNMR, pages 200–212, 2003.

Kadir Haspalamutgil, Can Palaz, Tansel Uras, Esra Erdem, and Volkan Patoglu.

A tight integration of task and motion planning in an execution monitoring framework.
In Proc. of the AAAI 2010 Workshop on Bridging The Gap Between Task And Motion Planning (BTAMP), 2010.

K. Inoue and C. Sakama.

Abductive framework for nonmonotonic theory change.
In IJCAI, pages 204–210, 1995.

Tomi Janhunen, Guohua Liu, and Ilkka Niemel.

Tight integration of non-ground answer set programming and satisfiability modulo theories.
In Working notes of the 1st Workshop on Grounding and Transformations for Theories with Variables, 2011.

Tae-Won Kim, Joohyung Lee, and Ravi Palla.

Circumscriptive event calculus as answer set programming.
In Proceedings of International Joint Conference on Artificial Intelligence (IJCAI), pages 823–829, 2009.

Henry Kautz and Bart Selman.

Planning as satisfiability.
In Proceedings of European Conference on Artificial Intelligence (ECAI), pages 359–363, 1992.

Steven M. Lavalle.

Rapidly-exploring random trees: A new tool for path planning.
Technical report, 1998.


Joohyung Lee.
Nondefinite vs. definite causal theories.
In Proceedings 7th Int'l Conference on Logic Programming and Nonmonotonic Reasoning, pages 141–153, 2004.


Joohyung Lee.
A model-theoretic counterpart of loop formulas.
In Proceedings of International Joint Conference on Artificial Intelligence (IJCAI), pages 503–508. Professional Book Center, 2005.


N. Leone, G. Greco, G. Ianni, V. Lio, G. Terracina, T. Eiter, W. Faber, M. Fink, G. Gottlob, R. Rosati, D. Lembo, M. Lenzerini, M. Ruzzi, E. Kalka, B. Nowicki, and W. Staniszkis.
The infomix system for advanced integration of incomplete and inconsistent data.
In Proc. of SIGMOD, pages 915–917, 2005.


Yuliya Lierler.
Abstract answer set solvers.
In Proceedings of International Conference on Logic Programming (ICLP), pages 377–391. Springer, 2008.


Yuliya Lierler.
Abstract answer set solvers with backjumping and learning.
Theory and Practice of Logic Programming, 11:135–169, 2011.


V. Lifschitz.
Answer set programming and plan generation.
Artif. Intell., 138(1-2):39–54, 2002.


Vladimir Lifschitz.
Answer set programming and plan generation.
Artificial Intelligence, 138:39–54, 2002.


Vladimir Lifschitz.
Logic programs with intensional functions.
In Proceedings of International Conference on Principles of Knowledge Representation and Reasoning (KR), 2012. This volume.


Joohyung Lee and Vladimir Lifschitz.

Loop formulas for disjunctive logic programs.
In Proceedings of International Conference on Logic Programming (ICLP), pages 451–465, 2003.


Joohyung Lee and Fangzhen Lin.

Loop formulas for circumscription.
In Proceedings of National Conference on Artificial Intelligence (AAAI), pages 281–286, 2004.


Joohyung Lee and Fangzhen Lin.

Loop formulas for circumscription.
Artificial Intelligence, 170(2):160–185, 2006.


Joohyung Lee, Vladimir Lifschitz, and Ravi Palla.

A reductive semantics for counting and choice in answer set programming.
In Proceedings of the AAAI Conference on Artificial Intelligence (AAAI), pages 472–479. AAAI Press, 2008.


Joohyung Lee and Yunsong Meng.

On reductive semantics of aggregates in answer set programming.
In Procedings of International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR), pages 182–195, 2009.


Joohyung Lee and Yunsong Meng.

Stable models of formulas with generalized quantifiers (preliminary report).
In Technical Communications of the 28th International Conference on Logic Programming, 2012.


Joohyung Lee and Ravi Palla.

System F2LP – computing answer sets of first-order formulas.
In Procedings of International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR), pages 515–521, 2009.


Joohyung Lee and Ravi Palla.

Situation calculus as answer set programming.
In Proceedings of the AAAI Conference on Artificial Intelligence (AAAI), pages 309–314, 2010.


Joohyung Lee and Ravi Palla.

Integrating rules and ontologies in the first-order stable model semantics (preliminary report).
In Proceedings of International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR), pages 248–253, 2011.

Joohyung Lee and Ravi Palla.

Reformulating temporal action logics in answer set programming.
In Proceedings of the AAAI Conference on Artificial Intelligence (AAAI), 2012.

Joohyung Lee and Ravi Palla.

Reformulating the situation calculus and the event calculus in the general theory of stable models and in answer set programming.
Journal of Artificial Inteligence Research (JAIR), 43:571–620, 2012.

Vladimir Lifschitz and Alexander Razborov.

Why are there so many loop formulas?
ACM Transactions on Computational Logic, 7:261–268, 2006.

H. Liu and P. Singh.

ConceptNet: A practical commonsense reasoning toolkit.
BT Technology Journal, 22, 2004.

Yuliya Lierler and Peter Schüller.

Parsing combinatory categorial grammar via planning in answer set programming.
In Correct Reasoning, pages 436–453, 2012.

Vladimir Lifschitz, Lappoon R. Tang, and Hudson Turner.

Nested expressions in logic programs.
Annals of Mathematics and Artificial Intelligence, 25:369–389, 1999.

Fangzhen Lin and Yuting Zhao.

ASSAT: Computing answer sets of a logic program by SAT solvers.
In Proceedings of National Conference on Artificial Intelligence (AAAI), pages 112–117. MIT Press, 2002.

Fangzhen Lin and Yuting Zhao.

ASSAT: Computing answer sets of a logic program by SAT solvers.

Artificial Intelligence, 157:115–137, 2004.

John McCarthy.

Circumscription—a form of non-monotonic reasoning.
Artificial Intelligence, 13:27–39,171–172, 1980.

Veena S. Mellarkod, Michael Gelfond, and Yuanlin Zhang.

Integrating answer set programming and constraint logic programming.
Annals of Mathematics and Artificial Intelligence, 2008.

A. Mileo, D. Merico, and R. Bisiani.

Wireless sensor networks supporting context-aware reasoning in assisted living.
In Proc. of PETRA, pages 1–2, 2008.

A. Mileo, D. Merico, and R. Bisiani.

Non-monotonic reasoning supporting wireless sensor networks for intelligent monitoring: The sindi system.
In Proc. of LPNMR, pages 585–590, 2009.

A. Mileo, T. Schaub, D. Merico, and R. Bisiani.

Knowledge-based multi-criteria optimization to support indoor positioning.
AMAI, 62(3–4):345–370, 2011.

M. Nogueira, M. Balduccini, M. Gelfond, R. Watson, and M. Barry.

An a-prolog decision support system for the space shuttle.
In Proc. of PADL, pages 169–183. Springer, 2001.

Robert Nieuwenhuis, Albert Oliveras, and Cesare Tinelli.

Solving SAT and SAT modulo theories: From an abstract Davis-Putnam-Logemann-Loveland procedure to DPLL(T).
Journal of the ACM, 53(6):937–977, 2006.

Mauricio Osorio and Victor Cuevas.

Updates in answer set programming: An approach based on basic structural properties.
TPLP, 7(4):451–479, 2007.

Johannes Oetsch, Jörg Pührer, and Hans Tompits.

The sealion has landed: An ide for answer-set programming—preliminary report.
INFSYS Research Report, 1843-11-06, 2011.

Nikolay Pelov, Marc Denecker, and Maurice Bruynooghe.
Well-founded and stable semantics of logic programs with aggregates.
TPLP, 7(3):301–353, 2007.

Jörg Pührer, Stijn Heymans, and Thomas Eiter.
Dealing with inconsistency when combining ontologies and rules using dl-programs.
In Proc. of ESWC (1), pages 183–197, 2010.

Enrico Pontelli, Tran Cao Son, Chitta Baral, and Gregory Gelfond.
Answer set programming and planning with knowledge and world-altering actions in multiple agent domains.
In Correct Reasoning, pages 509–526, 2012.

Francesco Ricca, Antonella Dimasi, Giovanni Grasso, Salvatore Maria Ielpa, Salvatore Iiritano, Marco Manna, and Nicola Leone.
A logic-based system for e-tourism.
Fundam. Inform., 105(1–2):35–55, 2010.

Francesco Ricca, Giovanni Grasso, Mario Alviano, Marco Manna, Vincenzino Lio, Salvatore Iiritano, and Nicola Leone.
Team-building with answer set programming in the gioia-tauro seaport.
Theory and Practice of Logic Programming, 12:361–381, 2012.

C. Sakama.
Learning by answer sets.
In Proc. of AAAI Spring Symposium:Answer Set Programming, 2001.

Chiaki Sakama.
Induction from answer sets in nonmonotonic logic programs.
ACM Trans. Comput. Log., 6(2):203–231, 2005.

Chiaki Sakama.
Dishonest reasoning by abduction.
In Proc. of IJCAI, pages 1063–1064, 2011.

Chiaki Sakama and Katsumi Inoue.
Brave induction: a logical framework for learning from incomplete information.
Machine Learning, 76(1):3–35, 2009.

Mantas Simkus.
Fusion of logic programming and description logics.
In Proc. of ICLP, pages 551–552, 2009.

T. Soininen and I. Niemelä.
Developing a declarative rule language for applications in product configuration.
In Proc. of PADL, pages 305–319, 1998.

Patrik Simons, Ilkka Niemelä, and Timo Soininen.
Extending and implementing the stable model semantics.
Artificial Intelligence, 138:181–234, 2002.

T. C. Son, E. Pontelli, and C. Sakama.
Logic programming for multiagent planning with negotiation.
In Proc. of ICLP, pages 99–114, 2009.

T. C. Son and C. Sakama.
Reasoning and planning with cooperative actions for multiagents using answer set programming.
In Proc. of DALT, pages 208–227, 2009.

T. Schaub and S. Thiele.
Metabolic network expansion with answer set programming.
In Proc. of ICLP, pages 312–326, 2009.

T. Schaub and K. Wang.
A comparative study of logic programs with preference.
In Proc. of IJCAI, pages 597–602, 2001.

Yi-Dong Shen and Kewen Wang.
Extending logic programs with description logic expressions for the semantic web.

In Proc. of ISWC (1), pages 633–648, 2011.

Tommi Syrjänen.

Including diagnostic information in configuration models.
In Proc. of Computational Logic, pages 837–851, 2000.

Luis Tari, Saadat Anwar, Shanshan Liang, Jörg Hakenberg, and Chitta Baral.

Synthesis of pharmacokinetic pathways through knowledge acquisition and automated reasoning.
In Proc. of Pacific Symposium on Biocomputing, pages 465–476, 2010.

N. Tran and C. Baral.

Reasoning about triggered actions in ansprolog and its application to molecular interactions in cells.
In Proc. of KR, pages 554–564, 2004.

F. Türe and E. Erdem.

Efficient haplotype inference with answer set programming.
In Proc. of AAAI, pages 1834–1835, 2008.

Phan Huy Tu, Tran Cao Son, Michael Gelfond, and A. Ricardo Morales.

Approximation of action theories and its application to conformant planning.
Artif. Intell., 175(1):79–119, 2011.

Juha Tiihonen, Timo Soininen, Ilkka Niemelae, and Reijo Sulonen.

A practical tool for mass-customising configurable products.
In Proc. of ICDE, pages 1290–1299, 2003.

Calvin Kai Fan Tang and Eugenia Ternovska.

Model checking abstract state machines with answer set programming.
Fundam. Inform., 77(1-2):105–141, 2007.

M. D. Vos, T. Crick, J. Padget, M. Brain, O. Cliffe, and J. Needham.

A multi-agent platform using ordered choice logic programming.
In In Declarative Agent Languages and Technologies (DALT'05), pages 72–88, 2005.

Marina De Vos and Dirk Vermeir.

Dynamic decision-making in logic programming and game theory.
In Proc. of AI, pages 36–47, 2002.

M. D. Vos and D. Vermeir.

Extending answer sets for logic programming agents.
Ann. Math. Artif. Intell., 42(1-3):103–139, 2004.

Yining Wu, Martin Caminada, and Dov M. Gabbay.

Complete extensions in argumentation coincide with 3-valued stable models in logic programming.
Studia Logica, 93(2-3):383–403, 2009.

Claudia Zepeda, José Luis Carballido, Mario Rossainz, and Mauricio Osorio.

Updates based on asp.
In Proc. of MICAI (Special Sessions), pages 63–66, 2010.