Beyond NP: Quantifying over Answer Sets

Giovanni Amendola[1], Francesco Ricca[1], Mirek Truszczynski[2]

[1] Department of Mathematics and Computer Science

[2] Department of Computer Science, University of Kentucky, USA

ICLP 2019
Las Cruces (New Mexico) – September 22, 2019

Outline

Context (1)

Answer Set Programming (ASP) [BET11]

- Declarative programming paradigm
- Non-monotonic reasoning and logic programming
- Roots in Datalog and Nonmonotonic Logic
- Stable model semantics [GL91]
- Robust and efficient systems [GLM+18]
  - DLV [AAC+18], Clingo [GKK+16], ...
- Effective in practical industrial-grade applications [EGL16]

Context (2)

Expressive KR Language

- Default negation, Disjunction, Aggregates, Constraints ...
- Basic ASP models up to Σ_2^P [DEGV01]

  → i.e., problems not (polynomially) translatable to SAT or CSP

Well-known facts about ASP

- Uniform and compact encodings

  → Fixed encoding, instances as facts, inductive definitions
- Modular solutions

  → Generate-Define-Test/Guess&Check methodology [Lif02, EFLP00]
- Compact and elegant modeling of problem in NP

Context (2)

Expressive KR Language

- Default negation, Disjunction, Aggregates, Constraints ...
- Basic ASP models up to Σ_2^P [DEGV01]

  → i.e., problems not (polynomially) translatable to SAT or CSP

Well-known facts about ASP

- Uniform and compact encodings

  → Fixed encoding, instances as facts, inductive definitions
- Modular solutions

  → Generate-Define-Test/Guess&Check methodology [Lif02, EFLP00]
- Compact and elegant modeling of problem in NP

Context (2)

Expressive KR Language

- Default negation, Disjunction, Aggregates, Constraints ...
- Basic ASP models up to Σ_2^P [DEGV01]

  → i.e., problems not (polynomially) translatable to SAT or CSP

Well-known facts about ASP

- Uniform and compact encodings

  → Fixed encoding, instances as facts, inductive definitions

- Modular solutions

  → Generate-Define-Test/Guess&Check methodology [Lif02, EFLP00]

- Compact and elegant modeling of problem in NP

The usual example

Example (3-col)

Problem: Given a graph, assign one color out of 3 colors to each node such that two adjacent nodes have always different colors.
Input: a Graph is represented by *node*(_) and *edge*(_, _).

% guess a coloring for the nodes
$(r) \quad col(X, red) \mid col(X, yellow) \mid col(X, green) \;\text{:-}\; node(X).$

% discard colorings where adjacent nodes have the same color
$(c) \quad \text{:-}\; edge(X, Y), col(X, C), \; col(Y, C).$

% NB: answer sets are subset minimal → only one color per node

"NP-complete problem modeled with only two rules!"

Motivation (1)

What about modeling beyond NP with ASP?

- It is possible...

Motivation (1)

What about modeling beyond NP with ASP?

- It is possible... with unrestricted disjunction [DEGV01]

  $\rightarrow$ Stable model checking in co-NP

Motivation (1)

What about modeling beyond NP with ASP?

- It is possible... with unrestricted disjunction [DEGV01]

  → Stable model checking in co-NP

- Rarely elegant and compact

  → Unless one can find a positive encoding

A rare example...

Example (Strategic Companies is Σ_2^P-complete)

Problem: There are various products, each one is produced by several companies. We now have to sell some companies. What are the minimal sets of strategic companies, such that all products can still be produced? A company also belong to the set, if all its controlling companies belong to it.

Input: *produced_by*(_, _, _) and *controlled_by*(_, _, _, _)

% Guess strategic companies
$strategic(Y) \mid strategic(Z) \coloneq produced\_by(X, Y, Z).$

% Ensure they are strategic
$strategic(W) \coloneq controlled\_by(W, X, Y, Z),$
$\quad strategic(X), strategic(Y), strategic(Z).$

Motivation (1)

What about modeling beyond NP with ASP?

- It is possible... to some extent
- Rarely elegant and compact

  → Unless one can find a positive encoding
  → Well-known strategic companies example

- Generate-define-test approach is no longer sufficient
- Saturation technique [EG95]
  - Exploits the minimality to check "for all" conditions
  - Difficult to use, not intuitive

    → Introduces constraints with no direct relation with the problem

Motivation (1)

What about modeling beyond NP with ASP?

- It is possible... to some extent
- Rarely elegant and compact

  → Unless one can find a positive encoding
  → Well-known strategic companies example

- Generate-define-test approach is no longer sufficient
- Saturation technique [EG95]
  - Exploits the minimality to check “for all” conditions
  - Difficult to use, not intuitive

    → Introduces constraints with no direct relation with the problem

Beyond NP (Saturation)

Example (Quantified Boolean Formulas by [EG95])

Problem: Given a QBF formula $\Phi = \exists X \forall Y \phi(X, Y)$, where ϕ is in 3-DNF form, determine an assignment for X that makes Φ satisfiable.

Input: $conj(X_1, S_{X_1}, X_2, S_{X_2}, X_3, S_{X_3})$ and $exist(X)$, $forall(Y)$

% Guess assignment for X
$asgn(X, true) \lor asgn(X, false) \leftarrow exist(X).$

% Guess assignment for Y
$asgn(Y, true) \lor asgn(Y, false) \leftarrow forall(Y).$

% Saturate Y
$asgn(Y, true) \leftarrow sat, forall(Y).$
$asgn(Y, false) \leftarrow sat, forall(Y).$

% check satisfiability Y
$sat \leftarrow conj(X_1, S_1, X_2, S_2, X_3, S_3), asgn(X_1, S_1), asgn(X_2, S_2), asgn(X_3, S_3).$
$\leftarrow \text{not } sat.$

Motivation and Goals

"Unlike the ease of common ASP modeling, [...] these techniques are rather involved and hardly usable by ASP laymen." [GKS11]

Goals

- Address the shortcomings of ASP beyond NP
- Make modeling natural as for NP

Motivation and Goals

“Unlike the ease of common ASP modeling, [...] these techniques are rather involved and hardly usable by ASP laymen.” [GKS11]

Goals

- Address the shortcomings of ASP beyond NP
- Make modeling natural as for NP

Contributions

1. Design ASP(Q): an extension of ASP with quantifiers
   → Inspired from Quantified Boolean formulas (QBFs)
   → Elegant expansion of ASP with a new form of quantifiers
2. Identify computational properties of ASP(Q)
3. Show the modeling capabilities of ASP(Q)
4. Compare ASP(Q) with alternative approaches
   → QBFs, Stable-unstable [BJT16], Meta-programming [Red17, GKS11],
   → Program transformations [EP06, Red17, FW11], etc.

ASP with Quantifiers: Syntax and Semantics

Definition (ASP with Quantifiers)

An ASP with Quantifiers (ASP(Q)) program Π is of the form:

$$\Box_1 P_1 \; \Box_2 P_2 \; \cdots \; \Box_n P_n : C, \tag{1}$$

$\Box_i \in \{\exists^{st}, \forall^{st}\}$; P_i a program; C a stratified normal program.

Intuitive semantics

Program $\Pi = \exists^{st} P_1 \forall^{st} P_2 \cdots \exists^{st} P_{n-1} \forall^{st} P_n : C$ is coherent if:

"There is an answer set M_1 of P_1 s.t. for each answer set M_2 of $P_2 \cup fix(M_1)$ there is an answer set M_3 of $P_3 \cup fix(M_2)$ such that ... for each answer set M_n of $P_n \cup fix(M_{n-1})$ there is an answer set of $C \cup fix(M_n)$"

where $fix_P(I) = \{a \mid a \in I\} \cup \{\leftarrow a \mid a \in B_P \setminus I\}$. M_1 quantified answer set of Π

Basic Example

Example (Quantified ASP Program)

Let $\Pi = \exists^{st} P_1 \forall^{st} P_2 : C$

- $P_1 = \{a(1) \vee a(2)\}$
- $P_2 = \{b(1) \vee b(2) \leftarrow a(1);\ b(2) \leftarrow a(2)\}$
- $C = \{\leftarrow b(1),\ not\ b(2)\}$

Basic Example

Example (Quantified ASP Program)

Let $\Pi = \exists^{st} P_1 \forall^{st} P_2 : C$

- $P_1 = \{a(1) \vee a(2)\}$
- $P_2 = \{b(1) \vee b(2) \leftarrow a(1);\ b(2) \leftarrow a(2)\}$
- $C = \{\leftarrow b(1),\ not\ b(2)\}$

- P_1 has two answer sets $\{a(1)\}$ and $\{a(2)\}$

Basic Example

Example (Quantified ASP Program)

Let $\Pi = \exists^{st} P_1 \forall^{st} P_2 : C$

- $P_1 = \{a(1) \vee a(2)\}$
- $P_2 = \{b(1) \vee b(2) \leftarrow a(1);\ b(2) \leftarrow a(2)\}$
- $C = \{\leftarrow b(1),\ not\ b(2)\}$

- P_1 has two answer sets $\{a(1)\}$ and $\{a(2)\}$
- $P'_2 = P_2 \cup fix_{P_1}(\{a(1)\})$, and $fix_{P_1}(\{a(1)\}) = \{a(1); \leftarrow a(2)\}$

Basic Example

Example (Quantified ASP Program)

Let $\Pi = \exists^{st} P_1 \forall^{st} P_2 : C$

- $P_1 = \{a(1) \vee a(2)\}$
- $P_2 = \{b(1) \vee b(2) \leftarrow a(1);\ b(2) \leftarrow a(2)\}$
- $C = \{\leftarrow b(1),\ not\ b(2)\}$

- P_1 has two answer sets $\{a(1)\}$ and $\{a(2)\}$
- $P_2' = \{b(1) \vee b(2) \leftarrow a(1);\ b(2) \leftarrow a(2); a(1); \leftarrow a(2)\}$

Basic Example

Example (Quantified ASP Program)

Let $\Pi = \exists^{st} P_1 \forall^{st} P_2 : C$

- $P_1 = \{a(1) \vee a(2)\}$
- $P_2 = \{b(1) \vee b(2) \leftarrow a(1);\ b(2) \leftarrow a(2)\}$
- $C = \{\leftarrow b(1),\ not\ b(2)\}$

- P_1 has two answer sets $\{a(1)\}$ and $\{a(2)\}$
- $P_2' = \{b(1) \vee b(2) \leftarrow a(1);\ b(2) \leftarrow a(2); a(1); \leftarrow a(2)\}$
- P_2' has two answer sets $\{a(1), b(1)\}$ and $\{a(1), b(2)\}$

Basic Example

Example (Quantified ASP Program)

Let $\Pi = \exists^{st} P_1 \forall^{st} P_2 : C$

- $P_1 = \{a(1) \vee a(2)\}$
- $P_2 = \{b(1) \vee b(2) \leftarrow a(1);\ b(2) \leftarrow a(2)\}$
- $C = \{\leftarrow b(1),\ not\ b(2)\}$

- P_1 has two answer sets $\{a(1)\}$ and $\{a(2)\}$
- $P_2' = \{b(1) \vee b(2) \leftarrow a(1);\ b(2) \leftarrow a(2); a(1); \leftarrow a(2)\}$
- P_2' has two answer sets $\{a(1), b(1)\}$ and $\{a(1), b(2)\}$
- **But** $C \cup fix_{P_2'}(\{a(1), b(1)\})$ is not coherent!

Basic Example

Example (Quantified ASP Program)

Let $\Pi = \exists^{st} P_1 \forall^{st} P_2 : C$

- $P_1 = \{a(1) \vee a(2)\}$
- $P_2 = \{b(1) \vee b(2) \leftarrow a(1);\ b(2) \leftarrow a(2)\}$
- $C = \{\leftarrow b(1),\ not\ b(2)\}$

- P_1 has two answer sets $\{a(1)\}$ and $\{a(2)\}$

Basic Example

Example (Quantified ASP Program)

Let $\Pi = \exists^{st} P_1 \forall^{st} P_2 : C$

- $P_1 = \{a(1) \vee a(2)\}$
- $P_2 = \{b(1) \vee b(2) \leftarrow a(1);\ b(2) \leftarrow a(2)\}$
- $C = \{\leftarrow b(1),\ not\ b(2)\}$

- P_1 has two answer sets $\{a(1)\}$ and $\{a(2)\}$
- $P_2' = P_2 \cup fix_{P_1}(\{a(2)\})$, and $fix_{P_1}(\{a(2)\}) = \{a(2); \leftarrow a(1)\}$

Basic Example

Example (Quantified ASP Program)

Let $\Pi = \exists^{st} P_1 \forall^{st} P_2 : C$

- $P_1 = \{a(1) \vee a(2)\}$
- $P_2 = \{b(1) \vee b(2) \leftarrow a(1);\ b(2) \leftarrow a(2)\}$
- $C = \{\leftarrow b(1),\ not\ b(2)\}$

- P_1 has two answer sets $\{a(1)\}$ and $\{a(2)\}$
- P_2' has one answer set $\{a(2), b(2)\}$

Basic Example

Example (Quantified ASP Program)

Let $\Pi = \exists^{st} P_1 \forall^{st} P_2 : C$

- $P_1 = \{a(1) \vee a(2)\}$
- $P_2 = \{b(1) \vee b(2) \leftarrow a(1);\ b(2) \leftarrow a(2)\}$
- $C = \{\leftarrow b(1),\ not\ b(2)\}$

- P_1 has two answer sets $\{a(1)\}$ and $\{a(2)\}$
- P_2' has one answer set $\{a(2), b(2)\}$
- Finally, $\{a(2), b(2)\}$ satisfies C!

Basic Example

Example (Quantified ASP Program)

Let $\Pi = \exists^{st} P_1 \forall^{st} P_2 : C$

- $P_1 = \{a(1) \vee a(2)\}$
- $P_2 = \{b(1) \vee b(2) \leftarrow a(1);\ b(2) \leftarrow a(2)\}$
- $C = \{\leftarrow b(1),\ not\ b(2)\}$

Π is coherent, and $\{a(2)\}$ is a quantified answer set of Π

- P_1 has two answer sets $\{a(1)\}$ and $\{a(2)\}$
- P_2' has one answer set $\{a(2), b(2)\}$
- Finally, $\{a(2), b(2)\}$ satisfies C!

Beyond NP (Saturation vs ASP(Q)) (1)

Example (Quantified Boolean Formulas)

Problem: Given a QBF formula $\Phi = \exists X \forall Y \phi(X, Y)$, where ϕ is in 3-DNF form, determine an assignment for X that makes Φ satisfiable.

Input: $conj(X_1, S_{X_1}, X_2, S_{X_2}, X_3, S_{X_3})$ and $exist(X)$, $forall(Y)$

% Guess assignment for X
$asgn(X, true) \;\vee\; asgn(X, false) \leftarrow exist(X).$

% Guess assignment for Y
$asgn(Y, true) \;\vee\; asgn(Y, false) \leftarrow forall(Y).$

% Saturate Y
$asgn(Y, true) \leftarrow sat, forall(Y).$
$asgn(Y, false) \leftarrow sat, forall(Y).$

% Check satisfiability Y
$sat \leftarrow conj(X_1, S_1, X_2, S_2, X_3, S_3), asgn(X_1, S_1), asgn(X_2, S_2), asgn(X_3, S_3).$
$\leftarrow \text{not } sat.$

Beyond NP (Saturation vs ASP(Q)) (2)

Example (Quantified Boolean Formulas)

Problem: Given a QBF formula $\Phi = \exists X \forall Y \phi(X, Y)$, where ϕ is in 3-DNF form, determine an assignment for X that makes Φ satisfiable.

Input: $conj(X_1, S_{X_1}, X_2, S_{X_2}, X_3, S_{X_3})$ and $exist(X)$, $forall(Y)$

Solution: $\Pi = \exists^{st} P_1 \forall^{st} P_2 : C$ such that:

% Guess assignment for X
$P_1 = \{ \; asgn(X, true) \;\vee\; asgn(X, false) \leftarrow exist(X). \; \}$

% Guess assignment for Y
$P_2 = \{ \; asgn(Y, true) \;\vee\; asgn(Y, false) \leftarrow forall(Y). \; \}$

% Check satisfiability Y
$C = \{$
$sat \leftarrow conj(X_1, S_1, X_2, S_2, X_3, S_3), asgn(X_1, S_1), asgn(X_2, S_2), asgn(X_3, S_3).$
$\leftarrow \text{not } sat.$
$\}$

Beyond NP (Π_2^P-complete)

Example (Quantified Boolean Formulas)

Problem: Given a QBF formula $\Psi = \forall X \exists Y \psi(X, Y)$, where ψ is in 3-CNF form, determine an assignment for X that makes Ψ satisfiable.

Input: $disj(X_1, S_{X_1}, X_2, S_{X_2}, X_3, S_{X_3})$ and $exist(X)$, $forall(Y)$

Solution: $\Pi = \forall^{st} P_1 \exists^{st} P_2 : C$ such that:

% Guess assignment for X
$P_1 = \{\ asgn(X, true) \ \vee\ asgn(X, false) \leftarrow forall(X).\ \}$

% Guess assignment for Y
$P_2 = \{\ asgn(Y, true) \ \vee\ asgn(Y, false) \leftarrow exist(Y).\ \}$

% Check satisfiability Y
$C = \{$
$\leftarrow disj(X_1, S_1, X_2, S_2, X_3, S_3), iasgn(X_1, S_1), iasgn(X_2, S_2), iasgn(X_3, S_3).$
$iasgn(X, false) \text{ :- } asgn(X, true).$
$iasgn(X, true) \text{ :- } asgn(X, false).$
$\}$

Theoretical Results

Theorem (ASP(Q) is a straightforward generalization of ASP)

Let P be an ASP program, and Π the ASP(Q) program $\exists^{st} P : C$. Then,

$$AS(P) = QAS(\Pi).$$

COHERENCE problem: Given Π, decide whether Π is coherent.

Theorem (Complexity)

The COHERENCE *problem is*

(i) *PSPACE-complete, even restricted to normal ASP(Q) programs;*

(ii) Σ_n^P*-complete for n-normal existential ASP(Q) programs;*

(iii) Π_n^P*-complete for n-normal universal ASP(Q) programs.*

Modeling Examples

Min-Max Clique [Ko95]

- Example of Π_2^P-complete problem
- Key role in game theory, optimization and complexity [CDG$^+$95]
- Approach can be adapted to model other minmax problems

Pebbling Number [MC06]

- Mathematical game
- Example of Π_2^P-complete problem

Vapnik-Chervonenkis Dimension (VC-Dimension) [BEHW89]

- Relevant problem in machine learning
- Measures the capacity of a space of functions that can be learned by a statistical classification algorithm
- Example of Σ_3^P-complete problem

Modeling Examples

Min-Max Clique [Ko95]

- Example of Π_2^P-complete problem
- Key role in game theory, optimization and complexity [CDG+95]
- Approach can be adapted to model other minmax problems

Pebbling Number [MC06]

- Mathematical game
- Example of Π_2^P-complete problem

Vapnik-Chervonenkis Dimension (VC-Dimension) [BEHW89]

- Relevant problem in machine learning
- Measures the capacity of a space of functions that can be learned by a statistical classification algorithm
- Example of Σ_3^P-complete problem

Minmax Clique: The Problem

Definition (Minmax Clique)

Given a graph G, sets of indices I and J, a partition $(A_{i,j})_{i \in I, j \in J}$, and an integer k, decide whether

$$\min_{f \in J^I} \max\{|Q| : Q \text{ is a clique of } G_f\} \geq k.$$

J^I is the set of all total functions from I to J, and G_f is the subgraph of G induced by $\bigcup_{i \in I} A_{i,f(i)}$.

In simpler words:

"For each total function $f \in J^I$, there exists a clique c in G_f, such that the size of c is larger than k"

Solution: An ASP(Q) program $\Pi = \forall^{st} P_1 \exists^{st} P_2 : C$.

Minmax Clique: The Solution

"For each total function $f \in J^I$"

$$P_1 = \left\{ \begin{array}{rcll} edge(a,b) & & & \forall (a,b) \in E \\ node(a) & & & \forall a \in N \\ v(i,j,a) & & & \forall i \in I,\ j \in J,\ a \in A_{i,j} \\ setI(X) & \leftarrow & v(X,\_,\_) & \\ setJ(X) & \leftarrow & v(\_,X,\_) & \\ 1\{f(X,Y) : setJ(Y)\}1 & \leftarrow & setI(X) & \end{array} \right\}$$

"There exists a clique c in G_f"

$$P_2 = \left\{ \begin{array}{rcl} inInduced(Z) & \leftarrow & v(X,Y,Z),\ f(X,Y) \\ edgeP(X,Y) & \leftarrow & edge(X,Y),\ inInduced(X), \\ & & inInduced(Y) \\ \{inClique(X)\ :\ inInduced(X)\} & & \\ & \leftarrow & inClique(X),\ inClique(Y), \\ & & not\ edgeP(X,Y) \end{array} \right\}$$

"Such that the size of c is larger than k"

$$C = \{ \leftarrow \ \#\text{count}\{X : inClique(X)\} < k \ \}$$

Pebbling Number: The Problem

Definition (Pebbling Number)

Given a digraph $G = \langle N, E \rangle$ with pebbles placed on (some of) its nodes.

- A pebbling move along (a, b) removes 2 pebbles from a and adds 1 to b
- The Pebbling number $\pi(G)$ is the smallest number of pebbles s.t. for each assignment of k pebbles and for each node w (the target), some sequence of pebbling moves results in a pebble on w

Problem: Is $\pi(G) \leq k$?

In simpler words:

"For each assignment of k pebbles to the nodes of G, and for each target node $t \in N$, there exists a sequence of pebble moves (at most $k - 1$ moves), such that some pebble is on w"

Solution: An ASP(Q) program $\Pi = \forall^{st} P_1 \exists^{st} P_2 : C$.

Pebbling Number: The Solution (1)

"For each assignment of k pebbles to the nodes of G, and for each target node $w \in N$"

$$P_1 =$$

$$\left\{ \begin{array}{rcl} edge(a,b) & & \forall (a,b) \in E \\ node(a) & & \forall a \in N \\ pebble(i) & & \forall i = 0, 1, \ldots, k \\ 1\{onNode(X,N) : pebble(N)\}1 & \leftarrow & node(X) \\ \leftarrow \#\text{sum}\{N, X : onNode(X,N)\} \neq k & & \\ 1\{target(X) : node(X)\}1 & & \end{array} \right\}$$

Pebbling Number: The Solution (2)

"There exists a sequence of pebble moves"

$$P_2 =$$

$$\left\{ \begin{array}{rcl} step(i) & & \forall i = 0, 1, \ldots, k-1 \\ 1\{endstep(S) : step(S)\}1 & & \\ onNode(X, N, 0) & \leftarrow & onNode(X, N) \\ 1\{move(X, Y, S) : edge(X, Y)\}1 & \leftarrow & step(S), endstep(T), 1 \leq S,\ S \leq T \\ & \leftarrow & move(X, Y, S),\ onNode(X, N, S), N < 2 \\ affected(X, S) & \leftarrow & move(X, Y, S) \\ affected(Y, S) & \leftarrow & move(X, Y, S) \\ onNode(X, N-2, S) & \leftarrow & onNode(X, N, S-1), move(X, Y, S) \\ onNode(Y, M+1, S) & \leftarrow & onNode(Y, M, S-1), move(X, Y, S) \\ onNode(X, N, S) & \leftarrow & onNode(X, N, S-1),\ not\ affected(X, S) \end{array} \right\}$$

"Such that some pebble is on w"

$$C = \{ \ \leftarrow \quad target(W), onNode(W, 0, T), endstep(T) \ \}$$

Vapnik-Chervonenkis Dimension: The Problem

Definition (VC Dimension)

Let k be an integer, U a finite set, $\mathcal{C} = \{S_1, \ldots, S_n\} \subseteq 2^U$ a collection of subsets of U represented by a program $P_{\mathcal{C}}$.

Problem: Is there $X \subseteq U$ of size at least k, s.t. for each $S \subseteq X$, there is S_i s.t. $S = S_i \cap X$?
(VC dimension of $\mathcal{C}$, $VC(\mathcal{C})$ is the maximum size of such a set X.)

Solution: An ASP(Q) program $\Pi = \exists^{st} P_1 \forall^{st} P_2 1 \exists^{st} P_3 : C$.

Vapnik-Chervonenkis Dimension: The Solution

"There is $X \subseteq U$ of size at least k"

$$P_1 = \left\{ \begin{array}{rr} inU(x) & \forall x \in U \\ k\{inX(X) : inU(X)\} & \end{array} \right\}$$

"Such that for each $S \subseteq X$"

$$P_2 = \{ \ \{inS(X) : inX(X)\} \ \}$$

"There is S_i"

$$P_3 = P_{\mathcal{C}}$$

"Such that $S = S_i \cap X$"

$$C = \left\{ \begin{array}{rcl} inIntersection(Z) & \leftarrow & true(Z),\ inX(Z) \\ & \leftarrow & inIntersection(Z),\ not\ inS(Z) \\ & \leftarrow & not\ inIntersection(Z),\ inS(Z) \end{array} \right\}$$

ASP(Q) vs Stable-Unstable (1)

Stable-Unstable Models [BJT16]

- Extends ASP up to the second level of PH
- Based on the concept of *parametrized stable model*
- Combined logic program: $\Pi = (P_g, P_t)$
- *"A stable unstable model is a parameterized stable model of P_g, say I, s.t. no parameterized stable model of P_t exists that coincides with I in the intersection of the two signatures"*
- Inspired by an internal working principle of ASP solvers

  $\rightarrow$ P_g guess candidate, P_t performs a co-NP check

- Generalized to capture PH

  $\rightarrow$ Recursive oracle calls

ASP(Q) vs Stable-Unstable: Summary

ASP(Q) vs Stable-Unstable

- Parameters are implicit in ASP(Q)
- Stable-unstable coincides with *existential* ASP(Q)
  - $\rightarrow$ Property holding in all models vs existence of counterexample
- Stable-unstable cannot model Π_k^P
  - $\rightarrow$ Unless the PH collapses
- ASP(Q) modeling often closer to the problem description
  - $\rightarrow$ Complex interplay of recursion, negation and recursive oracles

Conclusion

Contributions

1. A natural solution for modeling beyond NP with ASP
   - ASP(Q) extends ASP via quantifiers over stable models
2. A study of the computational properties of the language
3. Examples to show the modeling capabilities
4. A comparison with alternative approaches

"ASP(Q) models problems in the Polynomial Hierarchy in the same compact and elegant way as ASP models problems in NP"

Future Work

- Implementation: (i) by rewriting in QBF (ii) dedicated solvers

Conclusion

Contributions

1. A natural solution for modeling beyond NP with ASP
   - ASP(Q) extends ASP via quantifiers over stable models
2. A study of the computational properties of the language
3. Examples to show the modeling capabilities
4. A comparison with alternative approaches

"ASP(Q) models problems in the Polynomial Hierarchy in the same compact and elegant way as ASP models problems in NP"

Future Work

- Implementation: (i) by rewriting in QBF (ii) dedicated solvers

Conclusion

Contributions

1. A natural solution for modeling beyond NP with ASP
    - ASP(Q) extends ASP via quantifiers over stable models
2. A study of the computational properties of the language
3. Examples to show the modeling capabilities
4. A comparison with alternative approaches

"ASP(Q) models problems in the Polynomial Hierarchy in the same compact and elegant way as ASP models problems in NP"

Future Work

- Implementation: (i) by rewriting in QBF (ii) dedicated solvers

Acknowledgments

Thanks for your attention!

Questions?

Bonus slides

Bonus Slides

ASP(Q) vs ASP vs QBF

ASP vs ASP(Q)

- ASP(Q) is a natural extension of ASP
- Natural in Σ_2^P with disjunctive positive encodings
- Normal program sufficient to model PH

QBF vs ASP(Q)

- Both extend base language with some form of quantifier

  $\rightarrow$ variable assignments vs answer sets
- Same computational properties
- ASP(Q) supports variables and inductive definitions
- ASP(Q) inherits aggregates, choice rules, strong negation, and disjunction

ASP(Q) vs Stable-Unstable (1)

Stable-Unstable Models [BJT16]

- Extends ASP up to the second level of PH
- Based on the concept of *parametrized stable model*
- Combined logic program: $\Pi = (P_g, P_t)$
- Inspired by an internal working principle of ASP solvers

  $\rightarrow$ P_g guess candidate, P_t performs a co-NP check
- *"A stable unstable model is a parameterized stable model of P_g, say I, s.t. no parameterized stable model of P_t exists that coincides with I in the intersection of the two signatures"*
- Generalized to capture PH

  $\rightarrow$ Recursive oracle calls

ASP(Q) vs Stable-Unstable (2)

Problems in Σ_2^P

- Testing in ASP(Q): "for all stable models of some program, a certain property holds."
- Testing in Stable-Unstable: "there is no stable model of some program s.t. a certain property holds."
- Switching between ASP(Q) and Stable-Unstable is trivial
- Hence, they are on par for modeling problems in Σ_2^P.

Problems in Π_2^P

- Naturally represented in ASP(Q)
- Stable-unstable requires
  - An exponential encoding (quantifier expansion in QBF)
  - Pushing the computation in the oracle (one more quantifier)
- Combined programs model *complements* of Π_2^P problems and not the problems themselves

ASP(Q) vs Stable-Unstable (3)

Modeling problems beyond the second level

- Combined programs resort to a recursive definition

  → Force the programmer to think in terms of nested oracles

  → Recursion and negation make it harder to connect between problem description and oracles
- The interface between natural language problem description and ASP(Q) programs is transparent (as for QBF)

  → Explicitly supported by the quantifiers
- The difficulty of modeling problems in Π_2^P, noted above, appears in the general setting of problems in Π_k^P, for $k \geq 2$

References

[AAC+18] Weronika T. Adrian, Mario Alviano, Francesco Calimeri, Bernardo Cuteri, Carmine Dodaro, Wolfgang Faber, Davide Fuscà, Nicola Leone, Marco Manna, Simona Perri, Francesco Ricca, Pierfrancesco Veltri, and Jessica Zangari. The ASP system DLV: advancements and applications. KI, 32(2-3):177–179, 2018.

[BEHW89] Anselm Blumer, Andrzej Ehrenfeucht, David Haussler, and Manfred K. Warmuth. Learnability and the Vapnik-Chervonenkis dimension. J. ACM, 36(4):929–965, 1989.

[BET11] Gerhard Brewka, Thomas Eiter, and Miroslaw Truszczynski. Answer set programming at a glance. Commun. ACM, 54(12):92–103, 2011.

[BJT16] Bart Bogaerts, Tomi Janhunen, and Shahab Tasharrofi. Stable-unstable semantics: Beyond NP with normal logic programs. TPLP, 16(5-6):570–586, 2016.

References (cont.)

[CDG+95] Feng Cao, Ding-Zhu Du, Biao Gao, Peng-Jun Wan, and Panos M. Pardalos. Minimax Problems in Combinatorial Optimization, pages 269–292. Springer US, Boston, MA, 1995.

[DEGV01] Evgeny Dantsin, Thomas Eiter, Georg Gottlob, and Andrei Voronkov. Complexity and expressive power of logic programming. ACM Comput. Surv., 33(3):374–425, 2001.

[EFLP00] Thomas Eiter, Wolfgang Faber, Nicola Leone, and Gerald Pfeifer. Declarative problem-solving using the dlv system. In Logic-based Artificial Intelligence, pages 79–103. 2000.

[EG95] Thomas Eiter and Georg Gottlob. On the computational cost of disjunctive logic programming: Propositional case. Ann. Math. Artif. Intell., 15(3-4):289–323, 1995.

[EGL16] Esra Erdem, Michael Gelfond, and Nicola Leone. Applications of answer set programming. AI Magazine, 37(3):53–68, 2016.

References (cont.)

[EP06] Thomas Eiter and Axel Polleres. Towards automated integration of guess and check programs in answer set programming: a meta-interpreter and applications. TPLP, 6(1-2):23–60, 2006.

[FW11] Wolfgang Faber and Stefan Woltran. Manifold answer-set programs and their applications. In Logic Programming, Knowledge Representation, and Nonmonotonic Reasoning, volume 6565 of LNCS, pages 44–63, 2011.

[GKK+16] Martin Gebser, Roland Kaminski, Benjamin Kaufmann, Max Ostrowski, Torsten Schaub, and Philipp Wanko. Theory solving made easy with clingo 5. In ICLP (Technical Communications), volume 52 of OASICS, pages 2:1–2:15. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2016.

[GKS11] Martin Gebser, Roland Kaminski, and Torsten Schaub. Complex optimization in answer set programming. TPLP, 11(4-5):821–839, 2011.

References (cont.)

[GL91] Michael Gelfond and Vladimir Lifschitz. Classical negation in logic programs and disjunctive databases. New Generation Comput., 9(3/4):365–386, 1991.

[GLM+18] Martin Gebser, Nicola Leone, Marco Maratea, Simona Perri, Francesco Ricca, and Torsten Schaub. Evaluation techniques and systems for answer set programming: a survey. In IJCAI, pages 5450–5456. ijcai.org, 2018.

[Ko95] Chih-Long Ko, Ker-Iand Lin. On the Complexity of Min-Max Optimization Problems and their Approximation, pages 219–239. Springer US, Boston, MA, 1995.

[Lif02] Vladimir Lifschitz. Answer set programming and plan generation. Artif. Intell., 138(1-2):39–54, 2002.

[MC06] Kevin Milans and Bryan Clark. The complexity of graph pebbling. SIAM J. Discret. Math., 20(3):769–798, March 2006.

References (cont.)

[Red17] Christoph Redl. Explaining inconsistency in answer set programs and extensions. In LPNMR, volume 10377 of LNCS, pages 176–190. Springer, 2017.