Beyond NP: Quantifying over Answer Sets

Giovanni Amendola\textsuperscript{1}, Francesco Ricca\textsuperscript{1}, Mirek Truszczynski\textsuperscript{2}

\textsuperscript{1} Department of Mathematics and Computer Science
\textsuperscript{2} Department of Computer Science, University of Kentucky, USA

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

1. Introduction
2. ASP with Quantifiers
3. Related Work
**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]
Expressive KR Language

- Default negation, Disjunction, Aggregates, Constraints ...
- Basic ASP models up to $\Sigma^P_2$ [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
Expressive KR Language

- Default negation, Disjunction, Aggregates, Constraints ...
- Basic ASP models up to $\Sigma_2^P$ [DEGV01]
  $\rightarrow$ i.e., problems not (polynomially) translatable to SAT or CSP

Well-known facts about ASP

- Uniform and compact encodings
  $\rightarrow$ Fixed encoding, instances as facts, inductive definitions
- Modular solutions
  $\rightarrow$ Generate-Define-Test/Guess&Check methodology [Lif02, EFLP00]
- Compact and elegant modeling of problem in NP
Expressive KR Language
- Default negation, Disjunction, Aggregates, Constraints ...
- Basic ASP models up to $\Sigma_2^P$ [DEGV01]
  \[ \rightarrow \text{i.e., problems not (polynomially) translatable to SAT or CSP} \]

Well-known facts about ASP
- Uniform and compact encodings
  \[ \rightarrow \text{Fixed encoding, instances as facts, inductive definitions} \]
- Modular solutions
  \[ \rightarrow \text{Generate-Define-Test/Guess&Check methodology [Lif02, EFLP00]} \]
- 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
\( (r) \col(X, \text{red}) \mid \col(X, \text{yellow}) \mid \col(X, \text{green}) \vdash \text{node}(X). \)

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

% NB: answer sets are subset minimal \( \rightarrow \) only one color per node

“NP-complete problem modeled with only two rules!”
What about modeling **beyond NP** with ASP?

- It is possible...
What about modeling **beyond NP** with ASP?

- It is possible... with *unrestricted disjunction* [DEGV01]
  - Stable model checking in co-NP
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 $\Sigma^P_2$-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
\[
\text{strategic}(Y) \mid \text{strategic}(Z) \leftarrow produced\_by(X, Y, Z).
\]

% Ensure they are strategic
\[
\text{strategic}(W) \leftarrow controlled\_by(W, X, Y, Z),
\text{strategic}(X), \text{strategic}(Y), \text{strategic}(Z).
\]
Motivation

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
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
Example (Quantified Boolean Formulas by [EG95])

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

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

% Guess assignment for $X$

$$\text{asgn}(X, \text{true}) \lor \text{asgn}(X, \text{false}) \leftarrow \text{exist}(X).$$

% Guess assignment for $Y$

$$\text{asgn}(Y, \text{true}) \lor \text{asgn}(Y, \text{false}) \leftarrow \text{forall}(Y).$$

% Saturate $Y$

$$\text{asgn}(Y, \text{true}) \leftarrow \text{sat}, \text{forall}(Y).$$

$$\text{asgn}(Y, \text{false}) \leftarrow \text{sat}, \text{forall}(Y).$$

% check satisfiability $Y$

$$\text{sat} \leftarrow \text{conj}(X_1, S_1, X_2, S_2, X_3, S_3), \text{asgn}(X_1, S_1), \text{asgn}(X_2, S_2), \text{asgn}(X_3, S_3).$$

$$\leftarrow \text{not} \text{sat}.$$
“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
“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 $\Pi$ is of the form:

$$\square_1 P_1 \square_2 P_2 \cdots \square_n P_n : C,$$

where $\square_i \in \{\exists^{st}, \forall^{st}\}$; $P_i$ 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 $P_2 \cup \text{fix}(M_1)$ there is an answer set $M_3$ of $P_3 \cup \text{fix}(M_2)$ such that ... for each answer set $M_n$ of $P_n \cup \text{fix}(M_{n-1})$ there is an answer set of $C \cup \text{fix}(M_n)$”

where $\text{fix}_P(I) = \{a \mid a \in I\} \cup \{\leftarrow a \mid a \in B_P \setminus I\}$. $M_1$ quantified answer set of $\Pi$.
Example (Quantified ASP Program)

Let $\Pi = \exists^s t P_1 \land^s t P_2 : C$

- $P_1 = \{a(1) \lor a(2)\}$
- $P_2 = \{b(1) \lor b(2) \leftarrow a(1); b(2) \leftarrow a(2)\}$
- $C = \{\leftarrow b(1), \neg b(2)\}$
Example (Quantified ASP Program)

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

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

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

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

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

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

Example (Quantified ASP Program)

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

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

- $P_1$ has two answer sets $\{a(1)\}$ and $\{a(2)\}$
- $P'_2 = \{b(1) \lor b(2) \leftarrow a(1); b(2) \leftarrow a(2); a(1); \leftarrow a(2)\}$
Example (Quantified ASP Program)

Let \( \Pi = \exists^s P_1 \forall^s P_2 : C \)

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

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

Example (Quantified ASP Program)

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

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

- $P_1$ has two answer sets $\{a(1)\}$ and $\{a(2)\}$
- $P'_2 = \{b(1) \lor b(2) \leftarrow a(1); b(2) \leftarrow a(2); a(1); \leftarrow a(2)\}$
- $P'_2$ has two answer sets $\{a(1), b(1)\}$ and $\{a(1), b(2)\}$
- **But** $C \cup \text{fix}_P'\{\{a(1), b(1)\}\}$ is not coherent!
### Example (Quantified ASP Program)

Let $\Pi = \exists^s P_1 \land^s P_2 : C$

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

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

Example (Quantified ASP Program)

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

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

- $P_1$ has two answer sets $\{a(1)\}$ and $\{a(2)\}$
- $P'_2 = P_2 \cup \text{fix}_{P_1}(\{a(2)\})$, and $\text{fix}_{P_1}(\{a(2)\}) = \{a(2); \leftarrow a(1)\}$
Example (Quantified ASP Program)

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

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

- $P_1$ 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 $\Pi = \exists^st P_1 \forall^st P_2 : C$

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

- $P_1$ 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 $\Pi = \exists^s P_1 \forall^s P_2 : C$

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

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

- $P_1$ 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 $C$!
Example (Quantified Boolean Formulas)

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

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

% Guess assignment for $X$
\[
\text{asgn}(X, \text{true}) \lor \text{asgn}(X, \text{false}) \leftarrow \text{exist}(X).
\]

% Guess assignment for $Y$
\[
\text{asgn}(Y, \text{true}) \lor \text{asgn}(Y, \text{false}) \leftarrow \text{forall}(Y).
\]

% Saturate $Y$
\[
\text{asgn}(Y, \text{true}) \leftarrow \text{sat}, \text{forall}(Y).
\]
\[
\text{asgn}(Y, \text{false}) \leftarrow \text{sat}, \text{forall}(Y).
\]

% Check satisfiability $Y$
\[
\text{sat} \leftarrow \text{conj}(X_1, \text{S}_1, X_2, \text{S}_2, X_3, \text{S}_3), \text{asgn}(X_1, \text{S}_1), \text{asgn}(X_2, \text{S}_2), \text{asgn}(X_3, \text{S}_3). \leftarrow \text{not sat}.
\]
Example (Quantified Boolean Formulas)

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

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

**Solution:** $\Pi = \exists^* P_1 \forall^* P_2 : C$ such that:

- **Guess assignment for $X$**
  $P_1 = \{ \text{asgn}(X, \text{true}) \lor \text{asgn}(X, \text{false}) \leftarrow \text{exist}(X) \}$

- **Guess assignment for $Y$**
  $P_2 = \{ \text{asgn}(Y, \text{true}) \lor \text{asgn}(Y, \text{false}) \leftarrow \text{forall}(Y) \}$

- **Check satisfiability $Y$**
  $C = \{\text{sat} \leftarrow \text{conj}(X_1, S_{X_1}, X_2, S_{X_2}, X_3, S_{X_3}), \text{asgn}(X_1, S_1), \text{asgn}(X_2, S_2), \text{asgn}(X_3, S_3), \not \text{sat} \}$
Example (Quantified Boolean Formulas)

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

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

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

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

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

% Check satisfiability $Y$
$C = \{$
$\leftarrow \text{disj}(X_1, S_{X_1}, X_2, S_{X_2}, X_3, S_{X_3}), \text{iasgn}(X_1, S_{X_1}), \text{iasgn}(X_2, S_{X_2}), \text{iasgn}(X_3, S_{X_3}).$
$\text{iasgn}(X, false) \leftarrow \text{asgn}(X, true).$
$\text{iasgn}(X, true) \leftarrow \text{asgn}(X, false).$
$\}$
Theoretical Results

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

Let $P$ be an ASP program, and $\Pi$ the ASP(Q) program $\exists^* P : C$. Then,

$$\text{AS}(P) = \text{QAS}(\Pi).$$

COHERENCE problem: Given $\Pi$, decide whether $\Pi$ is coherent.

Theorem (Complexity)

The COHERENCE problem is

(i) $\text{PSPACE}$-complete, even restricted to normal ASP(Q) programs;

(ii) $\Sigma^P_n$-complete for $n$-normal existential ASP(Q) programs;

(iii) $\Pi^P_n$-complete for $n$-normal universal ASP(Q) programs.
Modeling Examples

**Min-Max Clique** [Ko95]
- Example of $\Pi_2^P$-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 $\Pi_2^P$-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 $\Sigma_3^P$-complete problem
Modeling Examples

**Min-Max Clique** [Ko95]
- Example of $\Pi^P_2$-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 $\Pi^P_2$-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 $\Sigma^P_3$-complete 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 $\Pi = \forall^{st} P_1 \exists^{st} P_2 : C$. 
“For each total function $f \in J^I$”

$P_1 = \left\{ \begin{array}{l}
\text{edge}(a, b) \quad \forall (a, b) \in E \\
\text{node}(a) \quad \forall a \in N \\
v(i, j, a) \quad \forall i \in I, \ j \in J, \ a \in A_{i,j} \\
\text{set}_I(X) \leftarrow v(X, \_ , \_ ) \\
\text{set}_J(X) \leftarrow v(\_ , X, \_ ) \\
1\{f(X, Y) : \text{set}_J(Y)\}1 \leftarrow \text{set}_I(X)
\end{array} \right\}$

“There exists a clique $c$ in $G_f$”

$P_2 = \left\{ \begin{array}{l}
in\text{Induced}(Z) \leftarrow v(X, Y, Z), \ f(X, Y) \\
\text{edge}_P(X, Y) \leftarrow \text{edge}(X, Y), \ in\text{Induced}(X), \ in\text{Induced}(Y) \\
\{\text{inClique}(X) : in\text{Induced}(X)\} \leftarrow in\text{Clique}(X), \ in\text{Clique}(Y), \ not \text{edge}_P(X, Y)
\end{array} \right\}$

“Such that the size of $c$ is larger than $k$”

$C = \{ \leftarrow \#\text{count}\{X : \text{inClique}(X)\} < k \ \}$
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 $\Pi = \forall^{st} P_1 \exists^{st} P_2 : C$. 

G. Amendola, F. Ricca, M. Truszczynski

Beyond NP: Quantifying over Answer Sets
“For each assignment of \( k \) pebbles to the nodes of \( G \), and for each target node \( w \in N \)”

\[
P_1 = \left\{ \begin{array}{l}
\text{edge}(a, b) \quad \forall (a, b) \in E \\
\text{node}(a) \quad \forall a \in N \\
\text{pebble}(i) \quad \forall i = 0, 1, \ldots, k \\
1\{\text{onNode}(X, N) : \text{pebble}(N)\} \leftarrow \text{node}(X) \\
\leftarrow \#\text{sum}\{N, X : \text{onNode}(X, N)\} \neq k \\
1\{\text{target}(X) : \text{node}(X)\} \\
\end{array} \right\}
\]
“There exists a sequence of pebble moves”

\[
P_2 = \\
\begin{cases}
\text{step}(i) & \forall i = 0, 1, \ldots, k - 1 \\
1\{\text{endstep}(S) : \text{step}(S)\} & 1 \\
\text{onNode}(X, N, 0) & \leftarrow \text{onNode}(X, N) \\
1\{\text{move}(X, Y, S) : \text{edge}(X, Y)\} & \leftarrow \text{step}(S), \text{endstep}(T), 1 \leq S, S \leq T \\
& \leftarrow \text{move}(X, Y, S), \text{onNode}(X, N, S), N < 2 \\
\text{affected}(X, S) & \leftarrow \text{move}(X, Y, S) \\
\text{affected}(Y, S) & \leftarrow \text{move}(X, Y, S) \\
\text{onNode}(X, N - 2, S) & \leftarrow \text{onNode}(X, N, S - 1), \text{move}(X, Y, S) \\
\text{onNode}(Y, M + 1, S) & \leftarrow \text{onNode}(Y, M, S - 1), \text{move}(X, Y, S) \\
\text{onNode}(X, N, S) & \leftarrow \text{onNode}(X, N, S - 1), \text{not affected}(X, S)
\end{cases}
\]

“Such that some pebble is on \( w \)”

\[
C = \left\{ \leftarrow \text{target}(W), \text{onNode}(W, 0, T), \text{endstep}(T) \right\}
\]
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 \( P_C \).

**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 \( \Pi = \exists^* P_1 \forall^* P_2 \exists^* P_3 : C \).
Vapnik-Chervonenkis Dimension: The Solution

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

\[ P_1 = \left\{ k \{ \text{in}U(X) : \text{in}U(X) \} \mid \forall x \in U \right\} \]

“Such that for each $S \subseteq X$”

\[ P_2 = \{ \{ \text{in}S(X) : \text{in}X(X) \} \mid \forall x \in U \right\} \]

“There is $S_i$”

\[ P_3 = P_C \]

“Such that $S = S_i \cap X$”

\[ C = \left\{ \text{inIntersection}(Z) \iff \text{true}(Z), \text{in}X(Z) \right. \]
\[ \left. \iff \text{inIntersection}(Z), \text{not in}S(Z) \right. \]
\[ \iff \text{not inIntersection}(Z), \text{in}S(Z) \right\} \]
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)
  - Property holding in all models vs existence of counterexample
- Stable-unstable cannot model $\Pi^P_k$
  - Unless the PH collapses
- ASP(Q) modeling often closer to the problem description
  - 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
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
Thanks for your attention!

Questions?
Bonus Slides
**ASP(Q) vs ASP vs QBF**

**ASP vs ASP(Q)**
- ASP(Q) is a natural extension of ASP
- Natural in $\Sigma^P_2$ 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
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

Problems in $\Sigma_2^P$

- 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 $\Sigma_2^P$.

Problems in $\Pi_2^P$

- 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 $\Pi_2^P$ problems and not the problems themselves
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 $\Pi^P_2$, noted above, appears in the general setting of problems in $\Pi^P_k$, for $k \geq 2$
References


References (cont.)


References (cont.)


References (cont.)