Logic Programming for Knowledge Representation

Mirosław Truszczyński

Department of Computer Science
University of Kentucky

September 10, 2007

McCarthy and Hayes on AI, 1969

- [...] intelligence has two parts, which we shall call the epistemological and the heuristic.
  The epistemological part is the representation of the world in such a form that the solution of problems follows from the facts expressed in the representation. The heuristic part is the mechanism that on the basis of the information solves the problem and decides what to do.

- Epistemological part ⟶ modeling – knowledge representation
- Heuristic part ⟶ automated reasoning – search for models or proofs
- There may be more to AI now – but KRR remains its core
- How to approach it? Use classical logic — it is “descriptively universal” and reasoning can be automated
  Early proposal of McCarthy

McCarthy and Hayes on AI, 1969

- [...] intelligence has two parts, which we shall call the epistemological and the heuristic.
  The epistemological part is the representation of the world in such a form that the solution of problems follows from the facts expressed in the representation. The heuristic part is the mechanism that on the basis of the information solves the problem and decides what to do.

- Epistemological part ⟶ modeling – knowledge representation
- Heuristic part ⟶ automated reasoning – search for models or proofs
- There may be more to AI now – but KRR remains its core
- How to approach it? Use classical logic — it is "descriptively universal" and reasoning can be automated
  Early proposal of McCarthy

Challenges

With FOL – things are not so easy

- Incomplete information
  (new information may invalidate earlier inferences – defeasible reasoning)
- Qualification problem
  (we do not check for potato in tailpipe before starting the engine)
- Frame problem
  (moving an object does not change its color)
- Rules with exceptions (defaults)
- Definitions – most notably *inductive* definitions

Non-monotonic logics

Proposed in response to challenges of KRR

- Language of logic with non-classical semantics
- Model preference
  - circumscription *(McCarthy 1977)*
- Fixpoint conditions defining belief sets
  - default logic *(Reiter 1980)*
  - autoepistemic logic *(Moore 1984)*
  - logic programming with stable-model semantics (more managable fragment of default logic) *(Gelfond-Lifschitz, 1988)*
  - ID-logic *(Denecker 1998, 2000; Denecker-Ternovska 2004)*
- The last two stem directly from LP research
- Emphasize both modeling and reasoning
- Main focus of this tutorial

About the tutorial

Prerequisites (will not spend much time on them)

- Answer-set semantics (Gelfond-Lifschitz, 1988)
- Well-founded semantics (Van Gelder-Ross-Schlipf, 1989)
- Answer-set programming (ASP) – logic programs encode problems so that answer sets encode solutions
  (Niemelä, 1999; Marek, T_ 1999)

Objectives

- Discuss some recent advances in ASP motivated by KRR needs
  - constraints
  - modularity
  - tools
- Present ID-logic as an alternative approach to KRR based in logic programming

About the tutorial

Prerequisites (will not spend much time on them)

- Answer-set semantics (Gelfond-Lifschitz, 1988)
- Well-founded semantics (Van Gelder-Ross-Schlipf, 1989)
- Answer-set programming (ASP) – logic programs encode problems so that answer sets encode solutions
  (Niemelä, 1999; Marek, T_ 1999)

Objectives

- Discuss some recent advances in ASP motivated by KRR needs
  - constraints
  - modularity
  - tools
- Present ID-logic as an alternative approach to KRR based in logic programming

Constraints and aggregates

Constraints and aggregates

Common in problems arising in practical applications

- *n*-queens problem
  - assign n queens to squares on the $n \times n$-chessboard so that
  - there is exactly one queen in each row
  - there is exactly one queen in each column
  - there is at most one queen in each diagonal
- With means to model constraints on sets, logic programs are shorter, more direct and easier to process
  - $idx(0).\ idx(1).\ \ldots\ idx(n-1).$
  - $1\{q(I,J) : idx(J)\}1 \leftarrow idx(I).$
  - $1\{q(I,J) : idx(I)\}1 \leftarrow idx(J).$
  - $\{q(I,I+R) : idx(I), I+R \leq n-1\}1 \leftarrow idx(R).$
  - $\ldots$

More examples ...

Constraints

- Constraints
  - exactly three atoms in a set are true
  - the total weight of atoms true in a set is at most 7
  - the average weight of atoms true in a set is at least 21

Constraint clauses or rules

- If the average weight of atoms true in a set is at most 4, then at least 2 atoms in another set must be true

More examples ...

Constraints

- Constraints
  - exactly three atoms in a set are true
  - the total weight of atoms true in a set is at most 7
  - the average weight of atoms true in a set is at least 21

Constraint clauses or rules

- If the average weight of atoms true in a set is at most 4, then at least 2 atoms in another set must be true

Not surprisingly then ...

Extending normal logic programming with constraints on sets ...

- Received much attention
  - Simons, Niemelä, and Soininen (weight atoms in *smodels*)
  - Dell'Armi, Faber, Ielpa, Leone, Pfeifer (aggregates for *dlv*)
  - Denecker, Pelov, and Bruynooghe (aggregates for LP - approximation theory)

Constraints and aggregates

Our goals ...

- A general theory of logic programs with abstract constraints
  - important: constraints may also appear in the heads
  - supported by *smodels*!
- A uniform foundation for extensions of logic programming with high-level aggregates
  - Marek, Niemelä, and T_ (monotone abstract constraints)
  - Liu, T_ (convex abstract constraints)
  - Marek, Remmel (arbitrary abstract constraints)
  - Son, Pontelli, Tu (arbitrary abstract constraints)
  - Liu, Son, Pontelli, T_ (arbitrary abstract constraints, later today)
- A clear link to *normal* logic programming

Basic concepts

Syntax – abstract constraints

- Propositional case; fixed set of propositional atoms At
- Abstract constraint – a pair $A = (X, C)$
  - $X \subseteq At$ – domain
  - $C \subseteq \mathcal{P}(X)$ – satisfiers
  - A_{dom} and A_{sat}
- B is the “negation” of A, if
  - $B_{dom} = A_{dom}$
  - $B_{sat} = \{X \subseteq B_{dom} \mid X \notin A_{sat}\}$
  - $Comp(A)$

Basic concepts

Syntax – constraint rules and programs

- $r = \quad H \leftarrow A_1, \ldots, A_m$

  H, A_i – abstract constraints
- $hd(r)$, $bd(r)$
- Headset of a rule r – $hset(r) = H_{dom}$
- Constraint programs – sets of constraint rules
- Headset of a program P – $hset(P) = \bigcup_{r \in P} hset(r)$

Basic concepts

Satisfiability

- $M \models A$ if $M \cap A_{dom} \in A_{sat}$
- $M \models (H \leftarrow A_1, \ldots, A_m)$ if:
  - $M \models H$, or
  - $M \not\models A_i$, for some i, $1 \leq i \leq m$
- $M \models P$
- r is M-applicable if $M \models bd(r)$
- $P(M)$ – the set of all M-applicable rules in P

Constraint rules serve as “inference rules”

Intended interpretation

- If the body of a rule has been derived, the rule provides support for deriving a set of atoms that satisfies its head
  - $2\{x, y, z\} \leftarrow \{a = 1, b = 2, c = 1, d = 1, e = 3\}4$
  - provides support for $\{x, y\}$, $\{x, z\}$, $\{y, z\}$ and $]\{x, y, z\}$
- Models of a constraint program do not correspond to sets of atoms that can be “derived” from (“justified” on the basis of) the program
- Restricted classes of models needed
- Operator-based approach (van Emden-Kowalski, Apt, Fitting) applies
- It exploits properties of operators on complete lattices (of interpretations) and of fixpoints of these operators
- There are essential differences – nondeterminism

One-step provability

Operator T_P^{nd}

- If an interpretation M satisfies the body of a rule r, then r supports any set of atoms M' such that
  - $M' \subseteq hset(r)$
    r provides no support for atoms that do not appear in the head of r
  - M' satisfies the head of r
    since r "fires", the constraint imposed by the head of r must hold
- A set M' is *nondeterministically one-step provable* from M by a constraint program P, if
  - $M' \subseteq hset(P(M))$ and $M' \models hd(r)$, for every $r \in P(M)$
- The *nondeterministic one-step provability operator* — $T_P^{nd} : \mathcal{P}(At) \to \mathcal{P}(\mathcal{P}(At))$
- $T_P^{nd}(M)$ – consists of all sets that are nondeterministically one-step provable from M by means of P

T_P^{nd} and models

For a normal logic program P

- M is a model of P iff $T_P(M) \subseteq M$
- Models of P = prefixpoints of T_P w.r.t. $\subseteq$

For a constraint program P

- M is a model of a constraint program P iff there is $M' \in T_P^{nd}(M)$ such that $M' \subseteq M$
- *Smyth* preorder
  - $\mathcal{A}, \mathcal{B}$ – families of sets
  - $\mathcal{A} \preceq_{Smyth} \mathcal{B}$ if for every $B \in \mathcal{B}$ there is $A \in \mathcal{A}$ such that $A \subseteq B$
- $T_P^{nd}(M) \preceq_{Smyth} \{M\}$
- Models = prefixpoints of T_P^{nd} w.r.t. $\preceq_{Smyth}$

T_P^{nd} and models

For a normal logic program P

- M is a model of P iff $T_P(M) \subseteq M$
- Models of P = prefixpoints of T_P w.r.t. $\subseteq$

For a constraint program P

- M is a model of a constraint program P iff there is $M' \in T_P^{nd}(M)$ such that $M' \subseteq M$
- *Smyth* preorder
  - $\mathcal{A}, \mathcal{B}$ – families of sets
  - $\mathcal{A} \preceq_{Smyth} \mathcal{B}$ if for every $B \in \mathcal{B}$ there is $A \in \mathcal{A}$ such that $A \subseteq B$
- $T_P^{nd}(M) \preceq_{Smyth} \{M\}$
- Models = prefixpoints of T_P^{nd} w.r.t. $\preceq_{Smyth}$

Supported models

For a normal logic program P

- M is a supported model if $T_P(M) = M$
- Fixpoint of T_P

For a constraint logic program P

- We define M to be a *supported* model of P if $M \in T_P^{nd}(M)$
  - fixpoint of a nondeterministic operator
- Supported models are indeed models: $M \in T_P^{nd}(M)$ implies $T_P^{nd}(M) \preceq_{Smyth} \{M\}$
- $1\{a\} \leftarrow 1\{a\}$ $\{a\}$ is supported but not "intened"
- Stable models?
  - General case (arbitrary constraints)– far from trivial (later today)

Supported models

For a normal logic program P

- M is a supported model if $T_P(M) = M$
- Fixpoint of T_P

For a constraint logic program P

- We define M to be a *supported* model of P if $M \in T_P^{nd}(M)$
  - fixpoint of a nondeterministic operator
- Supported models are indeed models: $M \in T_P^{nd}(M)$ implies $T_P^{nd}(M) \preceq_{Smyth} \{M\}$
- $1\{a\} \leftarrow 1\{a\}$ $\{a\}$ is supported but not “intened”
- Stable models?
  - General case (arbitrary constraints)– far from trivial (later today)

Special cases

Monotone constraints

- A – monotone (upward closed) if for every $X \in A_{sat}$ and for every Y such that $X \subseteq Y \subseteq A_{dom}$, $Y \in A_{sat}$
- $(\{a\}, \{\{a\}\})$ – written as a
  - "At least k" weight constraint, relative a weight function w from atoms into *non-negative* reals: $(X, \{Y \mid Y \subseteq X;\ k \leq \sum_{x \in Y} w(x)\})$

Antimonotone constraints

- A – antimonotone (downward closed) if for every $X \in A_{sat}$ and for every Y such that $Y \subseteq X$, $Y \in A_{sat}$
- $(\{a\}, \{\emptyset\})$ – written as **not**(a)

- $(\emptyset, \emptyset)$ both monotone and antimnonotone, written as $\bot$
- A monotone if and only if $Comp(A)$ antimonotone
- A antimonotone if and only if $Comp(A)$ monotone

Special cases

Monotone constraints

- A – monotone (upward closed) if for every $X \in A_{sat}$ and for every Y such that $X \subseteq Y \subseteq A_{dom}$, $Y \in A_{sat}$
- $(\{a\}, \{\{a\}\})$ – written as a
  - "At least k" weight constraint, relative a weight function w from atoms into *non-negative* reals: $(X, \{Y \mid Y \subseteq X;\ k \leq \sum_{x \in Y} w(x)\})$

Antimonotone constraints

- A – antimonotone (downward closed) if for every $X \in A_{sat}$ and for every Y such that $Y \subseteq X$, $Y \in A_{sat}$
- $(\{a\}, \{\emptyset\})$ – written as **not**(a)

- $(\emptyset, \emptyset)$ both monotone and antimnonotone, written as $\perp$
- A monotone if and only if $Comp(A)$ antimonotone
- A antimonotone if and only if $Comp(A)$ monotone

Special cases

Monotone constraints

- A – monotone (upward closed) if for every $X \in A_{sat}$ and for every Y such that $X \subseteq Y \subseteq A_{dom}$, $Y \in A_{sat}$
- $(\{a\}, \{\{a\}\})$ – written as a
  - "At least k" weight constraint, relative a weight function w from atoms into *non-negative* reals: $(X, \{Y \mid Y \subseteq X;\ k \leq \sum_{x \in Y} w(x)\})$

Antimonotone constraints

- A – antimonotone (downward closed) if for every $X \in A_{sat}$ and for every Y such that $Y \subseteq X$, $Y \in A_{sat}$
- $(\{a\}, \{\emptyset\})$ – written as **not**(a)

- $(\emptyset, \emptyset)$ both monotone and antimnonotone, written as $\perp$
- A monotone if and only if $Comp(A)$ antimonotone
- A antimonotone if and only if $Comp(A)$ monotone

Special cases

Convex constraints

- A – convex if for every X, Y, Z such that $X, Y \in A_{sat}$ and $X \subseteq Z \subseteq Y$, $Z \in A_{sat}$

Upward (monotone) and downward (antimonotone) closure

- A – convex
- $A^m_{dom} = A_{dom}$, $A^m_{sat} = \{X \subseteq A^m_{dom} \mid Y \subseteq X \text{ for some } Y \in A_{sat}\}$
  - A^m – monotone
- $A^a_{dom} = A_{dom}$, $A^a_{sat} = \{X \mid X \subseteq Y \text{ for some } Y \in A_{sat}\}$
  - A^a – antimonotone
- $M \models A$ iff $M \models A^m$ and $M \models A^a$
- "$A = A^m \wedge A^a$" or $A = A^m, A^a$

From now on monotone and convex constraints only

Special cases

Convex constraints

- A – convex if for every X, Y, Z such that $X, Y \in A_{sat}$ and $X \subseteq Z \subseteq Y$, $Z \in A_{sat}$

Upward (monotone) and downward (antimonotone) closure

- A – convex
- $A^m_{dom} = A_{dom}$, $\quad A^m_{sat} = \{X \subseteq A^m_{dom} \mid Y \subseteq X$ for some $Y \in A_{sat}\}$
  - A^m – monotone
- $A^a_{dom} = A_{dom}$, $\quad A^a_{sat} = \{X \mid X \subseteq Y$ for some $Y \in A_{sat}\}$
  - A^a – antimonotone
- $M \models A$ iff $M \models A^m$ and $M \models A^a$
- "$A = A^m \wedge A^a$" or $A = A^m, A^a$

From now on monotone and convex constraints only

Special cases

Convex constraints

- A – convex if for every X, Y, Z such that $X, Y \in A_{sat}$ and $X \subseteq Z \subseteq Y$, $Z \in A_{sat}$

Upward (monotone) and downward (antimonotone) closure

- A – convex
- $A^m_{dom} = A_{dom}$, $\quad A^m_{sat} = \{X \subseteq A^m_{dom} \mid Y \subseteq X \text{ for some } Y \in A_{sat}\}$
  - A^m – monotone
- $A^a_{dom} = A_{dom}$, $\quad A^a_{sat} = \{X \mid X \subseteq Y \text{ for some } Y \in A_{sat}\}$
  - A^a – antimonotone
- $M \models A$ iff $M \models A^m$ and $M \models A^a$
- "$A = A^m \wedge A^a$" or $A = A^m, A^a$

From now on monotone and convex constraints only

Computations

Monotone (“Horn”) constraint programs

- All constraints monotone
  - if heads of all rules in P have satisfiers, P has models
  - otherwise, it is not guaranteed – inconsistent monotone constraint programs:
    - $2\{a\} \leftarrow 1\{a, b, c\}$
    - $1\{b, c\}$
- Once a rule is applicable w.r.t M, it remains applicable w.r.t every superset of M
  - key property of normal Horn logic programs behind the bottom-up computation

Computations

- Assume finite domains for constraints (can be dropped)
- A *P-computation* is a sequence $(X_n)_{n=0,1,\ldots}$ such that $X_0 = \emptyset$ and, for every non-negative integer n:
  - $X_n \subseteq X_{n+1}$, and
  - $X_{n+1} \in T_P^{nd}(X_n)$
- $\bigcup_{n=0}^{\infty} X_n$ — the *result* of the computation $t = (X_n)_{n=0,1,\ldots}$ (notation — R_t)
- Results of computations are models (in fact, supported models)
- Converse does not hold — not all supported models are results of computations: $1\{a\} \leftarrow 1\{a\}$
- Only consistent monotone programs have computations

$$2\{a\} \leftarrow 1\{a,b,c\}$$
$$1\{b,c\}$$

Computations

Stable models of monotone constraint programs

- Results of P-computations — *derivable* models of P
- Generalization of stable models of Horn programs
- However for
  - $2\{a, b, d\} \leftarrow 1\{a, b, c\}$
  - $1\{b, c\}$

  $\{b, a, d\}$, $\{b, c, d\}$, $\{b, c, a, d\}$, ... all stable

Derivable models

Properties

- Every model of a monotone constraint program contains the *greatest* derivable model
- Every consistent monotone constraint program has a largest derivable model
- Every consistent monotone constraint program has a minimal derivable model
- Every minimal model of a monotone constraint program is derivable
- (These properties generalize properties of the least model of a normal Horn program)

Convex constraint programs

Key property of monotone constraint programs fails

- As interpretations grow, bodies of rules may cease to hold!
- However, if they do cease to hold, they will not hold again, as long as interpretations grow
- This is exactly the case with bodies of normal logic programs

Convex constraint programs – separating positive and negative

- View
  - $H \leftarrow A_1, \ldots, A_k$

  as
  - $H^m, H^a \leftarrow A_1^m, \ldots, A_k^m, A_1^a, \ldots, A_k^a$

  and then as
  - $H^m \leftarrow A_1^m, \ldots, A_k^m, A_1^a, \ldots, A_k^a$
  - $\perp \leftarrow Comp(H^a), A_1^m, \ldots, A_k^m, A_1^a, \ldots, A_k^a$

Convex constraint programs

Key property of monotone constraint programs fails

- As interpretations grow, bodies of rules may cease to hold!
- However, if they do cease to hold, they will not hold again, as long as interpretations grow
- This is exactly the case with bodies of normal logic programs

Convex constraint programs – separating positive and negative

- View
  - $H \leftarrow A_1, \ldots, A_k$

  as

  - $H^m, H^a \leftarrow A_1^m, \ldots, A_k^m, A_1^a, \ldots, A_k^a$

  and then as

  - $H^m \leftarrow A_1^m, \ldots, A_k^m, A_1^a, \ldots, A_k^a$
  - $\perp \leftarrow Comp(H^a), A_1^m, \ldots, A_k^m, A_1^a, \ldots, A_k^a$

Stable models

Reduct of P w.r.t M

- Remove r from P if for some $A \in bd(r)$, $M \not\models A^a$
- Replace each other rule
  - $H \leftarrow A_1, \ldots, A_k$

  with
  - $H^m \leftarrow A_1^m, \ldots, A_k^m$
  - $\bot \leftarrow Comp(H^a), A_1^m, \ldots, A_k^m$
- A set of atoms M is a stable model of P if M is a derivable model of the reduct P^M
- The definition mirrors that in normal logic programming
- Monotone programs are convex programs: both concepts of stability coincide
- Stable models of an program P are supported models of P

Connection to normal logic programming

Recall the notation a for $(\{a\}, \{\{a\}\})$ and $\mathbf{not}(a)$ for $(\{a\}, \{\emptyset\})$

- A logic program rule

$$a \leftarrow b_1, \ldots, b_m, \mathbf{not}(c_1), \ldots, \mathbf{not}(c_n)$$

  can be viewed as the convex constraint rule
- That mapping preserves all semantics (models, supported models, stable models)

More results, some applications

Theory extends

- Strong equivalence
- Uniform equivalence
- Program completion
- Tightness and Fages lemma
- Loop formulas

Applications

- If we restrtict to weight atoms with non-negative weights:
- *pb-models* – a fast program for computing stable models exploiting off-the-shelf PB(SAT) solvers

More results, some applications

Theory extends

- Strong equivalence
- Uniform equivalence
- Program completion
- Tightness and Fages lemma
- Loop formulas

Applications

- If we restrtict to weight atoms with non-negative weights:
- *pb-models* – a fast program for computing stable models exploiting off-the-shelf PB(SAT) solvers

Discussion

This theory generalizes earlier proposals

- Logic programs with cardinality atoms *(Marek, Niemalä, T_, LPNMR-04)*
- The formalism of logic programs with weight constraints as defined by Niemelä, Soininen and Simons (under a straightforward modular embedding)
- The formalism of *disjunctive* logic programs with the semantics of *possible* models of Inoue and Sakama
- possibility for further generalizations
  - Abstract algebraic theory of *non-deterministic* operators on complete lattices
    (along the lines of the approximation theory which, in particular, unifies default and autoepistemic logics and provides a comprehensive account of normal logic programming)

Modularity

Strong and uniform equivalence

Overview

- Strong equivalence (Lifschitz, Pearce, Valverde, 2001)
  - Two programs P and Q are strongly equivalent if for every program R, $P \cup R$ and $Q \cup R$ have the same stable models
- Uniform equivalence – equivalence for replacement w.r.t. extensions by sets of atoms (Eiter and Fink, 2003)

  Optimizing database queries expressed as logic programs
- Rich theory based on:
  - logic here-and-there (Lifschitz, Pearce, Valverde, 2001)
  - modal logic S4F (Cabalar, 2004; T_, 2007)
  - se-models (Turner, 2003)
  - approxmation theory (T_, 2006)

Strong and uniform equivalence — extensions

$(\mathcal{H}, \mathcal{B})$-equivalence, Woltran 2007

- Consider (disjunctive) programs over a fixed alphabet *At*
- For $\mathcal{H}, \mathcal{B} \subseteq At$: $\mathcal{C}_{\mathcal{H},\mathcal{B}}$ – all programs *P* such that $hd(P) \subseteq \mathcal{H}$ and $bd(P) \subseteq \mathcal{B}$
- Programs *P* and *Q* are $(\mathcal{H}, \mathcal{B})$-equivalent if for every program $R \in \mathcal{C}_{\mathcal{H},\mathcal{B}}$, $P \cup R$ and $Q \cup R$ have the same answer sets
- Characterization of $(\mathcal{H}, \mathcal{B})$-equivalence in terms of $(\mathcal{H}, \mathcal{B})$-models
- (At, At)-equivalence = strong equivalence
- $(At, \emptyset)$-equivalence = uniform equivalence

Stratification

- A program P is stratified if its atoms can be labeled with integers so that for every rule $r \in P$, the label of the head of r is:
  - greater than or equal to the labels of non-negated atoms in the body, and
  - strictly greater than the label of atoms negated in the body of r
- Every stratified logic program has a unique stable model, which can be computed in a bottom-up fashion

  Apt, et al 1986

Splitting Theorem

- For sets A and B of atoms

$$Lit(A,B) = A \cup \{\textbf{not}(b) \colon b \in B \setminus A\}$$

- $P_{\leftarrow L}$ – a simplification of a program P with respect to a set of literals L (“input”)
- Splitting Theorem: Let P and Q be logic programs such that no atom in the head of a rule in Q appears in P. Then M is a stable model of $P \cup Q$ if and only if $M = M_P \cup M_Q$, where M_P is a stable model of P and M_Q is a stable model of $Q_{\leftarrow Lit(M_P, At(P))}$
  Lifschitz, Turner, 1994

General Splitting Theorem

- Restriction: P and Q are logic programs such that no cycle in $DG^{+}(P \cup Q)$ contains atoms from both $hd(P)$ and $hd(Q)$
- Under this restriction: M is a stable model of $P \cup Q$ if and only if $M = M_P \cup M_Q$, where M_P is a stable model of $P_{\leftarrow Lit(M_Q, At(Q))}$ and M_Q is a stable model of $Q_{\leftarrow Lit(M_P, At(P))}$

  Janhunen et al, 2007

Tools for Answer-Set Programming

Tools for Answer-Set Programming

Brief overview

- ASP system
  - Grounder
  - Solver
- Basic approach
  - ground a program (problem encoding) w/r to an input instance (specific data)
  - compute answer sets of the resulting ground program

Tools for Answer-Set Programming

Some history

- *Lparse+smodels*

  Niemelä, Syrjänen, Simmons, 1996-2001
- *Dlv* (integrated grounded and solver for disjuntive logic programs)

  Leone, Eiter, Faber, Pfeifer et al, 1997-present
- *Cmodels* – exploit program completion and SAT techniques

  Lierler, Lifschitz, 2000-present
- *Assat* – program completion + loops formulas

  Lin, Zhao, 2002
- Much more now ...

Tools – 1st Answer-Set Programming Contest

Goals

- Establish methodology
- Collect benchmarks
- Assess available tools (grounders and solvers)
- Establish input/output formats

Acknowledgements

- Broad Community effort
- Run of the Asparagus Platform (University of Potsdam)
  - `http://asparagus.cs.uni-potsdam.de/contest`
- Associated with LPNMR 2007
- Reported in LPNMR Proceedings and on Asparagus site

Gebser, Liu, Namasivayam, Neumann, Schaub, T_, LPNMR 2007

Tools – 1st Answer-Set Programming Contest

Goals

- Establish methodology
- Collect benchmarks
- Assess available tools (grounders and solvers)
- Establish input/output formats

Acknowledgements

- Broad Community effort
- Run of the Asparagus Platform (University of Potsdam)
  - `http://asparagus.cs.uni-potsdam.de/contest`
- Associated with LPNMR 2007
- Reported in LPNMR Proceedings and on Asparagus site

Gebser, Liu, Namasivayam, Neumann, Schaub, T_, LPNMR 2007

ASP Contest – competition classes

Modeling, Grounding, Solving

- Benchmarks consist of:
  - a problem statement
  - a set of instances (given as sets of ground facts),
  - the names of the predicates and their arguments to to encode solutions
- Performance measured by total time for grounding and solving
- Success depends on
  - the quality of the problem encoding
  - the efficiency of a grounder
  - the speed of a solver.

ASP Contest – competition classes

Model computation – core language

- Benchmarks are ground logic programs
  - aggregates are not allowed
  - programs are classified into normal and disjunctive
- Performance measured by time to compute answer sets (or determine inconsistency)

Model computation – programs with weight atoms

- Benchmarks are ground programs with weight atoms (*lparse* format)
- Performance measured by time to compute answer sets (or determine inconsistency)

ASP Contest – competition classes

Model computation – core language

- Benchmarks are ground logic programs
  - aggregates are not allowed
  - programs are classified into normal and disjunctive
- Performance measured by time to compute answer sets (or determine inconsistency)

Model computation – programs with weight atoms

- Benchmarks are ground programs with weight atoms (*lparse* format)
- Performance measured by time to compute answer sets (or determine inconsistency)

ASP Contest – Benchmarks

Broad participation of the communty

- About 40 benchmark families, each with tens, even hundreds of instances
- Diversity in type and hardness

Selection process

- Random subject to some constraints
- Significant number of benchmarks for each competition class (around 100)
- Benchmarks not too hard (at least one competitior able to solve it)
- But also not too easy (no more than three competitors solve it in < 1 sec)

ASP Contest – Benchmarks

Broad participation of the commmunty

- About 40 benchmark families, each with tens, even hundereds of instances
- Diversity in type and hardness

Selection process

- Random subject to some constraints
- Significant number of benchmarks for each competition class (around 100)
- Benchmarks not too hard (at least one competitior able to solve it)
- But also not too easy (no more than three competitors solve it in < 1 sec)

ASP Contest

Particpants

Solver	Affiliation	MGS	SCore	SLparse
asper	Angers		×	
assat	Hongkong	×		×
clasp	Potsdam	×	×	×
cmodels	Texas	×	×	×
dlv	Vienna/Calabria	×	×	
gnt	Helsinki	×	×	×
lp2sat	Helsinki		×	
nomore	Potsdam	×	×	×
pbmodels	Kentucky	×	×	×
smodels	Helsinki	×	×	×

ASP Contest – Results

MGS

1. *dlv*
2. *pbmodels*
3. *clasp*

Core – normal LPs

1. *clasp*
2. *smodels*
3. *cmodels*

LP(weight atoms)

1. *clasp*
2. *pbmodels*
3. *smodels*

Core – disjunctive

1. *dlv*
2. *cmodels*
3. *gnt*

ASP Contest

Some lessons

- Old solvers getting faster
- New solvers springing up: *clasp*, *pbmodels*
- Need standards for input, output and intermediate formats
- Modeling crucial – modeling methodology and good practices
- Grounding – a critical bottleneck
  - Just one new grounder: *gringo* from UP

ID-logic

Moving outside LP domain

Model-expansion formalism

East and T_ 2000; Mitchell and Ternovska, 2005

- Given a FO formula φ and a finite structure A_I for vocabulary $\sigma \subseteq vocab(\varphi)$
- Is there a structure A — an expansion of A_I to $vocab(\varphi)$ — such that $A \models \varphi$?
- NEXPTIME-complete

Model-extension problem *parametrized*

- Fix φ and σ
- Input: finite structure A_I for the vocabulary σ
- Decision problem – NP-complete
- Its search form captures class NP-search (NPMV)

Moving outside LP domain

Model-expansion formalism

East and T_ 2000; Mitchell and Ternovska, 2005

- Given a FO formula φ and a finite structure A_I for vocabulary $\sigma \subseteq vocab(\varphi)$
- Is there a structure A — an expansion of A_I to $vocab(\varphi)$ — such that $A \models \varphi$?
- NEXPTIME-complete

Model-extension problem *parametrized*

- Fix φ and σ
- Input: finite structure A_I for the vocabulary σ
- Decision problem – NP-complete
- Its search form captures class NP-search (NPMV)

Example — graph-coloring problem

Formula φ_{col}

- Every vertex gets at least one color
  - $vtx(X) \rightarrow clrd(X, C)[C].$ $vtx(X) \rightarrow \exists C\ clrd(X, C)$
- Every edge has vertices colored differently
  - $edge(X, Y), clrd(X, C), clrd(Y, C) \rightarrow .$
    $edge(X, Y) \wedge clrd(X, C) \wedge clrd(Y, C) \rightarrow \bot$
- Typing
  - $clrd(X, C) \rightarrow vtx(X)$
  - $clrd(X, C) \rightarrow col(C)$

Signatures

- $vocab(\varphi) = \{col, vtx, edge, clrd\}$
- Signature of an input interpretation: $\sigma = \{col, vtx, edge\}$

Example — graph-coloring problem

Formula φ_{col}

- Every vertex gets at least one color
  - $vtx(X) \rightarrow clrd(X, C)[C].$ $\quad vtx(X) \rightarrow \exists C\ clrd(X, C)$
- Every edge has vertices colored differently
  - $edge(X, Y), clrd(X, C), clrd(Y, C) \rightarrow .$
    $edge(X, Y) \wedge clrd(X, C) \wedge clrd(Y, C) \rightarrow \bot$
- Typing
  - $clrd(X, C) \rightarrow vtx(X)$
  - $clrd(X, C) \rightarrow col(C)$

Signatures

- $vocab(\varphi) = \{col, vtx, edge, clrd\}$
- Signature of an input interpretation: $\sigma = \{col, vtx, edge\}$

Example — graph-coloring problem

Input: an interpretation I of σ

- A finite domain D^I of I
- Interpretation of *col*: relation $col^I(\cdot)$ over D^I
- Interpretation of *vtx*: relation $vtx^I(\cdot)$ over D^I
- Interpretation of *edge*: relation $edge^I(\cdot,\cdot)$ over D^I

Goal

- Find an extension for *clrd* that completes an input interpretation to a model of φ_{col}

Key property

- One-to-one correspondence between such extensions and graph colorings to a model of φ_{col}

Example — graph-coloring problem

Input: an interpretation I of σ

- A finite domain D^I of I
- Interpretation of *col*: relation $col^I(\cdot)$ over D^I
- Interpretation of *vtx*: relation $vtx^I(\cdot)$ over D^I
- Interpretation of *edge*: relation $edge^I(\cdot,\cdot)$ over D^I

Goal

- Find an extension for *clrd* that completes an input interpretation to a model of φ_{col}

Key property

- One-to-one correspondence between such extensions and graph colorings to a model of φ_{col}

Example — graph-coloring problem

Input: an interpretation I of σ

- A finite domain D^I of I
- Interpretation of *col*: relation $col^I(\cdot)$ over D^I
- Interpretation of *vtx*: relation $vtx^I(\cdot)$ over D^I
- Interpretation of *edge*: relation $edge^I(\cdot,\cdot)$ over D^I

Goal

- Find an extension for *clrd* that completes an input interpretation to a model of φ_{col}

Key property

- One-to-one correspondence between such extensions and graph colorings to a model of φ_{col}

Problem solving with MX

Coding a search problem Π

- Select the language – signature of the formula
- Select the input signature
- Construct a formula φ_Π that encodes Π

Solving for an interpretation I

- Ground φ with respect to I
  - replace universal quantification by conjunctions
  - replace existential quantification by disjunctioin
- What results is a propositional theory
  - Its models correspond to expansions of I to models of φ
- Search for a model
- Can use SAT solvers directly!

Problem solving with MX

Coding a search problem Π

- Select the language – signature of the formula
- Select the input signature
- Construct a formula φ_Π that encodes Π

Solving for an interpretation I

- Ground φ with respect to I
  - replace universal quantification by conjunctions
  - replace existential quantification by disjunctioin
- What results is a propositional theory
  - Its models correspond to expansions of I to models of φ
- Search for a model
- Can use SAT solvers directly!

What's missing from MX?

Inductive definitions

- Motivation: KRR applications
- Expressing some concepts neither straightforward nor concise
- Closed World Assumption
- More generaly, inductive definitions (IDs) for instance, transitive closure of a graph
- ID-logic addresses the problem!

ID-Logic, Denecker 1998

Integrates inductive definitions with FOL

- Inductive definitions
  - logic programs with the complete well-founded model
- Intuitive semantics
- Semantics grounded in algebra of operators on lattices and their fixpoints
- Directly extends MX logic
- Simple and concise encoding of logic programs with stable model semantics
- Addresses major KR problems

  Denecker-Ternovska on ID-logic and situation calculus, 2004

Example

Hamiltonian-path problem

- Constraints
  - $hp\_edge(X, Y) \rightarrow edge(X, Y).$
  - $hp\_edge(Y, X), start(X) \rightarrow .$
  - $hp\_edge(X, Y), hp\_edge(X, Z) \rightarrow Y = Z.$
  - $hp\_edge(Y, X), hp\_edge(Z, X) \rightarrow Y = Z.$
  - $visit(X).$
- Definitions
  - $visit(Y) \leftarrow visit(X), hp\_edge(X, Y).$
  - $visit(X) \leftarrow start(X).$
- Typing

Tools

- psgrnd/aspps – `www.cs.uky.edu/ai/aspps/`
- GidL/MidL – `www.cs.kuleuven.be/~dtai/krr/software/idp.html`
- MXG – `www.cs.sfu.cs/research/groups/mxp/mxg/`

Conclusions

Logic programming has much to offer KRR

- Robust solutions for KRR challenges (cf. frame axiom)
- Answer-set semantics and answer-set programming
- Well-founded semantics and the ID-logic

Recent research advances

- Effective modeling support (cf. aggregates/constraints on sets)
- Emerging understanding of modular structure of programs
- More and more powerful tools
- And much more outsie of the main themes covered here ...

Conclusions

Logic programming has much to offer KRR

- Robust solutions for KRR challenges (cf. frame axiom)
- Answer-set semantics and answer-set programming
- Well-founded semantics and the ID-logic

Recent research advances

- Effective modeling support (cf. aggregates/constraints on sets)
- Emerging understanding of modular structure of programs
- More and more powerful tools
- And much more outsie of the main themes covered here ...

Thank you!