GASP: Answer Set Programming with Lazy Grounding

A. Dal Palù[1] A. Dovier[2] E. Pontelli[3] G. Rossi[1]

1. Dip. Matematica, Univ. Parma

2. Dip. Matematica e Informatica, Univ. Udine

3. Dept. Computer Science, New Mexico State Univ.

Umbria Jazz 2008

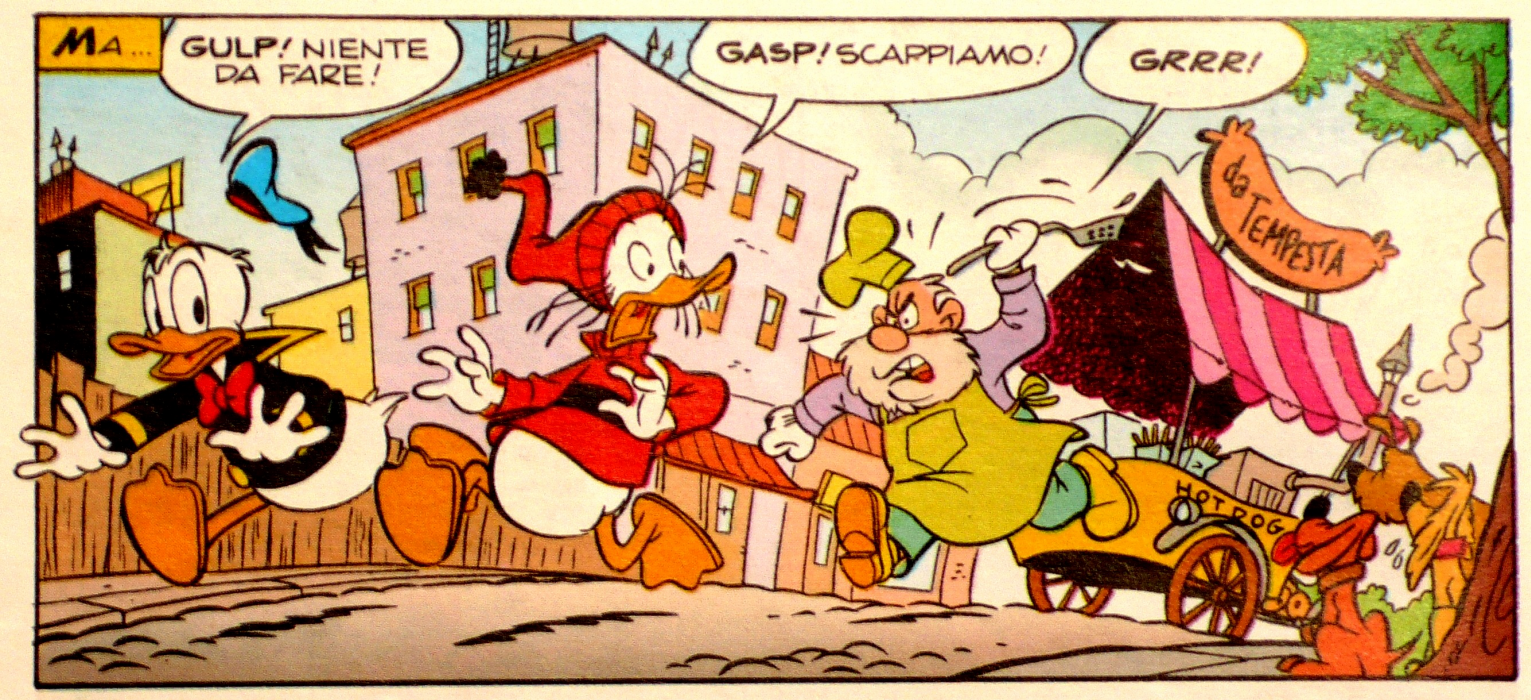
Introduction

Motivations

- We all know the importance of Answer Set Programming for
  - Knowledge representation and reasoning
  - Combinatorial Encoding
  - Emerging Challenging Applications (Bioinformatics, Semantic Web)
- ASP computations are split in two parts
  - Grounding (lparse, Gringo, DLV grounder, . . .)
  - Bottom-up Solving (smodels, cmodels, DLV, Clasp, . . .)
- For medium/large size applications grounding is a problem of time and mainly of space.
- The GULP community is working in this field

Introduction

Motivations



Introduction
Main Idea

- We 'lifted' a ground notion of ASP computation from [Liu,Pontelli,Tran,Truszczynski:ICLP07]
- We developed GASP a lazy grounding tool that grounds clauses as needed and only for some values
- The process is implemented by constraint programming on finite domains.

Introduction

Main Idea

- We 'lifted' a ground notion of ASP computation from [Liu,Pontelli,Tran,Truszczynski:ICLP07]
- We developed GASP a lazy grounding tool that grounds clauses as needed and only for some values
- The process is implemented by constraint programming on finite domains.

Introduction

Main Idea

- We 'lifted' a ground notion of ASP computation from [Liu,Pontelli,Tran,Truszczynski:ICLP07]
- We developed GASP a lazy grounding tool that grounds clauses as needed and only for some values
- The process is implemented by constraint programming on finite domains.

- A general program is a set of clauses of the form

$$H \leftarrow B_1, \ldots, B_m, \textbf{not } C_1, \ldots, \textbf{not } C_n$$

- H, B_i, C_j are atoms (ground or non-ground)
- If $m = 0$ and $n = 0$ they are facts
- ASP programs also contain *constraints* formulas of the form

$$\leftarrow B_1, \ldots, B_m, \textbf{not } C_1, \ldots, \textbf{not } C_n$$

- For stable models they are equivalent to

$$p \leftarrow B_1, \ldots, B_m, \textbf{not } C_1, \ldots, \textbf{not } C_n, \textbf{not } p$$

- A stable model is a minimal Herbrand model S of P.
- Given a ground model P and S, the reduct P^S is defined in the following way: For each clause of P

$$H \leftarrow B_1, \ldots, B_m, \textbf{not } C_1, \ldots, \textbf{not } C_n$$

  - If $\{C_1, \ldots, C_n\} \cap S = \emptyset$ add $H \leftarrow B_1, \ldots, B_m$ in P^S
- If S is the minimum model of P^S (a definite clause program) then S is a stable model of P

- Existence of stable model is NP-complete.
- When looking for stable models, it appears clear that some atoms must be present in all stable models.
- It appears also clear that some atoms cannot be present in any stable model.
- Example:

```
q .
r :- not s.
a :- not b.
b :- not a.
```

- $\{q, r\}$ always true. $\{s\}$ always false. a and b alternatively true or false.

- This suggest a 3-valued view of interpretations:
- The Herbrand base is split into the set I^+ of surely true, the set I^- of surely false, and the others.
- The well-founded model technique computes deterministically a unique pair: $\langle I^+, I^- \rangle$.
- If $I^+ \cup I^- = B_P$ then I^+ is a stable model (in this case P has an unique stable model).
- If $I^+ \cup I^- \neq B_P$ then I^+ is not guaranteed to be 'a model'.
- Anyway, well-founded computation can be seen as the computation of a shared core of stable models.

PRELIMINARIES

WELL-FOUNDED MODEL COMPUTATION

- Technique from [Zukowski, Brass, and Freitag, 1997]
- Given two sets of atoms I and J, let us define

$$T_{P,J}(I) = \{a \in \mathcal{A} \mid (a \leftarrow bd^+, \textbf{not}\ bd^-) \in P, \\ I \models bd^+, (\forall p \in bd^-)(J \not\models p)\}$$

- For computing the well-founded model, compute

$$\begin{cases} K_0 = \mathrm{lfp}(T_{P^+,\emptyset}) & U_0 = \mathrm{lfp}(T_{P,K_0}) \\ K_i = \mathrm{lfp}(T_{P,U_{i-1}}) & U_i = \mathrm{lfp}(T_{P,K_i}) \quad i > 0 \end{cases}$$

  where P^+ is the set of definite clauses of P.
- If $(K_i, U_i) = (K_{i+1}, U_{i+1})$ we stop and $I^+ = K_i, I^- = B_P \setminus U_i$ is the well-founded model.

- Consider $P = \{p \leftarrow \textbf{not}\ q. \quad q \leftarrow \textbf{not}\ p\}$
- It admits three Herbrand models $\{p\}, \{q\}, \{p, q\}$
- $\{p\}$ and $\{q\}$ are its stable models
- Its well-founded (partial) model is $I^+ = \emptyset, I^- = \emptyset$
- One can start from I^+ and I^- of the well-founded model to compute stable models guessing the remaining atoms.

- $I \models b_1, \ldots, b_m, \textbf{not } c_1, \ldots, \textbf{not } c_n$ iff $\{b_1, \ldots, b_m\} \subseteq I$ and $\{c_1, \ldots, c_n\} \cap I = \emptyset$
- $a \in T_P(I)$ iff there is a ground clause $a \leftarrow Body$ in P s.t. $I \models Body$

A computation of a program P is a sequence of 2 valued-interpretations $\emptyset = I_0, I_1, I_2, \ldots$ such that:

- $I_i \subseteq I_{i+1}$ for all $i \geq 0$ (*Persistence of Beliefs*)
- $\bigcup_{i=0}^{\infty} I_i$ is a model of P (*Convergence*)
- $I_{i+1} \subseteq T_P(I_i)$ for all $i \geq 0$ (*Revision*)
- if $a \in I_{i+1} \setminus I_i$ then there is a rule $a \leftarrow Body$ in P such that $I_j \models Body$ for each $j \geq i$ (*Persistence of Reason*).

Given a rule $a \leftarrow body$ and 3 valued-interpretation I, we say that the rule is applicable w.r.t. I if

$$body^+ \subseteq I^+ \text{ and } body^- \cap I^+ = \emptyset\,.$$

We extend the definition of applicable to a non ground rule R w.r.t. I iff there exists a grounding r of R that is applicable w.r.t. I.

ASP COMPUTATION

NON GROUND

A GASP computation of a program P is a sequence of 3-interpretations $I_0, I_1, I_2, \ldots$ that satisfies the following properties:

- $I_0 = \text{wf}(P)$
- $I_i \subseteq I_{i+1}$ (*Persistence of Beliefs*)
- if $I = \bigcup_{i=0}^{\infty} I_i$, then $\langle I^+, \mathcal{A} \setminus I^+ \rangle$ is a model of P (*Convergence*)
- for each $i \geq 0$ there exists a rule $a \leftarrow body$ in P that is applicable w.r.t. I_i and $I_{i+1} = \text{wf}(P \cup I_i \cup \langle body^+, body^- \rangle)$ (*Revision*)
- if $a \in I_{i+1} \setminus I_i$ then there is a rule $a \leftarrow body$ in P which is applicable w.r.t. I_j, for each $j \geq i$ (*Persistence of Reason*).

GASP computations lead to stable models

IMPLEMENTATION

INTERPETATIONS

- The implementation is based on CLP(FD)
- All constants are numbers in $[0, \mathbb{M} - 1]$.
- For each predicate p of arity n we need to store two sets of tuples $\langle t_1, \ldots, t_n \rangle$
- We convert them into a unique value $\tau(t_1, \ldots, t_n)$:

$$t_1 M^{n-1} + t_2 M^{n-2} + \cdots + t_{n-1} M + t_n$$

- In this way, computing T_P we will skip set operations (projections, intersections, joins, etc) and exploit constraint-based primitives.
- Each predicate p is associated to an atom: $\texttt{atom}(p, n, \{\tau(\text{true tuples})\}, \{\tau(\text{false tuples})\})$
- A 3-valued interpretation is a list of these atoms

Implementation
FDSETS

- The sets of tuples are stored as FDSETS
- The set $\{0, 1, 2, 3, 4, 5, 6, 7, 11, 12, 13, 14, 15\}$ is stored compactly as a list of disjoint intervals:

$$[[0|7], [11|15]]$$

- This allows to sensibly reduce representation space of sets/domains and also allows fast implementation of constraint propagation
- Several FDSETS primitives are available in SICStus Prolog.

IMPLEMENTATION

FDSETS: EXAMPLE

- Let p be of arity 3
- Let $\mathbb{M} = 10$
- Assume that
  $p(0,0,1), p(0,0,2), p(0,0,3), p(0,0,8),$
  $p(0,0,9), p(0,1,0), p(0,1,1), p(0,1,2)$
  hold.
- Then:
  $\{\tau(0,0,1), \tau(0,0,2), \tau(0,0,3), \tau(0,0,8),$
  $\tau(0,0,9), \tau(0,1,0), \tau(0,1,1), \tau(0,1,2)\}$
  can be represented simply by:

$$[[1|3], [8|12]]$$

IMPLEMENTATION

T_P COMPUTATION: MAIN IDEA

- Assume: $\texttt{atom}(p, 2, PosP, \_)$, $\texttt{atom}(q, 2, PosQ, \_)$, $\texttt{atom}(r, 1, PosR, \_)$
- Consider $r(X) : -p(X, Y), q(Y, Z)$
- We introduce FD constraints:
- $X \in 0..\mathbb{M} - 1, Y \in 0..\mathbb{M} - 1, Z \in 0..\mathbb{M} - 1$
- $\mathbb{M} * X + Y \in PosP \wedge \mathbb{M} * Y + Z \in PosQ \wedge X \notin PosR \Rightarrow \text{add}(X, PosR)$
- Namely, add to $PosR$ all the values of X such that $\tau(X, Y) \in PosP$ and $\tau(Y, Z) \in PosQ$ that are not already in $PosR$
- No need of joins, intersections, etc.

T_P COMPUTATION: RULE APPLICATION

T_P IN SICSTUS

```
apply_def_rule(rule([H],PBody,[]),I,
                [atom(F,ARITY,NEWPDOM,NEG)|PARTI]) :-
      copy_term([H,PBody],[H1,PBody1]),
      term_variables([H1,PBody1],VARS),
      bigM(M),M1 is M-1,domain(VARS,0,M1),
      build_constraint(PBody1,I,C1,pos),
      H1 =..[F|ARGS],
      build_arity(ARGS,VAR,ARITY),
      select(atom(F,ARITY,OLD,NEG),I,PARTI),
      nin_set(ARITY,VAR,OLD,C2),
      findall(X,
         (C1+C2 #= 2, X #= VAR,labeling([ff],VARS)),
         LIST),
      list_to_fdset(LIST,SET),
      fdset_union(OLD,SET,NEWPDOM).
```

T_P COMPUTATION: RULE APPLICATION

T_P COMPUTATION: CONSTRAINTS

```
build_constraint([R|Rs],I,C,pos):-
      R =.. [F|ARGS],!,
      build_arity(ARGS,VAR,ARITY),
      member(atom(F,ARITY,DOMF,NDOMF),I),
      C2  #<=> VAR in_set DOMF,
      build_constraint(Rs,I,C1,Sign),
      C #<=>  C1 #/\ C2.

build_arity(ARGS,VAL,ARITY) :-
       (ARGS = [],!, VAL=0, ARITY=0;
        ARGS = [VAL],!,  ARITY=1;
        ARGS = [Arg1,Arg2],!, bigM(M),
                ARITY=2, VAL #= M*Arg1+Arg2;
        ARGS = [Arg1,Arg2,Arg3], bigM(M),
                ARITY=3, VAL #= M*M*Arg1+M*Arg2+Arg3).
```

Implementation

T_P for well-founded

```
alternating_apply_rule(rule([H],PBody,NBody),
          J,I,[atom(F,ARITY,NEWPDOM,NEG)|PARTI]) :-
      copy_term([H,PBody,NBody],[H1,PBody1,NBody1]),
      term_variables([H1,PBody1],VARS),
      bigM(M),M1 is M-1,domain(VARS,0,M1),
      build_constraint(PBody1,I,C1,pos),
      build_constraint(NBody1,J,C2,neg),
      H1 =..[F|ARGS],
      build_arity(ARGS,VAR,ARITY),
      select(atom(F,ARITY,OLD,NEG),I,PARTI),
      nin_set(ARITY,VAR,OLD,C3),
      findall(X,
          (C1+C2+C3 #= 3, X #= VAR,labeling([ff],VARS)),
          LIST),
      list_to_fdset(LIST,SET),
      fdset_union(OLD,SET,NEWPDOM).
```

Implementation

GASP computation

```
apply_rule(rule([H],Bpos,Bneg),I1,I2,rule([H1],Bpos1,Bneg1)) :-
      Bneg \= [], %%% Must have negative literals
      copy_term([H,Bpos,Bneg],[H1,Bpos1,Bneg1]),
      H1 =.. [F|ARGS],
      term_variables([H1,Bpos1],VARS),
      bigM(M), M1 is M-1, domain(VARS,0,M1),
      build_constraint(Bpos1,I1,1,pos),
      build_constraint(Bneg1,I1,1,neg),
      build_arity(ARGS,Arg,ARITY),
      member(atom(F,ARITY,PDOM,_),I1),
      nin_set(ARITY,Arg,PDOM,1),
      labeling([ff],VARS),
      dom_update(Bneg1,I1,I2). %%% Update negative part
```

Problem of permutations... partially solved.

Implementation

Some extensions


```
1 { function(X,Y): range(Y) } 1 :- domain(X).
```

Naive encoding (assume `range(a1),...,range(an)`):

```
function(X,a1) :- domain(X),
       not function(X,a2), ..., not function(X,an).
                    ...
function(X,an) :- domain(X),
       not function(X,a1), ..., not function(X,an-1).
```

FD encoding: `function` is associated to a list of pairs $[[X_1, Y_1], \ldots, [X_m, Y_m]]$ where m is the size of `domain(X)`, and

- $[X_1, \ldots, X_m] :: \texttt{domain}(X)$,
- $X_i < X_{i+1}$,
- $[Y_1, \ldots, Y_m] :: \{a_1, \ldots, a_n\}$

IMPLEMENTATION

SOME EXTENSIONS

Consider the ASP constraint

```
:- number(X), number(Y), number(Z), part(P),
       inpart(X,P), inpart(Y,P), inpart(Z,P),
           Z = X+Y.
```

It "suggests" the following FD constraints:

```
for X: number(X) for Y:number(Y) for Z:number(Z)
  for P1:  part(P1) for P2:part(P2) for P3:part(P3)
    if inpart(X,P1) ∧ inpart(Y,P2) ∧ inpart(Z,P3) ∧X + Y = Z
      add the constraint P1 = P2 ⇒ P1 ≠ P3
```

Implementation

Some results

	N (n,p)	Lparse	Smodels	GASP	GASP/Smodels	GASP-fun
Def	32	0.06	0.03	0.14	4.6	0.14
	64	0.09	0.12	0.45	3.7	0.45
	128	0.26	0.79	1.89	2.4	1.89
	228	0.75	1.95	6.78	3.5	6.78
	256	0.65	Error	8.81	-	8.81
WF	32	0.06	0.08	0.98	12.2	0.98
	64	0.11	0.23	3.58	27.8	3.58
	128	0.32	1.07	15.38	14.4	15.38
	228	0.90	3.39	58.70	17.3	58.70
	256	0.68	Error	78.34	-	78.34
Schur	(6,3)	0.04	0.09	1.64	18.2	0.04
	(7,3)	0.05	0.18	6.59	36.6	0.06
	(8,3)	0.05	0.25	27.43	109.7	0.06
	(9,3)	0.06	0.48	113.59	236.65	0.06
	(10,3)	0.06	0.19	480.85	2530.8	0.09
	(30,4)	0.09	0.03	∞	-	0.36
	(35,4)	0.10	0.09	∞	-	0.44
	(40,4)	0.11	77.31	∞	-	0.50
	(45,4)	0.15	∞	∞	-	8315

Conclusions

- Is GASP a solution?
- Not yet, we should work:

- SGRUNT = Senza GRoUNding Totalmente