UNIVERSITÀ DELLA CALABRIA
DIPARTIMENTO DI MATEMATICA E INFORMATICA

An Infrastructure for Multi-Shot Reasoning with Incremental Grounding

Giovambattista Ianni Francesco Pacenza Jessica Zangari
Department of Mathematics and Computer Science, University of Calabria, Italy

34^{th} *Italian Conference on Computational Logic*

Introduction

Answer Set Programming (ASP)

- Declarative paradigm for Knowledge Representation and Reasoning
- Able to model incomplete knowledge and non-monotonic reasoning
- Successfully employed in both academy and industry

Core Syntax

- A **disjunctive logic program** is a (finite) set of **rules** of form:

$$\underbrace{a_1 \mid \ldots \mid a_n}_{\text{head atoms}} \texttt{ :- } \underbrace{b_1, \ldots, b_k, \texttt{not } b_{k+1}, \ldots, \texttt{not } b_m.}_{\text{body literals}}$$

- A **literal** is either **positive** a or **negative** $\texttt{not } a$ where a is an **atom**
- An **atom** has form $p(t_1, \ldots, t_n)$ (typical FO signature), where:
  - p is a **predicate** of arity n
  - $t_1, \ldots, t_n$ are **terms**
- A **term** is either a **variable** or a **constant**

ASP Syntax

Core Syntax

- A **disjunctive logic program** is a (finite) set of **rules** of form:

$$\underbrace{a_1 \mid \ldots \mid a_n}_{\text{head atoms}} \texttt{ :- } \underbrace{b_1, \ldots, b_k, \texttt{not } b_{k+1}, \ldots, \texttt{not } b_m.}_{\text{body literals}}$$

- A **literal** is either **positive** a or **negative** $\texttt{not } a$ where a is an **atom**
- An **atom** has form $p(t_1, \ldots, t_n)$ (typical FO signature), where:
  - p is a **predicate** of arity n
  - $t_1, \ldots, t_n$ are **terms**
- A **term** is either a **variable** or a **constant**

Linguistic Extensions

- Many extensions:
  - new term types: functional terms, arithmetic terms
  - new atom and literal types: aggregate literals, built-in atoms
  - new rule types: weak constraints, choice rules, queries
- Standard input language: ASP-Core-2

ASP Computation

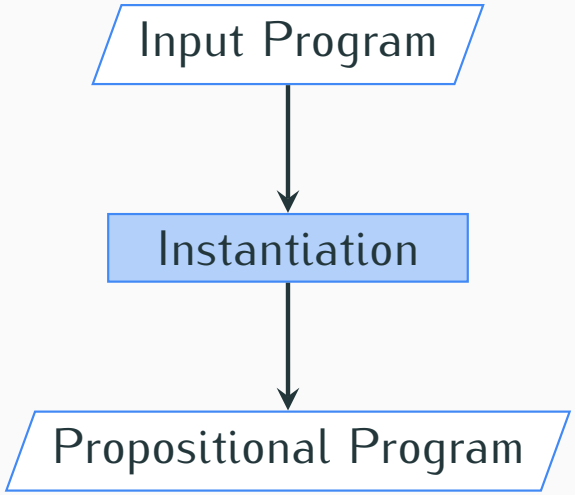
Input Program

Canonical approach to solve an ASP program π:

ASP Computation

Canonical approach to solve an ASP program π:

1. **Instantiation** (or **Grounding**) phase:
   - Produces a semantically equivalent ground program $P = Ground(\pi)$, with $P \subseteq P_H$ where $P_H = Ground_{Herbrand}(\pi)$

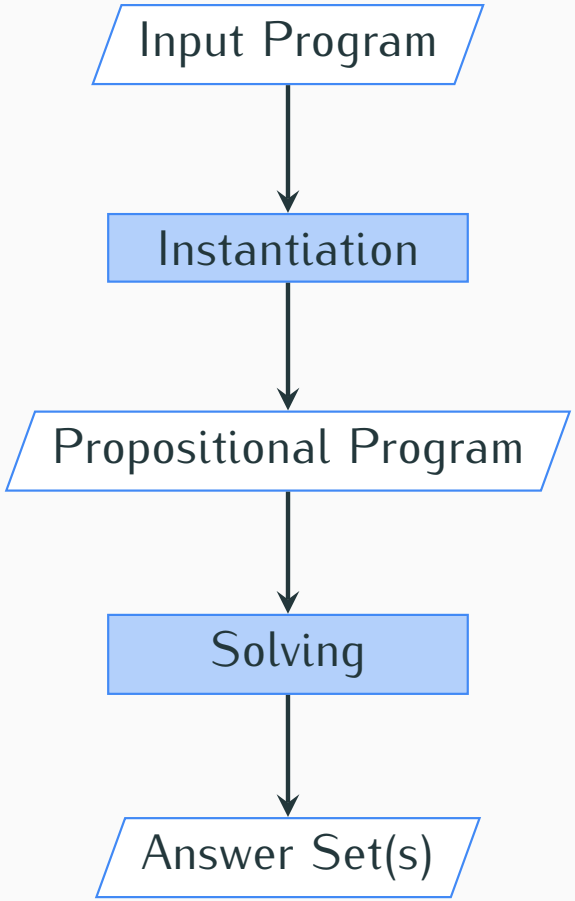


ASP Computation

Canonical approach to solve an ASP program π:

1. **Instantiation** (or **Grounding**) phase:
   - Produces a semantically equivalent ground program $P = Ground(\pi)$, with $P \subseteq P_H$ where $P_H = Ground_{Herbrand}(\pi)$
2. **Solving** phase:
   - Generates the Answer Set(s), $AS(P)$ of π

$$AS(\pi) \equiv AS(P) \equiv AS(P_H)$$



ASP Systems

Canonical approach: Ground&Solve

- Stand-alone grounders: Lparse, IDLV, Gringo
- Stand-alone solvers: Cmodels, Smodels, Clasp, Wasp
- Monolithic systems: DLV2, Clingo

ASP Systems

Canonical approach: Ground&Solve

- Stand-alone grounders: Lparse, IDLV, Gringo
- Stand-alone solvers: Cmodels, Smodels, Clasp, Wasp
- Monolithic systems: DLV2, Clingo

Other approaches

- Lazy Grounding: *Gasp*, *Asperix*, *Omiga*, *Alpha*
- Translation-based systems: LPtoSAT

Embedding-Programs

Model-Theoretic vs Embedding-Programs Semantics

Model-Theoretic Semantics

Set of *atoms*

Embedding-Programs Semantics

Set of *rules*

Model-Theoretic vs Embedding-Programs Semantics

Model-Theoretic Semantics

Set of *atoms*

Let I be an interpretation, P a program, $r = a \leftarrow b_1, ..., b_m$ a variable-free ground rule

Embedding-Programs Semantics

Set of *rules*

Let P be a program, F a set of facts, $S \subseteq (grnd(P) \cup F)$ a set of ground rules, $r \in (grnd(P) \cup F)$ a rule

Model-Theoretic vs Embedding-Programs Semantics

Model-Theoretic Semantics

Set of *atoms*

Let I be an interpretation, P a program, $r = a \leftarrow b_1, ..., b_m$ a variable-free ground rule

$I \models B(r)$, if $B(r) \in I$

Embedding-Programs Semantics

Set of *rules*

Let P be a program, F a set of facts, $S \subseteq (grnd(P) \cup F)$ a set of ground rules, $r \in (grnd(P) \cup F)$ a rule

$S \vdash_b r$, if $\forall a \in B^+(r)\ \exists r' \in S$ s.t. $a \in H(r')$

Model-Theoretic vs Embedding-Programs Semantics

Model-Theoretic Semantics

Set of *atoms*

Let I be an interpretation, P a program, $r = a \leftarrow b_1, ..., b_m$ a variable-free ground rule

$I \models B(r)$, if $B(r) \in I$

$I \models H(r)$, if $H(r) \in I$

Embedding-Programs Semantics

Set of *rules*

Let P be a program, F a set of facts, $S \subseteq (grnd(P) \cup F)$ a set of ground rules, $r \in (grnd(P) \cup F)$ a rule

$S \vdash_b r$, if $\forall a \in B^+(r)\ \exists r' \in S$ s.t. $a \in H(r')$

$S \vdash_h r$, if $r \in S$

Model-Theoretic vs Embedding-Programs Semantics

Model-Theoretic Semantics

Set of *atoms*

Let I be an interpretation, P a program, $r = a \leftarrow b_1, ..., b_m$ a variable-free ground rule

$I \models B(r)$, if $B(r) \in I$

$I \models H(r)$, if $H(r) \in I$

$I \models r$, if either: (i) $I \not\models B(r)$, or (ii) $I \models H(r)$

Embedding-Programs Semantics

Set of *rules*

Let P be a program, F a set of facts, $S \subseteq (grnd(P) \cup F)$ a set of ground rules, $r \in (grnd(P) \cup F)$ a rule

$S \vdash_b r$, if $\forall a \in B^+(r)\ \exists r' \in S$ s.t. $a \in H(r')$

$S \vdash_h r$, if $r \in S$

$S \vdash r$, if either: (i) $S \nvdash_b r$, or (ii) $S \vdash_h r$

Embedding example

Input Program

$r_1: \ a(1,1).$

$r_2: \ b(1,1).$

$r_3: \ c(1,1) \ \text{:-} \ a(1,1), \ not \ b(1,1).$

$r_4: \ d(1,1) \ \text{:-} \ a(1,1).$

$r_5: \ e(1) \ \text{:-} \ f(1).$

$r_6: \ g(1) \ \text{:-} \ e(1), \ not \ a(1,1).$

Related Embedding Program

Embedding example

Input Program

$r_1 :\ a(1,1).$

$r_2 :\ b(1,1).$

$r_3 :\ c(1,1)\ \text{:-}\ a(1,1),\ not\ b(1,1).$

$r_4 :\ d(1,1)\ \text{:-}\ a(1,1).$

$r_5 :\ e(1)\ \text{:-}\ f(1).$

$r_6 :\ g(1)\ \text{:-}\ e(1),\ not\ a(1,1).$

Related Embedding Program

$r_1 :\ a(1,1).$ ✓

Embedding example

Input Program

$r_1 :\ a(1,1).$

$r_2 :\ b(1,1).$

$r_3 :\ c(1,1)\ \text{:-}\ a(1,1),\ not\ b(1,1).$

$r_4 :\ d(1,1)\ \text{:-}\ a(1,1).$

$r_5 :\ e(1)\ \text{:-}\ f(1).$

$r_6 :\ g(1)\ \text{:-}\ e(1),\ not\ a(1,1).$

Related Embedding Program

$r_1 :\ a(1,1).$ ✓

$r_2 :\ b(1,1).$ ✓

Embedding example

Input Program

$r_1 :\ a(1,1).$

$r_2 :\ b(1,1).$

$r_3 :\ c(1,1)\ \text{:-}\ a(1,1),\ not\ b(1,1).$

$r_4 :\ d(1,1)\ \text{:-}\ a(1,1).$

$r_5 :\ e(1)\ \text{:-}\ f(1).$

$r_6 :\ g(1)\ \text{:-}\ e(1),\ not\ a(1,1).$

Related Embedding Program

$r_1 :\ a(1,1).$ ✓

$r_2 :\ b(1,1).$ ✓

$r_3 :\ c(1,1)\ \text{:-}\ a(1,1),\ not\ b(1,1).$ ✓

Embedding example

Input Program

$r_1 :\ a(1,1).$

$r_2 :\ b(1,1).$

$r_3 :\ c(1,1)\ \text{:-}\ a(1,1),\ not\ b(1,1).$

$r_4 :\ d(1,1)\ \text{:-}\ a(1,1).$

$r_5 :\ e(1)\ \text{:-}\ f(1).$

$r_6 :\ g(1)\ \text{:-}\ e(1),\ not\ a(1,1).$

Related Embedding Program

$r_1 :\ a(1,1).$ ✓

$r_2 :\ b(1,1).$ ✓

$r_3 :\ c(1,1)\ \text{:-}\ a(1,1),\ not\ b(1,1).$ ✓

$r_4 :\ d(1,1)\ \text{:-}\ a(1,1).$ ✓

Embedding example

Input Program

$r_1 :\ a(1,1).$

$r_2 :\ b(1,1).$

$r_3 :\ c(1,1)\ \text{:-}\ a(1,1),\ not\ b(1,1).$

$r_4 :\ d(1,1)\ \text{:-}\ a(1,1).$

$r_5 :\ e(1)\ \text{:-}\ f(1).$

$r_6 :\ g(1)\ \text{:-}\ e(1),\ not\ a(1,1).$

Related Embedding Program

$r_1 :\ a(1,1).$ ✓

$r_2 :\ b(1,1).$ ✓

$r_3 :\ c(1,1)\ \text{:-}\ a(1,1),\ not\ b(1,1).$ ✓

$r_4 :\ d(1,1)\ \text{:-}\ a(1,1).$ ✓

$r_5 :\ e(1)\ \text{:-}\ f(1).$ ×

Embedding example

Input Program

$r_1 :\ a(1,1).$

$r_2 :\ b(1,1).$

$r_3 :\ c(1,1)\ \text{:-}\ a(1,1),\ not\ b(1,1).$

$r_4 :\ d(1,1)\ \text{:-}\ a(1,1).$

$r_5 :\ e(1)\ \text{:-}\ f(1).$

$r_6 :\ g(1)\ \text{:-}\ e(1),\ not\ a(1,1).$

Related Embedding Program

$r_1 :\ a(1,1).$ ✓

$r_2 :\ b(1,1).$ ✓

$r_3 :\ c(1,1)\ \text{:-}\ a(1,1),\ not\ b(1,1).$ ✓

$r_4 :\ d(1,1)\ \text{:-}\ a(1,1).$ ✓

$r_5 :\ e(1)\ \text{:-}\ f(1).$ ×

$r_6 :\ g(1)\ \text{:-}\ e(1),\ not\ a(1,1).$ ×

Overgrounding Approach

Context

Given a program P and a sequence of set of facts F_1, ..., F_n, we need to perform a series of distinct evaluations for P over different input facts changing in each "shot"

- We aim at computing the sequence $AS(P \cup F_1)$, ..., $AS(P \cup F_n)$.

Context

Given a program P and a sequence of set of facts F_1, ..., F_n, we need to perform a series of distinct evaluations for P over different input facts changing in each "shot"

- We aim at computing the sequence $AS(P \cup F_1)$, ..., $AS(P \cup F_n)$.

Definition $Inst(P, S)$

Given a program P and a set of ground atoms S, we define the operator $Inst(P, S)$ as $Inst(P, S) = \{r \in grnd(P) \ s.t. \ B^+(r) \subseteq S \ \}$

Context

Given a program P and a sequence of set of facts F_1, ..., F_n, we need to perform a series of distinct evaluations for P over different input facts changing in each "shot"

- We aim at computing the sequence $AS(P \cup F_1)$, ..., $AS(P \cup F_n)$.

Definition $Inst(P, S)$

Given a program P and a set of ground atoms S, we define the operator $Inst(P, S)$ as $Inst(P, S) = \{r \in grnd(P) \ s.t. \ B^+(r) \subseteq S \ \}$

Theorem

Let $UF_k = \bigcup_{1 \leq i \leq k} F_i$. It holds that

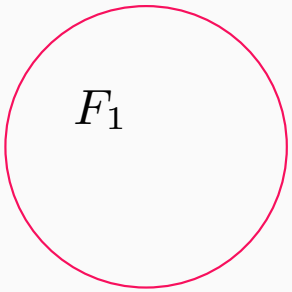
$$AS(Inst(P, UF_k)^\infty \cup F_k) = AS(P \cup F_k)$$

In particular, for each k,

$$Inst(P, UF_k)^\infty \subseteq Inst(P, UF_{k+1})^\infty \tag{1}$$

Example

Accumulated Set of Facts UF_k

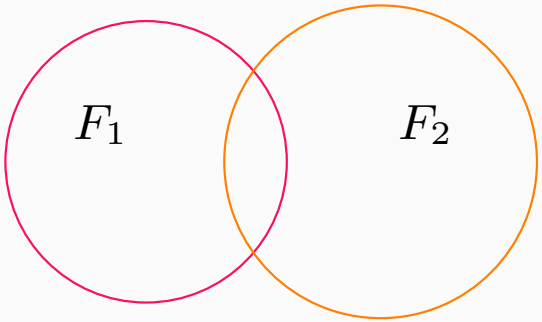


New Ground Program $Inst(P, UF_k)^\infty$

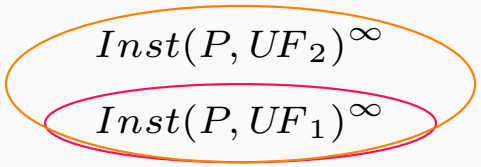
$Inst(P, UF_1)^\infty$

Example

Accumulated Set of Facts UF_k

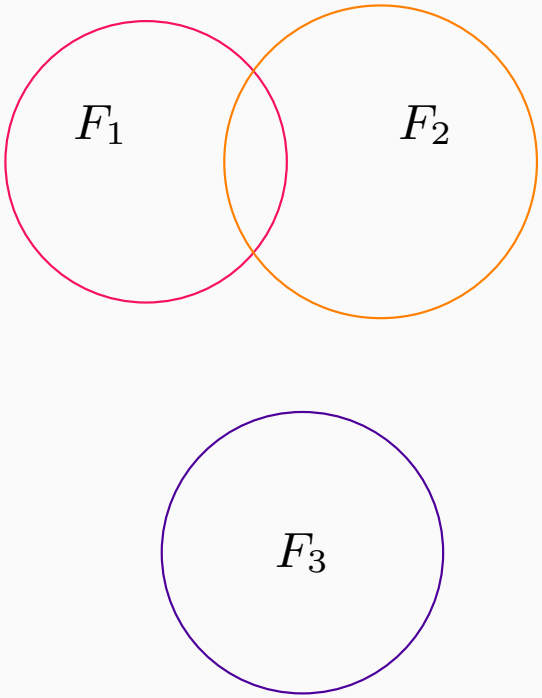


New Ground Program $Inst(P, UF_k)^\infty$

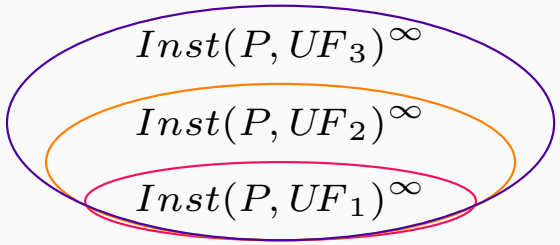


Example

Accumulated Set of Facts UF_k

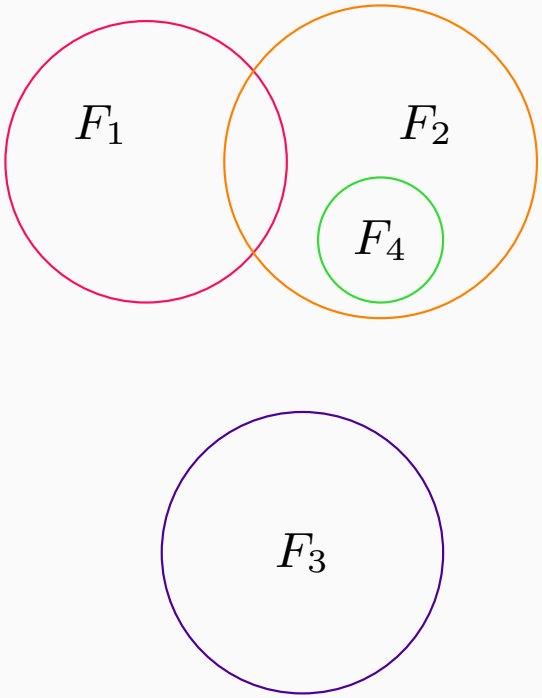


New Ground Program $Inst(P, UF_k)^\infty$

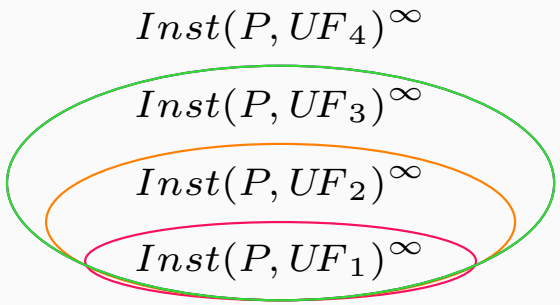


Example

Accumulated Set of Facts UF_k



New Ground Program $Inst(P, UF_k)^\infty$



Incremental Grounding Algorithm

Input: a stored ground program $G_k = inst^{\infty}(KB, UF_k)$
Input: union of "accumulated" input facts $UF_k = \bigcup_{1 \leq i \leq k} F_k$
Input: current input facts F_{k+1}
Output: updated ground program G_{k+1}, updated accumulated facts UF_{k+1}

Differential Algorithm

- At shot 1, we set $UF_1 = F_1$, and $G_1 = Inst(P, UF_1)^{\infty}$
- At generic shot k:
  1. we set $UF_k = UF_{k-1} \cup F_k$,
  2. we compute a set of additional ground rules ΔG_k, and
  3. we set $G_k = G_{k-i} \cup \Delta G_k$.

Overgrounding - Example

Consider the *fixed* program P_0:

$$r(X,Y) \text{ :- } e(X,Y),\ not\ ab(X). \qquad\qquad r(X,Z) \mid s(X,Z) \text{ :- } e(X,Y),\ r(Y,Z).$$

[1] With respect to F_1, F_2 features the additions $F^+ = \{e(a,d), ab(c)\}$ and the deletions $F^- = \{e(a,b)\}$

Overgrounding - Example

Consider the *fixed* program P_0:

$r(X,Y) \text{ :- } e(X,Y),\ not\ ab(X).$ $\qquad$ $r(X,Z) \mid s(X,Z) \text{ :- } e(X,Y),\ r(Y,Z).$

Iteration #1: $F_1 = \{e(c,a),\ e(a,b)\}$

$r_1 : r(a,b) \text{ :- } e(a,b),\ not\ ab(a).$ $\qquad$ $r_2 : r(c,b) \mid s(c,b) \text{ :- } e(c,a),\ r(a,b).$

$r_3 : r(c,a) \text{ :- } e(c,a),\ not\ ab(c).$

[1] With respect to F_1, F_2 features the additions $F^+ = \{e(a,d), ab(c)\}$ and the deletions $F^- = \{e(a,b)\}$

Overgrounding - Example

Consider the *fixed* program P_0:

$r(X,Y) \text{ :- } e(X,Y),\ not\ ab(X).$ $\qquad$ $r(X,Z) \mid s(X,Z) \text{ :- } e(X,Y),\ r(Y,Z).$

Iteration #1: $F_1 = \{e(c,a),\ e(a,b)\}$

$r_1 : r(a,b) \text{ :- } e(a,b),\ not\ ab(a).$ $\qquad$ $r_2 : r(c,b) \mid s(c,b) \text{ :- } e(c,a),\ r(a,b).$

$r_3 : r(c,a) \text{ :- } e(c,a),\ not\ ab(c).$

Iteration #2: $F_2 = \{e(c,a),\ e(a,d),\ ab(c)\}$ [1] $- UF_2 = F_1 \cup F_2$

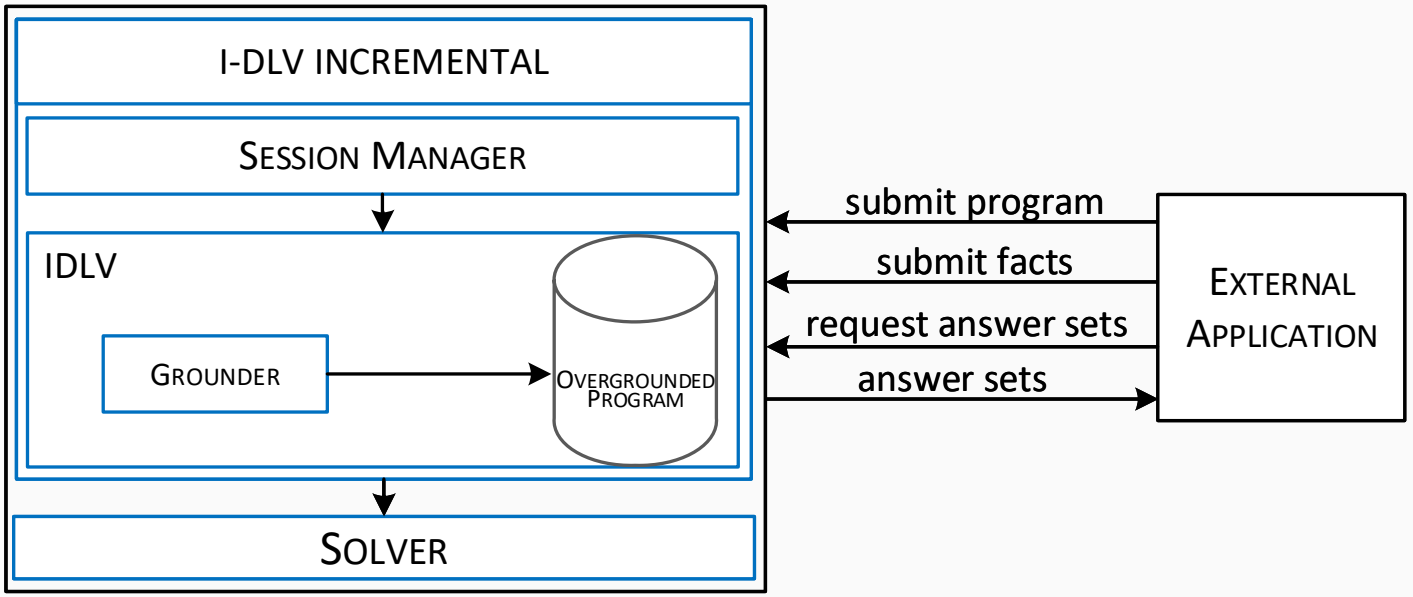
$r_1 : r(a,b) \text{ :- } e(a,b),\ not\ ab(a).$ $\qquad$ $r_2 : r(c,b) \mid s(c,b) \text{ :- } e(c,a),\ r(a,b).$

$r_3 : r(c,a) \text{ :- } e(c,a), not\ ab(c).$ $\qquad$ $\mathbf{r_4 : r(c,d) \mid s(c,d) \text{ :- } e(c,a),\ r(a,d).}$

$\mathbf{r_5 : r(a,d) \text{ :- } e(a,d),\ not\ ab(a).}$

[1] With respect to F_1, F_2 features the additions $F^+ = \{e(a,d), ab(c)\}$ and the deletions $F^- = \{e(a,b)\}$

System Architecture



Benchmark

Multi-shot Sudoku

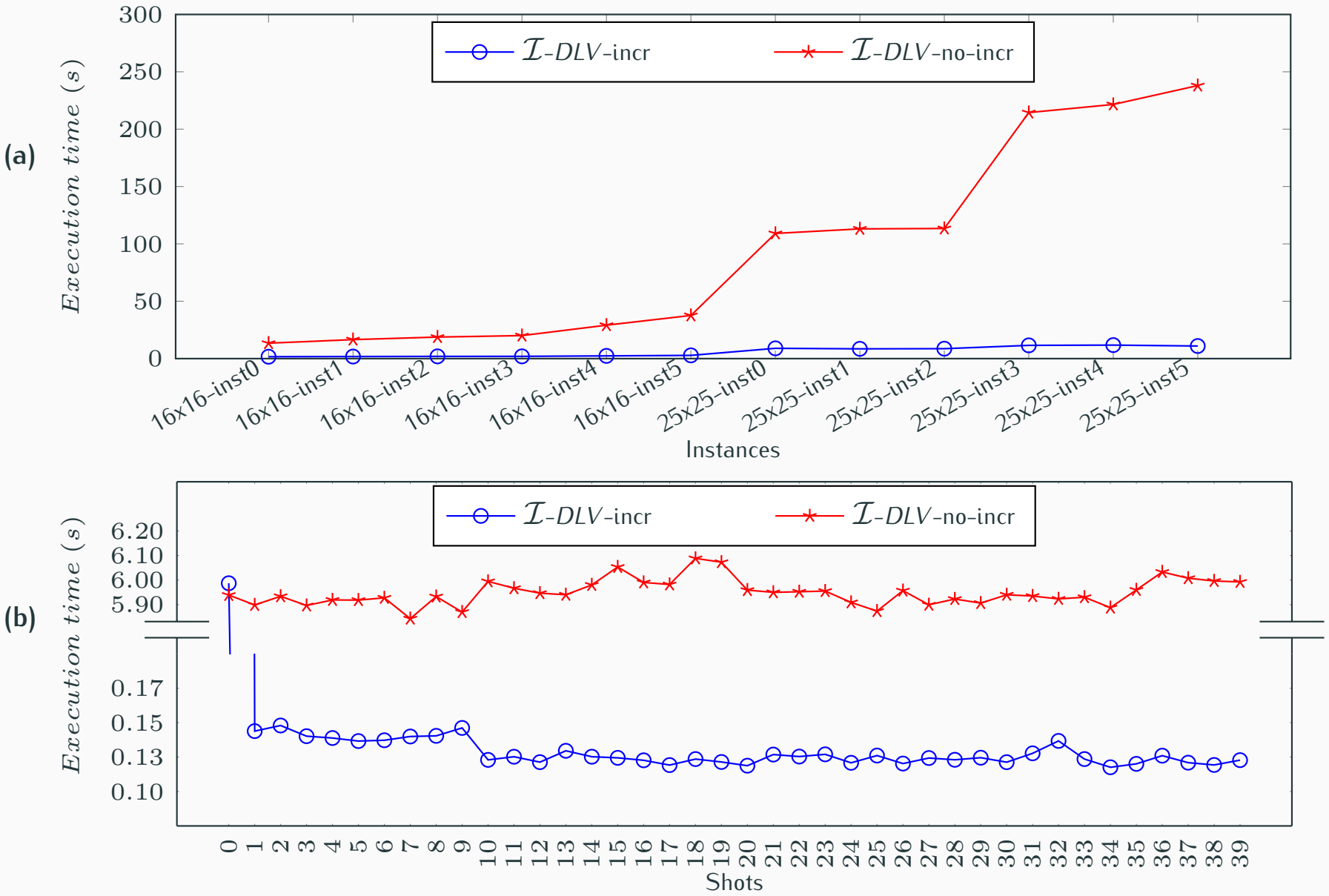


Figure 1: (a) Experiments on Sudoku benchmarks. (b) Grounding times for all iterations of a 25x25 Sudoku instance.

Multi-shot Pac-Man - Time Performance

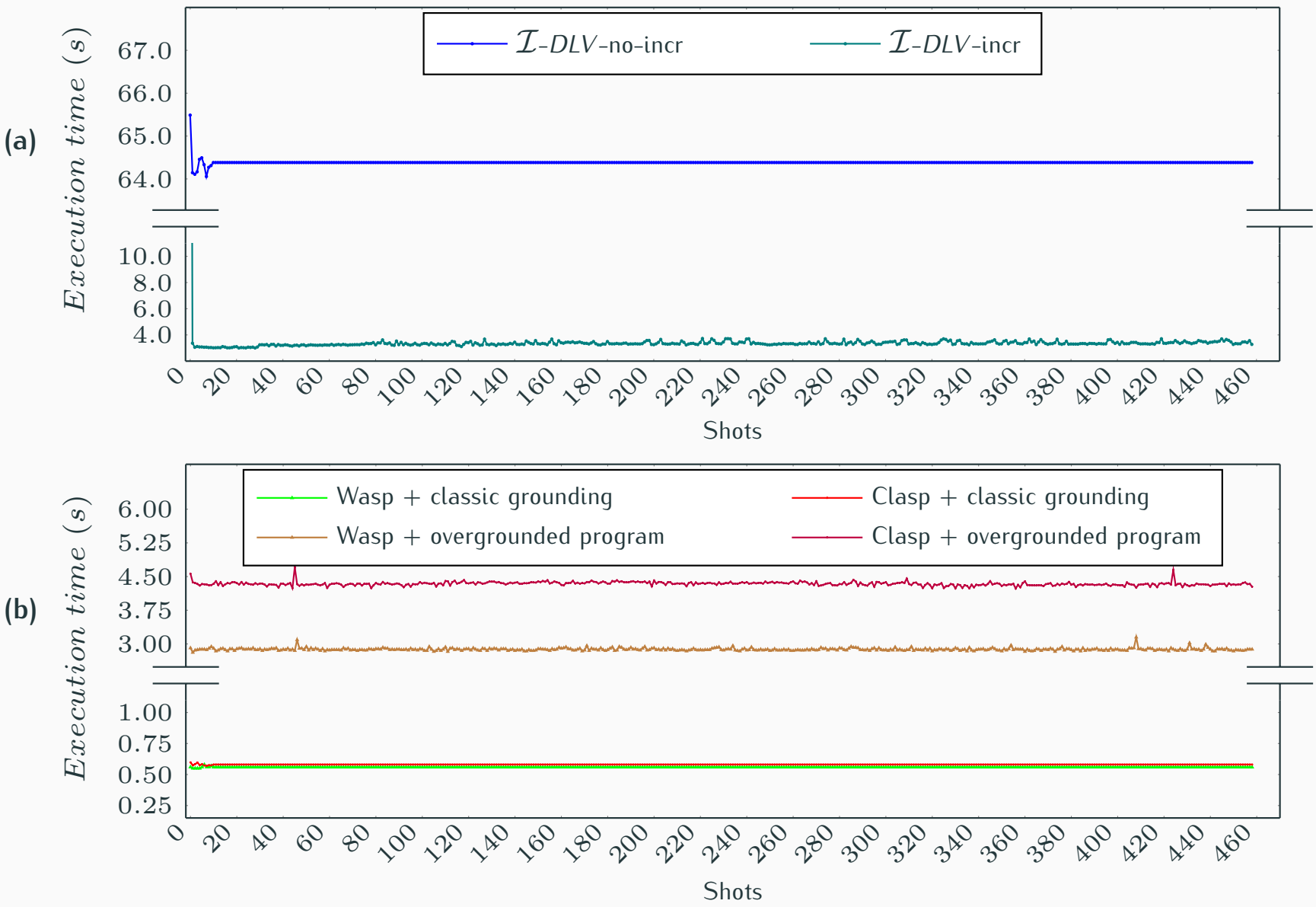


Figure 2: Pac-Man benchmark: (a) Grounding time. (b) Solving Time.

Multi-shot Pac-Man - Time Performance

(c)

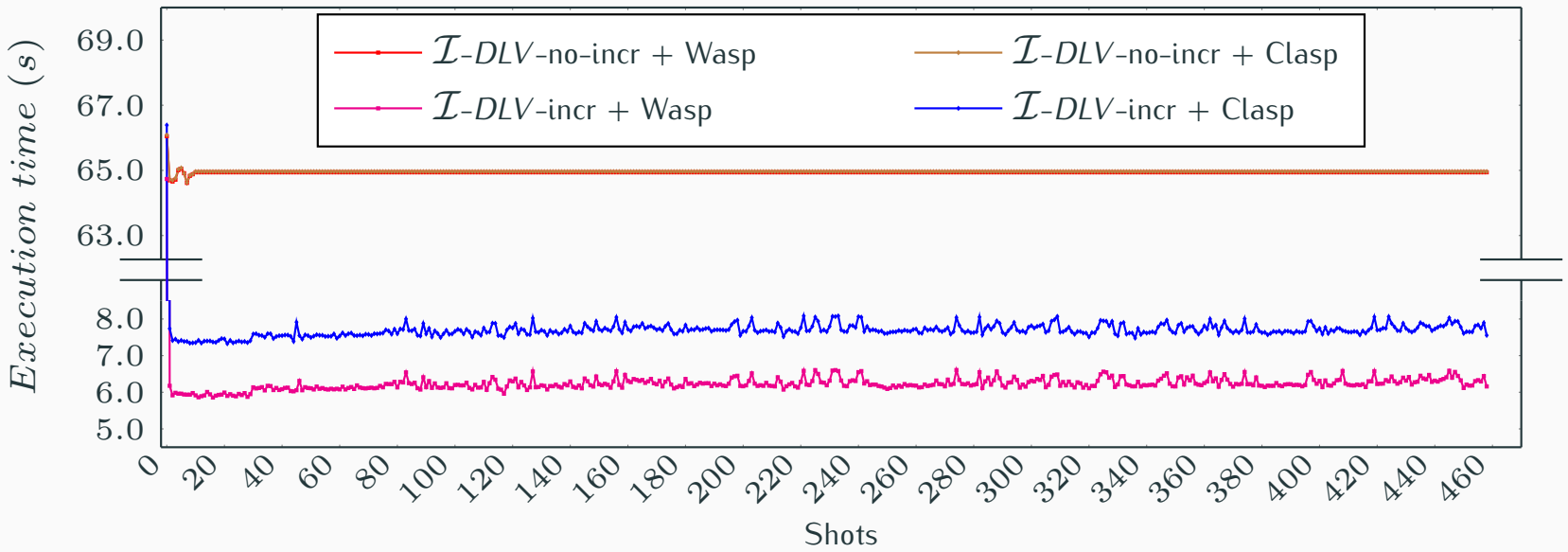
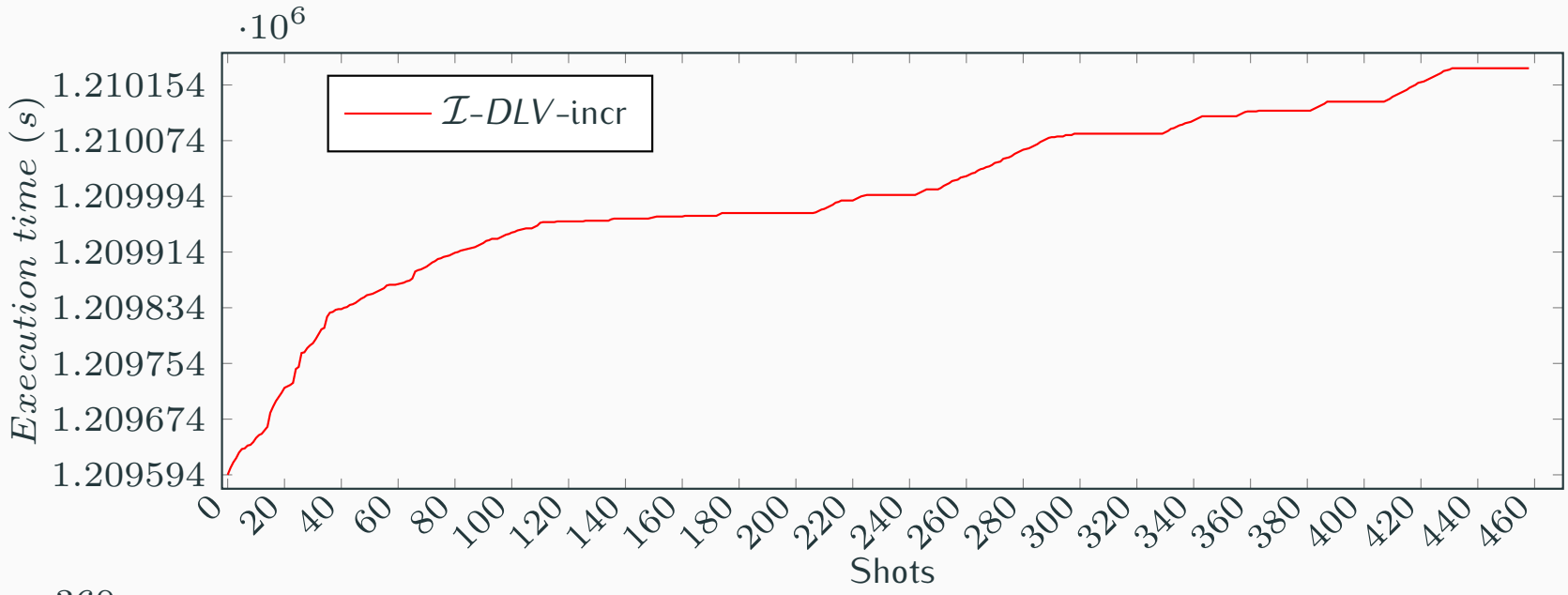


Figure 3: Pac-Man benchmark: (c) Total time.

Multi-shot Pac-Man - Memory Performance

(a)



(b)

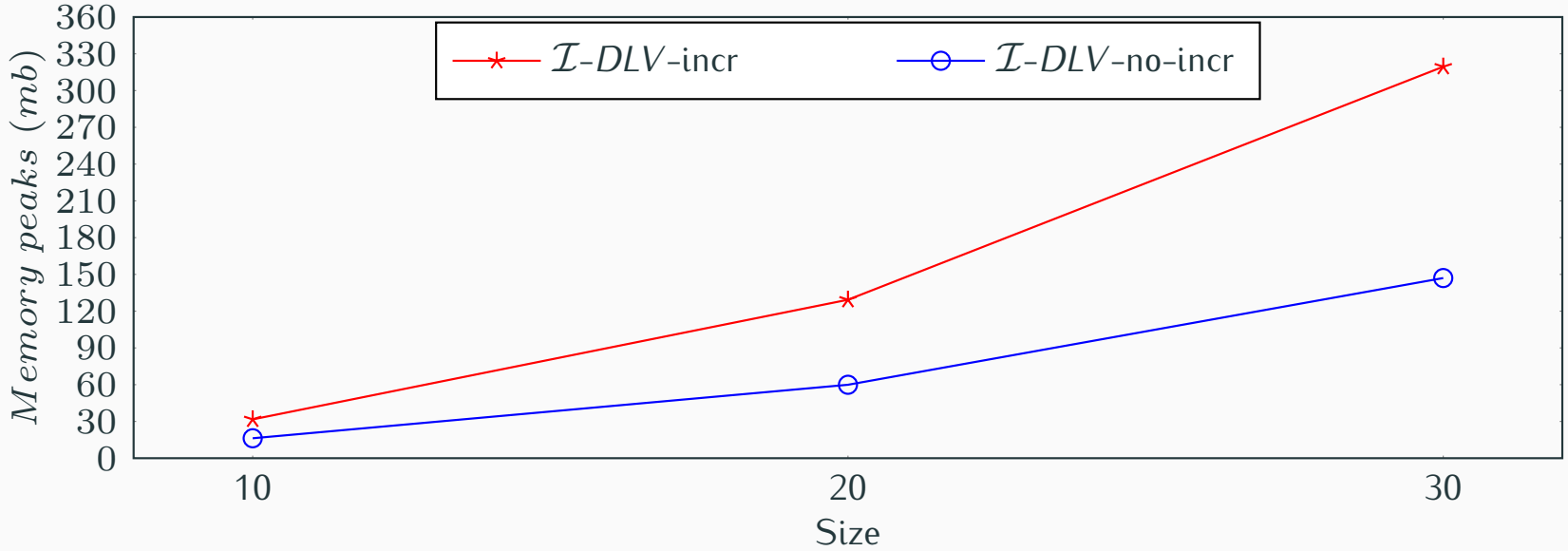


Figure 4: Pac-Man benchmark: (a) total number of rules after each iteration. (b) Memory peaks.

Conclusions

Conclusions and future work

Benefits

- Embeddings constitute good theoretical tool for modelling overgroundings
- Overgrounded programs do not require operational knowledge of the grounder
- Our early experiments show the potential of the approach

Future work

- We want to better define the theoretical foundations to show the classes of programs and the conditions over which “overgrounding” is possible
- We are interested in “interruptibility” of reasoning tasks

Thanks for your attention !