Techniques for Efficient Lazy-Grounding ASP Solving

Lorenz Leutgeb[1] Antonius Weinzierl[1,2]

September 19, 2017

[1] Knowledge-Based Systems Group
TU Wien, Vienna, Austria

[2] Department of Computer Science
Aalto University, Helsinki, Finland

Introduction

Motivation

- Answer-Set Programming (ASP): well-established KR formalism.
- Rule-based, nonmonotonic, expressive.
- Traditional ASP solving two-phased: ground and solve.
- Problem: grounding of exponential size in general and practice.

$\Rightarrow$ Grounding bottleneck of ASP.

Example

$$dom(1).\ dom(2).\ \ldots\ dom(20).$$

$$sel(X) \leftarrow dom(X), not\ nsel(X).$$
$$nsel(X) \leftarrow dom(X), not\ sel(X).$$

$$\leftarrow sel(Y), sel(X), X! = Y.$$

$$p(X_1, X_2, X_3, X_4, X_5, X_6) \leftarrow sel(X_1), sel(X_2), sel(X_3), sel(X_4), sel(X_5), sel(X_6).$$

Motivation

- Answer-Set Programming (ASP): well-established KR formalism.
- Rule-based, nonmonotonic, expressive.
- Traditional ASP solving two-phased: ground and solve.
- Problem: grounding of exponential size in general and practice.

$\Rightarrow$ Grounding bottleneck of ASP.

Example

$$dom(1).\ dom(2).\ \ldots\ dom(20).$$

$$sel(X) \leftarrow dom(X), not\ nsel(X).$$
$$nsel(X) \leftarrow dom(X), not\ sel(X).$$

$$\leftarrow sel(Y), sel(X), X! = Y.$$

$$p(X_1, X_2, X_3, X_4, X_5, X_6) \leftarrow sel(X_1), sel(X_2), sel(X_3), sel(X_4), sel(X_5), sel(X_6).$$

Lazy-Grounding ASP Solving

- Lazy-Grounding: interleave grounding and solving
  $\Rightarrow$ Avoid grounding bottleneck.
- Based on computation sequences.
- Existing solvers: Gasp, AsPeRiX, Omiga.
- No strong search space reducing techniques
  (like conflict-driven learning, backjumping, heuristics, etc).
- Grounding strongly-coupled to search (grounder backtracks).

$\Rightarrow$ Bad search performance.

The Alpha Solver

- Novel approach at lazy-grounding (first presented at LPNMR'17).
- Two components: lazy-grounder and modified CDNL solver.
- Blend lazy-grounding and state-of-the-art CDNL-based search.
- Represent rules by nogoods (similar to Clasp/ Clingo).
- Loose-coupling of lazy-grounding and search.
- Goal: Lazy-grounding with good search perfomance.

Overview of Alpha

Architecture of Alpha

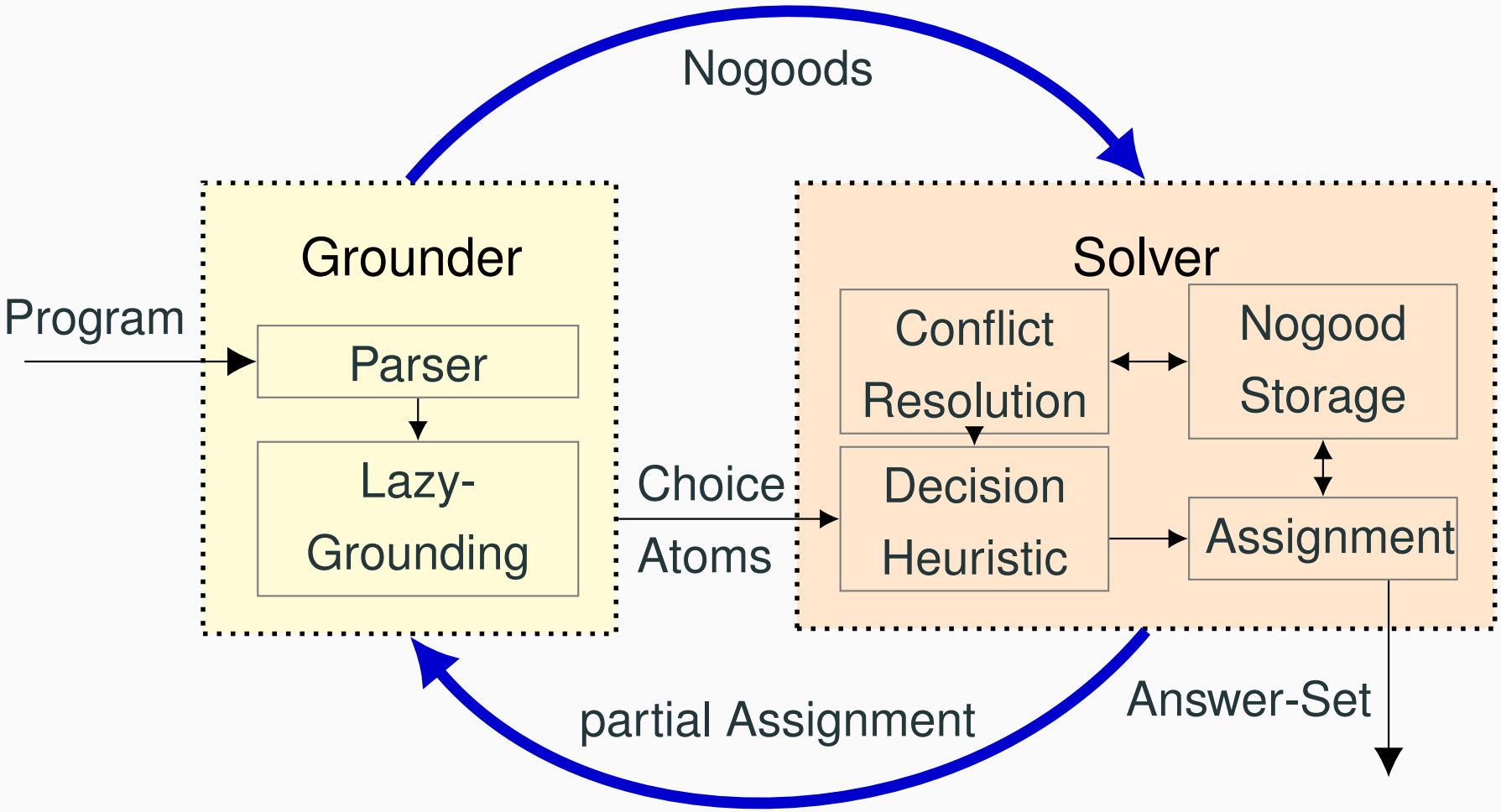


Figure 1: Alpha architecture, arrows indicate data flow.

Representing Rules

- Constraint: $\leftarrow sel(1), sel(2).$
- Nogood: $ng = \{\mathbf{T}sel(1), \mathbf{T}sel(2)\}$.
- Problem: distinguish $\leftarrow not\ a, b.$ from $a \leftarrow b.$
- State-of-the-art solution, add constraint expressing:

  "If a is true, there must be a rule firing with a in its head."
- Lazy-grounding cannot know all rules with a in the head:

  $$a \leftarrow p(X).$$
- Solution in Alpha:
  - Nogoods may have head and third truth value *must-be-true* .
  - Propagate: to *true* if head of nogood is implied, otherwise to *must-be-true* (and *false*).
  - Nogood $\{\mathbf{F}a, \mathbf{T}b\}$ distinct from $\{\underline{\mathbf{F}a}, \mathbf{T}b\}_1$ with head.

Representing Rules

- Constraint: $\leftarrow sel(1), sel(2).$
- Nogood: $ng = \{\mathbf{T}sel(1), \mathbf{T}sel(2)\}$.
- Problem: distinguish $\leftarrow not\ a, b.$ from $a \leftarrow b.$
- State-of-the-art solution, add constraint expressing:

  "If a is true, there must be a rule firing with a in its head."
- Lazy-grounding cannot know all rules with a in the head:

  $$a \leftarrow p(X).$$
- Solution in Alpha:
  - Nogoods may have head and third truth value *must-be-true* .
  - Propagate: to *true* if head of nogood is implied, otherwise to *must-be-true* (and *false*).
  - Nogood $\{\mathbf{F}a, \mathbf{T}b\}$ distinct from $\{\underline{\mathbf{F}a}, \mathbf{T}b\}_1$ with head.

Representing Rules by Nogoods

- Represent body of grounded rule $r\sigma$ with fresh atom $\beta(r, \sigma)$.

Example

- Let r be $sel(X) \leftarrow dom(X), not\ nsel(X).$
- Given assignment A with $dom(3)$ true.
- Grounder generates substitution $\sigma : X \mapsto 3$.
- Body representing atom: $\beta(r, 3)$.
- Generated nogoods:

$n_1 : \{\mathbf{F}\underline{\beta(r,3)}, \mathbf{T}dom(3), \mathbf{F}nsel(3)\}_1$ $\quad n_2 : \{\mathbf{T}\beta(r,3), \mathbf{F}dom(3)\}$

$n_3 : \{\mathbf{T}\beta(r,3), \mathbf{T}dom(3)\}$ $\quad n_4 : \{\mathbf{F}\underline{sel(3)}, \mathbf{T}\beta(r,3)\}_1$

Representing Rules by Nogoods

- Represent body of grounded rule $r\sigma$ with fresh atom $\beta(r, \sigma)$.

Example

- Let r be $sel(X) \leftarrow dom(X), not\ nsel(X).$
- Given assignment A with $dom(3)$ true.
- Grounder generates substitution $\sigma : X \mapsto 3$.
- Body representing atom: $\beta(r, 3)$.
- Generated nogoods:

$n_1 : \{\mathbf{F}\underline{\beta(r,3)}, \mathbf{T}dom(3), \mathbf{F}nsel(3)\}_1$ $\quad n_2 : \{\mathbf{T}\beta(r,3), \mathbf{F}dom(3)\}$

$n_3 : \{\mathbf{T}\beta(r,3), \mathbf{T}dom(3)\}$ $\quad n_4 : \{\mathbf{F}\underline{sel(3)}, \mathbf{T}\beta(r,3)\}_1$

Representing Rules by Nogoods

- Represent body of grounded rule $r\sigma$ with fresh atom $\beta(r, \sigma)$.

Example

- Let r be $\quad sel(X) \leftarrow dom(X), not\ nsel(X).$
- Given assignment A with $dom(3)$ true.
- Grounder generates substitution $\sigma : X \mapsto 3$.
- Body representing atom: $\beta(r, 3)$.
- Generated nogoods:

$n_1 : \{\mathbf{F}\underline{\beta(r,3)}, \mathbf{T}dom(3), \mathbf{F}nsel(3)\}_1 \qquad n_2 : \{\mathbf{T}\beta(r,3), \mathbf{F}dom(3)\}$

$n_3 : \{\mathbf{T}\beta(r,3), \mathbf{T}dom(3)\} \qquad n_4 : \{\mathbf{F}\underline{sel(3)}, \mathbf{T}\beta(r,3)\}_1$

Core Alpha Algorithm

Input: A (non-ground) program P.

Output: The answer-sets $\mathcal{AS}(P)$ of P.

Initialize $\mathcal{AS} = \emptyset$, assignment A, and nogood storage Δ.

Run lazy grounder, obtain initial nogoods Δ from facts.

while *search space not exhausted* **do**

- Propagate on Δ extending A.
- **if** *there exists conflicting nogood* **then**
  - Analyze conflict, learn new nogood, and backjump. **(a)**
- **else if** *propagation extended A* **then**
  - Run lazy grounder wrt. A and extend Δ. **(b)**
- **else if** *exists an applicable rule* **then** **(c)**
  - Guess as chosen by heuristic.
- **else if** *exists unassigned atom* **then** **(d)**
  - Assign all unassigned atoms to false.
- **else if** *no atom in A assigned to must-be-true* **then** **(e)**
  - $\mathcal{AS} \leftarrow \mathcal{AS} \cup \{A\}$
  - Add enumeration nogood and backtrack.
- **else** **(f)**
  - Backtrack.

return $\mathcal{AS}$

Propagation and 3WL

Propagation I

- Assignment A over **F**, **T**, **M**.

Example

Nogood $\delta = \{\mathbf{F}b, \mathbf{T}c, \mathbf{T}d\}$ $A = (\mathbf{F}b, \mathbf{T}c)$ implies $\mathbf{F}d$.

- All but one literal s of δ assigned the same value in A: δ is unit

$\Rightarrow$ Extend A with inverse of s.

- Unit-propagation on nogood δ two forms:
  - δ weakly-unit for s if $\delta \setminus A^{\mathcal{B}} = \{s\}$ and $\overline{s} \notin A^{\mathcal{B}}$,
    $\Rightarrow$ propagation to *must-be-true* and *false*.
  - δ strongly-unit for s if δ is nogood with head,
    $\delta \setminus A = \{s\}$, $hd(\delta) = s = \mathbf{F}a$, and $\mathbf{T}a \notin A$,
    $\Rightarrow$ propagation to *true*.

Propagation I

- Assignment A over **F**, **T**, **M**. Boolean-projection $A^{\mathcal{B}}$ maps **M** to **T**.

Example

Nogood $\delta = \{\mathbf{F}b, \mathbf{T}c, \mathbf{T}d\}$ $A = (\mathbf{F}b, \mathbf{T}c)$ implies $\mathbf{F}d$.

- All but one literal s of δ assigned the same value in A: δ is unit

$\Rightarrow$ Extend A with inverse of s.

- Unit-propagation on nogood δ two forms:
  - δ weakly-unit for s if $\delta \setminus A^{\mathcal{B}} = \{s\}$ and $\overline{s} \notin A^{\mathcal{B}}$,
    $\Rightarrow$ propagation to *must-be-true* and *false*.
  - δ strongly-unit for s if δ is nogood with head,
    $\delta \setminus A = \{s\}$, $hd(\delta) = s = \mathbf{F}a$, and $\mathbf{T}a \notin A$,
    $\Rightarrow$ propagation to *true*.

Propagation II

- Propagation: Given assignment A, nogoods Δ, find all $\delta \in \Delta$ that are weakly-unit and strongly-unit under A.

Definition

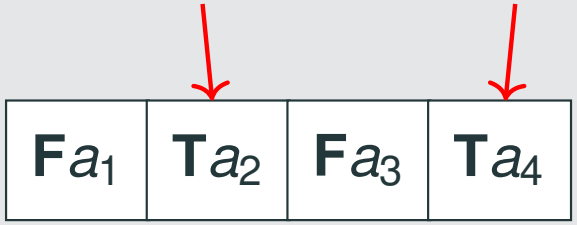
Let the *immediate unit-propagation* $\Gamma_\Delta(A)$, for set Δ of nogoods and assignment A be:

$$\begin{aligned}\Gamma_\Delta(A) = A \cup \{\mathbf{T}a \mid \exists \delta \in \Delta, \delta \text{ is strongly-unit under } A \text{ for } s = \mathbf{F}a\} \\ \cup \{\mathbf{M}a \mid \exists \delta \in \Delta, \delta \text{ is weakly-unit under } A \text{ for } s = \mathbf{F}a\} \\ \cup \{\mathbf{F}a \mid \exists \delta \in \Delta, \delta \text{ is weakly-unit under } A \text{ for } s = \mathbf{T}a\}\end{aligned}$$

Propagation is its least fixpoint, i.e., $\mathit{propagate}(A) = \mathit{lfp}\big(\Gamma_\Delta(A)\big)$.

Watched Literals

Example



$$A_1 = \emptyset$$

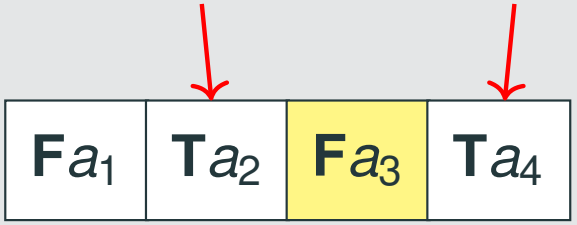
1. Watched are two (arbitrary) unassigned literals in nogood.
2. No watch on $\mathbf{F}a_3$, nothing to do.
3. Watched literal assigned, move watch.
4. Watch is moved, pointing at unassigned atom again.
5. Only one literal unassigned, nogood is unit. Assign $\mathbf{F}a_4$.

- Nogood with head, strongly-unit and weakly-unit?

$\Rightarrow$ Second set of watches; only 3 watches per nogood: 3WL.

Watched Literals

Example



$$A_1 = \{\mathbf{F}a_3 \qquad\qquad \}$$

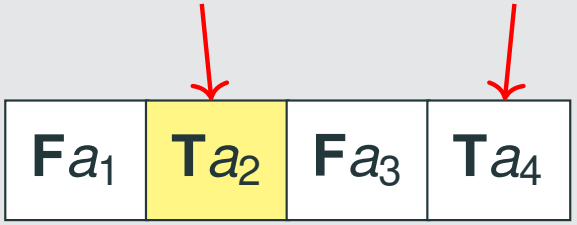
1. Watched are two (arbitrary) unassigned literals in nogood.
2. No watch on $\mathbf{F}a_3$, nothing to do.
3. Watched literal assigned, move watch.
4. Watch is moved, pointing at unassigned atom again.
5. Only one literal unassigned, nogood is unit. Assign $\mathbf{F}a_4$.

- Nogood with head, strongly-unit and weakly-unit?

$\Rightarrow$ Second set of watches; only 3 watches per nogood: 3WL.

Watched Literals

Example



$$A_1 = \{\mathbf{F}a_3, \mathbf{T}a_2 \qquad\qquad \}$$

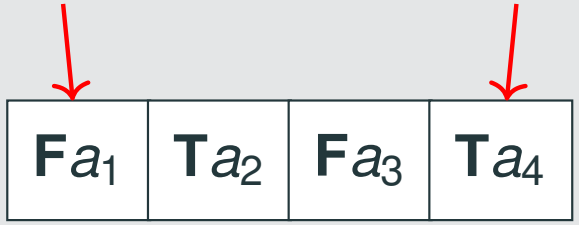
1. Watched are two (arbitrary) unassigned literals in nogood.
2. No watch on $\mathbf{F}a_3$, nothing to do.
3. Watched literal assigned, move watch.
4. Watch is moved, pointing at unassigned atom again.
5. Only one literal unassigned, nogood is unit. Assign $\mathbf{F}a_4$.

- Nogood with head, strongly-unit and weakly-unit?

$\Rightarrow$ Second set of watches; only 3 watches per nogood: 3WL.

Watched Literals

Example



$$A_1 = \{\mathbf{F}a_3, \mathbf{T}a_2 \qquad\qquad \}$$

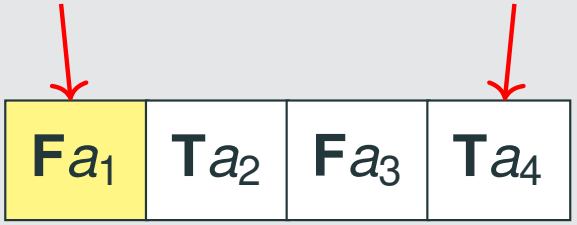
1. Watched are two (arbitrary) unassigned literals in nogood.
2. No watch on $\mathbf{F}a_3$, nothing to do.
3. Watched literal assigned, move watch.
4. Watch is moved, pointing at unassigned atom again.
5. Only one literal unassigned, nogood is unit. Assign $\mathbf{F}a_4$.

- Nogood with head, strongly-unit and weakly-unit?

$\Rightarrow$ Second set of watches; only 3 watches per nogood: 3WL.

Watched Literals

Example



$$A_1 = \{\mathbf{F}a_3, \mathbf{T}a_2, \mathbf{F}a_1 \qquad \}$$

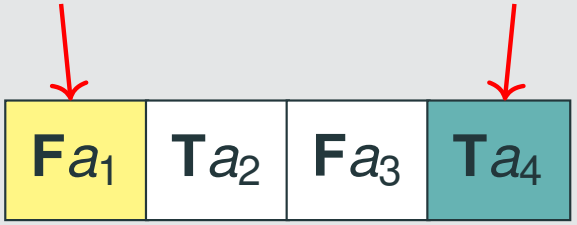
1. Watched are two (arbitrary) unassigned literals in nogood.
2. No watch on $\mathbf{F}a_3$, nothing to do.
3. Watched literal assigned, move watch.
4. Watch is moved, pointing at unassigned atom again.
5. Only one literal unassigned, nogood is unit. Assign $\mathbf{F}a_4$.

- Nogood with head, strongly-unit and weakly-unit?

$\Rightarrow$ Second set of watches; only 3 watches per nogood: 3WL.

Watched Literals

Example



$$A_1 = \{\mathbf{F}a_3, \mathbf{T}a_2, \mathbf{F}a_1, \mathbf{F}a_4\}$$

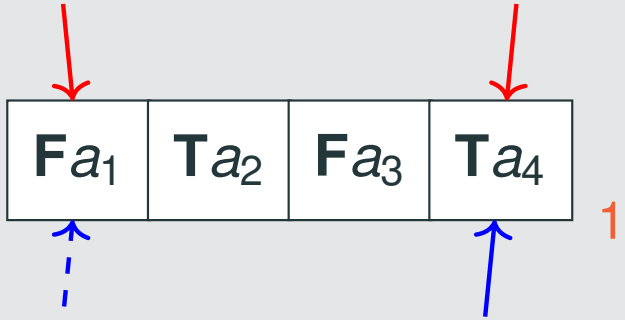
1. Watched are two (arbitrary) unassigned literals in nogood.
2. No watch on $\mathbf{F}a_3$, nothing to do.
3. Watched literal assigned, move watch.
4. Watch is moved, pointing at unassigned atom again.
5. Only one literal unassigned, nogood is unit. Assign $\mathbf{F}a_4$.

- Nogood with head, strongly-unit and weakly-unit?

$\Rightarrow$ Second set of watches; only 3 watches per nogood: 3WL.

Watched Literals

Example



1. Watched are two (arbitrary) unassigned literals in nogood.
2. No watch on $\mathbf{F}a_3$, nothing to do.
3. Watched literal assigned, move watch.
4. Watch is moved, pointing at unassigned atom again.
5. Only one literal unassigned, nogood is unit. Assign $\mathbf{F}a_4$.

- Nogood with head, strongly-unit and weakly-unit?

$\Rightarrow$ Second set of watches; only 3 watches per nogood: 3WL.

Watch Data Structure

Definition

A *watch structure* W for assignment A and set of nogoods Δ is mapping $W : \mathcal{A} \to \Delta^4$ with

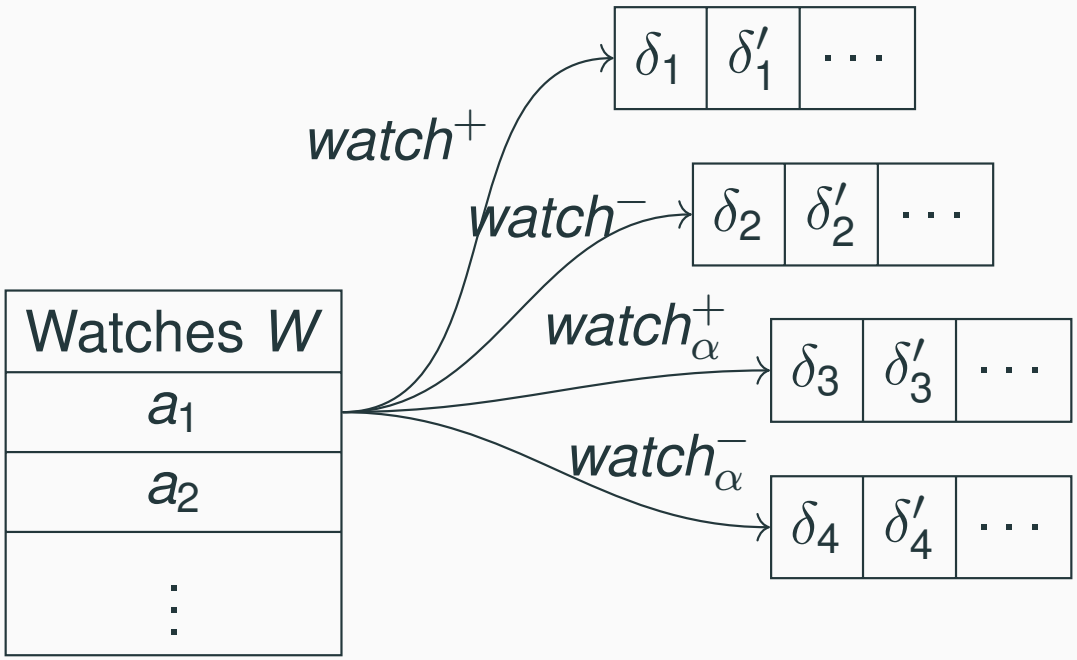
$$W(a) = \big(watch^+(a), watch^-(a), watch^+_\alpha(a), watch^-_\alpha(a)\big).$$

Watch Data Structure

Definition

A *watch structure* W for assignment A and set of nogoods Δ is mapping $W : \mathcal{A} \to \Delta^4$ with

$$W(a) = \big(watch^+(a), watch^-(a), watch^+_\alpha(a), watch^-_\alpha(a)\big).$$



Watch Structure Invariants

- Assignment grows (choices, propagation),
  and shrinks (backtracking, backjumping).
- Shrinking shall not cause watches to be moved.
- Intuitively: watched literals must be unassigned (growing) or
  if nogood no longer satisfied, watched literals must be unassigned
  (shrinking).
- The watches of a nogood point at either
  1. two unassigned literals, or
  2. one watched literal is assigned to satisfy the nogood and other watched literal is either unassigned or assigned at equal-or-higher decision level.
- For α-watch: *must-be-true* is considered unassigned.

Watch Structure Invariants

- Assignment grows (choices, propagation),
  and shrinks (backtracking, backjumping).
- Shrinking shall not cause watches to be moved.
- Intuitively: watched literals must be unassigned (growing) or
  if nogood no longer satisfied, watched literals must be unassigned
  (shrinking).
- The watches of a nogood point at either
  1. two unassigned literals, or
  2. one watched literal is assigned to satisfy the nogood and other watched literal is either unassigned or assigned at equal-or-higher decision level.
- For α-watch: *must-be-true* is considered unassigned.

Candidates for Watches

Definition (Ordinary Watches)

For nogood δ and assignment A, $s, s' \in \delta$ are *potential watches* if:

(i) $at(s)$ and $at(s')$ both unassigned in A; or

(ii) $at(s)$ complementary assigned, i.e., $\overline{s} \in A^{\mathcal{B}}$, and either s' unassigned in A or $dl^{w}(A, at(s')) \geq dl^{w}(A, at(s))$.

Definition (New α-watch)

For nogood δ with head, assignment A, $s_\alpha \in \delta$ with $hd(\delta) \neq s_\alpha$ is *potential α-watch* if:

(i) $at(s_\alpha)$ unassigned or assigned to *must-be-true* in A, or s_α complementary assigned in A; or

(ii) head is *true*, i.e., $\mathbf{T}at(hd(\delta)) \in A$ and $dl^{s}(A, at(s_\alpha)) \geq dl^{s}(A, at(hd(\delta)))$.

Consistent Watch Structure

Example

Consider $A = \{\mathbf{M}c, \mathbf{F}d\}$ with $dl^W(A, c) \leq dl^W(A, d)$.

Nogood	Potential watches	Potential α-watches
$\delta_1 = \{\mathbf{F}a, \mathbf{T}b, \mathbf{T}c, \mathbf{F}d, \mathbf{F}e\}_1$	$\mathbf{F}a, \mathbf{T}b, \mathbf{F}e$	$\mathbf{T}b, \mathbf{T}c, \mathbf{F}e$
$\delta_2 = \{\mathbf{F}c, \mathbf{F}d\}$	$\mathbf{F}c, \mathbf{F}d$	–
$\delta_3 = \{\mathbf{F}a, \mathbf{T}c\}_1$	–	$\mathbf{T}c$

- Watch structure W consistent for assignment A and nogoods Δ, if each $\delta \in \Delta$ watched by potential watches s, s' and a potential α-watch s_α if δ has head.

Consistent Watch Structure

Example

Consider $A = \{\mathbf{M}c, \mathbf{F}d\}$ with $dl^W(A, c) \leq dl^W(A, d)$.

Nogood	Potential watches	Potential α-watches
$\delta_1 = \{\mathbf{F}a, \mathbf{T}b, \mathbf{T}c, \mathbf{F}d, \mathbf{F}e\}_1$	$\mathbf{F}a, \mathbf{T}b, \mathbf{F}e$	$\mathbf{T}b, \mathbf{T}c, \mathbf{F}e$
$\delta_2 = \{\mathbf{F}c, \mathbf{F}d\}$	$\mathbf{F}c, \mathbf{F}d$	–
$\delta_3 = \{\mathbf{F}a, \mathbf{T}c\}_1$	–	$\mathbf{T}c$

- Watch structure W consistent for assignment A and nogoods Δ, if each $\delta \in \Delta$ watched by potential watches s, s' and a potential α-watch s_α if δ has head.

3WL Propagation

Input: Assignment A, new assignments Σ, watches W, nogoods Δ.

Output: Assignment A' or pair of A' and violated nogood d.

$A' \leftarrow A$

while $\Sigma \neq \emptyset$ **do**

  $\Sigma \leftarrow \Sigma \setminus \{\mathbf{X}a\}$ for some $\mathbf{X}a \in \Sigma$. `// Process new assignment.`

  $(\Delta, \Delta_\alpha) \leftarrow \begin{cases} (watch^+(a), watch^+_\alpha(a)) & \text{if } \mathbf{X} \in \{\mathbf{T}, \mathbf{M}\}, \text{ and} \\ (watch^-(a), watch^-_\alpha(a)) & \text{otherwise.} \end{cases}$

  foreach $\delta \in \Delta$ **do** `// Propagation to` $\mathbf{M}, \mathbf{F}$.

    if δ *is violated* **then**

      return (A', δ)

    else if δ *is weakly-unit for* s **then**

      Let $s' = \mathbf{M}b$ if $s = \mathbf{F}b$ and $s' = \mathbf{F}b$ otherwise.

      $(A', \Sigma) \leftarrow (A \cup \{s'\}, \Sigma \cup \{s'\})$

    Remove old watches from Δ, find new watches s, s'.

    $watch(at(s)) \leftarrow watch(at(s)) \cup \{\delta\}$, $watch(at(s')) \leftarrow watch(at(s')) \cup \{\delta\}$

  foreach $\delta \in \Delta_\alpha$ **do** `// Propagation to` $\mathbf{T}$.

    if δ *is strongly-unit* **then**

      $(A', \Sigma) \leftarrow (A \cup \{\mathbf{T}at(hd(\delta))\}, \Sigma \cup \{\mathbf{T}at(hd(\delta))\})$

    Remove old watches from Δ_α, find new watch s.

    $watch_\alpha(at(s)) \leftarrow watch_\alpha(at(s)) \cup \{\delta\}$

return A'

3WL Correctness

One can show:

Proposition

Let W be a watch structure W for a set of nogoods Δ that is consistent with an assignment A and let $A' \supseteq A$ be a larger assignment with $\Sigma = A' \setminus A$. Then, Algorithm 2 running on A, Σ, and W returns either

1. *a pair (A'', δ) such that A'' is a consequence of A' and Δ and $\delta \in \Delta$ is violated by A'', or*
2. *an assignment $A'' = propagate(A')$ and the modified watch structure is consistent with A'' and Δ.*

Evaluation

Benchmarks

- Compare Alpha (git-commit: 037b3f9), Omiga (git-commit: a65421f), AsPeRiX (2.5), Clingo (5.2.0).
- Performed on Linux machine, with two 12-core AMD Opteron 6176 SE CPUs and 128 GB RAM.
- Timeout 300 sec, memory limit 8GB.
- Request 10 answer sets from each solver.
- Available: `www.kr.tuwien.ac.at/research/systems/alpha`

Benchmark: Ground Explosion

Example

$dom(1). \ldots dom(20).$ $\quad sel(X) \leftarrow dom(X), not\ nsel(X).$ $\quad nsel(X) \leftarrow dom(X), not\ sel(X).$
$\leftarrow sel(Y), sel(X), X! = Y.$ $\quad p(X_1, X_2, X_3, X_4, X_5, X_6) \leftarrow sel(X_1), sel(X_2), sel(X_3), sel(X_4), sel(X_5), sel(X_6).$

Instance size	Alpha	Omiga	AsPeRiX	Clingo
8	1.37(0)	0.42(0)	5.54(0)	1.74(0)
10	1.48(0)	0.43(0)	0.02(0)	7.00(0)
16	1.60(0)	0.51(0)	0.03(0)	145.28(0)
18	1.64(0)	0.45(0)	0.03(0)	memout(1)
20	1.83(0)	0.46(0)	0.05(0)	memout(1)
50	1.31(0)	0.53(0)	0.11(0)	memout(1)
100	1.60(0)	0.81(0)	0.21(0)	memout(1)
500	2.19(0)	1.41(0)	1.06(0)	memout(1)
1000	2.30(0)	1.66(0)	2.21(0)	memout(1)

Table 1: Instance size is elements in domain. Runtime in seconds.

Benchmark: Graph-5-Colorability

- Similar to ASPCOMP encoding (average on 10 random instances).

Instance size	Alpha	Omiga	AsPeRiX	Clingo
10/40	1.41(0)	14.33(0)	31.10(1)	0.02(0)
20/80	1.53(0)	234.93(6)	128.79(4)	0.02(0)
30/120	1.59(0)	300.00(10)	230.23(7)	0.03(0)
40/160	2.54(0)	300.00(10)	217.17(7)	0.04(0)
50/200	2.31(0)	300.00(10)	300.00(10)	0.04(0)
100/400	4.24(0)	300.00(10)	300.00(10)	0.06(0)
400/1600	22.54(0)	300.00(10)	300.00(10)	0.45(0)
500/2000	33.85(0)	300.00(10)	300.00(10)	0.68(0)
750/3000	67.22(0)	300.00(10)	300.00(10)	1.46(0)
1000/4000	119.94(0)	300.00(10)	300.00(10)	2.66(0)

Table 2: Size is no. vertices / no. edges. Runtime in seconds.

Benchmark: Graph-5-Colorability

- Similar to ASPCOMP encoding.
- Easier instances.

Instance size	Alpha	Omiga	AsPeRiX	Clingo
10/40	1.41(0)	14.33(0)	31.10(1)	0.02(0)
20/80	1.53(0)	234.93(6)	128.79(4)	0.02(0)
50/50	1.88(0)	290.47(9)	0.24(0)	0.03(0)
50/100	2.05(0)	300.00(10)	0.45(0)	0.03(0)
50/200	2.31(0)	300.00(10)	300.00(10)	0.04(0)
50/300	74.39(2)	300.00(10)	300.00(10)	0.07(0)
50/400	253.80(8)	300.00(10)	300.00(10)	0.06(0)
50/500	168.76(4)	300.00(10)	300.00(10)	0.04(0)

Table 3: Size is no. vertices / no. edges. Runtime in seconds.

Benchmark: Cutedge

- Reachability after one edge removed from random graph.

Instance size	Alpha	Omiga	AsPeRiX	Clingo
100/30	12.59(0)	4.25(0)	0.78(0)	27.64(0)
100/50	11.87(0)	6.22(0)	1.79(0)	79.50(0)
200/30	22.90(0)	13.46(0)	13.29(0)	300.00(10)
200/50	45.95(0)	24.20(0)	35.18(0)	300.00(10)
300/10	16.92(0)	10.08(0)	8.54(0)	291.35(4)
300/30	59.58(0)	32.36(0)	72.09(0)	300.00(10)
500/10	62.46(0)	32.01(0)	70.38(0)	300.00(10)
500/30	300.00(10)	122.16(0)	300.00(10)	300.00(10)
400/10	40.97(0)	20.72(0)	27.61(0)	300.00(10)
400/30	300.00(10)	84.73(0)	284.71(4)	300.00(10)

Table 4: Size is no. vertices / percentage each edge present. Runtime in sec.

Benchmark: Reachability

- Reachability on random graph (positive program).

Instance size	Alpha	Omiga	AsPeRiX	Clingo
1000/4	2.13(0)	1.21(0)	0.77(0)	0.11(0)
1000/8	3.19(0)	1.63(0)	2.57(0)	0.21(0)
10000/2	10.95(0)	7.82(0)	31.11(0)	0.52(0)
10000/4	13.06(0)	22.55(0)	130.00(0)	1.09(0)
10000/8	16.62(0)	56.93(0)	300.00(10)	2.27(0)

Table 5: Size is no. vertices / multipler edges. Runtime in seconds.

Evaluation

Alpha is:

- on-par with other lazy-grounding solvers for grounding problems.
- much faster than other lazy-grounding solvers for search problems.
- slower than Clingo on search problems.
- not fastest, but already good compromise, especially if grounding problematic.

- Alpha freely available at:

  `https://github.com/alpha-asp/Alpha`

Conclusion

Related Work

- Alpha logical successor of Omiga (Dao-Tran et al., 2012):
  lazy-grounding (Rete), no efficient search.
- Computation sequence on GPUs (Dovier et al., 2016):
  nogoods-based, ground-and-solve, loop-nogoods etc. like Clasp.
- AsPeRiX (Lefèvre et al., 2017):
  lazy-grounding, no efficient search.
- More distanced:
  - s(ASP) (Marple et al., 2016):
    query-driven stable models, Prolog-like.
  - Lazy-MX (de Cat et al., 2015):
    lazy model expansion for FO(ID), no loops over negation, no ASP.

Conclusion

- Alpha combines lazy-grounding and CDNL search.
- Efficient propagation techniques 3WL for *must-be-true* and two forms of propagation.
- Benchmarks: Alpha effective where grounding and solving an issue.
- First lazy-grounding ASP solver for non-trivial search problems.

- Open issues / future work:
  - Using dependency information (e.g., SCCs) for faster solving.
  - Optimizations: forgetting, nogood minimization, stratified propagation.
  - Heuristics for grounding.

Thank you.

Conclusion

- Alpha combines lazy-grounding and CDNL search.
- Efficient propagation techniques 3WL for *must-be-true* and two forms of propagation.
- Benchmarks: Alpha effective where grounding and solving an issue.
- First lazy-grounding ASP solver for non-trivial search problems.

- Open issues / future work:
  - Using dependency information (e.g., SCCs) for faster solving.
  - Optimizations: forgetting, nogood minimization, stratified propagation.
  - Heuristics for grounding.

Thank you.