

Drop FCG  
 Drop disj Just  
 Split “transition systems” (for studying propagation power) and “proof theory” (for size of the constructed proof)  
 Wasp vs clasp?

TODO  
 TODO  
 TODO  
 TODO

## 1 Rationale and Positioning with Regard to State of the Art

Answer Set Programming (ASP) is well-established as a knowledge representation paradigm. ASP stems from the idea that stable model semantics of logic problems can be utilized to encode search problems. It is especially useful for encoding NP-complete problems and this property is recently exploited in application domains such as machine learning, robotics, data-integration, biology, planning, phylogenetic inference and product configuration. ASP has a rich first-order language ASP-Core2 to effectively represent knowledge in the mentioned domains and has efficient solvers built on top of conflict-driven clause learning (CDCL) from satisfiability solving and lazy clause generation.

Traditional ASP systems work in two phases: **grounding** and **solving**. This works well for problems where the grounding is not too large.

Programs written in ASP-Core2 language has first-order variables in them and these need to be eliminated to feed the program into an ASP solver. This process is called grounding. To this day, most of the effort on ASP focused on improving the efficiency of ASP solvers while the grounding methods changed very little. Recently, though, the community started showing interest in better grounding methods since there are interesting problems where grounding is the bottleneck [4]. Examples to such problems include planning with a potentially large number of time-steps; queries such as reachability on large graphs; and problems that have a lot of unnecessary information when fully grounded but are simple in nature. There are already ongoing efforts to address this grounding bottleneck. Intelligent grounding [10] and rule decomposition [6] aim to reduce the size of grounding where possible. For query problems, top-down evaluation utilizing *magic sets* [5, 2] is used. For planning problems, *incremental grounding* [17] is used to ground the next time steps only if a solution does not exist for the current number of time-steps. This general idea of grounding only as much as needed is called lazy grounding. Bottom-up lazy grounding systems include OMIGA [13], GASP [12], ASPeRiX [20] and the recently introduced ALPHA [25] that integrates lazy grounding with a CDCL solver. Also top-down lazy grounding techniques (for a language related to ASP) [14] and top-down stable model generation techniques that avoid grounding [22, 23] exist.

Lazy grounding systems interleave the two phases, with as aim to **only ground those parts** that are **relevant** for the solver, to overcome the **grounding bottleneck**.

While these lazy grounding methods outperform the usual ground-and-solve approaches on problems where grounding is the bottleneck, they are far from competitive on standard ASP problems [16]. This is not surprising since lazy grounding is tailored for the former kind of problems, but currently we do not have an understanding of where the disparity in performance on the latter comes from. One reason may be that ground-and-solve approaches are more matured since they have been being optimized for years by now, but some part of the performance difference should be caused by differences in algorithms, such as missed propagation due to not constructing parts of the grounding. In this project, we investigate all the factors that determine the differences in speed between these systems. Based on

that analysis, we develop new lazy grounding techniques and systems that circumvent the greatest bottlenecks. Our hypothesis is that these improvements will prove valuable both in traditional applications and on applications where lazy grounding systems are superior to ground-and-solve systems.

The focus of this project is on lazy grounding with a CDCL solver as backend, since CDCL has been responsible for one of the greatest performance boosts in SAT- and ASP-solving. We believe CDCL to be a strong foundation to build search algorithms on. In particular, the basis of this project will be ALPHA, the only CDCL-based lazy grounding algorithm for ASP. In this project, we investigate to which extent it is possible to integrate top-down (goal-directed) lazy grounding techniques (such as [23] and those developed for the IDP system [14]) in ALPHA and more generally, extend the ALPHA algorithm with techniques that compensate for its weaknesses, while preserving its strengths. In pilot research for the current project, we took a first, promising, step in this direction by using a justification-based top-down analysis whenever ALPHA arrives in a conflict it cannot easily resolve [7].

On traditional applications, lazy grounding systems are often orders of magnitude slower than ground-and-solve systems. We research **why** there is such a big performance gap and develop **novel lazy grounding algorithms** that do not suffer from this.

We ground our research by testing it on ASP models for Multi Agent Path Finding (**MAPF**), which can be modeled as a planning problem or a graph problem [?]. **MAPF** is both hard and related to real world applications such as automated warehouses. Also, it has variants which are practically harder to solve (and at least as hard theoretically) [REFER TO THESIS], which will allow for testing our approach on incrementally harder versions of the same fundamental problem.

The outcome of this project is manifold. First of all, there will be an increased understanding of how different techniques to avoid grounding relate. Second, there will be a clear overview of which factors influence the performance differences between lazy grounding systems and traditional ground-and-solve systems. Third, we develop novel lazy grounding techniques that combine elements of top-down and bottom-up reasoning. Fourth, we ground and evaluate our research on **MAPF** using ASP.

## 2 Scientific Research Objectives

We categorize our objectives as follows: **(i)** deeply understanding the literature on lazy grounding and analyzing them **(ii)** developing novel algorithms for lazy grounding, **(iii)** applying and evaluating said algorithms. To achieve **(i)**, we elaborate our objectives as follows:

**Objective 1.** Develop a clear understanding of existing lazy grounding algorithms for ASP and their relationships with each other, and the cases where they are best used. ▲

**Objective 2.** Identify the factors that make the existing algorithms perform poorly, and quantify the effects of these factors. ▲

To develop novel algorithms for lazy grounding **(ii)**, we believe a good starting point would be to combine bottom-up and top-down reasoning approaches. Also, we can increment on existing algorithms in consideration of their current shortcomings.

**Objective 3.** Extend the current algorithms by addressing the most important factors identified in **OBJ2**. ▲

**Objective 4.** Develop lazy grounding algorithms that combine top-down and bottom-up reasoning approaches ▲

We propose concrete ways to achieve **OBJ3** and **OBJ4** objectives in our work packages. We would like to evaluate (iii) our novel algorithms by applying them to challenging problems and existing lazy grounding benchmarks.

**Objective 5.** Evaluate our novel algorithms on **MAPF** and existing benchmarks for lazy grounding. ▲

The result of this project is a **clear analysis of the strengths and weaknesses** of existing lazy grounding techniques and **novel algorithms that overcome those weaknesses**. As a result, answer set programming will be applicable to a **broad class of problems** that are currently unsolvable due to the grounding bottleneck.

### 3 Research Methodology and Work Plan

To analyze the existing algorithms, we study the existing literature and test them on **MAPF**. This way, we can pinpoint the shortcomings of these algorithms on a relevant and practical problem.

For implementing both our novel algorithms and existing algorithms in the literature, we use the recently developed and promising ASP solver WASP [1]. WASP is a state-of-the-art solver with useful features focusing on extendibility. This means we can more easily modify the inner workings of the solver compared to other solvers, which will prove useful as we describe in our work packages. We believe using an existing and well-engineered solver like WASP will save us precious time from implementing our own. WASP is developed in University of Calabria and we plan a research visit there to utilize their expertise. Our strategy will be to implement our lazy grounding algorithms as custom theory propagators, which can be efficiently done using WASP [1]. For experimental analysis, we reimplement ALPHA on WASP in a modular way such that we can enable or disable different parts of the algorithm. This would allow us to compare different fundamental techniques instead of just their concrete implementations. For experimental validation of both our novel algorithms and existing lazy grounding algorithms, we will use existing benchmarks for ASP solvers (e.g. those in ASP competitions), other benchmarks that are used for lazy grounding specifically and existing models for **MAPF**.

Fourth, in its design, a strong focus was put on extensibility, for instance supporting by the addition of custom theory propagators [15]. Our strategy is to implement lazy grounding techniques on top of this in the form of such propagators.

For the **experimental analysis**, we reimplement ALPHA on top of WASP in such a way that parts of the algorithm can be enabled/disabled to achieve a fair comparison between fundamental techniques, not concrete implementations of them. This reimplementation also forms the basis to build our novel lazy grounding techniques on.

For the **experimental validation** of our algorithms (and existing techniques) we use traditional ASP benchmarks (e.g., those encountered in ASP competitions), benchmarks previously used for lazy grounding, and novel ASP models of fluid construction grammar (which we develop in **WP??**). While not explicitly repeated in the descriptions below, each of the work packages **6-??** contains a validation component consisting of testing the performance of our algorithms and comparing them with other approaches on the benchmarks just mentioned.

**Work Package 1.** Work Package 1: Utilize existing MAPF models for ASP to test the shortcomings of existing lazy grounding algorithms. Optimization variants of MAPF are NP-complete and the problem is highly relevant for many industrial applications. Due to the

exploding number of possible paths as the path lengths increase, grounding step is a time-consuming part of the solving process. This is currently best addressed by multi-shot ASP solving, which can be considered a problem specific lazy grounding method. Furthermore, there are even more challenging variants of **MAPF** [my paper] and this could allow us to evaluate our algorithms on incrementally more challenging versions of the same fundamental problem. ▲

**Work Package 2.** Work Package 2: Literature review and comparison (OBJ1). We thoroughly study existing ASP methods that utilize lazy grounding, i.e., methods that either intertwine grounding and solving or skip the grounding step altogether. ▲

**Work Package 3.** *Identify the different factors responsible for the performance gap between alpha and state-of-the-art ASP systems.* (OBJ2) We do this by theoretically analyzing the search algorithms performed by the different solvers. In a preliminary analysis, we have already identified several factors.

**Missing completion** The ALPHA algorithm does not detect when all rules that have a certain atom in the head have been grounded.<sup>1</sup> Since it does not detect this, the algorithm loses the ability to propagate top-down.

**Propagation in ungrounded parts** Some rules/constraints are not grounded by ALPHA, even though they would propagate. A simple example is the constraint “:  $\neg p(X)$ .”, which states that all atoms over  $p$  are false. In ALPHA, this constraint is only presented to the solver when some  $p(x)$  is assigned true. Thus, for such constraints, ALPHA essentially employs a generate-and-test strategy.

**Missing unfounded-set propagation** One important propagation mechanism, that distinguishes ASP solvers from SAT solvers is unfounded-set propagation. That is: as soon as there is only one possible non-cyclic justification for a true atom, that justification is propagated to hold. In other words, this propagation ensures that the final solution will contain no unfounded sets. This kind of propagator is implemented in all modern native ground ASP solvers. Again, this kind of propagation is not present in ALPHA due to the fact that the solver does not have access to all the rules that can derive a given atom.

**Choice variables** ALPHA only makes choices on certain variables, namely those representing whether a certain rule fires or not. However, making such a restriction results, for standard ground-and-solve systems in an exponentially weaker proof system compared to allowing choices both on whether a rule fires and on whether an atom holds or not [3].<sup>2</sup>

**Visible choice variables** ALPHA also limits its choices in a different way. Namely, it only allows for choices on variables that are actually seen by the solver. Initially, this set is very small, but it grows during run-time.

**Engineering** A last factor that influences the performance difference between ALPHA and state-of-the-art ground-and-solve systems, is the fact that in the latter, years of optimization of low-level data-structures, and engineering have been spent, while ALPHA is a one-man java implementation. This is a factor that is not of scientific interest; still, it is interesting to know how much of the difference in speed is explained by this factor.

Besides these factors, more things can be in play. Identifying them is one of the challenges of this work package. ▲

In order to quantify the importance of the effects discovered in **WP3** on applications, we will slow down WASP by (one by one and jointly) handicapping it to lose propagation power or limit its other options. For evaluating certain criteria, such as the choice heuristics, we

---

<sup>1</sup>In very rare case, it does detect this.

<sup>2</sup>In the context of an ongoing project, Antonius Weinzierl is currently researching the ramifications of allowing choices on all variables instead of using the restriction that is currently implemented in ALPHA.

need to know what ALPHA would do at any point of the search. The following work package will achieve this.

**Work Package 4. *Implementation of alpha techniques in wasp.* (with Calabria)**

We first reimplement the techniques from ALPHA in WASP in such a way that it can run in “ghost mode”, i.e., that during runtime of WASP, we keep track of which rules would be visible to the solver without actually interfering in the search process. This ghost-mode implementation will be vital for tackling **WP5**. Next to using it for experiments, this reimplementation of ALPHA techniques also serves as the basis for our improvements detailed in **WP6–9**. From the perspective of fundamental research, this work package may seem trivial, but it is key to accurately and thoroughly tackle **WP5–9**. ▲

**Work Package 5. *Quantify effect of different factors.* (OBJ3,9)**

For each factor identified in **WP3** we evaluate its effect on performance. In order to do this, we limit WASP’s propagation or choice power (or limit it in other ways, depending on which factors are found). We will both test individual handicaps and joint handicaps. For two of the above mentioned factors, namely **visible choice variables** and **propagation in ungrounded parts**, we need to be able to simulate ALPHA’s behavior during runtime of WASP; in particular we need to know which rules would have been grounded by ALPHA at that point during the search. For other factors, such as disabling top-down propagation, this is not required. All that is needed there is postponing this kind of propagation until a complete assignment is found.

We will compare the unmodified version of WASP with handicapped versions on all the benchmarks mentioned above, i.e., on the ASP competition, benchmarks designed for lazy grounding and on **MAPF**. While solving time is the metric we aim at reducing in the end, here we focus on metrics such as number of conflicts and number of choices made by the solver. The reason to do this is that the solving time metric is polluted by the ghost-mode implementation running in the background. ▲

**Work Package 6. *Domain estimation.***

We develop techniques that, for each rule in a (non-ground) ASP program, determine bounds on the set of its relevant instantiations. These bounds can be symbolically represented or explicitly enumerated (though the latter should only be done when they are small enough). These bounds will (in upcoming work packages) be used either to check if all instantiations of a given rule have been added to the grounding, or to generate instantiations of the rule. We distinguish two types of estimations: *static analysis*, where the domains are determined solely based on the input program and *dynamic analysis*, where the domains are also based on the state of the solver. Many techniques in this direction have been developed in a variety of contexts, ranging from symbolic techniques to derive a first-order representation of such bounds [26], to syntactic restrictions on ASP programs that guarantee a finite grounding [24, 18, 9], and intelligent grounding techniques [10]. The aim of this work package is to investigate which of these are usable in the context of lazy grounding, which guarantees they offer (e.g., with respect to finiteness) and if need be, develop new techniques that derive an estimation of all the relevant instantiations of a given rule in a given program. ▲

**Work Package 7. *Top-down propagation.* (OBJ5,6,9) (with Calabria)**

To compensate for missing top-down propagation, we see several options.

- (i) Using the techniques from **WP6**, the grounder can observe that for each element in the domain, an instantiation of the rule was added to the solver. When this happens for all rules that can derive a certain atom, the completion (i.e., the constraint that if the atom in question holds, at least one rule should derive it) for that atom can be given to the solver.
- (ii) We develop a dynamic algorithm that, during search, keeps track of which atoms are completed *given the current assignment*. Such information is directly available for facts

(input of the program) and propagates bottom-up through the program: whenever each variable in a given rule is bound by a completed predicate, this rule itself is completed. When all rules defining a given predicate are completed, this predicate is also completed. However, we expect that working on a predicate level will rarely be sufficient and we will research more fine-grained mechanisms to keep track of this. Given this information, we can device a custom propagator that propagates top-down for completed atoms.

(iii) We develop a novel propagation mechanism that performs top-down propagation even though not all rules have been grounded. The way it works is: whenever an atom is true in a given interpretation, but no more rules exist that can derive it, a call to the grounder is made to request to ground more rules that can derive it. In addition, this idea can be refined to include a two-watched literal scheme so that the solver always has at least two rules that can derive each true atom. The biggest challenge to get this working is researching how the grounder can get new instantiations for rules deriving an atom when the novel propagator requests them. We investigate several ways to achieve this:

(a) The domain estimation techniques from **WP6** provide an upper bound on the set of all possible instantiations of a rule. If the derived domain is finite, we can iterate over it to get new instantiations of that rule.

(b) In [7], we used a justification analysis to explain *why* there are no rules that can derive a certain atom. A similar justification analysis but performed earlier in the search process can explain which instantiations are worthwhile considering in the current assignment.

Successful completion of such a propagator would effectively allow our solver to reason and ground goal-directed. To illustrate this, consider a planning problem over a fixed (discrete) interval. In this case, the most important constraint is that the goal is reached in the final time. Our top-down propagation would then immediately generate (at least) one rule that can achieve this. I.e., it finds an action at the one-but-last timepoint that achieves the goal. After making a choice on this, our propagator can either generate more rules at that point in time or go back in time one more step. ▲

**Work Package 8. *Unfounded set propagation.* (OBJ5,6,9)** We develop a variant of the unfounded set algorithm that works when not all rules for certain atoms have been grounded. To achieve this, we follow a strategy similar to approach (iii) of **WP7**. That is, we develop a new propagator that behaves like the classical unfounded set propagation (the source pointer approach) except that whenever it detects an unfounded set (or whenever the classical algorithm would propagate), it goes back to the grounder and requests more rules for at least one of the atoms involved (those that are suspected to be part of an unfounded set). If the grounder can generate more such rules, there is the possibility of “escaping” out of the unfounded set; if not, the planned propagation can take place. As with **WP7(iii)**, the biggest challenge is figuring out where the grounder can get more instantiations for a given rule. That problem is already tackled in that work package. We will also research how such a propagator can be optimized, effectively taking the new information it now has into account. ▲

**Work Package 9. *Non-ground propagation.* (OBJ5,6,9)** To compensate for missed propagation on parts of the grounding that are not constructed, we develop a custom propagator that instantiates rules on the moment that their instantiation would propagate. Research in this direction has already been done in the group of Francesco Ricca [11], limited to constraints. The restriction to constraints is quite limiting, since this means the techniques no longer work for instance when a constraint is reified. In this project, we extend their ideas for arbitrary rules and furthermore, integrate this in our lazy grounding algorithm. Effectively, this boils down to developing a *lazy clause generation* propagator for arbitrary rules. There are different possibilities with respect to *how lazy* we wish this propagator to be, for instance

by also allowing so-called *lazy explanations* [19]. ▲

**Work Package 10. Algorithms for improved lazy grounding. (OBJ5,9)** In this work package, we develop algorithms that compensate for the factors impacting the speed of lazy grounding solvers identified in **WP3**, except for those covered in **WP6–9**. Few details about this work package can be given, except that we will allocate time to each of the factors depending on its impact on performance, as identified in **WP5**. ▲

## Organization

We plan a research visit to University of Calabria to work on WPsomething to benefit from their expertise on WASP. Timing of the visit will coincide with the time allocated for WPsomething. It will be a short visit of no longer than two months and will not cover all the time allocated for WPsomething.

## Analysis of Risks and Bottlenecks

The research goals of this project reasonably ambitious; with the exception of **WP4** and WPMAPF, each individual work package is innovative and will lead to publications in high-impact conferences and journals. With high gains, often high risks come. We mitigated them by designing the work packages so that the dependencies are limited. Figure 2 contains an overview of the dependencies between all the work packages. Here, full arrows denote a crucial dependency, where the second work package cannot be completed (in the form as described above) unless the first is successful. Dashed arrows denote a dependency where the second work packages can still be performed even if the first is only partly completed/successful.

The work package on which the most other packages depend is **WP4**. However, this work package is concerned with re-implementing existing techniques from ALPHA in WASP; this work package consists of no research. Since we collaborate for this Francesco Ricca (the head of the group that developed WASP), the risk of failure of **WP4** is extremely low. In the unlikely case that this work package should fail, we would consider implementing our novel techniques directly on the existing ALPHA solver, but in this case, we have to drop **WP5**.

There is also a dependency of work packages **WP7** and **WP8** on **WP6**, since the former two use techniques from the latter two to find instantiations of variables. This dependency is not crucial since the proposal contains several different ideas as to how the instantiations can be found. The (strong) dependency between **WP3** and **WP10** is since in **WP10** we develop algorithms to compensate for shortcomings identified in **WP3**.

In case we experience set-backs, such as certain methods not working out in general, we will not hesitate to restrict our attention to a subset of all ASP programs for which we can get the techniques to work. Examples of such subsets include normal logic programs,  $\omega$ -restricted programs [24],  $\lambda$ -restricted programs [18], and generate-define-test programs [21]. These classes cover the majority of practical problems modeled in ASP, but impose syntactic restrictions that might enable techniques that do not work in general.

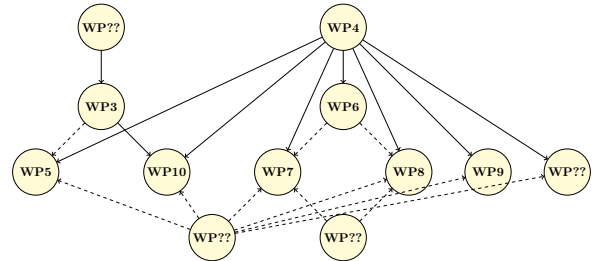
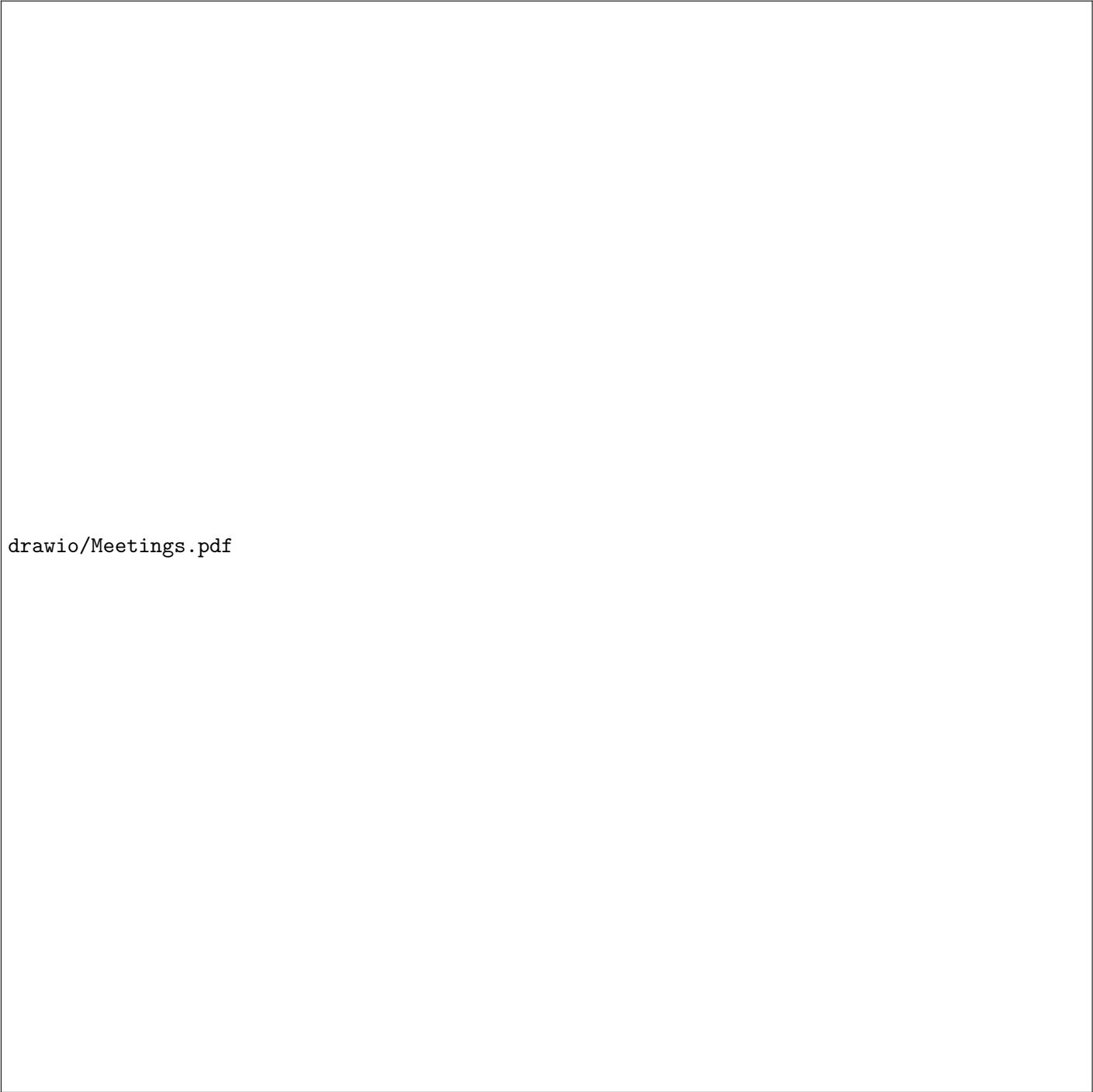


Figure 2: Dependencies between the different work packages.

## References

- [1] Mario Alviano, Carmine Dodaro, Wolfgang Faber, Nicola Leone, and Francesco Ricca. WASP: A native ASP solver based on constraint learning. In Cabalar and Son [8], pages 54–66.
- [2] Mario Alviano, Wolfgang Faber, Gianluigi Greco, and Nicola Leone. Magic sets for disjunctive datalog programs. *Artif. Intell.*, 187:156–192, 2012.
- [3] Christian Anger, Martin Gebser, Tomi Janhunen, and Torsten Schaub. What’s a head without a body? In Gerhard Brewka, Silvia Coradeschi, Anna Perini, and Paolo Traverso, editors, *ECAI*, pages 769–770. IOS Press, 2006.
- [4] Marcello Balduccini, Yuliya Lierler, and Peter Schüller. Prolog and ASP inference under one roof. In Cabalar and Son [8], pages 148–160.
- [5] Francois Bancilhon, David Maier, Yehoshua Sagiv, and Jeffrey D Ullman. Magic sets and other strange ways to implement logic programs (extended abstract). In *Proceedings of the Fifth ACM SIGACT-SIGMOD Symposium on Principles of Database Systems*, PODS ’86, pages 1–15, New York, NY, USA, 1986. ACM.
- [6] Manuel Bichler, Michael Morak, and Stefan Woltran. lpopt: A rule optimization tool for answer set programming. In Manuel V. Hermenegildo and Pedro López-García, editors, *Logic-Based Program Synthesis and Transformation - 26th International Symposium, LOPSTR 2016, Edinburgh, UK, September 6-8, 2016, Revised Selected Papers*, volume 10184 of *Lecture Notes in Computer Science*, pages 114–130. Springer, 2016.
- [7] Bart Bogaerts and Antonius Weinzierl. Exploiting justifications for lazy grounding of answer set programs. In Jérôme Lang, editor, *Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence, IJCAI 2018, July 13-19, 2018, Stockholm, Sweden.*, pages 1737–1745. ijcai.org, 2018.
- [8] Pedro Cabalar and Tran Cao Son, editors. *Logic Programming and Nonmonotonic Reasoning, 12th International Conference, LPNMR 2013, Corunna, Spain, September 15-19, 2013. Proceedings*, volume 8148 of *LNCS*. Springer, 2013.
- [9] Marco Calautti, Sergio Greco, Francesca Spezzano, and Irina Trubitsyna. Checking termination of bottom-up evaluation of logic programs with function symbols. *TPLP*, 15(6):854–889, 2015.
- [10] Francesco Calimeri, Davide Fuscà, Simona Perri, and Jessica Zangari. I-DLV: the new intelligent grounder of DLV. *Intelligenza Artificiale*, 11(1):5–20, 2017.
- [11] Bernardo Cuteri, Carmine Dodaro, Francesco Ricca, and Peter Schüller. Constraints, lazy constraints, or propagators in ASP solving: An empirical analysis. *TPLP*, 17(5-6):780–799, 2017.
- [12] Alessandro Dal Palù, Agostino Dovier, Enrico Pontelli, and Gianfranco Rossi. GASP: Answer set programming with lazy grounding. *Fundam. Inform.*, 96(3):297–322, 2009.
- [13] Minh Dao-Tran, Thomas Eiter, Michael Fink, Gerald Weidinger, and Antonius Weinzierl. Omiga: An open minded grounding on-the-fly answer set solver. In Luis Fariñas del Cerro, Andreas Herzig, and Jérôme Mengin, editors, *JELIA*, volume 7519 of *LNCS*, pages 480–483. Springer, 2012.
- [14] Broes De Cat, Marc Denecker, Maurice Bruynooghe, and Peter J. Stuckey. Lazy model expansion: Interleaving grounding with search. *J. Artif. Intell. Res. (JAIR)*, 52:235–286, 2015.
- [15] Carmine Dodaro, Francesco Ricca, and Peter Schüller. External propagators in WASP: preliminary report. In Stefano Bistarelli, Andrea Formisano, and Marco Maratea, editors, *Proceedings of the 23rd RCRA International Workshop on Experimental Evaluation of Algorithms for Solving Problems with Combinatorial Explosion 2016 (RCRA 2016) A workshop of the XV International Conference of the Italian Association for Artificial Intelligence (AI\*IA 2016), Genova, Italy, November 28, 2016.*, volume 1745 of *CEUR Workshop Proceedings*, pages 1–9. CEUR-WS.org, 2016.
- [16] Martin Gebser, Marco Maratea, and Francesco Ricca. The sixth answer set programming competition. *J. Artif. Intell. Res.*, 60:41–95, 2017.
- [17] Martin Gebser, Orkunt Sabuncu, and Torsten Schaub. An incremental answer set programming based system for finite model computation. *AI Commun.*, 24(2):195–212, 2011.
- [18] Martin Gebser, Torsten Schaub, and Sven Thiele. GrinGo: A new grounder for Answer Set Programming. In Chitta Baral, Gerhard Brewka, and John S. Schlipf, editors, *LPNMR*, volume 4483 of *Lecture Notes in Computer Science*, pages 266–271. Springer, 2007.
- [19] Ian P. Gent, Ian Miguel, and Neil C. A. Moore. Lazy explanations for constraint propagators. In Manuel Carro and Ricardo Peña, editors, *Practical Aspects of Declarative Languages, 12th International Symposium, PADL 2010, Madrid, Spain, January 18-19, 2010. Proceedings*, volume 5937 of *Lecture Notes in Computer Science*, pages 217–233. Springer, 2010.
- [20] Claire Lefevre and Pascal Nicolas. The first version of a new ASP solver: ASPeRiX. In Esra Erdem, Fangzhen Lin, and Torsten Schaub, editors, *LPNMR*, volume 5753 of *LNCS*, pages 522–527. Springer, 2009.
- [21] Vladimir Lifschitz. Answer set programming and plan generation. *Artif. Intell.*, 138:39–54, 2002.
- [22] Kyle Marple and Gopal Gupta. Galliwasp: A goal-directed answer set solver. In Elvira Albert, editor, *Logic-Based Program Synthesis and Transformation, 22nd International Symposium, LOPSTR 2012, Leuven, Belgium, September 18-20, 2012, Revised Selected Papers*, volume 7844 of *Lecture Notes in Computer Science*, pages 122–136. Springer, 2012.
- [23] Kyle Marple, Elmer Salazar, and Gopal Gupta. Computing stable models of normal logic programs without grounding. *CoRR*, abs/1709.00501, 2017.
- [24] Tommi Syrjänen. Omega-restricted logic programs. In Thomas Eiter, Wolfgang Faber, and Mirosław Truszczyński, editors, *LPNMR*, volume 2173 of *LNCS*, pages 267–279. Springer, 2001.
- [25] Antonius Weinzierl. Blending lazy-grounding and CDNL search for answer-set solving. In Marcello Balduccini and Tomi Janhunen, editors, *Logic Programming and Nonmonotonic Reasoning - 14th International Conference, LPNMR 2017, Espoo, Finland, July 3-6, 2017, Proceedings*, volume 10377 of *Lecture Notes in Computer Science*, pages 191–204. Springer, 2017.
- [26] Johan Wittocx, Maarten Mariën, and Marc Denecker. Grounding FO and FO(ID) with bounds. *J. Artif. Intell. Res. (JAIR)*, 38:223–269, 2010.





drawio/Meetings.pdf

Figure 1: Overview of the work division and the project meetings.