**PROJECT POSITIONING**

**Explain how this project fits with the research activities of each involved research group**

My promotor Bart Bogaerts and his group work on knowledge representation and automated reasoning. The project directly interests the group and the international knowledge representation community, this is apparent by some recent publications on the topic [REF RICCA, BART, ALPHA]. It can be said that this project is the continuation of these recent efforts. Furthermore, Bart Bogaerts is already working on lazy grounding specifically.

**Position the project in a national and international context.**

ASP became a well-established paradigm in knowledge representation and reasoning community over the years [REF?]. Lazy grounding for ASP solvers is of great interest to numerous research groups internationally [REF?]. Efficient lazy grounding has the potential to increase the performance of existing ASP applications as well as open the door for it to be applied to a wider range of problems, thus the international and increasing interest. [????]

**Describe the collaboration/coordination/work distribution between the different participating research groups as well as the role/complementarity of the different research groups/(co-)supervisors**

Bart Bogaerts is the supervisor for the project. Research group of Francesco Ricca in U Calabria are experienced in ASP tool development and lazy grounding and they will have a consultory role in this regard.

**SCIENCE COMMUNICATION**

**Indicate whether you think the results of the proposed research will be suitable to be communicated to a non-expert audience and how you would undertake such communication.**

The main results of our research could be demonstrated via comparison between our lazy grounding approach and previous full grounding approaches in terms of performance. Due to the nature of the problem, solving times for different approaches can be drastically different, measured in seconds, which would be recognizable by anyone.

ASP is praised for its intuitiveness and expressiveness, meaning a non-expert could easily understand what a simple ASP program intends to achieve with little guidance from an expert. We believe this will allow for (interactive) demos to be made for science events. [???????? Need some kind of ref for first sentence of par]

**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, product configuration, … [REFERENCES HERE]. 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. 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 and rule decomposition [REF] aim to reduce the size of grounding where possible. For query problems, top-down evaluation utilizing magic sets is used. For planning problems, iterative grounding 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, GASP, ASPeRiX and the recently introduced alpha that integrates lazy grounding with a CDCL solver [REFS].* *Also top-down lazy grounding techniques (for a language related to ASP) [17] and top-down stable model generation techniques that avoid grounding [26, 27] 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 [REF]. 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 (goaldirected) lazy grounding techniques (such as [27] and those developed for the IDP system [17]) 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 [8].*

*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 newly developed [OR EXISTING, REFER TO ESRA ERDEM’s RG] ASP models for Multi Agent Path Finding (MAPF), which can be modeled as a planning problem or a graph problem [REFER TO OWN PAPER]. 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 OWN PAPER], 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, our research is grounded on an ASP-based implementation of fluid construction grammar.*

**Scientific Research Objectives**

We categorize our objectives as follows: 1) deeply understanding the literature on lazy grounding and analyzing them, 2) developing novel algorithms for lazy grounding, 3) applying and evaluating said algorithms.

To achieve (1), 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.

Objective 3: Quantify to effects of different factors discovered in OBJ2.

To develop novel algorithms for lazy grounding (2), 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 4: Extend the current algorithms by addressing the most important factors identified in OBJ2 and OBJ3.

Objective 5: Develop lazy grounding algorithms that combine top-down and bottom-up reasoning approaches

Although the above objectives are a bit vague, we propose concrete ways to achieve them in our work packages.

We would like to evaluate our novel algorithms (3) by applying them to challenging problems and existing lazy grounding benchmarks.

Objective 6: Model the Multi Agent Path Finding (MAPF) problem and its variants of interest in ASP.

Objective 7: Evaluate our novel algorithms on MAPF and existing benchmarks for lazy grounding.

**Research Methodology and Work Plan**

To analyze the existing algorithms, we study the existing literature. [ANY MORE DETAILS?]

For implementing both our novel algorithms and existing algorithms in the literature, we use the recently developed and promising ASP solver WASP [ REF]. 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 and enable us to utilize the experience of its developers in the research group of Francesco Ricca.

Our strategy will be to implement our lazy grounding algorithms as custom theory propagators, which can be efficiently done using WASP [REF].

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 our novel models for MAPF [novel? Maybe refer to Esra Erdem’s (or mine) papers].

Work Package 1: 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 2: Develop [OR UTILIZE EXISTING] MAPF models for ASP (OBJ6). 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 [REFER TO MAPDC (my studies)] and this could allow us to evaluate our algorithms on incrementally more challenging versions of the same fundamental problem.

*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. 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 “: −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 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]*

*Visible choice variables alpha also limits it 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 oneman 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. The biggest challenge, however, is formally proving that we have found all factors, which we do in collaboration with Antonius Weinzierl, the architect of alpha. To do this, we follow two strategies. First, we exploit our characterizations as transition systems from WP1. Second, we use techniques from proof complexity theory [13] to study the relationship between proofs found by alpha and proofs found by classical ground-and solve systems.*

*Work Package 4. Implementation of alpha techniques in wasp. (OBJ3,4)*

*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 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. (OBJ3,5) (with Calabria, Vienna) 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 fluid construction grammar While solving the 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. (OBJ5,9) (with Vienna) 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 [34], to syntactic restrictions on ASP programs that guarantee a finite grounding [30, 22, 11], and intelligent grounding techniques [12]. 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 [8], 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) (with Calabria) 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 a 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) (with Calabria) 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 [14], 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 [23]. N Work Package 10. Algorithms for improved lazy grounding. (OBJ5,9)(with Calabria, Vienna) 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.*