Advances in Cognitive Systems 2 (2013) 1-18 Submitted 6/2013; published soon

LUIGi: A Goal-Driven Autonomy Agent Reasoning with Ontologies

Dustin Dannenhauer DTD212@LEHIGH.EDU
Héctor Muñoz-Avila HEM4@LEHIGH.EDU
Department of Computer Science and Engineering, Lehigh University, Bethlehem, PA 18015 USA

Abstract

In this paper we present LUIGi, a goal-driven autonomy (GDA) agent that uses ontological representations. Like other GDA agents, LUIGi reasons about (1) its own expectations when following a course of action, (2) the discrepancies between these expectations and actual situations encountered, (3) explanations for those discrepancies and (4) new goals based on the explanations. Unlike other GDA agents, LUIGi uses ontologies to reason about these four GDA elements enabling inference capabilities relating high-level concepts with low-level chunks of information. We demonstrate LUIGi's capabilities in the context of a real-time strategy game.

1. Introduction

Many cognitive scientists believe that humans learn and reason with skills of different levels of abstraction (Langley & Choi, 2006). Perhaps as a result, stratified goal reasoning, the study of agents that can reason about what goals at different levels of abstraction they should dynamically pursue, has been a central topic of cognitive architectures. SOAR (Laird, 2012), Companion (Forbus & Hinrichs, 2006) and ICARUS (Langley & Choi, 2006) uses HTN planning techniques for subgoal generation: goals are recursively decomposed from high-level goals into simpler ones. Goals are not only generated as subgoals for a goal to be achieved; for example, if an impasse on the use of knowledge occurs, SOAR will generate a goal to solve this impasse.

Goal-driven autonomy (GDA) (Muñoz-Avila et al., 2010b; Molineaux, Klenk, & Aha, 2010) is a model of goal reasoning in which an agent revises its goals by reasoning about discrepancies it encounters during plan execution monitoring (i.e., when its expectations are not met) and their explanations. Discrepancies arise when solving tasks in environments where either there are multiple possible outcomes for the agent's actions or changes in the environment may take place independent of the agent's actions. GDA agents continuously monitor the current plan's execution and assess whether the actual states visited during the current plan's execution match expectations. When mismatches occur (i.e., when the state does not meet expectations), a GDA monitor will suggest alternative goals that, if accomplished, would fulfill its overarching objectives.

A shortcoming in the current state of the art on GDA research is the lack of structured, high-level representations in their formalisms. Most rely on STRIPS representations. For example, the agent's expectations are defined as either specific states (i.e., collection of atoms) or specific atoms in the state (e.g., after executing the action move(x,y) the agent expects the atom location(y) to be

in the state) expected to be true. Goals are desired states to be reached in the state or desired atoms in the state (e.g., the agent is at a certain location).

In this paper we present LUIGi, a GDA agent that uses ontologies to enhance its representation of GDA elements whereby high-level concepts can be directly related, and defined in terms of low-level chunks of information. Another motivation for using ontologies is that STRIPS representations are too limited to represent events that happen in the real world (or complex domains), more structured representations are needed in order to capture more complex constraints (Gil & Blythe, 2000). Drawbacks to using ontologies include the knowledge engineering effort required for construction and maintenance of the ontology as well as the running time performance during reasoning. In past years, taking an ontological approach may have seemed intractable for systems acting in real-time domains with large state spaces. With the growth of the semantic web, description logics, and fast reasoners, ontologies may begin to become viable. As demonstrated in this work, we are reaching a point where ontologies are useful for fast-acting systems such as agents that play Real-Time Strategy games. This work investigates the use of ontologies for a GDA agent playing the Real-Time Strategy game Starcraft.

2. Related Work

Cognitive architectures have long pointed to the importance of using ontological knowledge. The Disciple cognitive architecture uses *isa* and *subclass* ontological relations to represent complex relations between concepts (Tecuci et al., 1999). This enables Disciple to generalize categories of objects by looking at their *isa* relations. The Companions cognitive architecture maps two structured representations (the base and the target) as part of its analogical reasoning process (Forbus & Hinrichs, 2006). It uses an analogy ontology to formalize the kinds of relations that occur during its analogical reasoning process. This ontology enables Companions to combine reuse knowledge (akin to case-based reasoning) with first-principles reasoning (akin to planning). The SOAR cognitive architecture uses ontologies to facilitate representing structured declarative information. It also defines types of impasses that are shared across domains. Our work borrows from these experiences by introducing ontologies into GDA.

Robotics research is seeing an increase in the use of ontologies. The work of (Tenorth & Beetz, 2009) is similar to the work presented here, differing in the task and environment, where the agent is concerned with manipulating and acting on objects in a household environment. (Hawes et al., 2011) use ontologies in a conceptual layer of map knowledge. The ontology allows the robot to identify areas as either Rooms or Passages. This is similar to our work, where the ontology allows labeling regions in Starcraft maps as *Controlled*, *Contested*, or *Unknown*, but the task of the agent and use of the labels are different. The goals of the robot are to explore areas, or spaces, where as goals in LUIGi are to defeat the enemy via different strategies.

GDA research has garnered a lot of attention in recent years. (Cox, 2007) introduces the notions of expectation and explanation, which are two of the main components of GDA. When an anomaly happens, an agent generates an explanation; this process involves finding the culprit of the anomaly. The culprit may stem from a defect in the agent's cognitive process (including its knowledge). An intelligent agent may need to blame itself and its own cognitive process in order to improve: this

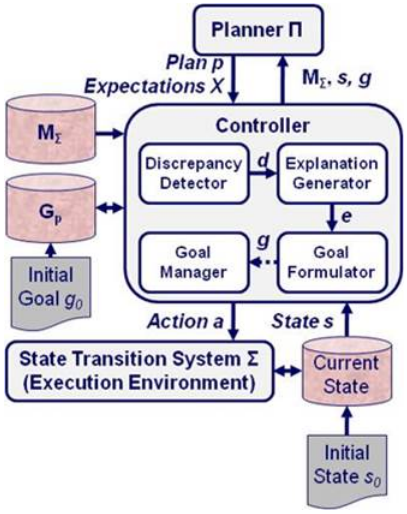

Figure 1: The GDA Cycle

can be done by generating a Learning Goal. Learning Goals are used in pursuit of improving the agent's cognitive process (i.e. filling a gap in knowledge), which are different from Achievement Goals: the goals the agent pursues to accomplish its task.

GDA models can be seen as consisting of four steps, as shown within the Controller component in Figure 1. This particular GDA model, with some small variations, has been used in a number of publications (Muñoz-Avila et al., 2010a; Muñoz-Avila et al., 2010b; Molineaux, Klenk, & Aha, 2010; Jaidee, Muñoz-Avila, & Aha, 2011; Powell, Molineaux, & Aha, 2011; Aha, 2011)

The DiscoverHistory GDA agent (Molineaux, Aha, & Kuter, 2011) uses a Hierarchical Task Network planner to generate a plan. Since the planner cannot account for every plausible event in the domain; it makes assumptions that the conditions of the environment are static to generate its plan. The system then uses the DiscoverHistory algorithm whenever one of these assumptions fails. The way DiscoverHistory works is by modifying the generated plan, introducing new elements that solve discrepancies, or by modifying the initial assumptions. Often, this process introduces new contradictions, which are then recursively solved until the whole plan is consistent, or until a maximum number of modifications have been made.

More recently, Klenk et al. (2012) presented ARTUE, which uses a direct application of explanations in a strategy simulation. ARTUE is based on a modified version of the SHOP2 planner, and it explains discrepancies similarly to DiscoverHistory. When ARTUE cannot create an explanation, it discards the discrepancy entirely.

None of these GDA agents use ontological information to reason about the GDA steps. To the best of our knowledge, LUIGi is the first GDA agent to use ontological information to reason about the GDA elements at varied levels of abstraction.

Description logics used by semantic web languages, and as we adopted in our work, follow principles established by early work on knowledge representation. For example, Classic (Patel-Schneider et al., 1991) is a framed-based knowledge representation language that belongs to the

family of description logics and is a descendant of KL-ONE (Brachman & Schmolze, 1985). Description logics focus on the definition of terms in order to provide precise semantics. Description logics can automatically classify a concept (i.e., automatically insert the concept at the appropriate place in the ontology). Classic allows the description of objects in such a way that it is possible (and tractable) to determine automatically whether one object is subsumed by another. In the parlance of Classic, information can be told or derived. Told information is the one that is described explicitly in the ontology. For example, a UNIX symbolic link can be defined to be a subclass of file. Derived information is information that is derived through various mechanisms. Classic uses open world assumption. That is, Classic doesn't assume that all information is known. Hence, the absence of a fact doesn't imply its negation.

3. Goal-Driven Autonomy

The GDA model, shown in Figure 1, extends the conceptual model of online planning (Nau, 2007). The components include a Planner, a Controller, and a State Transition System $\sigma = (S, A, E, \gamma)$ with states S, actions A, exogenous events E, and state transition function $\gamma\colon S \times (A \cup E) \rightarrow 2^S$. In the GDA model, the Controller is centric. It receives as input a planning problem (M, s_c, g_c), where M_σ is a model of σ, s_c is the current state (e.g., initially the starting state), and $g_c \in G$ is a goal that can be satisfied by some set of states $S_g \in S$. It passes this problem to the Planner Π, which generates a sequence of actions $A_c = \langle a_c, \ldots, a_{c+n} \rangle$ and a corresponding sequence of expectations $X_c = \langle x_c, \ldots, x_{c+n} \rangle$, where each $x_i \in Xc$ is a set of constraints that are predicted to hold in the corresponding sequence of states $\langle s_{c+1}, \ldots, s_{c+n+1} \rangle$ when executing A_c in s_c using M_σ. The Controller sends a_c to σ for execution and retrieves the resulting state s_{c+1}, at which time it performs the following knowledge-intensive (and GDA-specific) tasks:

1. Discrepancy detection: GDA detects unexpected states before deciding how to respond to them. This task compares observations s_{c+1} with expectation x_c. If one or more discrepancies (i.e., unexpected observations) $d \in D$ are found in s_{c+1}, then explanation generation is performed to explain them.

2. Explanation generation: This module explains a detected discrepancy d. Given also state s_c, this task hypothesizes one or more explanations $ex \in Ex$ of their cause.

3. Goal formulation: Resolving a discrepancy may warrant a change in the current goal(s). This task generates goal $g \in G$ given a discrepancy d, its explanation ex, and current state s_c.

4. Goal management: New goals are added to the set of pending goals $GP \subseteq G$, which may also warrant other changes (e.g., removal of other goals). The Goal Manager will select the next goal $g \in GP$ to be given to the Planner. (It is possible that $g = g$.)

Common to most current GDA systems in the literature, the GDA elements are defined using STRIPS planning conventions:

- **State**: a collection of atoms

- **Goal**: a desired atom in the state
- **Goal state**: any state that contains the goal(s)
- **Expectation**: predicted state following an action
- **Discrepancy**: Whenever the expected state X is different from the actual S obtained (i.e., $X \neq S$). The discrepancy are the atoms in $X \setminus S$
- **Explanation**: any $ex \in Ex$ that caused the discrepancy

4. Example: Exploiting Ontologies in GDA

We now walk through an example that uses ontological knowledge to reason at a high-level in a real-time strategy game. Real-Time Strategy (RTS) games involve two or more players whose goal is defeat their opponent's armies, and commonly require the construction of army units (i.e. soldiers) while managing economic resources. RTS games require managing different tasks concurrently, including managing an economy, developing an army, and attacking the enemy. Some RTS games have achieved worldwide popularity, such as Starcraft and Warcraft. Starcraft has become so popular that a professional competitive league televises and awards considerable prize money to expert players. We use Starcraft as a testbed for GDA research because the environment is complex, real-time, dynamic, has partial visibility, and a vast state space with size on the order of $10^{10,000}$ with map sizes of up to 256x256 tiles (Weber, 2012). An important aspect of Starcraft is "fog of war": each player's visibility of the map is determined from the radius of its units. From the player's perspective a region is unknown if it is enveloped in the fog of war.

LUIGi maintains an ontology that contains low-level facts of the environment (such as the x and y coordinates of each unit) as well as high-level concepts that can be inferred from low-level facts. One such high-level concept is the notion of controlling a region (a bounded area of the map). A player controls a region provided two conditions hold. First, the player must have at least one unit in that region. Second, the player must own all units in that region. Regions in which both players have units are considered contested. A region can only be one of unknown, contested, or controlled.

Once we have these concepts of unknown, controlled, and contested regions, we can use rich expectations. For example, in the beginning of the game when we are constructing the buildings and producing our first army, we expect that we control our region. If this expectation is violated (producing a discrepancy), it means that the enemy is attacking our base early in the game. When this happens LUIGi explains the discrepancy and pursues a goal focused on defending its base region instead of building a considerable army. In real-time strategy games the order in which you construct buildings and produce fighting units is crucial in the beginning of a match. There is a trade-off between constructing the buildings and harvesting the resources that are needed to build more powerful units in the future versus building weaker units more quickly. A rush attack is when one player builds many cheap units to attack the enemy, before he has produced any defensive capabilities. It is important to note that if a rush fails, the player who rushed may then be at a disadvantage, having less resources then the defending player who was planning for the long term.

When the expectation "I control my starting region" fails during an enemy rush attack, the agent can choose a new goal to quickly defend their region. This is a higher reasoning level than many RTS bots who simply focus on optimizing micro-unit attacks. While micro-management is important, without high-level reasoning, bots are still at a disadvantage to humans. One reason for this that has been pointed out by domain experts is that the automated players are still weak in reasoning with high-level strategies (Churchill, 2013).

5. Ontologies for Expectations and Explanations

One of the main benefits of using an ontology with GDA is the ability to provide formal definitions of the different elements of a GDA system. The ontology used here chooses facts as its representation of atoms, where facts are represented as triples $\langle$subject, predicate, object$\rangle$. A fact can be an initial fact (e.g. $\langle$unit5, hasPosition, (5,6)$\rangle$ which is directly observable) or an inferred fact (e.g. $\langle$player1, hasPresenceIn, region3$\rangle$). In the previous example we use an ontology to represent the high-level concept of controlling a region. By using a semantic web ontology, that abides by the open-world assumption, it is technically not possible to infer that a region is controlled by a player, unless full knowledge of the game is available. Starcraft is one such domain that intuitively seems natural to abide by the open world assumption because of fog of war. However, we can assume local closed world for the areas that are within visual range of our own units. For example, if a region is under visibility of our units and there are no enemy units in that region, we can infer the region is not *contested*, and therefore we can label the region as *controlled*. Similarly, if none of our units are in a region, then we can infer the label of *unknown* for that region.

The following are formal definitions for a GDA agent using a semantic web ontology:

- **State** S: collection of facts
- **Inferred State** S^i: $S \cup \{$ facts inferred from reasoning over the state with the ontology $\}$
- **Goal** g: a desired fact $g \in S^i$
- **Expectation** x: one or more facts contained within the S^i associated with some action. We distinguish between primitive expectations, x_p, and compound expectations, x_c. x_p is a primitive expectation if $x_p \in S$ and x_c is a compound expectation if $x_c \in \{S^i - S\}$. $\{S^i - S\}$ denotes the set difference of S^i and S, which is the collection of facts that are strictly inferred.
- **Discrepancy** d: Given an inferred state S^i and an expectation, x, a discrepancy d is defined as:

  1. $d = x$ if $x \notin S^i$, or
  2. $d = \{x\} \cup S^i$ if $\{x\} \cup S^i$ is inconsistent with the ontology

- **Explanation** e: Explanations are directly linked to an expectation. For primitive expectations, such as $x_p = (player1, hasUnit, unit5)$ the explanation is simply the negation of the expectation when that expectation is not met: $\neg x_p$. For compound expectations, x_c (e.g. expectations that are the consequences of rules or facts that are inferred from description

logic axioms), the explanation is the trace of the facts that lead up to the relevant rules and/or axioms that cause the ontology to be inconsistent.

Example of Explanation

Assume we have the following description logic axiom (1) and semantic web rules (2) and (3):

$$KnownRegion \equiv DisjointUnionOf(ContestedRegion, ControlledRegion) \quad (1)$$

The class KnownRegion is equivalent to the disjoint union of the classes ContestedRegion and ControlledRegion. This axiom allows the ontology to infer that if an individual is an instance of ContestedRegion it cannot be an instance of ControlledRegion, and vice-versa. This also encodes the relationship that ContestedRegion and ControlledRegion are subclasses of KnownRegion.

$$Player(?p) \wedge Region(?r) \wedge Unit(?u) \wedge isOwnedBy(?u, ?p) \wedge isInRegion(?u, ?r) \rightarrow hasPresenceIn(?p, ?r) \quad (2)$$

With this rule, the ontology reasoner can infer that a player has a presence in a region if that player owns a unit that is located in that region.

$$Player(?p1) \wedge Player(?p2) \wedge DifferentFrom(?p1, ?p2) \wedge Region(?r) \wedge hasPresenceIn(?p1, ?r) \wedge hasPresenceIn(?p2, ?r) \rightarrow ContestedRegion(?r) \quad (3)$$

This rule allows the inference of a region to be an instance of ContestedRegion if two different players each have a presence in that region.

Figure 2 shows an example where the unsatisfied expectation is ⟨player0, controlsRegion, regionA⟩ in which the explanation is that regionA is contested. The explanation trace begins with the primitive facts of each player owning one or more units and those units' being located in regionA. Using rule (2), the next level of the tree is the inferred facts: ⟨player0,hasPresenceIn,regionA⟩ and ⟨player1,hasPresenceIn,regionA⟩. Now using rule (3) with the second level inferred facts, we infer that ⟨regionA,instanceOf,ContestedRegion⟩. From this level, the expectation ⟨player0, controlsRegion, regionA⟩, of which the fact ⟨regionA, instanceOf,ControlledRegion⟩ is inferred, and combined with the ⟨regionA,instanceOf,ContestedRegion⟩, axiom (1) produces an inconsistent ontology because a region can not be both a contested region and a controlled region (disjointUnionOf).

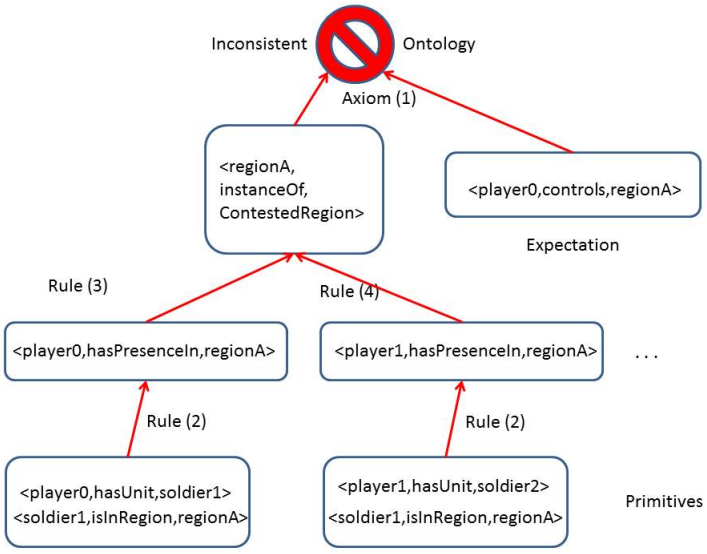

Figure 2: Inconsistency Explanation Trace

In this example, the explanation was that the region was contested, and the trace both provides an explanation and shows how the ontology reasoner is able to produce an inconsistent ontology. An inconsistent ontology is the result of the unmet compound expectation ⟨player0, controlsRegion, regionA⟩. Given the explanation, a viable goal to pursue next would be to build a stronger group of units to attack the enemy units in regionA. As discussed before, an alternative explanation could have been that the player does not control the region because it did not have any of its units in the region. In that case, the inconsistency in the ontology would result from regionA being labeled as *Unknown* and a viable goal would be to travel to the region along a different path or a different medium (flying vs. ground units).

High-level concepts with an ontology provide an abstraction over low-level facts and conditional relationships among the facts. For example, we can either have a group of facts such as ⟨player0, hasUnit, unit1⟩, ⟨player0, hasUnit, unit2⟩, ⟨unit1, isAtLocation, (x1,y1)⟩, ⟨unit2, isAtLocation, (x2,y2)⟩, ⟨player1, hasUnit, unit3⟩, ⟨unit3, isAtLocation, (x3,y3)⟩ accompanied with the conditional relationships equivalent to the description logic rule (1) and rules (2) and (3) or we can have the high-level concept *Contested Region* accompanied with the ontology. The conditional relationships are needed for both being able to count how many units each of the players has in the region, and being able to infer what region each unit is located in based on its (x,y). These numerical conditions are what prevents STRIPS from being a practical representation. The explanation trace shows how the concepts are built from the low-level facts and the complex (often hierarchical) relationships among them (via description logic axioms and rules). This abstraction provides smaller expectation representations than what would be required from equivalent STRIPS representations.

The explanation trace makes apparent the conditional relationship details and facts of which the high level expectation (i.e. ContestedRegion) is composed. This allows the root causes of the

unmet expectation to be identified. The benefit is richer expectations and explanations being able to express not only atoms (facts) but conditions and relations on those atoms as well.

6. Scenario Demonstrations

LUIGi is implemented in two components, a bot and a GDA component. The bot component interfaces directly with the Starcraft game engine to play the game. The GDA component runs as a separate program, and connects to the bot component via a TCP/IP connection with both the bot and GDA component running on the same computer. The bot is responsible for building up an initial base and executing tasks sent to it by the GDA component. The bot component contains knowledge of how to execute individual plan steps and knowledge of when a task has finished so that it can request the next plan step from the GDA component. During gameplay the bot dumps all of its game state data every n frames (in these scenarios $n = 20$), making it available for the GDA component to use during ontological reasoning.

The bot component is implemented in a C++ DLL file that is loaded into Starcraft and the GDA component is implemented in a Java standalone application. Running the reasoner, Pellet (Sirin et al., 2007), over the ontology takes on average between 1-2 seconds. Such an amount of time would normally be unacceptable for micro-management strategies controlling individual units in battle, but for more general strategies, such as what units to produce next and how to attack the enemy, the following scenarios demonstrated that this amount of time is acceptable when played at the speed setting used in human vs. human tournaments. This is because executing a strategy takes anywhere from 20 seconds to minutes in these scenarios. We now describe three scenarios where LUIGi was able to successfully detect a discrepancy and choose a new goal using the ontology.

Scenario 1: Early rush attack

In this scenario LUIGi gets rushed by the enemy early in the game. A discrepancy is detected soon after enemy units enter LUIGi's starting region. The discrepancy was that LUIGi does not control its base region because the region is contested and an explanation trace similar to Figure 2 is generated. LUIGi sends the explanation to the goal formulator component which chooses a new goal to defend the region. As part of the new goal, LUIGi recalls all units (including those in other regions) and uses them to attack the enemy forces in its base region. Figure 3a shows LUIGi in the process of producing troops while controlling the region. Figure 3b shows LUIGi pursuing a newly generated goal to defend the region after detecting and explaining the discrepancy of not controlling the region (i.e., the blue units are enemy units).

Scenario 2: Units do not reach destination

In a second scenario, LUIGi successfully builds ground units and sends them to attack the enemy base. However, the enemy has set up a defense force at its base region's perimeter, which destroys LUIGi's ground units before they make it to the enemy region. The expectation that is violated is a primitive expectation (e.g. $\langle$unit5, isInRegion, region8$\rangle$) and the discrepancy is that LUIGi expects unit5 to be in the region region8. The explanation is simply the negation of the expectation.

(a) No Discrepancy

(b) Discrepancy due to enemy Rush Attack

Figure 3: Screenshots of LUIGi building an initial army

LUIGi chooses a new goal in which to build units that fly in order to bypass the units defending the enemy's base. However, there are multiple valid goals worth pursuing in this situation, such as building different units capable of defeating the enemy units defending the enemy base's perimeter, or taking a different route leading into the enemy base.

Scenario 3: Units reach destination but do not defeat enemy

In a third scenario, LUIGi's units were produced and moved to the enemy region successfully, but were not able to defeat the enemy in its base region. The expectation that LUIGi controlled the enemy base region was unmet after LUIGi's units finished executing the plan step to attack the enemy units in the base and LUIGi's units were killed. The corresponding explanation, informally, was that LUIGi had no units in the region when the plan step of attacking had finished. While the expectation is the same as scenario 1, the explanation is different (i.e. the traces are different). As a result LUIGi chooses a different goal than in Scenario 1. In this case, LUIGi chooses a goal that does not involve directly attacking the enemy but instead to secure and defend other locations that contain valuable resources. This type of strategy gives LUIGi an economic advantage over its opponent yielding a more substantial army later in the match.

7. Discussion and Future Work

There is a big gap in the performance of the best automated players compared to the best human players for real-time strategy games such as Starcraft. Part of the reason for this gap is the inability of automated players to reason about high-level strategies. We believe that GDA combined with high-level reasoning as enabled by inferencing capabilities over ontologies could pave the way to mimic the kinds of strategic decisions that humans make. While restricted to play particular scenarios instead of complete games, LUIGi exhibits some of the needed high-level reasoning capabilities

relating high-level concepts with low-level chunks of information that none of the current Starcraft automated players possess.

In future work, we will like to extend LUIGi to play complete Starcraft games. Currently we are prevented from doing so not by conceptual difficulties with LUIGi itself but by low-level programming details of the way automated players "hack" into the game engine to change its game-playing code.[1] More interesting future work, from a research perspective, is to explore to reason with GDA elements as the ontologies might change over time. For example, imagine a game that allows players to change the terrain over time (e.g., constructing a bridge between two regions previously disconnected). Such a capability to reason with these changing elements would be of particular interest for GDA agents that interact for long-durations in an environment (e.g., an agent interacting in a persistent world).

References

Aha, D.W., M. M. . K. M. (2011). *Goal-Driven Autonomy* (Technical Report). NRL Review.

Brachman, R. J., & Schmolze, J. G. (1985). An Overview of the KL-ONE Knowledge Representation System. *Cognitive science*, *9*, 171–216.

Churchill, D. (2013). 2013 AIIDE StarCraft AI Competition Report. `http://webdocs.cs.ualberta.ca/~cdavid/starcraftaicomp/report2013.shtml`.

Cox, M. T. (2007). Perpetual Self-Aware Cognitive Agents. *AI magazine*, *28*, 32.

Forbus, K. D., & Hinrichs, T. R. (2006). Companion Cognitive Systems: A Step toward Human-Level AI. *AI Magazine*, *27*, 83.

Gil, Y., & Blythe, J. (2000). How Can a Structured Representation of Capabilities Help in Planning. *Proceedings of the AAAI–Workshop on Representational Issues for Realworld Planning Systems*.

Hawes, N., Hanheide, M., Hargreaves, J., Page, B., Zender, H., & Jensfelt, P. (2011). Home Alone: Autonomous Extension and Correction of Spatial Representations. *Robotics and Automation (ICRA), 2011 IEEE International Conference on* (pp. 3907–3914).

Jaidee, U., Muñoz-Avila, H., & Aha, D. W. (2011). Integrated Learning for Goal-Driven Autonomy. *Proceedings of the Twenty-Second international joint conference on Artificial Intelligence-Volume Volume Three* (pp. 2450–2455).

Laird, J. E. (2012). *The Soar Cognitive Architecture*. The MIT Press.

Langley, P., & Choi, D. (2006). Learning Recursive Control Programs from Problem Solving. *The Journal of Machine Learning Research*, *7*, 493–518.

Molineaux, M., Aha, D. W., & Kuter, U. (2011). *Learning Event Models that Explain Anomalies* (Technical Report). DTIC Document.

Molineaux, M., Klenk, M., & Aha, D. W. (2010). Goal-Driven Autonomy in a Navy Strategy Simulation. *AAAI*.

Muñoz-Avila, H., Aha, D. W., Jaidee, U., Klenk, M., & Molineaux, M. (2010a). Applying Goal Driven Autonomy to a Team Shooter Game. *FLAIRS Conference*.

1. Automated players must "inject" their code into the game engine in a manner reminiscent of how malicious software hacks into a system

Muñoz-Avila, H., Jaidee, U., Aha, D. W., & Carter, E. (2010b). Goal-Driven Autonomy with Case-Based Reasoning. In *Case-based reasoning. research and development*, 228–241. Springer.

Nau, D. S. (2007). Current Trends in Automated Planning. *AI magazine*, *28*, 43.

Patel-Schneider, P. F., McGuinness, D. L., Brachman, R. J., & Resnick, L. A. (1991). The CLASSIC Knowledge Representation System: Guiding Principles and Implementation Rationale. *ACM SIGART Bulletin*, *2*, 108–113.

Powell, J., Molineaux, M., & Aha, D. W. (2011). Active and Interactive Discovery of Goal Selection Knowledge. *FLAIRS Conference*.

Sirin, E., Parsia, B., Grau, B. C., Kalyanpur, A., & Katz, Y. (2007). Pellet: A practical OWL-DL reasoner. *Web Semantics: science, services and agents on the World Wide Web*, *5*, 51–53.

Tecuci, G., Boicu, M., Wright, K., Lee, S. W., Marcu, D., & Bowman, M. (1999). An Integrated Shell and Methodology for Rapid Development of Knowledge-Based Agents. *AAAI/IAAI* (pp. 250–257).

Tenorth, M., & Beetz, M. (2009). KnowRob âĂŤ Knowledge Processing for Autonomous Personal Robots. *Intelligent Robots and Systems, 2009. IROS 2009. IEEE/RSJ International Conference on* (pp. 4261–4266).

Weber, B. (2012). *Integrating Learning in a Multi-Scale Agent*. Doctoral dissertation, University of California, Santa Cruz.