Tractable First-Order Golog with Disjunctive Knowledge Bases

Jens Claßen and **Gerhard Lakemeyer**
Department of Computer Science
RWTH Aachen University
Germany
⟨classen|gerhard⟩@cs.rwth-aachen.de

Abstract

While based on the Situation Calculus, current implementations of the agent control language Golog typically avoid offering full first-order capabilities, but rather resort to the closed-world assumption for the sake of efficiency. On the other hand, realistic applications need to deal with incomplete world knowledge including disjunctive information. Recently Liu, Lakemeyer and Levesque proposed the logic of limited belief $\mathcal{SL}$, which lends itself to efficient reasoning in incomplete first-order knowledge bases. In particular, $\mathcal{SL}$ defines levels of belief which limit reasoning by cases in a principled way. In this paper, we propose to apply $\mathcal{SL}$-based reasoning in the context of a Golog system. Central to our approach is a new search operator that finds plans only within a fixed belief level k, and an iterative-deepening-style variant where instead of considering plans with increasing length, the belief level k is incremented in each cycle. Thus, not the shortest plans are preferred, but those which are the computationally cheapest to discover.

Introduction

The agent language Golog (Levesque et al. 1997) has already been applied in many application scenarios, including the control of autonomous mobile robots (Ferrein and Lakemeyer 2008). The language is based on the Situation Calculus (McCarthy and Hayes 1969; Reiter 2001), which in the theoretical formalization is a dialect of first-order predicate calculus. However, current implementations of Golog typically avoid to offer full first-order capabilities, but rather resort to the closed-world and/or domain closure assumptions for the sake of efficiency of reasoning. On the other hand, in realistic applications such as mobile robotics, almost inevitably one has to cope with incomplete world knowledge, in particular in the form of disjunctive information. Furthermore, as the task of an autonomous robot is usually open-ended, not all individuals (persons or objects) it has or will have to deal with are known in advance.

$\mathcal{SL}$, the subjective logic of limited belief proposed by Liu, Lakemeyer and Levesque (2004) is a formalism for efficient reasoning with incomplete first-order knowledge bases. They define a family of believe operators $\mathbf{B}_0$, $\mathbf{B}_1$, $\mathbf{B}_2$,... where intuitively $\mathbf{B}_0$ corresponds to the agent's explicit belief and implicit beliefs become only available at higher belief levels, where the greater k, the computationally more expensive, roughly measured in terms of the number of nested case distinctions. When k is fixed, then whether $\mathbf{B}_0 KB$ implies $\mathbf{B}_k \phi$ is decidable, and when the KB is in a certain form, reasoning is also tractable. Furthermore, the inference is classically sound in the sense that when $\mathbf{B}_0 KB$ implies $\mathbf{B}_k \phi$ in $\mathcal{SL}$, then KB entails ϕ in classical predicate logic.

For the above mentioned reasons, we believe that it is beneficial to apply $\mathcal{SL}$-based reasoning in the context of a Golog system. Apart from the fact that reasoning within a fixed level k can be done efficiently, belief levels offer the possibility to define a new planning operator that prefers plans with least computational costs. To illustrate the idea, consider a Golog program of the following form:

$$\psi?; a \mid \varphi?; b; c$$

There is a nondeterministic choice (|) between two branches: if formula ψ holds, action a can be executed, or when formula φ holds, action sequence $b; c$ could be performed. Planning here means to resolve the nondeterminism and thus commit to one or the other branch. A classical Golog system will typically test the branches in the presented order, meaning it first checks whether the action sequence $\langle a \rangle$ constitutes a legal execution, which involves checking if ψ is known to hold according to the system's knowledge base. Formula φ and sequence $\langle b, c \rangle$ will only be tested once Golog found out that it is not possible to execute the left branch successfully. Alternatively, the system might apply some iterative deepening strategy, which always yields the shortest action sequences. Still, when the left branch in the above example is executable, the system would prefer it over the right one. In any case, no attention is paid to the computational effort involved.

Now assume that φ can instantly be inferred from the agent's knowledge base (say if it is a fact that is explicitly known to be true), but ψ is quite complicated and requires extensive reasoning. In particular when the decision has to be made quickly (e.g. think of robot soccer) or when the additional reasoning time outweighs the time saved by performing fewer actions, it may pay off to prefer possibly longer plans that however involve less reasoning.

As our running example, consider a mobile robot working in an office environment. There is one employee, Carol, who wants to have a look at a certain book. The department

possesses two copies, where one ($book1$) is usually located in the library (lib) and the other one ($book2$) in the lab (lab). She gives the robot the following orders: "If $book1$ is in the library, bring it to me *or* if $book2$ is in the lab, bring it after unlocking the lab door." This might be expressed as follows in a Golog program:

$$\mathit{At}(book1, lib)?; get(book1, lib) \mid \\ \mathit{At}(book2, lab)?; unlock(lab); get(book2, lab)$$

Now assume that the robot explicitly knows from a recent observation that $book2$ actually is in the lab. On the other hand, it only knows that it saw $book1$ yesterday in the office shared by Ann and Bob, meaning one of them borrowed it. The robot also knows that whoever borrows a book will return it to the library in the evening on the same day. As working hours have just begun, all books that were borrowed yesterday will now be in the library. Obviously, this knowledge is sufficient to deduce that $book1$ is in the library, but whereas retrieving the explicit fact $\mathit{At}(book2, lab)$ from the knowledge base basically requires no reasoning at all, deriving $\mathit{At}(book1, lib)$ involves one case distinction: Either Ann or Bob borrowed the book, but in any case, it has been returned. Therefore intuitively, $\mathit{At}(book2, lab)$ is already available at belief level zero, but $\mathit{At}(book1, lib)$ only at greater levels.

In this paper we propose to apply the idea of iterative deepening on belief levels instead of on action sequence lengths. It will first be tested whether any of the program's possible execution traces can be verified to succeed by reasoning at level zero. Only if this is not the case, the level will be increased to one etc. Thus, the first successful execution trace to be found will also be the one that needs the least computational effort, which allows to obtain solutions much quicker in many cases.

The remainder of the paper is organized as follows. In the next section, we give the formal syntax and semantics of the logic on which our approach is based. Next, we present some results that relate the formalism to existing languages. The following section contains the main contribution of this paper in form of the new planning operator we propose. Finally, we sketch possible directions for future work.

Definitions

In this section, we introduce our new logic $\mathcal{SLA}$ formally. The language is basically an extension of Liu, Lakemeyer and Levesque's (2004) logic of limited belief $\mathcal{SL}$ by aspects of the modal Situation Calculus variant $\mathcal{ES}$ (Lakemeyer and Levesque 2004) for modelling action and change.

Syntax

Terms The *terms* of the language come in two sorts: *object* and *action*. A term of sort object is either an object variable ($x_1, x_2, \ldots$) or an object constant ($d_1, d_2, \ldots$, e.g. lab). An action term is either an action variable ($a_1, a_2, \ldots$) or of the form $g(t_1, \ldots, t_n)$, where g is an action function of arity n (e.g. $unlock$) and the t_i are object terms.

Formulas The *objective formulas* form the least set where

1. any atom of the form $F(t_1, \ldots, t_n)$ is an objective formula, where F is a fluent predicate symbol of arity n (e.g. *At*) and the t_i are object terms;
2. when t_1 and t_2 are terms of sort object, then $(t_1 = t_2)$ is an objective formula;
3. when t is a non-variable term of sort action, x an object variable, and ϕ, ϕ' are objective formulas, then so are $[t]\phi$, $\mathit{Poss}(t)$, $\exists x\phi$, $\neg\phi$, and $\phi \vee \phi'$.

We read $[t]\phi$ as "ϕ holds after doing action t" and $\mathit{Poss}(t)$ as "action t is possible to execute". We further call an objective formula *static* when it does not contain any action terms. Note that we disallow equalities and quantification over actions. The *subjective formulas* form the least set with

1. if ϕ is an objective formula and $k \geq 0$, then $\mathbf{B}_k\phi$ is a subjective formula and called a *believe atom* at level k;
2. if t_1 and t_2 are terms of sort object, then $(t_1 = t_2)$ is a subjective formula;
3. if φ_1 and φ_2 are subjective formulas and x is a variable of sort object, then $\neg\varphi_1$, $(\varphi_1 \vee \varphi_2)$ and $\exists x\varphi_1$ are also subjective formulas.

The language $\mathcal{SLA}$ is the set of all subjective formulas as defined above. Therefore, very much similar to $\mathcal{SL}$, all (fluent) predicates other than equality must occur within the scope of a $\mathbf{B}_k$ operator, which must not be nested. Here, we further require that also $[t]$ and $\mathit{Poss}(t)$ operators do not appear outside of $\mathbf{B}_k$. The fact that we only study formulas talking about the agent's beliefs about the world state is why the language is called *subjective logic*, or in our case, *subjective logic of actions*. $(\varphi_1 \wedge \varphi_2)$, $\forall x\varphi$, $(\varphi_1 \supset \varphi_2)$ and $(\varphi_1 \equiv \varphi_2)$ are treated as the usual abbreviations.

Programs Programs are composed according to the following grammar:

$$\delta ::= t \mid \phi? \mid (\delta_1; \delta_2) \mid (\delta_1|\delta_2) \mid \pi x.\delta \mid \delta^*$$

Here, t is any (not necessarily ground) term of sort action, ϕ can be any objective formula, and x an object variable. In the presented order, the constructs mean a primitive action, a test, sequence of programs, nondeterministic choice between programs, nondeterministic choice of argument, and nondeterministic iteration.

Regression and Basic Action Theories

Before we define the logic's formal semantics, we introduce basic action theories and regression, following (Lakemeyer and Levesque 2004). The language for basic action theories consists of the objective formulas defined above and extended by another modal operator $\Box$, where $\Box\alpha$ reads "α holds after every sequence of actions.", as well as equality atoms $(t_1 = t_2)$ among action terms, where at most one of the t_i is a variable. A formula without $\mathit{Poss}(t)$ and $[t]$, but possibly containing such action equalities, is called *quasi-static*.

Definition 1 (Basic Action Theory) *Given a set of fluent predicates $\mathcal{F}$, a set of sentences Σ is called a* basic action theory *over $\mathcal{F}$ iff it only mentions the fluents in $\mathcal{F}$ and is of the form $\Sigma = \Sigma_0 \cup \Sigma_d$, where $\Sigma_d = \Sigma_{pre} \cup \Sigma_{post}$ and*

- Σ_0 *is a finite set of static sentences,*
- Σ_{pre} *is a singleton of the form* $\Box(Poss(a) \equiv \pi)$*, where* π *is quasi-static with* a *being the only free variable;*
- Σ_{post} *is a finite set of successor state axioms of the form* $\Box([a]F(\vec{x})) \equiv \gamma_F)$*, one for each fluent* $F \in \mathcal{F}$*, where* γ_F *is a quasi-static formula whose free variables are among* $\vec{x}$ *and* a*.*

In our example, we might have an initial KB Σ_0 containing

$$\begin{array}{l} At(book2, lab), \\ Borrowed(ann, book1) \vee Borrowed(bob, book1), \\ \forall x \forall y Borrowed(x, y) \supset At(y, lib) \end{array} \quad (1)$$

where $Borrowed(x, y)$ means that person x borrowed y yesterday. The precondition axiom Σ_{pre} is given by:

$$\begin{array}{l} \Box Poss(a) \equiv \ \exists x(a = unlock(x) \vee a = lock(x)) \ \vee \\ \quad \exists x \exists y((a = get(x, y) \vee a = put(x, y)) \wedge \neg Locked(y)) \end{array}$$

That is locking or unlocking is always possible, but putting or getting something only when the according location is not locked. The successor state axioms in Σ_{post} are

$$\begin{array}{l} \Box[a]At(x, y) \equiv a = put(x, y) \vee At(x, y) \wedge a \neq get(x, y) \\ \Box[a]Locked(x) \equiv a = lock(x) \vee Locked(x) \wedge a \neq unlock(x) \end{array}$$

Whereas Lakemeyer and Levesque (2004) provide a complete model-theoretic semantics for $\Box$ and $[\cdot]$ within their logics $\mathcal{ES}$, we here adapt a view similar to (Liu, Lakemeyer, and Levesque 2004), i.e. we are interested in the implicit conclusions an agent can draw, given certain explicit beliefs, and the computational costs for doing so. In our encoding, the precondition and successor state axioms of basic action theories are part of the agent's explicit belief, and conclusions about future situations are drawn using regression.

Regression is a method for computing projections by syntactically transforming a formula talking about future situations (after performing certain actions) into an equivalent formula that only talks about the current situation. We use an adaptation of Lakemeyer and Levesque's $\mathcal{ES}$ variant of Reiter's (2001) regression operator as follows:

Definition 2 (Regression) *Formally, for any objective formula* α*, let* $\mathcal{R}[\Sigma_d, \alpha]$*, the regression of* α *wrt* Σ_d*, be the formula* $\mathcal{R}[\Sigma_d, \langle\,\rangle, \alpha]$*, where for any sequence of action terms* σ *(not necessarily ground),* $\mathcal{R}[\Sigma_d, \sigma, \alpha]$ *is defined inductively on* α *by:*

1. $\mathcal{R}[\Sigma_d, \sigma, (t_1 = t_2)] = (t_1 = t_2)$*, where the* t_i *are object terms;*
2. $\mathcal{R}[\Sigma_d, \sigma, (g_1(\vec{t_1}) = g_2(\vec{t_2}))] = \bot$*, where* g_1 *and* g_2 *are distinct action symbols;*
3. $\mathcal{R}[\Sigma_d, \sigma, (g(\vec{t_1}) = g(\vec{t_2}))] = (\vec{t_1} = \vec{t_2})$*;*
4. $\mathcal{R}[\Sigma_d, \sigma, \neg\alpha] = \neg\mathcal{R}[\Sigma_d, \sigma, \alpha]$*;*
5. $\mathcal{R}[\Sigma_d, \sigma, (\alpha \vee \beta)] = (\mathcal{R}[\Sigma_d, \sigma, \alpha] \vee \mathcal{R}[\Sigma_d, \sigma, \beta])$*;*
6. $\mathcal{R}[\Sigma_d, \sigma, \exists x\alpha] = \exists x\mathcal{R}[\Sigma_d, \sigma, \alpha]$*;*
7. $\mathcal{R}[\Sigma_d, \sigma, [t]\alpha] = \mathcal{R}[\Sigma_d, \sigma \cdot t, \alpha]$*;*
8. $\mathcal{R}[\Sigma_d, \sigma, Poss(t)] = \mathcal{R}[\Sigma_d, \sigma, \pi_t^a]$*;*
9. $\mathcal{R}[\Sigma_d, \sigma, F(\vec{t})]$ *is defined inductively on* σ *by:*
   (a) $\mathcal{R}[\Sigma_d, \langle\,\rangle, F(\vec{t})] = F(\vec{t})$*;*
   (b) $\mathcal{R}[\Sigma_d, \sigma \cdot t, F(\vec{t})] = \mathcal{R}[\Sigma_d, \sigma, (\gamma_F)_{\vec{t}}^{a\vec{x}}]$.

Lemma 3 *For any* α*,* $\mathcal{R}[\Sigma_d, \alpha]$ *is static.*

Semantics

For defining the logic's semantics, we need the following definitions from (Liu, Lakemeyer, and Levesque 2004):

Definition 4 (Unit Propagation) *A* clause *is a disjunction of literals, where a* literal *is either a ground atom* $F(\vec{t})$ *or its negation* $\neg F(\vec{t})$*. In a* unit resolution step*, we infer a clause* c *from a unit clause* $\{l\}$ *and some clause* $\{\bar{l}\} \cup c$*, where* $\bar{l}$ *refers to the complement of literal* l*. Let* s *be a (possibly infinite) set of ground clauses. A* unit derivation *of a clause* c *from* s *is given by a sequence* $c_1, \ldots, c_n$*, where* c_n *is* c *and each* c_i *is either an element from* s *or derivable from previous clauses by unit resolution. We then denote the closure of* s *under unit resolution by* $UR(s)$*, which is the set of clauses* c *such that there is some unit derivation of* c *from* s*. Further,* $US(s)$ *is the set of ground clauses* c *such that* c *is subsumed by some clause in* $UR(s)$*.*

Definition 5 (Belief Reduction)

1. $(\mathbf{B}_k c)\downarrow = \mathbf{B}_k c$*, where* c *is a clause;*
2. $(\mathbf{B}_k(t = t'))\downarrow = (t = t')$*;*
3. $(\mathbf{B}_k\neg(t = t'))\downarrow = \neg(t = t')$*;*
4. $(\mathbf{B}_k\neg\neg\phi)\downarrow = \mathbf{B}_k\phi$*;*
5. $(\mathbf{B}_k(\phi \vee \psi))\downarrow = (\mathbf{B}_k\phi \vee \mathbf{B}_k\psi)$*, where* $\phi \vee \psi$ *is not a clause;*
6. $(\mathbf{B}_k\neg(\phi \vee \psi))\downarrow = (\mathbf{B}_k\neg\phi \wedge \mathbf{B}_k\neg\psi)$*;*
7. $(\mathbf{B}_k\exists x\phi)\downarrow = \exists x\mathbf{B}_k\phi$*;*
8. $(\mathbf{B}_k\neg\exists x\phi)\downarrow = \forall x\mathbf{B}_k\neg\phi$.

We can now define the semantics of formulas. A semantic model is given by two things: a *setup* s, which is a (possibly infinite) set of nonempty ground clauses, and represents the agent's explicit beliefs about the current world state. Furthermore we need some $\Sigma_d = \Sigma_{\text{pre}} \cup \Sigma_{\text{post}}$, which represents the agent's explicit beliefs about the world's dynamics.

Definition 6 (Semantics of Formulas)

1. $s \models_{\Sigma_d} (d_1 = d_2)$ *iff* d_1 *and* d_2 *are identical object constants;*
2. $s \models_{\Sigma_d} \neg\varphi$ *iff* $s \not\models_{\Sigma_d} \varphi$*;*
3. $s \models_{\Sigma_d} \varphi_1 \vee \varphi_2$ *iff* $s \models_{\Sigma_d} \varphi_1$ *or* $s \models_{\Sigma_d} \varphi_2$*;*
4. $s \models_{\Sigma_d} \exists x\varphi$ *iff* $s \models_{\Sigma_d} \varphi_d^x$ *for some object constant* d*;*
5. $s \models_{\Sigma_d} \mathbf{B}_k\phi$ *iff one of the following holds:*
   (a) subsumption*:* $k = 0$*,* ϕ *is a clause* c*, and* $c \in US(s)$*;*
   (b) reduction*:* ϕ *is static, but not a clause and* $s \models_{\Sigma_d} (\mathbf{B}_k\phi)\downarrow$*;*
   (c) splitting*:* $k > 0$*,* ϕ *is static and there is some* $c \in s$ *such that for all* $\rho \in c$*,* $s \cup \{\rho\} \models_{\Sigma_d} \mathbf{B}_{k-1}\phi$*;*
   (d) regression*:* ϕ *is not static, and* $s \models_{\Sigma_d} \mathbf{B}_k(\mathcal{R}[\Sigma_d, \phi])$.

The notation φ_t^x denotes φ with all free occurrences of x replaced by t. Apart from item 5d and the extra Σ_d argument, the semantical definition is identical to the one in (Liu, Lakemeyer, and Levesque 2004). Item 5a says that

anything derivable from s by unit propagation is also available at belief level zero, since unit derivations are computationally cheap. According to item 5b, something is believed at level k when a corresponding simpler formula is already believed. Item 5c encodes that case distinctions make reasoning computationally expensive: ϕ is believed at level k when for some clause we can make a case distinction over all its literals and ϕ holds at level $k-1$ in any case. Finally, our addition of item 5d means that a formula involving actions is believed at level k iff its regression is. Because the regression is a static formula, one of the other three cases needs to be applied subsequently.

A sentence α is valid wrt Σ_d, written $\models_{\Sigma_d} \alpha$, if for every setup s, $s \models_{\Sigma_d} \alpha$. If α does not contain any actions, we often also leave out the Σ_d subscript. Typically, we are interested in checking whether, given a set of explicit belief Σ_0, some ϕ holds at believe level k. We therefore introduce the notation $\Sigma_0 \cup \Sigma_d \models_k \phi$ as an abbreviation for $\models_{\Sigma_d} \mathbf{B}_0\Sigma_0 \supset \mathbf{B}_k\phi$, again possibly leaving out Σ_d when no actions are involved.

Properties

When restricted to static formulas, $\mathcal{SLA}$ is identical to $\mathcal{SL}$:

Theorem 7 *Let ϕ be static. Then*

$$\mathbf{B}_0\Sigma_0 \models_{\Sigma_d} \mathbf{B}_k\phi \text{ iff } \models_{\mathcal{SL}} \mathbf{B}_0\Sigma_0 \supset \mathbf{B}_k\phi.$$

Furthermore, we have the following soundness result in terms of entailment of $\mathcal{ES}$ formulas:

Theorem 8 *If $\Sigma_0 \cup \Sigma_d \models_k \phi$, then $\Sigma_0 \cup \Sigma_d \models_{\mathcal{ES}} \phi$.*

This result also establishes the connection to the classical Situation Calculus, of which $\mathcal{ES}$ may be considered a modal dialect. For the details of the two formalisms' relation, we refer the interested reader to (Lakemeyer and Levesque 2005).

We can now reuse results related to these two logics, in particular concerning efficient reasoning with proper$^+$ knowledge bases as defined in (Liu and Levesque 2005):

Definition 9 (Proper$^+$ KBs) *A KB is proper$^+$ if it is a non-empty set of formulas of the form $\forall(e \supset c)$, where e is an ewff and c is a disjunction of literals whose arguments are distinct variables. An* ewff *is a static, quantifier-free formula without fluents and equalities among action terms.*

It is easy to see that the example Σ_0 (1) can be represented in proper$^+$ form. Reasoning with such KBs is tractable in the following sense:

Theorem 10 ((Liu and Levesque 2005)) *If Σ_0 is proper$^+$, ϕ static, and Σ_0 and ϕ use at most j different variables, then whether $\Sigma_0 \models_k \phi$ can be decided in time $O((ln^{j+1})^{k+1})$, where l is the size of ϕ, and n the size of Σ_0.*

That is, reasoning is only exponential in the number of variables used and the belief level. When the ϕ in question is not static, it first needs to be regressed. As the result again is a static formula, the same reasoning procedure can be used. It should however be noted that in the worst case, the length l of the regression result may be in turn exponential in the number of nested occurrences of $[t]$, since in each regression step, a fluent atom is replaced by an entire formula.

Programs

Our program semantics follows the one in (Claßen and Lakemeyer 2008), which is an adaptation of the single step semantics of (De Giacomo, Lespérance, and Levesque 2000). Given a setup s, some Σ_d, a believe level k, and a sequence z of already executed actions, a program δ is mapped to a set of action sequences z', which we call *program execution traces*. The definition uses the notion of program configurations (δ, z), where δ is a program (intuitively what remains to be executed) and z a sequence of ground actions (that have already been performed). A final configuration is one where program execution may legally and successfully terminate, and single step transitions t turn a configuration (δ, z) into a new configuration $(\delta', z \cdot t)$.

Formally, the set of final configurations $\mathcal{F}^{\Sigma_d}_{s,k}$ is the smallest set such that for all δ, δ_1, δ_2, static ϕ and z:

1. $(\phi?, z) \in \mathcal{F}^{\Sigma_d}_{s,k}$ if $s \models_{\Sigma_d} \mathbf{B}_k([z]\phi)$;
2. $(\delta_1; \delta_2, z) \in \mathcal{F}^{\Sigma_d}_{s,k}$ if $(\delta_1, z) \in \mathcal{F}^{\Sigma_d}_{s,k}$ and $(\delta_2, z) \in \mathcal{F}^{\Sigma_d}_{s,k}$;
3. $(\delta_1|\delta_2, z) \in \mathcal{F}^{\Sigma_d}_{s,k}$ if $(\delta_1, z) \in \mathcal{F}^{\Sigma_d}_{s,k}$ or $(\delta_2, z) \in \mathcal{F}^{\Sigma_d}_{s,k}$;
4. $(\pi x.\delta, z) \in \mathcal{F}^{\Sigma_d}_{s,k}$
   if $(\delta^x_d, z) \in \mathcal{F}^{\Sigma_d}_{s,k}$ for some object constant d;
5. $(\delta^*, z) \in \mathcal{F}^{\Sigma_d}_{s,k}$.

Thus, a configuration $(\phi?, z)$ whose remaining program is a test is final wrt s, Σ_d and k if the formula[1] $[z]\phi$ is believed at level k in s and Σ_d. From the above it also follows that $(t, z) \notin \mathcal{F}^{\Sigma_d}_{s,k}$ for atomic t, i.e. if some action t remains to be done, the configuration cannot be final. Further, sequences are only final when the involved subprograms are both final etc. The transition relation among program configurations is given as follows (the empty program nil abbreviates $\top?$):

1. $(t, z) \underset{s,\Sigma_d,k}{\rightarrow} (nil, z \cdot t)$;
2. $(\delta_1; \delta_2, z) \underset{s,\Sigma_d,k}{\rightarrow} (\gamma; \delta_2, z \cdot t)$ if $(\delta_1, z) \underset{s,\Sigma_d,k}{\rightarrow} (\gamma, z \cdot t)$;
3. $(\delta_1; \delta_2, z) \underset{s,\Sigma_d,k}{\rightarrow} (\delta', z \cdot t)$
   if $(\delta_1, z) \in \mathcal{F}^{\Sigma_d}_{s,k}$ and $(\delta_2, z) \underset{s,\Sigma_d,k}{\rightarrow} (\delta', z \cdot t)$;
4. $(\delta_1|\delta_2, z) \underset{s,\Sigma_d,k}{\rightarrow} (\delta', z \cdot t)$
   if $(\delta_1, z) \underset{s,\Sigma_d,k}{\rightarrow} (\delta', z \cdot t)$ or $(\delta_2, z) \underset{s,\Sigma_d,k}{\rightarrow} (\delta', z \cdot t)$;
5. $(\pi x.\delta, z) \underset{s,\Sigma_d,k}{\rightarrow} (\delta', z \cdot t)$
   if $(\delta^x_d, z) \underset{s,\Sigma_d,k}{\rightarrow} (\delta', z \cdot t)$ for some object constant d;
6. $(\delta^*, z) \underset{s,\Sigma_d,k}{\rightarrow} (\gamma; \delta^*, z \cdot t)$ if $(\delta, z) \underset{s,\Sigma_d,k}{\rightarrow} (\gamma, z \cdot t)$.

If $\underset{s,\Sigma_d,k}{\overset{*}{\rightarrow}}$ is the reflexive transitive closure of $\underset{s,\Sigma_d,k}{\rightarrow}$, then

$$\{z' \mid (\delta, z) \underset{s,\Sigma_d,k}{\overset{*}{\rightarrow}} (\delta', z \cdot z') \text{ and } (\delta', z \cdot z') \in \mathcal{F}^{\Sigma_d}_{s,k}\}$$

is the set $\|\delta\|^{s,k}_{\Sigma_d}(z)$ of execution traces of δ, given s, k, Σ_d, at z. Such a trace therefore corresponds to a (possibly empty) sequence of transition steps that lead into a final configuration. Note that because of rule 1, the actions contained in the

[1] We extend $[\cdot]$ to sequences: $[\langle\rangle]\phi \overset{def}{=} \phi$, $[z \cdot t]\phi \overset{def}{=} [z][t]\phi$.

trace are not necessarily all executable according to *Poss*, to allow for reasoning about hypothetical situations including non-reachable ones. In case we are only interested in the actually executable traces of our program δ, we simply have to substitute each occurrence of an action t by $Poss(t)?; t$ in δ.

Given an initial KB Σ_0, we further define

$$\|\delta\|^{\Sigma_0,k}_{\Sigma_d}(z) \stackrel{def}{=} \bigcap \{\|\delta\|^{s,k}_{\Sigma_d}(z) \mid s \models \mathbf{B}_0\Sigma_0\}$$

to be the set of execution traces common to all setups s where Σ_0 is explicitly believed. Our program semantics is sound (but not complete) wrt the program semantics of $\mathcal{ESG}$ as presented in (Claßen and Lakemeyer 2008) as follows:

Theorem 11 *If $z' \in \|\delta\|^{\Sigma_0,k}_{\Sigma_d}(z)$, then for any semantic model w of $\mathcal{ESG}$ such that $w \models \Sigma_0 \cup \Sigma_d$, also $z' \in \|\delta\|^w(z)$.*

Again the relation to classical Golog is given by the results presented in (Claßen and Lakemeyer 2008) and (Lakemeyer and Levesque 2005).

Execution of Programs

In classical Golog, programs are executed off-line, meaning the interpreter first analyzes the entire program to search for a conforming execution trace before performing any actions in the real world. This soon becomes infeasible, in particular when the program is large, the agent has only incomplete world knowledge and has to use sensing to gather information at run-time. IndiGolog (Sardina et al. 2004) therefore executes programs on-line, which means that there is no general look-ahead, but the system just does the next possible action in each step, treating nondeterminism like random choices. Look-ahead is only applied to parts of the program that are explicitly marked by the search operator $\Sigma(\delta)$, thus giving the programmer the control over where the system should spend computational effort for searching. However the search does not pay attention to the computational costs of plans. The main contribution of this paper is to propose the following two new offline search operators:

- $\Lambda_k(\delta, z)$,
  where the set of solution traces is $\|\delta\|^{\Sigma_0,k}_{\Sigma_d}(z)$;
- $\Lambda_*(\delta, z)$,
  where the set of solution traces $\bigcup_{k=0}^{\infty} \|\delta\|^{\Sigma_0,k}_{\Sigma_d}(z)$.

Whereas Λ_k only finds solutions obtainable by reasoning up to belief level k, Λ_* considers all belief levels, where the idea is that lower level solutions will be tested before ones at higher levels, thus preferring plans that require the least computational costs.

The algorithms we are going to present here for computing according solutions make use of the notion of characteristic program graphs as presented in (Claßen and Lakemeyer 2008). Due to space reasons, we will not repeat the (entire) definition here. Intuitively, a program δ is mapped to a graph $\mathcal{G}_\delta = \langle V, E, v_0 \rangle$, with[2]

[2]Here we assume that the program in question does not contain any π operators, which keeps things much simpler. In the future work section we discuss how to extend our approach to this case.

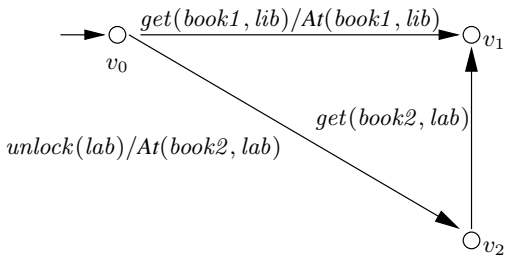

Figure 1: Characteristic Graph of Example Program

- V is a set of vertices of the form $\langle \delta', \varphi' \rangle$, where the δ' is some remaining subprogram and φ' is an objective formula encoding a condition under which program execution might terminate at that node.
- E is a set of labelled edges of the form $v' \stackrel{t/\phi}{\rightarrow} v''$, where the intuition is that a transition with ground action t may be taken from node v' to node v'' when the objective formula ϕ holds.
- $v_0 = \langle \delta, \varphi_0 \rangle \in V$ is the initial node.

The graph for the example program δ from the introduction is shown in Figure 1. The nodes are $v_0 = \langle \delta, \bot \rangle$, $v_1 = \langle nil, \top \rangle$, and $v_2 = \langle get(book2, lab), \bot \rangle$, where $\top$ denotes truth (definable as $\forall x(x = x)$) and $\bot$ falsity ($\neg\top$).

Definition 12 *Let ξ be a path in the characteristic graph. The* path formula $PF(\xi)$ *is defined inductively on its length:*

- $PF(v) = \varphi'$, *if* $v = \langle \delta', \varphi' \rangle$;
- $PF(\xi) = \psi \wedge [t]PF(\xi')$, *if* $\xi = v \stackrel{t/\psi}{\rightarrow} \xi'$.

Further, the path trace $PT(\xi)$ *is defined as*

- $PT(v) = \langle\,\rangle$;
- $PT(\xi) = t \cdot PT(\xi')$, *if* $\xi = v \stackrel{t/\psi}{\rightarrow} \xi'$.

For a fixed believe level k, our method now tests paths of increasing lengths.

Procedure 1 COMP$\Lambda_k(\delta, z)$

```
Determine G_δ = ⟨V, E, v_0⟩
for l = 0, 1, 2 ... do
  for all paths ξ of length l starting in v_0 do
    if Σ_0 ∪ Σ_d |=_k [z]PF(ξ) then
      return PT(ξ)
```

When the set of possible paths is finite like in our example, COMP$\Lambda_k(\delta, z)$ will always terminate for any k and z. In this case we can call that procedure for increasing belief level k:

Procedure 2 COMP$\Lambda_*(\delta, z)$

```
for k = 0, ..., ∞ do
  COMPΛ_k(δ, z)
```

Let us apply COMP$\Lambda_*(\delta, \langle\,\rangle)$ to our example program δ, and let us assume that no actions have been performed so far by the agent, i.e. $z = \langle\,\rangle$. We first call COMP$\Lambda_0(\delta, \langle\,\rangle)$ for belief level $k = 0$. The program graph contains four different paths

that start in the initial node: the only path of length zero, $\xi_0 = v_0$, further two paths of length one, $\xi_{11} = v_0 \rightarrow v_1$ and $\xi_{12} = v_0 \rightarrow v_2$, and finally one path of length two, namely $\xi_2 = v_0 \rightarrow v_2 \rightarrow v_1$. Their respective path formulas are:

$$\begin{array}{lcl} PF(\xi_0) & = & \bot \\ PF(\xi_{11}) & = & \mathit{At}(\mathit{book1}, \mathit{lib}) \wedge [\mathit{get}(\mathit{book1}, \mathit{lib})]\top \\ PF(\xi_{12}) & = & \mathit{At}(\mathit{book2}, \mathit{lab}) \wedge [\mathit{unlock}(\mathit{lab})]\bot \\ PF(\xi_2) & = & \mathit{At}(\mathit{book2}, \mathit{lab}) \wedge \\ & & [\mathit{unlock}(\mathit{lab})](\top \wedge [\mathit{get}(\mathit{book2}, \mathit{lab})]\top) \end{array}$$

Then we need to check for each ξ whether $\Sigma_0 \cup \Sigma_d \models_0 PF(\xi)$, which is the same as $\models_{\Sigma_d} \mathbf{B}_0\Sigma_0 \supset \mathbf{B}_0 PF(\xi)$, which according to rule 5d of the semantics means $\models_{\Sigma_d} \mathbf{B}_0\Sigma_0 \supset \mathbf{B}_0(\mathcal{R}[\Sigma_d, PF(\xi)])$. The regressed versions of the path formulas are, with simplifications:

$$\begin{array}{ll} \mathcal{R}[\Sigma_d, PF(\xi_0)] = \bot & \mathcal{R}[\Sigma_d, PF(\xi_{11})] = \mathit{At}(\mathit{book1}, \mathit{lib}) \\ \mathcal{R}[\Sigma_d, PF(\xi_{12})] = \bot & \mathcal{R}[\Sigma_d, PF(\xi_2)] = \mathit{At}(\mathit{book2}, \mathit{lab}) \end{array}$$

To see that both $\not\models_{\Sigma_d} \mathbf{B}_0\Sigma_0 \supset \mathbf{B}_0 \mathit{At}(\mathit{book1}, \mathit{lib})$ as well as $\not\models_{\Sigma_d} \mathbf{B}_0\Sigma_0 \supset \mathbf{B}_0\bot$, let s be the setup given by

$\{\mathit{At}(\mathit{book2}, \mathit{lab}),$
$\mathit{Borrowed}(\mathit{ann}, \mathit{book1}) \vee \mathit{Borrowed}(\mathit{bob}, \mathit{book1}),$
$\neg \mathit{Borrowed}(d, \mathit{book1}) \vee \mathit{At}(\mathit{book1}, \mathit{lib}) | d$ an obj. const.$\}$.

Then $s \models \mathbf{B}_0\Sigma_0$. As both $\bot$ and $\mathit{At}(\mathit{book1}, \mathit{lib})$ are clauses ($\bot$ is the empty clause), the only possibility is that they are believed by subsumption. However, in this case $UR(s) = s$ (no unit propagation is possible) and there is no clause in s that subsumes $\bot$ or $\mathit{At}(\mathit{book1}, \mathit{lib})$, hence $s \not\models \mathbf{B}_0\bot$ and $s \not\models \mathbf{B}_0 \mathit{At}(\mathit{book1}, \mathit{lib})$. On the other hand, when s is some arbitrary setup with $s \models \mathbf{B}_0\Sigma_0$, then $s \models \mathbf{B}_0 \mathit{At}(\mathit{book2}, \mathit{lab})$ by reduction (treating a set as a conjunction), therefore $\models_{\Sigma_d} \mathbf{B}_0\Sigma_0 \supset \mathbf{B}_0 \mathit{At}(\mathit{book2}, \mathit{lab})$. $\textsc{Comp}\Lambda_0(\delta, \langle\,\rangle)$ thus returns

$$PT(\xi_2) = \langle \mathit{unlock}(\mathit{lab}), \mathit{get}(\mathit{book2}, \mathit{lab}) \rangle.$$

When $k = 1$, we get $\models_{\Sigma_d} \mathbf{B}_0\Sigma_0 \supset \mathbf{B}_1 \mathit{At}(\mathit{book1}, \mathit{lib})$ as follows. Let again s be a setup with $s \models \mathbf{B}_0\Sigma_0$. Then s will contain a clause that subsumes $\mathit{Borrowed}(\mathit{ann}, \mathit{book1}) \vee \mathit{Borrowed}(\mathit{bob}, \mathit{book2})$, and we can split over this clause. As s also must contain a clause subsuming $\neg \mathit{Borrowed}(\mathit{ann}, \mathit{book1}) \vee \mathit{At}(\mathit{book1}, \mathit{lib})$, $\mathit{At}(\mathit{book1}, \mathit{lib})$ can be obtained from $s \cup \{\mathit{Borrowed}(\mathit{ann}, \mathit{book1})\}$ by unit propagation. Similarly for $\mathit{Borrowed}(\mathit{bob}, \mathit{book1})$, therefore $s \models \mathbf{B}_1 \mathit{At}(\mathit{book1}, \mathit{lib})$. Only the $k = 1$ cycle of $\textsc{Comp}\Lambda_*(\delta, \langle\,\rangle)$ will hence yield the solution

$$PT(\xi_{11}) = \langle \mathit{get}(\mathit{book1}, \mathit{lib}) \rangle.$$

Future Work

The approach presented here is work in progress. We are currently working on implementing the method by integrating a corresponding reasoner and search operator into the IndiGolog agent framework (Sardina et al. 2004) to be able to also evaluate it empirically against existing techniques.

There are furthermore many directions for future work at the conceptual level. Instead of solely using regression-based reasoning, we might extend our approach using recent tractability results for the progression of proper$^+$ knowledge bases (Liu and Lakemeyer 2009). As the naive loop of Λ_k obviously will not terminate once the program δ contains an iteration, Λ_* will in this case get stuck at belief level zero, even if there are possibly solutions at higher levels. One may try to apply some sort of dove-tailing technique here, where for any k, only solutions up to a length $l(k)$ are considered.

It is further conceivable to combine our language with an appropriate model for action time costs to be able to study trade-offs between the required reasoning and actual execution time of plans. Also, a variant of our method that computes conditional plans may be useful. In particular, it might be necessary to adapt the notion of *epistemic feasibility* as discussed in (Sardina et al. 2004): Consider a conditional plan in the form of the following program:

$$\phi?; a \;\mid\; \neg\phi?; b$$

where at plan time, the truth value of ϕ was unknown. To be able to execute this program we have to ensure that ϕ will become known at run time, or the executor gets stuck not knowing what step to take next. We therefore also need to extend our formalism appropriately to allow for *sensing actions* that the agent can use in order to gather the necessary information at run time, possibly in combination with knowledge-based programs as described in (Claßen and Lakemeyer 2006).

Finally, integrating the π operators we omitted in the previous section is straightforward in principle, but somewhat tedious. The idea is that whenever some πx is encountered on a path, the corresponding x in the path formula is substituted by a fresh variable x' as different quantifiers may use identical variable names. The obtained path formula then contains a number of free variables. The tractable reasoning procedure presented in (Liu and Levesque 2005) is able to deal with open queries for which it computes a set of variable substitutions. Each such substitution, applied to a path trace with free variables, then corresponds to one possible solution trace.

Conclusion

In this paper we introduced a new logic called $\mathcal{SLA}$ for tractable reasoning with limited beliefs in the presence of action and change. Based on this, we proposed a new planning operator that considers increasing levels of belief, thus preferring solution plans that are the computationally cheapest to discover.

Acknowledgements

This work was supported by the German National Science Foundation (DFG) under grant La 747/14-1. We also thank the anonymous reviewers for their helpful comments.

References

Claßen, J., and Lakemeyer, G. 2006. Foundations for knowledge-based programs using ES. In Doherty, P.; Mylopoulos, J.; and Welty, C. A., eds., *KR*, 318–318. AAAI Press.

Claßen, J., and Lakemeyer, G. 2008. A logic for non-terminating Golog programs. In Brewka, G., and Lang, J., eds., *KR*, 589–599. AAAI Press.

De Giacomo, G.; Lespérance, Y.; and Levesque, H. 2000. ConGolog, a concurrent programming language based on the situation calculus. *Artificial Intelligence* 121(1–2):109–169.

Ferrein, A., and Lakemeyer, G. 2008. Logic-based robot control in highly dynamic domains. *Robot. Auton. Syst.* 56(11):980–991.

Lakemeyer, G., and Levesque, H. J. 2004. Situations, si! situation terms, no! In *KR*, 516–526. AAAI Press.

Lakemeyer, G., and Levesque, H. J. 2005. Semantics for a useful fragment of the situation calculus. In Kaelbling, L. P., and Saffiotti, A., eds., *IJCAI*, 490–496. Professional Book Center.

Levesque, H.; Reiter, R.; Lespérance, Y.; Lin, F.; and Scherl, R. B. 1997. GOLOG: A logic programming language for dynamic domains. *Journal of Logic Programming* 31:59–84.

Liu, Y., and Lakemeyer, G. 2009. On first-order definability and computability of progression for local-effect actions and beyond. In *IJCAI*.

Liu, Y., and Levesque, H. J. 2005. Tractable reasoning in first-order knowledge bases with disjunctive information. In *AAAI*, 639–644. AAAI Press.

Liu, Y.; Lakemeyer, G.; and Levesque, H. 2004. A logic of limited belief for reasoning with disjunctive information. In *KR*, 587–597.

McCarthy, J., and Hayes, P. 1969. Some philosophical problems from the standpoint of artificial intelligence. In *Machine Intelligence 4*. New York: American Elsevier. 463–502.

Reiter, R. 2001. *Knowledge in Action: Logical Foundations for Specifying and Implementing Dynamical Systems*. MIT Press.

Sardina, S.; De Giacomo, G.; Lespérance, Y.; and Levesque, H. J. 2004. On the semantics of deliberation in Indigolog—from theory to implementation. *Annals of Mathematics and Artificial Intelligence* 41(2-4):259–299.