Adapting Golog for Composition of Semantic Web Services

Sheila McIlraith
KSL, Dept. Computer Science
Stanford University
Gates Bldg 2A-248
Stanford, CA 94305
sam@ksl.stanford.edu

Tran Cao Son
Computer Science Department
New Mexico State University
PO Box 30001, MSC CS
Las Cruces, NM 88003, USA
tson@cs.nmsu.edu

Abstract

Motivated by the problem of automatically composing network accessible services, such as those on the World Wide Web, this paper proposes an approach to building agent technology based on the the notion of generic procedures and customizing user constraint. We argue that an augmented version of the logic programming language Golog provides a natural formalism for composing Semantic Web services. To this end, we adapt and extend the Golog language to enable programs that are generic, customizable and usable in the context of the Web. Further, we define logical criteria for these generic procedures that define when they are *knowledge self-sufficient* and *physically self-sufficient*. To support information-gathering combined with search, we propose a middle-ground Golog interpreter that operates under an assumption of reasonable persistence of certain information. These contributions are realized in our augmentation of a ConGolog interpreter that combines online execution of information-providing Web services with offline simulation of world-altering Web services, to determine a sequence of Web Services for subsequent execution. Our implemented system is currently interacting with services on the Web.

1 INTRODUCTION

Two important trends are emerging in the World Wide Web (WWW). The first is the proliferation of so-called Web Services – self-contained, Web-accessible software applications. Familiar examples of Web services include information-gathering services such as the map service at yahoo.com, and world-altering services such as the book-buying service at amazon.com. Today's web was designed primarily for human interpretation and use. The second WWW trend is the emergence of the so-called Semantic Web – a vision for a future Web that is computer-interpretable through description in an unambiguous, semantically well-defined markup language [3]. The semantic Web vision is to mark-up Web content, their properties and relations, in a reasonably expressive *semantic Web markup language* such the description-logic based DAML+OIL [5, 11, 12].

Our interest is in the confluence of these two trends, and we have been developing a semantic markup of the content and capabilities of Web services in an ontology of DAML+OIL [11, 12], called DAML-S [4, 19]. Several metaphors have been exploited in the development of this markup, including the conception of Web services as primitive and complex actions with inputs, outputs, preconditions and effects. In the full paper we will discuss this markup further and tie it in to the technical contributions in this abstract.

The provision of, effectively, a knowledge representation of the properties and capabilities of Web services enables the automation of many tasks. One task of particular interest is automated Web service *selection, composition and interoperation* (e.g., "Make the travel arrangements for my KR'02 conference trip."). Other tasks will be described in the full paper. Disregarding network issues, automated Web service composition (WSC) is an instance of a planning and plan execution problem. However, this application domain has many distinctive features that require and support tailoring. We briefly outline some of these characteristics in the abstract and discuss further in the full paper.

WSC is inherently a task of planning and execution with very incomplete information. Several sequenced information gathering services may be required, that

culminate in the execution of only a few world-altering services. (Imagine making your travel plans on the Web.) Since our actions are the execution of software programs, preconditions are almost always limited to knowledge preconditions – the input parameters of the program. Information gathering services (aka sensors) don't fail (network issues aside). Exogenous events affect the things being sensed. Persistence of knowledge has a temporal extent associated with it (contrast stock prices to the price of a shirt at the Gap), which affects the sequencing of services. Many services perform similar functions. Generally, many plans exist to realize a user goal, and plans are usually short; The search space is short and broad. User input and constraints are a key element of WSC. They serve to prune this space (e.g., choosing from the multitude of airlines and flights available) and to distinguish desirable plan(s).

The unique features of WSC serve to drive the work presented in this paper. Rather than realizing WSC simply as planning, we argue that a number of the activities a user may wish to perform on the (semantic) WWW or within some networked service environment, can be viewed as customizations of reusable, high-level generic procedures. To wit, we all use approximately the same generic procedure to make our travel plans. This helps dictate what Web services we select, but the devil is in the customization to our personal preferences. Our vision is to construct such reusable, high-level generic procedures, and to archive them in sharable generic procedures ontologies so that multiple users can access them. A user could then select a task-specific generic procedure from the ontology and submit it to their agent for execution. The agent would automatically customize the procedure with respect to the user's personal or group-inherited constraints, the current state of the world, and available services, to generate and execute a sequence of requests to Web services to perform the task.

We realize this vision by adapting and extending the logic programming language Golog (e.g., [15, 26, 7]). The adaptations and extensions described in the sections to follow are designed to address the following desiderata of our WSC task. **Generic:** We want to build a class of programs that are sufficiently generic to meet the needs of a variety of different users. Thus programs will often have a high degree of nondeterminism to embody the variability desired by different users. **Customizable:** We want our programs to be easily customizable by individual users. **Usable:** We want our programs to be usable by different agents with different a priori knowledge. As a consequence, we need to ensure that the program accesses all the knowledge it needs, or that certain knowledge is stipulated as a prerequisite to executing the program. Similarly, the program ensures the actions it might use are $Poss$-ible. The programs must be self-sufficient.

2 PRELIMINARIES: GOLOG

Golog (e.g., [15, 26, 7]) is a high-level logic programming language, developed at the University of Toronto, for the specification and execution of complex actions in dynamical domains. It is built on top of the situation calculus (e.g.,[26]), a first-order logical language for reasoning about action and change. Originally, Golog programs operate under the completeness assumption, i.e., in any state of the world fluents are either true or false. Since our intention is to develop agents operating on the Web, it would be unrealistic for us to assume that our agents have complete information about the world in which they are operating. Consequently, we will rely on the situation calculus with knowledge (or with sensing actions) e.g., [28, 26] which we will refer to as situation calculus for short. This also means that our agents will not only have knowledge about world-altering actions but also information-gathering actions that they can perform. In the situation calculus, the state of the world is expressed in terms of functions and relations (fluents) relativized to a particular situation s, e.g., $f(\vec{x}, s)$. To deal with sensing actions in the situation calculus [22], a special fluent K whose first argument is also a situation is introduced. Informally, $K(s', s)$ holds if the agent is in the situation s but thinks that she/he might be in s'. A situation s is a history of the primitive actions (e.g., a) performed from an initial, distinguished situation S_0. The function $do(a, s)$ maps a situation and action into a new situation. A situation calculus theory $\mathcal{D}$ comprises the following sets of axioms (See [26]):

- domain independent foundational axioms of the situation calculus, Σ,
- successor state axioms, $\mathcal{D}_{SS}$, one for K and for every domain fluent, F[1].
- action precondition axioms, $\mathcal{D}_{ap}$, one for every action a in the domain, that serve to define $Poss(a, s)$.
- axioms describing the initial situation, $\mathcal{D}_{S_0}$ (including axioms about K).
- unique names axioms for actions, $\mathcal{D}_{una}$,
- domain closure axioms for actions, $\mathcal{D}_{dca}$[2]

[1] In this paper, we do not impose any conditions on K. For example, one might require that K is reflexive and transitive. See [22, 28].

[2] Not always necessary, but we will require it in 2.1.

We follow the work of Levesque and others (e.g., [14]), and represent the effects of sensing actions by including a sensed fluent axiom for each primitive action a in our domain, $SF(a,s) \equiv \phi_a(s)$.

Golog (alGOl in LOGic) [15] builds on top of the situation calculus by providing a set of extralogical constructs for assembling primitive actions, defined in the situation calculus, into complex actions that collectively comprise a program, δ. Constructs include the following (see [15, 26, 7] for details).

a — primitive actions
$\delta_1;\delta_2$ — sequences
$\phi?$ — tests
$\delta_1 \vert \delta_2$ — nondeterministic choice of actions
$\pi(x)\delta$ — nondeterministic choice of parameters
if ϕ **then** δ_1 **else** δ_2 – conditionals
while ϕ **do** δ — while loops
proc $P(\vec{v})$ δ **endProc** — procedure

These constructs can be used to write programs in the language of a domain theory, e.g.,

$bookAirTicket(\vec{x})$;
if far **then** $bookCar(\vec{y})$ **else** $bookTaxi(\vec{y})$.

Given a domain theory, $\mathcal{D}$ and Golog program δ, program execution must find a sequence of actions $\vec{a}$ such that: $\mathcal{D} \models Do(\delta, S_0, do(\vec{a}, S_0))$. $Do(\delta, S_0, do(\vec{a}, S_0))$ denotes that the Golog program δ, starting execution in S_0 will legally terminate in situation $do(\vec{a}, S_0)$, where $do(\vec{a}, S_0)$ abbreviates $do(a_n, do(a_{n-1}, \ldots, do(a_1, S_0)))$. More details in the full paper. (See [26].)

From DAML-S to Situation Calculus & Golog In the full paper, we describe how to translate Web service represented in DAML-S [4] into our situation calculus-Golog representation. Primitive services are analogous to primitive actions but with inputs and outputs, in addition to preconditions and effects. Preconditions and effects are translated in an obvious manner. Inputs and outputs are translated to knowledge-preconditions and knowledge-effects respectively. Composite Web services are translated into a subset of Golog.

2.1 CUSTOMIZING GOLOG PROGRAMS

In this section we extend Golog to enable individuals to customize a Golog program by specifying personal constraints. To this end, we introduce a new distinguished fluent in the situation calculus called $Desirable(a,s)$, i.e., action a is desirable in situation s. We contrast this with $Poss(a,s)$, i.e. action a is physically possible is situation s. We further restrict the cases in which an action is executable by requiring not only that an action a is $Poss(a,s)$ but further that it is $Desirable(a,s)$. This further constrains the search space for actions when realizing a Golog program. The set of $Desirable$ fluents, one for each action, is referred to as $\mathcal{D}_D$. $Desirable(a,s) \equiv true$ unless otherwise noted.

An individual specifies her personal constraints in our semantic Web markup language. The constraints are expressed in the situation calculus as *necessary conditions for an action a to be desirable*, $\mathcal{D}_{necD}$ of the form: $Desirable(a,s) \supset \omega_i$, and personal constraints $\mathcal{D}_{PC}$ which are formulae, C in the situation calculus.

For example, Marielle's personal constraint is that she would like to book an airline ticket from origin, o to destination, d if the driving time between these two locations is greater than 3 hours. Thus $Desirable(bookAirTicket(o,d,dt,s)) \supset gt(DriveTime(o,d),3,s)$ is included in $\mathcal{D}_{necD}$. Similarly, Marielle has specified dates she must be at home and her constraint is not to be away on those dates. Thus, $\mathcal{D}_{PC}$ includes: $\neg(Away(dt,s) \wedge MustbeHome(dt,s))$ Using $\mathcal{D}_{necD}$ and $\mathcal{D}_{PC}$, and exploiting our successor state axioms and domain closure axioms for actions, $\mathcal{D}_{SS}$ and $\mathcal{D}_{dca}$, we define $Desirable(a,s)$ for every action a as follows:

$$Desirable(A(\vec{x}),s) \equiv \Omega_A \wedge \bigwedge_{C \in D_{PC}} \Omega_{PC},$$

s.t. $\Omega_A = \omega_1 \vee \ldots \vee \omega_n$, for each ω_i of (2.1). E.g., $\Omega_{bookAirTicket} = gt(DriveTime(o,d),3,s)$ and $\Omega_{PC} \equiv \mathcal{R}^*[C(do(A(\vec{x}),s))]$ where $\mathcal{R}^*$ is repeated regression rewriting (e.g., [26]) of $C(do(A(\vec{x}),s))$, the constraints relativized to $do(a,s)$, using the successor state axioms, $\mathcal{D}_{SS}$ from $\mathcal{D}$. E.g.,

$$\Omega_{PC} \equiv \mathcal{R}^*[\neg(Away(dt, do(bookAirTicket(o,d,dt,s))) \wedge MustbeHome(dt, do(bookAirTicket(o,d,dt,s))))]$$

We rewrite this expression using the successor state axioms for fluents $Away(dt,s)$ and $MustbeHome(dt,s)$. E.g.,

$$Away(dt, do(a,s)) \equiv [(a = bookAirTicket(o,d,dt) \wedge d \neq Home) \vee (Away(dt,s) \wedge \neg(a = bookAirTicket(o,d,dt) \wedge d = Home))]$$

$$MustbeHome(dt, do(a,s)) \equiv MustbeHome(dt,s)$$

From this we determine:

$$Desirable(bookAirTicket(o,d,dt,s)) \equiv gt(DriveTime(o,d),3,s) \wedge (d = Home \vee \neg MustbeHome(dt,s))$$

Having computed $\mathcal{D}_D$, we include it in $\mathcal{D}$. In addition to computing $\mathcal{D}_D$, the set of *Desirable* fluents, we also modify the computational semantics of our dialect of Golog. In particular, we adopt the *computational semantics* for Golog. (See [7] for details.) Two predicates are used to define the semantics. $Trans(\delta, s, \delta', s')$ is intended to say that the program δ in situation s may legally execute one step, ending in situation s' with the program δ' remaining. $Final(\delta, s)$ is intended to say that the program δ may legally terminate in situation s. We require one change in the definition to incorporate *Desirable*. In particular, (1) is replaced by (2).

$$Trans(a, s, \delta', s') \equiv Poss(a[s], s) \wedge \delta' = nil \wedge s' = do(a[s], s) \quad (1)$$

$$Trans(a, s, \delta', s') \equiv Poss(a[s], s) \wedge Desirable(a[s], s) \wedge \delta' = nil \wedge s' = do(a[s], s) \quad (2)$$

This approach has many advantages. In the full paper we provide an extended discussion of the merits and limitations of this approach, comparing it to solutions for dealing with other constraints, e.g., [16]. To summarize, this approach is elaboration tolerant [18], individual's customized $\mathcal{D}_D$ are easily added to an existing action theory and easily updated through modular rewriting. *Desirable* has advantages over many other approaches to determining preferred sequences of actions since it prunes the search space for terminating situations, rather than pruning situations after they have been found. *Desirable* is easily implemented as an augmentation of most existing Golog interpreters, and the general approach lends itself to similar specification of other deontic constraints.

2.2 ADDING THE ORDER CONSTRUCT

In the previous subsection we described a way to customize Golog programs by incorporating user constraints. In order for Golog programs to be customizable and generic, they must have some nondeterminism to enable a variety of different choice points to incorporate user's constraints. Golog's nondeterministic choice of actions construct ($|$) and nondeterministic choice of arguments construct (π) both provide for nondeterminism in Golog programs.

In contrast, the sequence construct (;) provides no such flexibility, and can be overly constraining. Consider the program: $bookAirTicket(\vec{x}); bookCar(\vec{y})$. The ; construct dictates that $bookCar(\vec{y})$ must be performed in the situation resulting from performing $bookAirTicket(\vec{x})$ and that $Poss(bookCar(\vec{y}), do(bookAirTicket(\vec{x}), s)$ must be true, otherwise the program will fail. Imagine that the precondition $Poss(bookCar(\vec{y}), s)$ dictates that the user's credit card not be over its limit. If $Poss$ is not true, we would like for the agent executing the program to have the flexibility to perform a sequence of actions to reduce the credit card balance, in order to achieve this precondition, rather than having the program fail. The sequence construct ; does not provide for this flexibility.

To enable the insertion of actions in between a dictated sequence of actions for the purposes of achieving preconditions, we define a new construct called *order*, designated by the : connective[3] Informally, $a_1 : a_2$ will perform the sequence of action $a_1; a_2$ whenever $Poss(a_2, do(a_1, s))$ is true. However, when it is false, the : construct dictates that Golog search for a sequence of actions $\vec{a}$ that achieves $Poss(a_2, do(\vec{a}, do(a_1, s)))$. This can be achieved by a planner that searches for a sequence of actions $\vec{a}$ to achieve the goal $Poss(a_2, do(\vec{a}, do(a_1, s)))$. For the purpose of this paper, we simplify the definition, restricting a_2 to be a primitive action. The definition is easily extended to an order of complex actions $\delta_1 : \delta_2$. Thus, $a_1 : a_2$ is equivalent to the program $a_1; (\textbf{while } (\neg Poss(a_2)) \textbf{ do } (\pi a)[Poss(a)?; a]); a_2$.

It is easy to see that **while** $(\neg Poss(a_2))$ **do** $(\pi a)[Poss(a)?; a]$ will eventually achieve the precondition for a_2 if they can be achieved.

We extend the computational semantics as follows to include :.

$$Trans(\delta : a, s, \delta', s') \equiv Trans((\delta; achieve(Poss(a[s])); a, s, \delta', s') \quad (3)$$

$$Final(\delta : a, s) \equiv Final(\delta; achieve(Poss(a[s])); a, s) \quad (4)$$

where $achieve(G) =$ **while** $(\neg G)$ **do** $(\pi a)[Poss(a)?; a]$. Since *achieve* is defined in terms of existing Golog constructs, the definitions of $Trans$ and $Final$ follow from previous definitions.

Note that the order construct, ':' introduces undirected search into the instantiation process of Golog programs and though well-motivated for many programs, should be used with some discretion because of the potential computational overhead. In the full paper we provide a more extensive discussion of the order construct including improvements upon the simplistic action selection mechanism used by *achieve*, the simple extension to : to incorporate *Desirable*, and the utility of : and variants as a tool for specifying narrative, as first proposed in [24].

[3] Strictly speaking, this is shorthand for a combination of existing constructs.

2.3 SELF-SUFFICIENT PROGRAMS

Now that our Golog programs are customizable and can be encoded generically, we wish them to be usable. Sensing actions are used when the agent has incomplete knowledge of the initial state (almost always for nontrivial WSC), or when exogenous actions exist that change the world in ways the agent's theory of the world does not predict. Web service compositions often have the characteristic of sequences of information-gathering services, performed to distinguish subsequent world-altering services. In our work, we need to define Golog programs that can be used by a variety of different agents without making assumptions about what the agent knows. As such, we want to ensure that our Golog programs are self-sufficient with respect to obtaining the knowledge that they require to execute the program. Further, we wish our programs to ensure that all preconditions for an action are realized if in question.

To make this concrete, we define the notion of a Golog program δ being *self-sufficient* with respect to an action theory $\mathcal{D}$ and *kernel initial state*, $Init_\delta$. $Init_\delta$ is a formula relativized to (suppressed) situation s, denoting the necessary preconditions for executing δ. To characterize self-sufficiency, we introduce the predicate $ssf(\delta, s)$. $ssf(\delta, s)$ is defined inductively over the structure of δ.

$$ssf(nil, s) \equiv true \tag{5}$$

$$ssf(\phi?, s) \equiv \mathbf{KWhether}^4(\phi, s) \tag{6}$$

$$ssf(a, s) \equiv \mathbf{KWhether}(Poss(a[s], s)) \wedge \mathbf{KWhether}(Desirable(a[s], s)) \tag{7}$$

$$ssf(if\ \phi\ then\ \delta_1\ else\ \delta_2, s) \equiv \mathbf{KWhether}(\phi, s) \wedge (\phi[s] \supset ssf(\delta_1, s)) \wedge (\neg\phi[s] \supset ssf(\delta_2, s)) \tag{8}$$

$$ssf(\delta_1; \delta_2, s) \equiv ssf(\delta_1, s) \wedge \forall s'.[Trans(\delta_1, s, nil, s') \supset ssf(\delta_2, s')] \tag{9}$$

$$ssf(\delta_1|\delta_2, s) \equiv ssf(\delta_1, s) \wedge ssf(\delta_2, s) \tag{10}$$

$$ssf(while\ \phi\ do\ \sigma, s) \equiv \mathbf{KWhether}(\phi, s) \wedge (\phi[s] \supset \forall s'.[Trans(\sigma, s, nil, s') \supset ssf(while\ \phi\ do\ \sigma, s')]) \tag{11}$$

$$ssf(\Pi.x[\phi(x), \sigma], s) \equiv \exists x.[\mathbf{KWhether}(\phi(x), s) \wedge ssf(\sigma, s)] \tag{12}$$

$$ssf(\delta_1 : \delta_2, s)\ follows\ from\ (5) - (12) \tag{13}$$

[4] $\mathbf{KWhether}(\phi, s)$ abbreviates a formula indicating that the truth value of ϕ is known [28].

Definition 1 (KSSF: Knowledge Self-Sufficient Program) *$KSSF(\delta, Init_\delta)$, Golog program δ is knowledge self-sufficient relative to action theory $\mathcal{D}$ and kernel initial state $Init_\delta$ iff $\mathcal{D} \cup Init_\delta(S_0)$ is satisfiable and $\mathcal{D} \cup Init_\delta(S_0) \models ssf(\delta, S_0) \equiv true$.*

$KSSF(\delta, Init_\delta)$ ensures that given $Init_\delta$, execution of the Golog program δ will not fail for lack of knowledge. However, the program may fail because it may be im-*Poss*-ible to perform an action. $KSSF$ ensures that the agent knows whether $Poss$ is true, but not that it actually *is* true. To further ensure that our generic procedures are physically self-sufficient, we define $PSSF(\delta, Init_\delta)$.

Definition 2 (PSSF: Physically Self-Sufficient Program) *$PSSF(\delta, Init_\delta)$, Golog program δ is physically self-sufficient relative to a action theory $\mathcal{D}$ and kernel initial state $Init_\delta$ iff $KSSF(\delta, Init_\delta)$ and $\mathcal{D} \cup Init_\delta(S_0) \models \exists s'.[Trans(\delta, S_0, \sigma, s') \wedge Final(\sigma, s')]$.*

We wish to highlight the work of Ernie Davis [6], which we became aware of when we first presented ssf [21]. There are many similarities to our work. We discuss this in more detail in the full paper. One significant difference is that there is no distinction between (what we distinguish as) knowledge sufficiency and physically sufficiency in his framework, i.e., for a plan to be executable, he requires that the agent has the knowledge to execute it and that it must be physically possible. Further, we construct the ssf condition from the situation calculus theory for primitive actions that can be regressed over situations and verified in the initial situation. He develops a set of rules that can be used to check for plan executability. The set of rules, is sufficient but not necessary.

3 MIDDLE-GROUND EXECUTION

In building a Golog interpreter that incorporates sensing actions, the interplay between sensing and execution of world-altering actions can be complex and a number of different approaches have been discussed (e.g., [8, 13, 25]). While [8] and [25] advocate the use of an online interpreter to reason with sensing actions, [13] suggests the use of an offline interpreter with conditional plans. The trade-off is clear. An online interpreter is incomplete because no backtracking is allowed while an offline interpreter is computationally expensive due to the much larger search space, and the need to generate conditional plans, if sensing actions are involved. The choice between an online or offline interpreter depends on properties of the domain, and in particular, since exogenous actions can affect the value

of fluents, on the temporal extent of the persistence of the information being sensed. In a mobile robotics domain, an online interpreter is often more appropriate, whereas an offline interpreter is more appropriate for contingency planning.

We define a middle ground between offline and online execution, which we argue is appropriate for a large class of semantic Web WSC applications. Our middle-ground interpreter (MG) senses online to collect the relevant information needed in the Golog program, while only simulating the effects of world-altering actions. By executing sensing actions rather than branching and creating a conditional plan, MG reduces search space size, while maintaining the ability to backtrack by merely simulating world-altering actions, initially. The outcome is a sequence of world-altering actions that are subsequently executed[5]. Humans often follow this approach, collecting information on the Web (e.g., flight schedules) while only simulating the world-altering actions (buying tickets, etc.) in their head until they have a completed plan to execute.

Of course, the veracity of MG is predicated on an important assumption – that the information being gathered, and upon which world-altering actions are being selected, persists. We assume that the fluents MG is sensing persist for a reasonable period of time, and that none of the actions in the program cause this assumption to be violated. This assumption is generally true of much of the information we access on the Web (e.g., flight schedules, store merchandise), but not all (e.g., stock prices). This assumption is much less pervasive in mobile robotic applications where we may assume persistence for milliseconds, rather than minutes or hours. We formalize this assumption as follows.

Definition 3 (Conditioned-on Fluent) *Fluent C is a conditioned-on fluent in Golog program δ iff for some conditional action A with condition ϕ of the form:* **if** ϕ **then** δ_1 **else** δ_2; **while** ϕ **do** δ; $\phi?$; $\Pi x[\phi, \delta]$. *C appears in the formula ϕ.*

Definition 4 (Invocation and Reasonable Persistence (IRP) Assumption) *Golog program and kernel initial state $(\delta, Init_\delta)$ adhere to the invocation and reasonable persistence assumption if*

1. *Non-knowledge preconditions for sensing actions are true in $D_{S_0} \cup Init_\delta(S_0)$.*

2. *Knowledge of preconditions for actions and conditioned-on fluents C in δ, doesn't change exogenously. persists*[6].

Condition 1 ensures that all sensing actions can be executed by the MG interpreter. Condition 2 ensures that decisions are predicated on correct information. Condition 1 may seem extreme, but, as we argued earlier in this paper, by their nature, Web services generally only have knowledge preconditions. The persistence of knowledge in Condition 2, trivially holds from the frame assumption for knowledge. This condition addresses change by subsequent or exogenous actions.

In the full paper we establish the veracity of MG under the IRP Assumption. Space precludes introducing all the notation necessary to state this result formally. It is in the full paper. The following is only intended to be descriptive.

Informal Theorem Description 1 (Veracity of MG) *Given an action theory D and Golog program δ such that $PSSF(\delta, Init_\delta)$, and $(\delta, Init_\delta)$ adheres to IRP, let $\vec{a}$ be the sequence of world-altering actions selected by the middle-ground interpreter for subsequent execution, then assuming no sensor errors*[7]*, the middle-ground interpreter yields the same states for all fluents $F \in \mathcal{F}$ as an online interpreter with an oracle that chooses $\vec{a}$ at the appropriate branch points.*

In cases where the IRP Assumption is at risk of being violated, the sequence of world-altering actions generated by MG could be executed with an online execution monitoring system that re-performs sensing actions to verify, immediately prior to execution, that critical persistence assumptions have not been violated. In the case where the IRP Assumption does not hold for some or all conditioned-on fluents in a Golog program, MG could be integrated with an interpreter that builds conditional plans for branch points that do not adhere to IRP, following the approach proposed in [13]. The explicit encoding of search areas in a program, as proposed by [8] through the addition of their Σ search construct, can achieve some of the same functionality as our middle-ground interpreter. Indeed, the principle defined above, together with an annotation of the temporal extent of conditioned-on fluents within the action theory provides a means of automatically generating programs with embedded search operators Σ, as proposed in [8]. This will be detailed in the full paper.

[5] At this stage they can alternately be shown to a human for approval before execution. Our interpreter can also generate and present multiple alternate courses of action.

[6] I.e., no subsequent actions in the program change the value of sensed fluents.

[7] Trivially true of virtually all current-day information-gathering Web services.

3.1 MIDDLE-GROUND INTERPRETER

We have modified the ConGolog offline interpreter in [8, 7] to account for user constraints and the order construct. We have also provided a means of encoding the sensed fluent axioms that realizes the strategy for our middle-ground interpreter, described above. We discuss each of the modifications in further detail.

User customizing constraints: We have modified the ConGolog interpreter in [8, 7] to take into consideration personal constraints in a rather straightforward and elegant way. We replaced the following code:

```
trans(A,S,R,S1) :- primAct(A), (poss(A,S), R=nil, S1=do(A,S)); fail.
```

of the ConGolog interpreter with

```
trans(A,S,R,S1) :- primAct(A),(poss(A,S), desirable(A,S),
    R=nil, S1=do(A,S)); fail.
```

This ensures that every action selected by the interpreter is also *Desirable*.

Order Construct: To include the order construct :, we added the following rules to our interpreter:

```
final(P:A, S):- action(A), final([P;achieve(poss(A),0);A],S).
trans(P:A,S,R,S1):- action(A),
    trans([P;achieve(poss(A),0);A],S,R,S1).
```

where achieve(Goal,0) is an A*-planner, adapted from the 'World Simplest Breath First Planner' (wsbfp) developed by Reiter [26]. We appeal to its simplicity and the soundness and completeness of the A* algorithm. We include the code in the full paper, but leave it out here to save space. Obviously any planner can be used to accomplish this task. We are currently investigating the effectiveness of other planners (e.g., regression planners).

Sensing Actions: A common approach to incorporating sensing actions into a situation calculus theory is through the use of the distinguished sensed-fluent predicate, SF. $SF(a, s)$ states that action a returns the binary sensing result *true* when the fluent sensed by a, F is true in the situation s. Assuming each sensing action senses one fluent, we would include one sensed-fluent axiom for each action, $SF(a, s) \equiv F_a(s)$ [26].

To accommodate both backtracking and sensing, we assume that the truth value of $SF(a, s)$ can be determined by executing *an external function call*, denoted by $exec(a, s)$. Under this assumption, the truth value of F_a in s can be determined by making the external function call $exec(a, s)$. Whenever the execution succeeds, F_a is true; otherwise, it is false. This, together with the invocation and reasonable persistence assumption, allows us to write the successor state axiom of F_a in the following form:

$$F_a(do(a, s)) \equiv exec(a, s) \tag{14}$$

Observe that guarded action theories [9] are similar to action theories encoded in this way in that they no longer contain the fluent SF. This allows a treatment of sensed fluents as ordinary fluents if the external function call $exec(a, s)$ can be made and its termination code can be incorporated. This is simple to implement in a PROLOG interpreter. Equation (14) is translated into PROLOG as follows

```
holds(F, do(A,S)):- exec(A,S).
```

where F denotes F_a. Thus, we only need to provide the set of rules that call the action A, i.e., for each sensing action A, the theory contains a rule of the form

```
exec(A,S):- execute A ...
```

Notice that the execution of action A is domain dependent, and hence, the above rule will be a part of the situation calculus action theory rather than a part of the Golog interpreter[8].

Theorem 1 *Given an action theory $\mathcal{D}$ and Golog program δ such that $PSSF(\delta, Init_\delta)$, and $(\delta, Init_\delta)$ adheres to IRP, if $Prolog(\mathcal{D}, Init_\delta) \vdash_R Do(\delta, S_0, S)$ then there exists a model $\mathcal{M}$ of $\mathcal{D} \cup Init_\delta(S_0)$ such that $\mathcal{M} \models Do(\delta, S_0, S)$, where $Prolog(\mathcal{D}, Init_\delta)$ is the set of Prolog rules representing $\mathcal{D}$ and $Init_\delta(S_0)$ and $\vdash_R$ is proof by our Golog interpreter[9].*

We note that action theories in this paper are definitional theories in the sense of [26] if the initial situation axioms $\mathcal{D}_{S_0}$ is complete, i.e., for each fluent, it contains a definition. This can be achieved by making the closed-world assumption (CWA), which, since our programs are self-sufficient, is less egregious. Thus, the next proposition follows immediately from the Implementation Theorem of [26].

Proposition 1 *Given an action theory $\mathcal{D}$ and Golog program δ such that $PSSF(\delta, Init_\delta)$, and $(\delta, Init_\delta)$ adheres to IRP. Then, for all situations S,*

$$Prolog(\mathcal{D}, I_{CWA(S_0)}) \vdash_R Do(\delta, S_0, S) \text{ iff}$$
$$\mathcal{D} \cup CWA(S_0) \cup Init_\delta(S_0) \models Do(\delta, S_0, S),$$

where $\vdash_R$ is proof by our Golog interpreter, $CWA(S_0)$ is defined as $\{F(S_0) \equiv false \mid$ there exists no definition

[8] The offline interpreter with the search operator in [8] can also be modified and used here.

[9] In the full version of the paper, we detail the process of translating an action theory into a Prolog program for use with our interpreter.

of F in $\mathcal{D}_{S_0} \cup Init_\delta(S_0)\}$, and $Prolog(\mathcal{D}, I_{CWA(S_0)})$ is the set of Prolog rules representing $\mathcal{D}$ and $I_{CWA(S_0)} = Init_\delta(S_0) \cup CWA(S_0)$.

4 IMPLEMENTATION

A significant aspect of our contribution is that the research described to this point is also implemented in a running system. In the full paper we describe the complete architecture for our system. A partial description of the architecture can be found at [19]. We also put it all together by presenting our full generic procedure for the travel booking example discussed throughout the paper. We show how the same procedure has generated numerous instantiations based on different user customization constraints, highlighting the versatility of our approach to programming the semantic Web. In this abstract we have not fully connected this work to the semantic Web. This will be addressed in the full paper. Indeed this work is predicated on the existence of semantic markup of Web services. Integration with the semantic markup language DAML-S is ongoing as the language develops and stabilizes [4, 20]. We report on this. This extended abstract, simply provides an overview of the salient features of our implementation.

To realize our agent technology, we started with a simple implementation of an offline ConGolog interpreter in Quintus Prolog 3.2. We have modified and extended this interpreter as described in the previous section. The interpreter was also modified to communicate with the Open Agent Architecture (OAA) agent brokering system [17], to send requests for services, and to relay responses to Golog. Since commercial Web services currently do not utilize semantic markup in order to provide a computer-interpretable API, and computer-interpretable output, we use an information extraction program, World Wide Web Wrapper Factory (W4), to extract the information we need from the HTML output of Web services. All information-gathering actions are performed this way. For obvious practical (and financial!) reasons, world-altering services are not actually executed.

5 SUMMARY & RELATED WORK

In this paper we addressed the problem of automated Web service composition and execution for the semantic Web. We developed and extended theoretical research in reasoning about action and cognitive robotics, implemented it and experimented with it. We addressed the WSC problem through the provision of high-level generic procedures and customizing constraints. We proposed Golog as a natural formalism for this task. As an alternative to planning, our approach does not change the computational complexity of the task of generating a composition. Nevertheless, most Web service compositions are short, and the search space is broad. Consequently, our approach has the potential to drastically reduce the search space, making it computationally advantageous, in addition to being compelling, and easy for the average Web user to use and customize. Our goal was to develop Golog generic procedures that were easy to use, generic, customizable, and that were usable by a variety of users under varying conditions. We augmented Golog with the ability to include customizing user constraints. We also added a new programming construct called *order* that relaxes the notion of *sequence*, enabling the insertion of actions to achieve the precondition for the next action to be performed by the program. This construct facilitates customization as well as enabling more generic procedures. Finally, we defined the notion of knowledge and physically self-sufficient programs that are executable with minimal assumptions about the agent's initial state of knowledge, or the state of the world. Adherence to these criteria makes our generic procedures amenable to wide-spread use. These contributions were implemented as modifications to an existing ConGolog interpreter, along with an implementation of sensing actions (for information-gathering services). We have tested our results with a generic procedure for travel and a variety of different customizing constraints that showcase the effectiveness of our approach. The ConGolog interpreter communicates with Web services through an agent brokering systems and an HTML information extractor. Though our work was focused on Web service composition, the work presented in this paper has broad relevance to a variety of cognitive robotic tasks.

Related Golog work was identified in the body of this paper, with the work in [8] being most closely related. Other important related work which we wish to acknowledge and which we will further discuss in the full paper includes [10], [30], [27], [29], [2], the recent knowledge interpreter in [26], [1] and in particular to [6] and [23].

ACKNOWLEDGEMENTS

We would like to thank the Cognitive Robotics Group at the University of Toronto for providing an initial ConGolog interpreter that we have extended and augmented, and SRI for the use of the Open Agent Architecture software. Finally, we gratefully acknowledge the financial support of the US Defense Advanced Re-

search Projects Agency DAML Program grant number F30602-00-2-0579-P00001. The second author would also like to acknowledge the support of NSF grant NSF-EIA-981072.

References

[1] F. Bacchus and R. Petrick. Modeling an agent's incomplete knowledge during planning and execution. In *Proc. of the Sixth International Conference on Principles of Knowledge Representation and Reasoning*, pages 432–443, 1998.

[2] V. Benjamins et al. IBROW3: An Intel-ligent Brokering Service for Knowledge-Component Reuse on the World Wide Web. In *Proceedings of the 11th Banff Knowledge Acquisition for Knowledge-Based System Workshop (KAW 98), Banff, Canada*, 1998.

[3] T. Berners-Lee and M. Fischetti. *Weaving the Web: The Original Design and Ultimate Destiny of the World Wide Web by its Inventor*. Harper, 1999.

[4] D.-S. Coalition. Daml-s: A daml+oil ontology for web services. Technical report, 2001. http://www.daml.org/services.

[5] DAML-ONT. http://www.daml.org/2000/10/daml-ont.html, 2000.

[6] E. Davis. Knowledge Preconditions for Plans. *Journal of Logic and Computation*, 4(5):721–766, 1994.

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

[8] G. De Giacomo and H. Levesque. An incremental interpreter for high-level programs with sensing. In *Logical Foundations for Cognitive Agents, Contributions in Honor of Ray Reiter*, pages 86–102, 1999.

[9] G. De Giacomo and H. Levesque. Projection using regression and sensors. In *Proc. of the Sixteen International Joint Conference on Artificial Intelligence (IJCAI'99)*, pages 160–165, 1999.

[10] O. Etzioni and D. Weld. A softbot-based interface to the internet. *JACM*, pages 72–76, July 1994.

[11] J. Hendler and D. McGuinness. The DARPA Agent Markup Language. In *IEEE Intelligent Systems Trends and Controversies*, November/December 2000. Available from http://www.ksl.stanford.edu/people/dlm/papers/ieee-daml01-abstract.html.

[12] I. Horrocks, F. van Harmelen, P. Patel-Schneider, T. Berners-Lee, D. Brickley, D. Connolly, M. Dean, S. Decker, D. Fensel, P. Hayes, J. Heflin, J. Hendler, O. Lassila, D. McGuinness, and L. Stein. Daml+oil, March 2001. Available online at: http://www.daml.org/2001/03/daml+oil-index.

[13] G. Lakemeyer. On sensing and off-line interpreting in Golog. In *Logical Foundations for Cognitive Agents, Contr. in Honor of Ray Reiter*, pages 173–187, 1999.

[14] H. Levesque. What is planning in the presence of sensing? In *AAAI 96*, pages 1139–1146, 1996.

[15] H. Levesque, R. Reiter, Y. Lesperance, F. Lin, and R. Scherl. GOLOG: A logic programming language for dynamic domains. *Journal of Logic Programming*, 31(1-3):59–84, April-June 1997.

[16] F. Lin and R. Reiter. State constraints revisited. *Journal of Logic and Computation*, 4(5):655–678, 1994. Special Issue on Action and Processes.

[17] D. L. Martin, A. J. Cheyer, and D. B. Moran. The open agent architecture: A framework for building distributed software systems. *Applied Artificial Intelligence*, 13:91–128, January-March 1999.

[18] J. McCarthy. Mathematical logic in artificial intelligence. *Daedalus*, pages 297–311, Winter, 1988.

[19] S. McIlraith, T. Son, and H. Zeng. Mobilizing the semantic Web with DAML-enabled Web services. In *Proceedings of the Second International Workshop on the Semantic Web (SemWeb'01), Hongkong*, pages 46–53, 2001.

[20] S. McIlraith, T. Son, and H. Zeng. Semantic Web services. In *IEEE Intelligent Systems (Special Issue on the Semantic Web)*, March/April 2001. To appear.

[21] S. McIlraith and T. C. Son. Adapting ConGolog for Programming the Semantics Web. In *Working Notes of Common Sense – The Fifth International Symposium on Logical Formalization of Commonsense Reasoning*, pages 195–202, 2001.

Also in Proceedings of the 4th Workshop on Nonmonotonic Reasoning and Action, IJCAI, August 2001.

[22] R. Moore. A formal theory of knowledge and action. In J. Hobbs and R. Moore, editors, *Formal theories of the commonsense world*. Ablex, Norwood, NJ, 1985.

[23] L. Morgenstern. Knowledge preconditionsfor actions and plans. In *Proceedings of the Tenth International Joint Conference on Artificial Intelligence*, pages 867–874, 1987.

[24] R. Reiter. Narratives as programs. In *Proc. of the Seventh International Conference on Knowledge Representation and Reasoning (KR2000)*, pages 99–108, 2000.

[25] R. Reiter. On knowledge-based programming with sensing in the situation calculus. In *Proc. of the Second International Cognitive Robotics Workshop, Berlin*, 2000.

[26] R. Reiter. *KNOWLEDGE IN ACTION: Logical Foundations for Specifying and Implementing Dynamical Systems*. The MIT Press, 2001.

[27] S. Sardiña. Local conditional high-level robot programs. In *Proceedings of the 4th Workshop on Nonmonotonic Reasoning and Action, IJCAI, August 2001*, pages 195–202, 2001.

[28] R. Scherl and H. Levesque. The frame problem and knowledge producing actions. In *Proceedings of the 12th National Conference on Artificial Intelligence*, pages 689–695. AAAI Press, 1993.

[29] S. Shapiro. Embodied Cassie. In *Proceedings of the AAAI Fall Symposium - Cognitive Robotics*, pages 136–146, 1998.

[30] R. Waldinger. Deductive composition of Web software agents. In *Proc. NASA Wkshp on Formal Approaches to Agent-Based Systems, LNCS*. Springer-Verlag, 2000.