Chapter 2

Background

This chapter covers general background material for the thesis and provides a brief overview of the related literature. We defer more specific technical details and discussion of related work to the individual chapters that follow, where it can be presented in the appropriate context.

Readers familiar with the situation calculus are encouraged to briefly review this chapter. While it does not present any new results, it does introduce some novel notation and definitions which will be needed later in the thesis. They are introduced here to maintain consistency of the presentation. The introductory material on the Mozart programming platform may also be helpful.

We begin by introducing the base language of the situation calculus in Section 2.1, illustrated using examples from the "cooking agents" domain. Section 2.2 introduces the Golog family of programming languages, which are the standard formalism for representing complex tasks in the situation calculus. Reasoning about the knowledge of an agent, or *epistemic reasoning*, is covered in Section 2.3. Related formalisms for reasoning about action and change are briefly discussed in Section 2.4. Finally, Section 2.5 introduces the Mozart programming system, which will be used to implement our multi-agent Golog variant. Basic familiarity with formal logic is assumed throughout; readers requiring background on such material may find a gentle introduction in [43] and a more detailed treatment in [31].

2.1 The Situation Calculus

The situation calculus is a powerful formalism for describing and reasoning about dynamic worlds. It was first introduced by McCarthy and Hayes [70] and has since been significantly expanded and formalised [85, 92]. We use the particular variant due to Reiter et. al. at the University of Toronto, sometimes called the "Toronto school" or "situations-as-histories" version. The formalisation below is based on the standard definitions from [59, 85, 91], but has been slightly generalised to accommodate several existing extensions to the situation calculus, as well as our own forthcoming extensions, in a uniform manner.

Readers familiar with the situation calculus should therefore note some modified notation: the unique names axioms $\mathcal{D}_{una}$ are incorporated into a general background theory $\mathcal{D}_{bg}$; the *Poss* fluent is subsumed by a general class of *action description predicates* defined in $\mathcal{D}_{ad}$; we parameterise the "future situations" predicate $s \sqsubset s'$ to assert that all intermediate actions satisfy a given predicate using $s <_\alpha s'$; and we use the single-step variant of the regression operator, with corresponding definitions of regressable formulae.

2.1.1 Notation

The language $\mathcal{L}_{sitcalc}$ of the situation calculus is a many-sorted language of first-order logic with equality, augmented with a second-order induction axiom, containing the following disjoint sorts:

- Action terms are functions denoting individual instantaneous events that can cause the state of the world to change;
- Situation terms are histories of the actions that have occurred in the world, with the initial situation represented by S_0 and successive situations built using the function $do : Action \times Situation \rightarrow Situation$;
- Object terms represent any other object in the domain.

Fluents are predicates or functions that represent properties of the world that may change between situations, and so take a situation term as their final argument. Predicates and functions that do not take a situation term are called *rigid*. We use the term *primitive fluent* to describe fluents that are directly affected by actions, rather than being defined in terms of other fluents. No functions other than S_0 and *do* produce values of sort Situation.

For concreteness, let us present some formulae from an example domain that will be used throughout the thesis. In the "cooking agents" domain a group of robotic chefs inhabit a kitchen containing various ingredients and utensils, and they must cooperate to prepare a meal. Some example statements from this domain include "Joe does not have the knife initially", "Jim has the knife after he acquires it" and

"It is only possible to acquire an object if nobody else has it". Formally:

$$\neg HasObject(Joe, Knife1, S_0)$$
$$HasObject(Jim, Knife1, do(acquire(Jim, Knife1), S_0))$$
$$Poss(acquire(agt, obj), s) \equiv \neg\exists agt_2 : HasObject(agt_2, obj, s)$$

Here *HasObject* is a primitive fluent, while *Poss* is defined in terms of it.

$\mathcal{L}_{sitcalc}$ contains the standard alphabet of logical connectives, constants $\top$ and $\bot$, countably infinitely many variables of each sort, countably infinitely many predicates of each arity, etc; for a complete definition, consult the foundational paper by Pirri and Reiter [85]. We follow standard naming conventions for the situation calculus: upper-case roman names indicate constants; lower-case roman names indicate variables; greek characters indicate meta-variables or formula templates. All axioms universally close over their free variables at outermost scope. The notation $\bar{t}$ indicates a vector of terms of context-appropriate arity and type. The connectives $\wedge$, $\neg$, $\exists$ are taken as primitive, with $\vee$, $\rightarrow$, $\equiv$, $\forall$ defined in the usual manner.

In multi-agent domains it is customary to introduce a distinct sort Agent to explicitly represent the agents operating in the world, and we will do so here. As seen in the example formulae above, the first argument of each action term gives the performing agent, which can be accessed by the function $actor(a)$.

Complex properties of the state of the world are represented using *uniform formulae.* These are basically logical combinations of fluents referring to a common situation term.

Definition 1 (Uniform Terms). *Let σ be a fixed situation term, r an arbitrary rigid function symbol, f an arbitrary fluent function symbol, and x a variable that is not of sort* Situation. *Then the terms uniform in σ are the smallest set of syntactically-valid terms satisfying:*

$$\tau ::= x \,|\, r(\bar{\tau}) \,|\, f(\bar{\tau}, \sigma)$$

Definition 2 (Uniform Formulae). *Let σ be a fixed situation term, R an arbitrary rigid predicate, F an arbitrary primitive fluent predicate, τ an arbitrary term uniform in σ, and x an arbitrary variable that is not of sort* Situation. *Then the formulae uniform in σ are the smallest set of syntactically-valid formulae satisfying:*

$$\phi ::= F(\bar{\tau}, \sigma) \,|\, R(\bar{\tau}) \,|\, \tau_1 = \tau_2 \,|\, \phi_1 \wedge \phi_2 \,|\, \neg\phi \,|\, \exists x : \phi$$

We will call a formula *uniform* if it is uniform in some situation. The important aspect of this definition is that the formula refers to no situation other than σ, which appears as the final argument of all fluents in the formula. In particular, uniform formulae cannot quantify over situations or compare situation terms, and cannot contain non-primitive fluents.

The meta-variable ϕ is used throughout to refer to an arbitrary uniform formula. Since they represent some aspect of the state of the world, it is frequently useful to evaluate uniform formulae at several different situation terms. The notation $\phi[s']$ represents a uniform formula with the particular situation s' inserted into all its fluents. We may also completely suppress the situation term to simplify the presentation, using ϕ^{-1} to represent a uniform formula with the situation argument removed from all its fluents. For example, given:

$$\phi = HasObject(Jim, Knife1, s) \wedge HasObject(Joe, Bowl2, s)$$

Then we have:

$$\begin{aligned} \phi[s'] &= HasObject(Jim, Knife1, s') \wedge HasObject(Joe, Bowl2, s') \\ \phi^{-1} &= HasObject(Jim, Knife1) \wedge HasObject(Joe, Bowl2) \end{aligned}$$

Note that these are strictly meta-level operations, corresponding to possibly quite complex sentences from the underlying logic. They are *not* terms or operators from the logic itself.

2.1.2 Axioms

The dynamics of a particular domain are captured by a set of sentences from $\mathcal{L}_{sitcalc}$ called a *basic action theory*. Queries about the behaviour of the world are posed as logical entailment queries relative to this theory.

Definition 3 (Basic Action Theory)**.** *A basic action theory, denoted $\mathcal{D}$, is a set of situation calculus sentences (of the specific syntactic form outlined below) describing a particular dynamic world. It consists of the following disjoint sets: the foundational axioms of the situation calculus (Σ); action description axioms defining preconditions etc for each action ($\mathcal{D}_{ad}$); successor state axioms describing how primitive fluents change between situations ($\mathcal{D}_{ssa}$); axioms describing the value of primitive fluents in the initial situation ($\mathcal{D}_{S_0}$); and axioms describing the static background facts of the domain ($\mathcal{D}_{bg}$):*

$$\mathcal{D} = \Sigma \cup \mathcal{D}_{ad} \cup \mathcal{D}_{ssa} \cup \mathcal{D}_{S_0} \cup \mathcal{D}_{bg}$$

These axioms must satisfy some simple consistency criteria to constitute a valid domain description; see [85] for the details. This definition is slightly broader than the standard definitions found in the literature [59, 85, 91] and is designed to accommodate a variety of extensions to the situation calculus in a uniform manner.

We assume an arbitrary, but fixed, basic action theory $\mathcal{D}$.

Background Axioms

The set $\mathcal{D}_{bg}$ characterises the static aspects of the domain, and contains all axioms defining rigid predicates or functions. In particular, it must contain a set of unique names axioms asserting that action terms with different types or arguments are in fact different, e.g.:

$$\begin{gathered} acquire(agt, obj) \neq release(agt, obj) \\ acquire(agt_1, obj_1) = acquire(agt_2, obj_2) \rightarrow agt_1 = agt_2 \wedge obj_1 = obj_2 \end{gathered}$$

It also contains domain closure axioms for the sorts Action, Agent and Object, and defines the function $actor(a)$ to give the agent performing an action. The background axioms are a generalisation of the set $\mathcal{D}_{una}$ commonly found in the literature, which contains only the unique names axioms.

Successor State Axioms

The set $\mathcal{D}_{ssa}$ contains one *successor state axiom* for each primitive fluent in the domain. These axioms provide an elegant monotonic solution to the frame problem for that fluent [92] which has been instrumental to the popularity and utility of the situation calculus. They have the following general form:

$$F(\bar{x}, do(a, s)) \equiv \Phi_F(\bar{x}, a, s)$$

Here Φ_F is uniform in s. While we will make no assumptions about the internal structure of Φ_F, it typically takes the form shown below, which may help elucidate the purpose of these axioms:

$$F(\bar{x}, do(a, s)) \equiv \Phi_F^+(\bar{x}, a, s) \ \vee \ F(\bar{x}, s) \wedge \neg\Phi_F^-(\bar{x}, a, s)$$

Here Φ_F^+ and Φ_F^- are formulae uniform in s, representing the positive and negative effect axioms for that fluent. This may be read as "F is true after performing a if a made it true, or it was previously true and a did not make it false". For example, the dynamics of the *HasObject* fluent may be specified using:

$$\begin{aligned} HasObject(agt, obj, do(a, s)) \equiv\ & a = acquire(agt, obj) \\ & \vee\ HasObject(agt, obj, s) \wedge a \neq release(agt, obj) \end{aligned}$$

For functional fluents, $\mathcal{D}_{ssa}$ contains a similar axiom to specify the value v of the fluent after an action has occurred:

$$f(\bar{x}, do(a, s)) = v \ \equiv\ \Phi_f(v, \bar{x}, a, s)$$

Action Description Predicates

The set $\mathcal{D}_{ad}$ generalises the standard *action precondition axioms* [85] to define fluents that describe various aspects of the performance of an action, which we call *action description predicates*. These are the only non-primitive fluents permitted in a basic action theory. The predicate $Poss(a, s)$ is the canonical example, indicating whether it is possible to perform an action in a given situation. The set $\mathcal{D}_{ad}$ contains a single axiom of the following form, defining the complete set of preconditions for the action variable a, where Π_{Poss} is a formula uniform in s:

$$Poss(a, s) \ \equiv\ \Pi_{Poss}(a, s)$$

Note that this is a slight departure from the standard approach of [85], in which the preconditions for each action type are enumerated individually. The more restrictive approach presented here embodies a domain-closure assumption on the Action sort. If there are finitely many action types then Π_{Poss} is simply the completion of the precondition axioms for each action type. The single-axiom form is necessary when quantifying over "all possible actions" and has been widely used in the literature [96, 124].

In principle, any number of predicates and functions can be defined in this way; a common example is the sensing-result function $SR(a, s)$ which we will describe in Chapter 4. The general notion of an action description predicate allows us to treat all of them in a uniform manner. We will use the meta-variable α to represent an arbitrary action description predicate, and allow the action and situation arguments to be suppressed in a similar way to situation-suppressed uniform formulae.

In preparation for the coming material on extensions to the situation calculus in Section 2.1.4, let us introduce an action description predicate *Legal* that identifies actions that can be legally executed in the real world. In the basic situation calculus, it is simply equivalent to *Poss*:

$$Legal(a,s) \equiv Poss(a,s)$$

As shown by the above, it is often useful to define new action description predicates in terms of simpler existing ones, rather than directly in terms of the primitive fluents of the domain. As long as these definitions are well-founded they can be expanded down to primitive fluents when constructing the basic action theory.

Foundational Axioms

The foundational axioms Σ ensure that situations form a branching-time account of the world state. There is a distinguished situation S_0 called the *initial situation*. Situations in general form a tree structure with the initial situation at the root and $do(a,s)$ constructing the successor situation resulting when the action a is performed in situation s. All situations thus produced are distinct:

$$do(a_1,s_1) = do(a_2,s_2) \rightarrow a_1 = a_2 \wedge s_1 = s_2$$

We abbreviate the performance of several successive actions by writing:

$$do([a_1,\ldots,a_n],s) \overset{\text{def}}{=} do(a_n, do(\ldots, do(a_1,s)))$$

There is also a second-order induction axiom asserting that all situations must be constructed in this way, which is needed to prove statements that universally quantify over situations [89]:

$$\forall P : [P(S_0) \wedge \forall s,a : (P(s) \rightarrow P(do(a,s)))] \rightarrow \forall s : P(s)$$

The relation $s \sqsubset s'$ indicates that s' is in the future of s and is defined as follows:

$$\neg(s \sqsubset S_0)$$
$$s \sqsubset do(a,s') \equiv s \sqsubseteq s'$$

Here $s \sqsubseteq s'$ is the standard abbreviation for $s \sqsubset s' \vee s = s'$. This notion of "in the future of" can be extended to consider only those futures in which all actions satisfy a particular action description predicate. We define as a macro the relation $<_\alpha$ for an arbitrary action description predicate α, with the following definition:

$$s <_\alpha s' \overset{\text{def}}{=} s \sqsubset s' \wedge \forall a, s'' : \big(s \sqsubset do(a,s'') \sqsubseteq s' \rightarrow \alpha[a,s'']\big)$$

It is straightforward to demonstrate that this macro satisfies the following properties, which are analogous to the definition of $\sqsubset$:

$$\neg\,(s <_\alpha S_0)$$
$$s <_\alpha do(a, s') \equiv s \leq_\alpha s' \wedge \alpha[a, s']$$

The *legal situations* are those in which every action was legal to perform in the preceding situation. These are of fundamental importance, as they are the only situations that could be reached in the real world:

$$Legal(s) \overset{\text{def}}{=} S_0 \leq_{Legal} s$$

Initial State Axioms

The set $\mathcal{D}_{S_0}$ describes the actual state of the world before any actions are performed. It is a collection of sentences uniform in S_0 stating what holds in the initial situation. In many domains the initial state can be completely specified, so $\mathcal{D}_{S_0}$ is often in a closed form suitable for efficient automated reasoning.

Note that, unlike [59, 85, 91], we include static facts about the domain in $\mathcal{D}_{bg}$ rather than $\mathcal{D}_{S_0}$. This is entirely a cosmetic change to allow us to talk about these static facts separately from the initial database.

2.1.3 Reasoning

An important feature of the situation calculus is the existence of effective reasoning procedures for certain types of query. These are generally based on syntactic manipulation of a query into a form that is more amenable to reasoning, for example because it can be proven without using some of the axioms from $\mathcal{D}$.

Types of Reasoning

In the general case, answering a query about a basic action theory $\mathcal{D}$ is a theorem-proving task in second-order logic (denoted SOL) due to the induction axiom included in the foundational axioms:

$$\mathcal{D} \models_{SOL} \psi$$

This is clearly problematic for effective automated reasoning, but fortunately there exist particular syntactic forms for which some of the axioms in $\mathcal{D}$ are not required.

If a query only performs *existential* quantification over situation terms, it can be answered without the induction axiom (denoted I) and thus using only first-order logic (FOL) [85]:

$$\mathcal{D} \models_{SOL} \exists s : \psi(s) \ \text{ iff } \ \mathcal{D} - \{I\} \models_{FOL} \exists s : \psi(s)$$

While this is a substantial improvement over requiring a second-order theorem prover, it is still far from an effective technique. Effective reasoning requires that the set of axioms be reduced as much as possible.

In their work on state constraints, Lin and Reiter [66] show how to reduce the task of verifying a state constraint to a reasoning task we call *static domain reasoning*, where only the background axioms need to be considered:

$$\mathcal{D}_{bg} \models_{FOL} \forall s : \phi[s]$$

Since the axioms in $\mathcal{D}_{bg}$ do not mention situation terms, the leading quantification in such queries has no effect – ϕ will be entailed for all s if and only if it is entailed for some s. This is a major improvement because universal quantification over situation terms usually requires the second-order induction axiom. Their work has shown that this requirement can be circumvented in some cases.

Simpler still are queries uniform in the initial situation, which can be answered using only first-order logic and a limited set of axioms:

$$\mathcal{D} \models_{SOL} \phi[S_0] \quad \text{iff} \quad \mathcal{D}_{S_0} \cup \mathcal{D}_{bg} \models_{FOL} \phi[S_0]$$

We call such reasoning *initial situation reasoning.* Since the axioms $\mathcal{D}_{S_0} \cup \mathcal{D}_{bg}$ often satisfy the closed-world assumption, provers such as Prolog can be employed to handle this type of query quite effectively.

Regression

The principle tool for effective reasoning in the situation calculus is the regression meta-operator $\mathcal{R}_{\mathcal{D}}$, a syntactic manipulation that encodes the preconditions and effects of actions into the query itself, meaning fewer axioms are needed for the final reasoning task [85]. The idea is to reduce a query about some future situation to a query about the initial situation only.

There are two styles of regression operator commonly defined in the literature: the single-pass operator as defined in [85] which reduces to S_0 in a single application, the the single-step operator as defined in [98] which operates one action at a time.

We use the single-step variant because it is the more expressive of the two – while it is straightforward to define the single-pass operator in terms of the single-step operator, the reverse is not the case.

Regression is only defined for a certain class of formulae, the *regressable formulae.*

Definition 4 (Regressable Terms)**.** *Let σ be an arbitrary situation term, x an arbitrary variable not of sort situation, r an arbitrary rigid function and f an arbitrary fluent function. Then the regressable terms are the smallest set of syntactically-valid terms satisfying:*

$$\nu ::= \sigma \,|\, x \,|\, f(\bar{\nu}, \sigma) \,|\, r(\bar{\nu})$$

Definition 5 (Regressable Formulae)**.** *Let σ be an arbitrary situation term, x an arbitrary variable not of sort situation, ν an arbitrary regressable term, R an arbitrary rigid predicate, F an arbitrary primitive fluent predicate, and α an arbitrary action description predicate. Then the regressable formulae are the smallest set of syntactically-valid formulae satisfying:*

$$\varphi ::= F(\bar{\nu}, \sigma) \,|\, \alpha(\bar{\nu}, a, \sigma) \,|\, R(\bar{\nu}) \,|\, \nu_1 = \nu_2 \,|\, \neg\varphi \,|\, \varphi_1 \wedge \varphi_2 \,|\, \exists x : \varphi$$

Regressable formulae are more general than uniform formulae. In particular, they can contain action description predicates and may mention different situation terms. They cannot, however, quantify over situation terms or compare situations using the $\sqsubset$ predicate.

The regression operator is then defined using a series of *regression rules* such as those shown below, which mirror the structural definition of regressable formulae.

Definition 6 (Regression Operator)**.** *Let R be a rigid predicate, α be an action description predicate with axiom $\alpha(\bar{\nu}, a, s) \equiv \Pi_\alpha(a, s)$ in $\mathcal{D}_{ad}$, and F be a primitive fluent with axiom $F(\bar{x}, s) \equiv \Phi_F(\bar{x}, s)$ in $\mathcal{D}_{ssa}$. Then the regression of ϕ, denoted $\mathcal{R}_\mathcal{D}(\phi)$, is defined according to the following structural rules:*

$$\mathcal{R}_\mathcal{D}(\varphi_1 \wedge \varphi_2) \stackrel{def}{=} \mathcal{R}_\mathcal{D}(\varphi_1) \wedge \mathcal{R}_\mathcal{D}(\varphi_2)$$
$$\mathcal{R}_\mathcal{D}(\exists x : \varphi) \stackrel{def}{=} \exists x : \mathcal{R}_\mathcal{D}(\varphi)$$
$$\mathcal{R}_\mathcal{D}(\neg\varphi) \stackrel{def}{=} \neg\mathcal{R}_\mathcal{D}(\varphi)$$
$$\mathcal{R}_\mathcal{D}(\alpha(\bar{\nu}, a, \sigma)) \stackrel{def}{=} \mathcal{R}_\mathcal{D}(\Pi_\alpha(\bar{\nu}, a, \sigma))$$
$$\mathcal{R}_\mathcal{D}(F(\bar{\nu}, do(a, \sigma))) \stackrel{def}{=} \Phi_F(\bar{\nu}, a, \sigma)$$
$$\mathcal{R}_\mathcal{D}(F(\bar{\nu}, s)) \stackrel{def}{=} \Phi_F(\bar{\nu}, a, s)$$
$$\mathcal{R}_\mathcal{D}(F(\bar{\nu}, S_0)) \stackrel{def}{=} \Phi_F(\bar{\nu}, a, S_0)$$

We have omitted some technical details here, such as the handling of functional fluents; consult [85] for the details. The key point is that each application of the regression operator replaces action description predicates with their definitions from $\mathcal{D}_{ad}$ and primitive fluents with their successor state axioms from $\mathcal{D}_{ssa}$, "unwinding" a single action from each $do(a, \sigma)$ situation term in the query. If the situation term is not constructed using do, it is left unchanged.

Since $\mathcal{D}$ is fixed, we will henceforth drop the subscript and simply write $\mathcal{R}$ for the regression operator. When dealing with situation-suppressed uniform formulae, we will use a two-argument operator $\mathcal{R}(\phi, a)$ to indicate the regression of ϕ over the action a. It should be read as a shorthand for $\mathcal{R}(\phi[do(a, s)])^{-1}$ using the situation-suppression operator from Section 2.1.1.

Let us briefly state some important properties of the regression operator. First, and most importantly, it preserves equivalence of formulae:

Proposition 1. *Let φ be a regressable formula, then $\mathcal{D} \models \varphi \equiv \mathcal{R}(\varphi)$*

Any formula uniform in $do(a, s)$ is regressable, and the result is uniform in s:

Proposition 2. *Let ϕ be uniform in $do(a, s)$, then $\mathcal{R}(\phi)$ is uniform in s*

Let $\mathcal{R}^*$ denote repeated applications of $\mathcal{R}$ until the formula remains unchanged. Such applications can transform a query about some future situation into a query about the initial situation only:

Proposition 3. *Let ϕ be uniform in $do([a_1 \dots a_n], S_0)$, then $\mathcal{R}^*(\phi)$ is uniform in S_0*

This last property is key to effective reasoning in the situation calculus, as it allows one to answer the *projection problem.* To determine whether ϕ holds in a given future situation, it suffices to determine whether $\mathcal{R}^*(\phi)$ holds in the initial situation. As discussed above, queries uniform in S_0 are much easier to answer. The axioms $\mathcal{D}_{ad}$ and $\mathcal{D}_{ssa}$ are essentially "compiled into the query" by the $\mathcal{R}^*$ operator. While an efficiency gain is not guaranteed, regression has proven a very effective technique in practice [62, 85].

Decidability

Even given the use of regression to reduce the number of axioms required, reasoning still requires first-order logic and is thus only semi-decidable in general. Practical systems implemented on top of the situation calculus typically enforce additional restrictions on the domain in order to gain decidability.

A common restriction is to assume that the Action and Object domains are finite. This allows quantification over these variables to be replaced with finite conjunctions or disjunctions, essentially "propositionalising" the domain [13, 20, 91]. Both static domain and initial situation reasoning can then be performed using propositional logic, which is decidable. This may also be combined with special-purpose decision procedures for particular objects in the domain, such as deciding linear constraints over the integers or reals [91, 93].

A similar, but slightly less onerous restriction, is to ensure that the construction of function terms is well-founded [13]. This prevents building the arbitrarily-nested terms from the Herbrand universe that cause non-termination in first-order theorem provers, again gaining decidability.

Recent work by Gu and Soutchanski [38] has shown how to model some situation calculus domains using to the two-variable fragment of first-order logic. Since this fragment is decidable in general, both static domain and initial situation reasoning are decidable in such domains.

Inductive Reasoning

One class of query that cannot be answered effectively using regression are formulae that universally quantify over situations. Examples of such queries include verifying state constraints ("for all situations, the constraint is satisfied") and determining the impossibility of a goal ("for all situations, the goal is not satisfied"). The difficulty here comes from the induction axiom.

Reiter [89] has shown why the induction axiom is necessary to prove statements that universally quantify over situation terms. This work demonstrates the use of the axiom in manual proofs, but offers no procedure for answering such queries automatically. Other work considering inductive reasoning has focused exclusively on verifying state constraints [11, 66]. While it is possible to automate this verification in some cases, there are currently no general-purpose tools for effectively handling queries that universally quantify over situation terms.

It is this limitation, more than any other, that has restricted the situation calculus to synchronous domains. In asynchronous domains agents must account for potentially arbitrarily-long sequences of hidden actions, which requires universal quantification over situation terms. In Chapter 6 we develop a new reasoning technique to help overcome this limitation.

Progression

While regression has proven quite an effective technique in practice, it has an obvious shortcoming in domains with long histories – the computation required to reason about the current state of the world increases with each action performed.

An alternative approach is *progression*, in which the initial state of the world $\mathcal{D}_{S_0}$ is updated with each action performed, to give a new set of axioms describing the state of the current situation. Although this increases the upfront complexity when an action is performed, this work is amortised over many queries about the updated state. Thielscher [114] makes a compelling case that progression gives better runtime performance in domains with many actions. Why, then, do we focus only on regression in this thesis?

The theoretical foundations of progression in the situation calculus were laid out by Lin and Reiter [67] and come with an important caveat: the progression of a first-order database is not always first-order definable. This conjecture was recently proven by Vassos and Levesque [124], who show that while it is possible to define first-order progressions of a database that are valid for restricted classes of query, a first-order progressed database cannot be complete in general. As such, work on progression in the situation calculus has focused on restricted queries or restricted databases for which first-order progressions exist [68, 123, 125]. By contrast, the regression operator is sound and complete for answering a broad range of queries.

In this thesis, we develop formalisms and reasoning techniques for problems which have not been approached before in the situation calculus. Our first priority must be a sound and complete reasoning tool, for which regression is a good match. Advanced techniques such a progression are considered future work at this stage.

2.1.4 Extensions

The base language of the situation calculus may seem simplistic, lacking many features that would be desirable for modelling rich multi-agent domains. However, it is possible to significantly enrich the domain features that can be modelled while maintaining the elegance and simplicity of the base situation calculus. We now discuss several such extensions that are important in multi-agent domains.

Concurrent Actions

In the basic situation calculus only a single action can occur at any instant. While suitable for most single-agent domains, this limitation is emphatically not suitable for multi-agent systems – several actions can easily occur simultaneously if performed by different agents. Modelling this *true concurrency* is necessary to avoid problems with conflicting or incompatible actions. There is also the potential to utilise concurrency to execute tasks more efficiently. Clearly a solid account of concurrency is required for reasoning about multi-agent teams.

The work of [65, 83, 93] adds true concurrency to the situation calculus by replacing action terms with *sets* of actions that are performed simultaneously. The additional sort CONCURRENT is added to $\mathcal{L}_{sitcalc}$, and the appropriate axioms for set theory are added to $\mathcal{D}_{bg}$. All functions and predicates that take an ACTION term are are modified to take a CONCURRENT term instead. For example, $do(a, s)$ becomes $do(\{a_1, a_2, ...\}, s)$. Successor state axioms are modified to test for set membership rather than equality of action terms. For example, the successor state axiom for $HasObject$ would become:

$$\begin{aligned} HasObject(agt, obj, do(c, s)) \equiv\ & acquire(agt, obj) \in c \\ & \vee\ HasObject(agt, obj, s) \wedge release(agt, obj) \notin c \end{aligned}$$

Since it operates solely by replacing formulae with their equivalents, the regression operator is unchanged by this extension and effective reasoning is still possible.

There is, however, a subtle complication in axiomatising action description predicates such as $Poss$: interaction between primitive actions. A combination of actions is not guaranteed to be possible even if each of the individual actions are. For example, two agents may not be able to acquire the same resource at the same time. This is known as the precondition interaction problem and has undergone extensive research [79, 80, 83]. We make no explicit commitment towards a solution for this problem. Rather, we assume that the axioms in $\mathcal{D}_{ad}$ contain the necessary logic to account for interaction for all action description predicates.

To avoid unintuitive behaviour, we assume that the domain entails the following consistency requirements for the empty set of actions:

Definition 7 (Empty Action Consistency). *A basic action theory $\mathcal{D}$ using concurrent actions must entail the following consistency requirements for the empty set of actions:*

$$\mathcal{D} \models \forall s : \neg Legal(\{\}, s)$$
$$\mathcal{D} \models \forall s : \phi[do(\{\}, s)] \equiv \phi[s]$$

Since true concurrency is such an important aspect of multi-agent systems, we will assume concurrent actions are in use throughout the rest of the thesis.

Time

An explicit notion of time can make coordination between agents easier, as joint actions may be performed at a particular time. It also allows a richer description of the world, particularly in domains such as the cooking agents where time can play an important part in tasks to be performed.

The standard approach to time in the situation calculus is that of [82, 83, 93]. An additional sort TIMEPOINT is introduced, which can be any appropriately-behaved sequence such as integers or reals. The axiomatisation of timepoints is added to $\mathcal{D}_{bg}$, and each action gains an extra argument indicating the time at which is was performed. The functions *time* and *start* are introduced to give the performance time of an action and the start time of a situation respectively. The start time of the initial situation may be defined arbitrarily, but is typically taken to be zero.

However, this approach does not integrate cleanly with concurrent actions: it requires an additional predicate *Coherent* to ensure that the performance time is the same for all members in a set of concurrent actions [93]. The legal situations must be restricted to those in which all actions are coherent.

To avoid this extra complexity, we follow the approach taken in the related formalism of the fluent calculus [69] and attach the temporal component to the set of concurrent actions itself, rather than to each individual action. A similar approach is used in [97] to avoid problems when combining knowledge and time.

Predicates and functions taking terms of sort ACTION are modified to take CONCURRENT#TIMEPOINT pairs, e.g. $do(c, s)$ becomes $do(c\#t, s)$. The new function *start* is added to the foundational axioms with the following definition:

$$start(do(c\#t, s)) = t$$

We must ensure that successor situations have later start times than their preceding situations, by modifying the definition of *Legal*:

$$Legal(c\#t, s) \equiv Poss(c\#t, s) \wedge start(s) < t$$

Introducing timepoints does not affect the regression operator, but does increase the complexity of reasoning as $\mathcal{D}_{bg}$ now contains the axioms of number theory. In practice, we limit predicates about time to express only *linear* relationships, and employ a linear constraint solver for decidable reasoning over the temporal component.

Natural Actions

Natural actions are a special class of exogenous actions, those actions which occur outside of an agent's control [93]. They are classified according to the following requirement: natural actions must occur if it is possible for them to occur, unless an earlier action prevents them. For example, a timer will ring at the time it was set for, unless it is switched off. Such actions are used to model the predictable behaviour of the environment.

Natural actions are identified by the truth of the predicate $Natural(a)$. The times at which natural actions may occur are specified by the $Poss$ predicate. For example, suppose that the fluent $TimerSet(m, s)$ represents the fact that a timer is set to ring in m minutes in situation s. The possibility predicate would entail:

$$Poss(ringTimer\#t, s) \equiv \exists m : \; [TimerSet(m, s) \wedge t = start(s) + m]$$

The timer may thus ring only at its predicted time. To enforce the requirement that natural actions *must* occur whenever possible, the action description predicate $Legal(c\#t, s)$ is adjusted to ensure that $c\#t$ is not legal if natural actions could occur at some earlier time:

$$\begin{aligned} Legal(c\#t, s) \equiv\; & Poss(c\#t, s) \\ & \wedge \; \forall a, t' : \; \big[Natural(a) \wedge Poss(\{a\}\#t', s) \; \rightarrow \; \big(a \in c \vee t < t'\big)\big] \end{aligned}$$

Thus it is only legal to perform actions c at time t if no natural actions can occur in s at a time less than t.

Following this intuition, the *least natural time point* (or "LNTP") of a situation is defined as the earliest time at which a natural action may occur [91]. Rather than adding another axiom, this can be defined using a simple macro:

$$\begin{aligned} \mathbf{LNTP}(s, t) \overset{\text{def}}{=} \; & \exists a : [Natural(a) \wedge Poss(\{a\}\#t, s)] \wedge \\ & \forall a', t' : \; \big[Natural(a') \wedge Poss(\{a'\}\#t, s) \rightarrow t \leq t'\big] \end{aligned}$$

We assume that the theory of action avoids certain pathological cases identified in [91], so that absence of an LNTP implies that no natural actions are possible. That is to say, we assume the following is a consequence of the basic action theory:

$$\mathcal{D} \models [\exists a, t : \; Natural(a) \wedge Poss(\{a\}\#t, s)] \rightarrow [\exists t : \; \mathbf{LNTP}(s, t)]$$

The LNTP is important when planning in the presence of natural actions – one cannot plan to perform some actions at time t if t is greater than the least natural timepoint of the current situation. We also define a related concept, the set of *pending natural actions*, as the set of all natural actions that are possible at the least natural time point:

$$\mathbf{PNA}(s, c) \overset{\text{def}}{=} \exists t : \; \mathbf{LNTP}(s, t) \wedge \forall a : [Natural(a) \wedge Poss(\{a\}\#t, s) \equiv a \in c]$$

Long-Running Tasks

Although all actions in the situation calculus are instantaneous, it is still possible to model long-running tasks that have a finite duration. They are modelled by decomposing them into instantaneous $beginTask$ and $endTask$ actions, and a fluent $DoingTask$ indicating that a task is in progress [83].

In the presence of long-running tasks, a robust account of natural actions is very important – $endTask$ must be a natural action to ensure that any task that is initiated eventually terminates at the appropriate time.

Summary

As can be seen from the discussion above, it is possible to enrich the situation calculus with some very powerful domain features while still maintaining the basic structure of the language, and retaining regression as the principle tool for effective automated reasoning.

While we assume concurrent actions are in use through the rest of this thesis, we shall only refer explicitly to other rich domain features – such as time and natural actions – when we wish to make a special point about their treatment. By uniformly using the predicate $Legal$ to identify actions that can legally be performed in the world, rather than the base $Poss$ predicate, we ensure that our techniques are applicable regardless of the particular domain features being used.

2.2 Golog

Golog is a declarative agent programming language that is the standard approach to specifying complex behaviours in the situation calculus [62]. Testimony to its success are its wide range of applications and many extensions to provide additional functionality [17, 21, 27]. For simplicity, we use the general name "Golog" to refer to the standard family of languages based on this technique, including ConGolog [21] and IndiGolog [17].

2.2.1 Notation

To program an agent using Golog one specifies a situation calculus theory of action, and a program consisting of actions from the theory connected by programming constructs such as if-then-else, while loops, and nondeterministic choice. Table 2.1 lists the standard operators available in various incarnations of the language.

Readers familiar with dynamic logic will recognise some of these operators, but others are unique to first-order formalisms such as Golog. Many Golog operators are

Operator	Meaning
Nil	The empty program
a	Execute action a in the world
$\phi?$	Proceed if condition ϕ is true
$\delta_1;\delta_2$	Execute $\delta_1$followed by δ_2
$\delta_1\|\delta_2$	Execute either δ_1 or δ_2
$\pi(x,\delta(x))$	Nondet. select arguments for δ
$\delta*$	Execute δ zero or more times
$\textbf{if}\,\phi\,\textbf{then}\,\delta_1\,\textbf{else}\,\delta_2$	Exec. δ_1 if ϕ holds, δ_2 otherwise
$\textbf{while}\,\phi\,\textbf{do}\,\delta$	Execute δ while ϕ holds
$\textbf{proc}P(\overrightarrow{x})\delta(\overrightarrow{x})\textbf{end}$	Procedure definition
$\delta_1\|\|\delta_2$	Concurrent execution
$\delta_1 \ll \delta_2$	Prioritised concurrency
$\delta^{\|\|}$	Concurrent iteration
$\Sigma(\delta)$	Plan execution offline

Table 2.1: Operators used in Golog and its descendants

nondeterministic and may be executed in several different ways. It is the task of the agent to plan a deterministic instantiation of the program, a sequence of actions that can legally be performed in the world. Such a sequence is called a *legal execution* of the program.

To get a feel for how these operators can be used, consider some example programs. Figure 2.1 shows a simple program for Jim to wash the dishes. It makes use of the nondeterministic "pick" operator to select and clean a dish that needs washing, and does so in a loop until no dirty dishes remain. The legal executions of this program are sequences of $clean(Jim, d)$ actions, one for each dirty dish in the domain, performed in any order.

```
while $\exists d : Dirty(d)$ do
    $\pi(d, clean(Jim, d))$
end
```

Figure 2.1: A Golog program for washing the dishes

Figure 2.2 shows a program that we will return to in subsequent chapters, giving instructions for how to prepare a simple salad. The procedure *ChopTypeInto* (not shown) directs the specified agent to acquire an ingredient of the specified type, chop it, and place it into the indicated bowl. The procedure *MakeSalad* nondeterministically selects an agent to do this for a lettuce, a carrot, and a tomato. Note the nondeterminism in this program: the agent assigned to handling each ingredient is not specified (π construct), nor is the order in which they should be processed ($||$ construct). There is thus considerable scope for cooperation between agents to effectively carry out this task.

$$\begin{array}{c}
\textbf{proc}\, MakeSalad(dest) \\
[\pi(agt, ChopTypeInto(agt, Lettuce, dest))\,||\\
\pi(agt, ChopTypeInto(agt, Carrot, dest))\,||\\
\pi(agt, ChopTypeInto(agt, Tomato, dest))]\ ;\\
\pi(agt, [acquire(agt, dest)\,;\\
beginTask(agt, mix(dest, 1))\,;\\
endTask(agt, mix(dest, 1))\,;\\
release(agt, dest)])\ \textbf{end}
\end{array}$$

Figure 2.2: A Golog program for making a salad

2.2.2 Semantics

The original semantics of Golog were defined using macro-expansion [62]. The macro $\mathbf{Do}(\delta, s, s')$ was defined to be true if program δ could be successfully executed in situation s, leaving the world in situation s'. However, these semantics could not support the concurrent execution of two programs and were modified with the introduction of ConGolog [21] to use two predicates $Trans(\delta, s, \delta', s')$ and $Final(\delta, s)$ which are capable of representing single steps of execution of the program.

The predicate $Trans(\delta, s, \delta', s')$ holds when executing a step of program δ can cause the world to move from situation s to situation s', after which δ' remains to be executed. It thus characterises single steps of computation. The predicate $Final(\delta, s)$ holds when program δ may legally terminate its execution in situation s. We base our work on the semantics of IndiGolog [17], which builds on ConGolog [21] and is the most feature-full of the standard Golog variants. The full semantics are available in the references, but we present some illustrative examples below.

The transition rule for a program consisting of a single action is straightforward – it transitions by performing the action, provided it is possible in the current situation. Such a program may not terminate its execution since the action remains to be performed:

$$\begin{aligned}
Trans(a, s, \delta', s') &\equiv Poss(a, s) \wedge \delta' = Nil \wedge s' = do(a, s)\\
Final(a, s) &\equiv \bot
\end{aligned}$$

The transition rule for a test operator proceeds only if the test is true, leaving the situation unchanged, and likewise cannot terminate execution until the test has been satisfied:

$$\begin{aligned} Trans(?\phi, s, \delta', s') &\equiv \phi[s] \wedge \delta' = Nil \wedge s' = s \\ Final(?\phi, s) &\equiv \bot \end{aligned}$$

Now consider a simple nondeterministic operator, the "choice" construct that executes one of two alternate programs:

$$\begin{aligned} Trans(\delta_1|\delta_2, s, \delta', s') &\equiv Trans(\delta_1, s, \delta', s') \vee Trans(\delta_2, s, \delta', s') \\ Final(\delta_1|\delta_2, s) &\equiv Final(\delta_1, s) \vee Final(\delta_2, s) \end{aligned}$$

It is possible for this operator to transition in two different ways - by executing a step of execution from the first program, or a step of execution from the second program. Slightly more complicated, but of fundamental important in the next chapter, is the semantics of the concurrency operator:

$$\begin{aligned} Trans(\delta_1||\delta_2, s, \delta', s') &\equiv \exists\gamma : Trans(\delta_1, s, \gamma, s') \wedge \delta' = (\gamma||\delta_2) \\ &\quad \vee \exists\gamma : Trans(\delta_2, s, \gamma, s') \wedge \delta' = (\delta_1||\gamma) \\ Final(\delta_1||\delta_2, s) &\equiv Final(\delta_1, s) \wedge Final(\delta_2, s) \end{aligned}$$

This rule specifies the concurrent-execution operator as an *interleaving* of computation steps. It states that it is possible to single-step the concurrent execution of δ_1 and δ_2 by performing either a step from δ_1 or a step from δ_2, with the remainder γ left to execute concurrently with the other program

Clearly there are two notions of concurrency to be considered in the situation calculus: the possibility of performing several actions at the same instant (*true concurrency*), and the possibility of interleaving the execution of several programs (*interleaved concurrency*). Baier and Pinto [5] have modified ConGolog to incorporate sets of concurrent actions in an attempt to integrate these two forms of concurrency. However, their semantics may call for actions to be performed that are not possible and can also produce unintuitive program behaviour in some cases. A key aspect of our work in Chapter 3 is a robust integration of these two notions of concurrency.

We have omitted many details here that are not relevant to this thesis, such as the second-order axioms necessary to handle recursive procedure definitions. We will denote by $\mathcal{D}_{golog}$ the standard axioms defining $Trans$ and $Final$ [17, 21].

Algorithm 1 The Golog/ConGolog Execution Algorithm for program δ

```
Find a situation $s$ such that:
    $\mathcal{D} \cup \mathcal{D}_{golog} \models \exists s : \mathbf{Do}(\delta, S_0, s)$
for each action in the resulting situation term do
    execute that action
end for
```

2.2.3 Execution Planning

Planning an execution of a Golog program δ can be reduced to a theorem proving task as shown in equation (2.1). Here $Trans^*$ indicates the standard second-order definition for the reflexive transitive closure of $Trans$.

$$\mathcal{D} \cup \mathcal{D}_{golog} \models \exists s, \delta' : \left[Trans^*(\delta, S_0, \delta', s) \wedge Final(\delta', s)\right] \tag{2.1}$$

A constructive proof of this query would produce an instantiation of s, a situation term giving a sequence of actions constituting a legal execution of the program. These actions are then executed one-by-one in the world. Since the program remaining after termination is often not important, the macro **Do** is re-defined in terms of $Trans$ and $Final$ to specify only the resulting situation:

$$\mathbf{Do}(\delta, s, s') \overset{\text{def}}{=} \exists \delta' : Trans^*(\delta, s, \delta', s') \wedge Final(\delta', s')$$

In the original Golog and in ConGolog this forms the entirety of the execution planning process, as these variants require a full legal execution to be planned before any actions are performed in the world. This is referred to as *offline execution.* The Golog execution algorithm is presented in Algorithm 1.

By contrast, IndiGolog allows agents to proceed without planning a full terminating execution of their program, instead searching for a legal next step a in the current situation σ such that:

$$\mathcal{D} \cup \mathcal{D}_{golog} \models \exists a, \delta' : Trans^*(\delta, \sigma, \delta', do(a, \sigma))$$

This next step is then performed immediately, and the process repeats until a terminating configuration is reached. This is referred to as *online execution.* The IndiGolog execution algorithm is presented in Algorithm 2.

In order to incorporate planning into this execution algorithm, IndiGolog introduces an explicit "search" operator $\Sigma(\delta)$, which can only make a transition if the

Algorithm 2 The IndiGolog Execution Algorithm for program δ

```
$\sigma \Leftarrow S_0$
**while** $\mathcal{D} \cup \mathcal{D}_{golog} \not\models Final(\delta, \sigma)$ **do**
  Find an action $a$ and program $\delta'$ such that:
      $\mathcal{D} \cup \mathcal{D}_{golog} \models Trans^*(\delta, \sigma, \delta', do(a, \sigma))$
  Execute the action $a$
  $\sigma \Leftarrow do(a, \sigma)$
  $\delta \Leftarrow \delta'$
**end while**
```

program is guaranteed to eventually terminate successfully:

$$Trans(\Sigma(\delta), s, \delta', s') \equiv \exists s'', \delta'' : Trans(\delta, s, \delta'', s') \wedge \mathbf{Do}(\delta'', s', s'') \wedge \delta' = \Sigma(\delta'')$$

This approach gives the programmer powerful control over the amount of non-determinism in the program, and the amount of planning required to find a legal execution. It also allows the programmer to avoid planning over sensing actions, which can cause an exponential blowup in planning complexity. Sensing actions are simply performed outside the scope of a search operator.

2.2.4 Extensions

There have been a wide range of Golog extensions developed which we will not consider in this thesis. Among them have been extensions to include decision-theoretic [12] and game-theoretic aspects [28, 29], additional control operators such as partially-ordered sequences of actions [6] and hierarchical task networks [34, 111], synchronisation between the individual programs of a team of agents [26], and accounting for continuous change and event triggering [36].

While we will not consider these Gologs in any detail, we do note that each has been a relatively straightforward matter of extending the underlying situation calculus theory and/or the semantics of the Golog operators, and as a result there has been rich cross-pollination between these different works. We therefore hope that our work may in turn be combined with some of these extensions to provide an even richer formalism.

2.3 Epistemic Reasoning

Reasoning about the knowledge of an agent and the combined knowledge of a group of agents, referred to in general as *epistemic reasoning*, is a fundamental aspect of

References

[1] P. Abate, R. Gore, and F. Widmann. An On-the-fly Tableau-based Decision Procedure for PDL-Satisfiability. In *Proceedings of the Fifth Workshop on Methods for Modalities*, 2007.

[2] Pietro Abate and Rajeev Gor. System Description: The Tableau Work Bench. In *Proc. Fifth Workshop on Methods for Modalities*, 2007.

[3] C. Backstrom. Computational Aspects of Reordering Plans. *Journal of Artificial Intelligence Research*, 9:99–137, 1998.

[4] J. Baier and S. McIlraith. On Planning with Programs what Sense. In *Proceedings of the 10th International Conference on Principles of Knowledge Representation and Reasoning (KR'06)*, pages 492–502, 2006.

[5] Jorge Baier and Javier Pinto. Integrating True Concurrency into the Robot Programming Language GOLOG. In *Proceedings of the XIX International Conference of the Chilean Computer Science Society*, pages 179–186, 1999.

[6] C. Baral and T. Son. Extending ConGolog to allow partial ordering. In *Proceeings of the 6th International Workshop on Agent Theories, Architectures, and Languages*, pages 188–204, 2000.

[7] Bradley Bart, James P. Delgrande, and Oliver Schulte. Knowledge and Planning in an Action-Based Multi-agent Framework: A Case Study. In *Advances in Artificial Intelligence*, volume 2056 of *LNAI*, pages 121–130. Springer, 2001.

[8] Alexandru Batlag, Lawrence S. Moss, and Slawomir Solecki. The Logic of Public Announcements and Common Knowledge and Private Suspicions. In *Proceedings of the 7th conference on Theoretical Aspects of Rationality and Knowledge (TARK'98)*, pages 43–56, 1998.

[9] Kristof Van Belleghem, Marc Denecker, and Danny De Schreye. Combining Situation Calculus and Event Calculus. In *Proceedings of the 12th International Conference on Logic Programming*, pages 83–97, 1995.

[10] Kristof Van Belleghem, Marc Denecker, and Danny De Schreye. On the relation between situation calculus and event calculus. *Journal of Logic Programming*, 31:3–37, 1997.

[11] Leopoldo E. Bertossi, Javier Pinto, Pablo Saez, Deepak Kapur, and Mahadevan Subramaniam. Automating Proofs of Integrity Constraints in Situation Calculus. In *Proceedings of the 9th International Syposium on Methodologies for Intelligent Systems*, Lecture Notes in Artificial Intelligence, pages 212–222. Springer, 1996.

[12] Craig Boutilier, Raymond Reiter, Mikhail Soutchanski, and Sebastian Thrun. Decision-Theoretic, High-Level Agent Programming in the Situation Calculus. In *Proceedings of the 17th National Conference on Artificial Intelligence and 12th Conference on Innovative Applications of Artificial Intelligence AAAI'00/IAAI'00*, pages 355–362, 2000.

[13] Ronald J. Brachman and Hector J. Levesque. *Knowledge Representation and Reasoning*. Morgan Kaufmann Publishers, 2004.

[14] Jens Claßen and Gerhard Lakemeyer. A Logic for Non-Terminating Golog Programs. In *Proceedings of the 11th International Conference on Principles of Knowledge Representation and Reasoning (KR'08)*, pages 589–599, 2008.

[15] Patrick Cousot and Radhia Cousot. Constructive Versions of Tarski's Fixed Point Theorems. *Pacific Journal of Mathematics*, 82(1):43–57, 1979.

[16] Ernest Davis and Leora Morgenstern. A First-order Theory of Communication and Multi-agent Plans. *Journal of Logic and Computation*, 15(5):701–749, October 2005.

[17] Giuseppe De Giacomo and Hector Levesque. An Incremental Interpreter for High-Level Programs with Sensing. In *Logical foundation for cognitive agents: contributions in honor of Ray Reiter*. Springer, 1999.

[18] Giuseppe De Giacomo, Evgenia. Ternovska, and Ray. Reiter. Non-terminating processes in the situation calculus. In *In Proceedings of the AAAI'97 Workshop on Robots, Softbots, Immobots: Theories of Action, Planning and Control*, 1997.

[19] Giuseppe De Giacomo, Raymond Reiter, and Mikhail Soutchanski. Execution Monitoring of High-Level Robot Programs. In *Proceedings of the 6th International Conference on Principles of Knowledge Representation and Reasoning (KR'98)*, pages 453–465, 1998.

[20] Giuseppe De Giacomo, Luca Iocchi, Daniele Nardi, and Riccardo Rosati. A Theory and Implementation of Cognitive Mobile Robots. *Journal of Logic and Computation*, 9:759–785, 1999.

[21] Giuseppe De Giacomo, Yves Lespérance, and Hector Levesque. ConGolog, A Concurrent Programming Language Based on the Situation Calculus. *Artificial Intelligence*, 121(1-2):109–169, 2000.

[22] Keith Decker and Victor Lesser. Designing a Family of Coordination Algorithms. In *Proceedings of the 1st International Conference on Multi-Agent Systems (ICMAS'95)*, 1995.

[23] Robert Demolombe and Maria del Pilar Pozos Parra. A Simple and Tractable Extension of Situation Calculus to Epistemic Logic. In *Proceedings of the 12th International Symposium on Foundations of Intelligent Systems (ISMIS'00)*, pages 515–524, 2000.

[24] Silvio do Lago Pereira and Leliane Nunes de Barros. High-Level Robot Programming: An Abductive Approach Using Event Calculus. In *Proceedings of the 17th Brazilian Symposium on Artificial Intelligence*, Advances in Artificial Intelligence, 2004.

[25] Ronald Fagin, Joseph Y. Halpern, Yoram Moses, and Moshe Y. Vardi. *Reasoning about Knowledge.* The MIT Press, Cambridge, Massachesetts, 1995.

[26] Alessandro Farinelli, Alberto Finzi, and Thomas Lukasiewicz. Team Programming in Golog under Partial Observability. In *Proceedings of the 20th International Joint Conference on Artificial Intelligence (IJCAI'07)*, pages 2097–2012, 2007.

[27] A. Ferrein, Ch. Fritz, and G. Lakemeyer. Using Golog for Deliberation and Team Coordination in Robotic Soccer. *Kunstliche Intelligenz*, I:24–43, 2005.

[28] Alberto Finzi and Thomas Lukasiewicz. Game-Theoretic Agent Programming in Golog. In *Proceedings of the 16th biennial European Conference on Artificial Intelligence (ECAI'03)*, 2003.

[29] Alberto Finzi and Thomas Lukasiewicz. Game-Theoretic Golog under Partial Observability. In *Proceedings of the 4th International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS'05)*, pages 1301–1302, 2005.

[30] Michael J. Fischer, Nancy A. Lynch, and Michael S. Paterson. Impossibility of distributed consensus with one faulty process. *Journal of the ACM*, 32(2): 374–382, 1985.

[31] Melvin Fitting. *First-order logic and automated theorem proving (2nd ed.).* Springer-Verlag New York, Inc., Secaucus, NJ, USA, 1996.

[32] Melvin Fitting and Richard L. Mendelsohn. *First-order Modal Logic.* Springer, 1998.

[33] Christian Fritz, Jorge A. Baier, and Sheila A. McIlraith. ConGolog, Sin Trans: Compiling ConGolog into Basic Action Theories for Planning and Beyond. In *Proceedings of the 11th International Conference on Principles of Knowledge Representation and Reasoning (KR'08)*, 2008.

[34] Alfredo Gabaldon. Programming Hierarchical Task Networks in the Situation Calculus. In *Proceedings of the 6th International Conference on AI Planning and Scheduling: Workshop on On-line Planning and Scheduling*, 2002.

[35] Hojjat Ghaderi, Hector Levesque, and Yves Lespérance. A Logical Theory of Coordination and Joint Ability. In *Proceedings of the 22nd AAAI Conference on Artificial Intelligence (AAAI'07)*, pages 421–426, 2007.

[36] Henrik Grosskreutz and Gerhard Lakemeyer. cc-Golog: Towards More Realistic Logic-Based Robot Controllers. In *Proceedings of the 17th National Conference on Artificial Intelligence and 12th Conference on Innovative Applications of Artificial Intelligence AAAI'00/IAAI'00*, pages 476–482, 2000.

[37] Barbara J. Grosz and Sarit Kraus. Collaborative Plans for Complex Group Action. *Artificial Intelligence*, 86(2):269–357, 1996.

[38] Yilian Gu and Mikhail Soutchanski. Decidable Reasoning in a Modified Situation Calculus. In *Proceedings of the 20th International Joint Conference on Artificial Intelligence (IJCAI'07)*, pages 1891–1897, 2007.

[39] Joseph Y. Halpern and Yoram Moses. Knowledge and common knowledge in a distributed environment. *Journal of the ACM*, 37(3):549–587, 1990.

[40] D. Harel, D. Kozen, and J. Tiuryn. *Dynamic Logic*. Foundations of Computing. MIT Press, 2000.

[41] Sief Haridi and Nils Franzén. Tutorial of Oz, 1999. available at http://www.mozart-oz.org/.

[42] L. Kalantari and E. Ternovska. A Model Checker for Verifying ConGolog Programs. In *Proceedings of the 18th AAAI Conference on Artificial Intelligence (AAAI'02)*, 2002.

[43] John Kelly. *The Essence of Logic*. Prentice Hall, Inc., 1996.

[44] Ryan F. Kelly and Adrian R. Pearce. Towards High-Level Programming for Distributed Problem Solving. In *Proceedings of the IEEE/WIC/ACM International Conference on Intelligent Agent Technology (IAT'06)*, pages 490–497, 2006.

[45] Ryan F. Kelly and Adrian R. Pearce. Knowledge and Observations in the Situation Calculus. In *Proceedings of the 6th International Joint Conference on Autonomous Agents and Multi-Agent Systems (AAMAS'07)*, pages 841–843, 2007.

[46] Ryan F. Kelly and Adrian R. Pearce. Property Persistence in the Situation Calculus. In *Proceedings of the 20th International Joint Conference on Artificial Intelligence (IJCAI'07)*, pages 1948–1953, 2007.

[47] Ryan F. Kelly and Adrian R. Pearce. Complex Epistemic Modalities in the Situation Calculus. In *Proceedings of the 11th International Conference on Principles of Knowledge Representation and Reasoning (KR'08)*, pages 611–620, 2008.

[48] S. Khan and Y. Lespérance. ECASL: A Model of Rational Agency for Communicating Agents. In *Proceedings of the 4th International Joint Conference on Autonomous Agents and Multiagent Systems*, 2005.

[49] Barteld Kooi. Dynamic Term-Modal Logic. In *Proceedings of the Workshop on Logic, Rationality and Interaction*, volume 8 of *Texts in Computing*, pages 173–185, Beijing, 2007. College Publications, London.

[50] R. Kowalski and M. Sergot. A logic-based calculus of events. *New Generation Computing*, 4(1):67–95, 1986. ISSN 0288-3635.

[51] Robert Kowalski and Fariba Sadri. Reconciling the Event Calculus with the Situation Calculus. *Journal of Logic Programming*, 31:39–58, 1997.

[52] Gerhard Lakemeyer. On sensing and off-line interpreting in Golog. In *Logical foundation for cognitive agents: contributions in honor of Ray Reiter*. Springer, 1999.

[53] Martin Lange. Model Checking Propositional Dynamic Logic with All Extras. *Journal of Applied Logic*, 4(1):39–49, 2005.

[54] Y. Lespérance, H. J. Levesque, and R. Reiter. A Situation Calculus Approach to Modeling and Programming Agents. In Rao A. and M. Wooldridge, editors, *Foundations and Theories of Rational Agency*, pages 275–299. Kluwer, 1999.

[55] Y. Lespérance, H. J. Levesque, F. Lin, and R. B. Scherl. Ability and Knowing How in the Situation Calculus. *Studia Logica*, 66(1):165–186, October 2000.

[56] Yves Lespérance. On the Epistemic Feasibility of Plans in Multiagent Systems Specifications. In *Proceedings of the 8th International Workshop on Agent Theories, Architectures, and Languages*, volume 2333 of *Lecture Notes in Artificial Intelligence*, pages 69–85, 2001.

[57] Yves Lespérance and Ho-Kong Ng. Integrating Planning into Reactive High-Level Robot Programs. In *In Proceedings of the Second International Cognitive Robotics Workshop*, pages 49–54, 2000.

[58] Yves Lespérance, Todd G. Kelley, John Mylopoulos, and Eric S. K. Yu. Modeling Dynamic Domains with ConGolog. In *Conference on Advanced Information Systems Engineering*, pages 365–380, 1999.

[59] H. Levesque, F. Pirri, , and R. Reiter. Foundations for the Situation Calculus. *Electronic Transactions on Artificial Intelligence,*, 2(3-4):159–178, 1998.

[60] Hector Levesque. Planning with Loops. In *Proceedings of the 19th International Joint Conference on Artificial Intelligence (IJCAI'05)*, pages 509–515, 2005.

[61] Hector Levesque. What is Planning in the Presence of Sensing? In *Proc. of the 13th National Conference on Artificial Intelligence*, pages 1139–1146. AAAI, 1996.

[62] Hector J. Levesque, Ray Reiter, Yves Lespérance, Fangzhen Lin, and Richard B. Scherl. GOLOG: A Logic Programming Language for Dynamic Domains. *Journal of Logic Programming*, 31(1-3):59–83, 1997.

[63] H. R. Lewis and C. H. Papadimitriou. *Elements of the Theory of Computation.* Prentice Hall, Inc., 1981.

[64] F. Lin and H. Levesque. What robots can do: robot programs and effective achievability. *Artificial Intelligence*, 101:201–226, 1998.

[65] F. Lin and Y. Shoham. Concurrent actions in the Situation Calculus. In *Proceedings of the 10th National Conference on Artifical Intelligence (AAAI'92)*, 1992.

[66] Fangzhen Lin and Ray Reiter. State Constraints Revisited. *Journal of Logic and Computation*, 4(5):655–678, 1994.

[67] Fangzhen Lin and Ray Reiter. How to progress a database. *Artificial Intelligence*, 92:131–167, 1997.

[68] Y. Liu and H. J. Levesque. Tractable reasoning with incomplete first-order knowledge in dynamic systems with context-dependent actions. In *Proceedings of the 19th International Joint Conference on Artificial Intelligence (IJCAI'05)*, 2005.

[69] Yves Martin. The Concurrent, Continuous FLUX. In *Proceedings of the 18th International Joint Conference on Artificial Intelligence (IJCAI'03)*, 2003.

[70] John McCarthy and Patrick J. Hayes. Some Philosophical Problems from the Standpoint of Artificial Intelligence. In B. Meltzer and D. Michie, editors, *Machine Intelligence 4*, pages 463–502. Edinburgh University Press, 1969.

[71] Robert C. Moore. Reasoning about Knowledge and Action. Technical Note 191, SRI International, October 1980.

[72] Mogens Nielsen, Gordon D. Plotkin, and Glynn Winskel. Petri Nets, Event Structures and Domains. In *Proceedings of the International Symposium on Semantics of Concurrent Computation*, volume 70 of *Lecture Notes in Computer Science*, pages 266–284, 1979.

[73] Eric Pacuit. Some Comments on History Based Structures. *Journal of Applied Logic*, 5(4):613–624, 2007.

[74] Rohit Parikh and R. Ramanujam. Distributed Processes and the Logic of Knowledge. In *Proceedings of the Conference on Logic of Programs*, pages 256–268. Springer-Verlag, 1985.

[75] M. Peot and D. Smith. Conditional Nonlinear Planning. In *Proceedings of the 1st International Conference on AI Planning Systems*, pages 189–197, 1992.

[76] Ron Petrick and Hector Levesque. Knowledge equivalence in Combined Action Theories. In *Proceedings of the 8th International Conference on Principles of Knowledge Representation and Reasoning (KR'02)*, 2002.

[77] Ronald P. A. Petrick. *A Knowledge-level approach for effective acting, sensing, and planning.* PhD thesis, Department of Computer Science, University of Toronto, Toronto, Ontario, Canada, 2006.

[78] Ronald P. A. Petrick. Cartesian Situations and Knowledge Decomposition in the Situation Calculus. In *Proceedings of the 11th International Conference on Principles of Knowledge Representation and Reasoning (KR'08)*, pages 629–639, 2008.

[79] Javier Pinto. Concurrency and Action Interaction. Unpublished work, submitted for publication: November, 2000. URL `http://www.scs.carleton.ca/~bertossi/javier/papers/pinto00.ps.gz`.

[80] Javier Pinto. Concurrent Actions and Interacting Effects. In *Proceedings of the Sixth International Conference on Principles of Knowledge Representation and Reasoning (KR'98)*, pages 292–303, 1998.

[81] Javier Pinto. Using histories to model observations in theories of action. In *Selected Papers from the Workshop on Reasoning with Incomplete and Changing Information and on Inducing Complex Representations: Learning and Reasoning with Complex Representations*, volume 1359 of *Lecture Notes In Computer Science*, pages 221 – 233, 1998.

[82] Javier Pinto and Raymond Reiter. Reasoning About Time in the Situation Calculus. *Annals of Mathematics and Artificial Intelligence*, 14(2-4):251–268, 1995.

[83] Javier A. Pinto. *Temporal Reasoning in the Situation Calculus.* PhD thesis, Department of Computer Science, University of Toronto, Toronto, Ontario, Canada, 1994.

[84] Fiora Pirri and Ray Reiter. Planning with natural actions in the situation calculus. In *Logic-Based Artificial Intelligence.* Kluwer Press, 2000.

[85] Fiora Pirri and Ray Reiter. Some contributions to the metatheory of the situation calculus. *Journal of the ACM*, 46(3):325–361, 1999.

[86] David A. Plaisted and Yunshan Zhu. Situation Calculus with Aspect. In *Proceedings of the IASTED International Conference on Artificial Intelligence and Soft Computing*, pages 17–20, 1997.

[87] Joachim Posegga and Peter H. Schmitt. Automated Deduction with Shannon Graphs. *Journal of Logic and Computation*, 5(6):697–729, 1995.

[88] Vaughan R. Pratt. Modeling Concurrency with Geometry. In *Conference Record of the 18th Annual ACM Symposium on Principles of Programming Languages*, pages 311–322, 1991.

[89] Ray Reiter. Proving Properties of States in the Situation Calculus. *Artificial Intelligence*, 64:337–351, 1993.

[90] Ray Reiter. Sequential, Temporal GOLOG. In *Proceedings of the Sixth International Conference on Principles of Knowledge Representation and Reasoning (KR'98)*, pages 547–556, Trento, Italy, 1998.

[91] Ray Reiter. *Knowledge in Action: Logical Foundations for Specifying and Implementing Dynamical Systems.* The MIT Press, 2001.

[92] Ray Reiter. The frame problem in situation the calculus: a simple solution (sometimes) and a completeness result for goal regression. In Vladimir Lifschitz, editor, *Artificial intelligence and mathematical theory of computation: papers in honor of John McCarthy*, pages 359–380. Academic Press Professional, Inc., 1991.

[93] Ray Reiter. Natural Actions, Concurrency and Continuous Time in the Situation Calculus. In *Proceedings of the 5th International Conference on Principles of Knowledge Representation and Reasoning (KR'96)*, pages 2–13, 1996.

[94] Peter Van Roy. Logic Programming in Oz with Mozart. In Danny De Schreye, editor, *International Conference on Logic Programming*, pages 38–51. The MIT Press, November 1999.

[95] Sebastian Sardina, Giuseppe De Giacomo, Yves Lespénce, and Hector Levesque. On the Semantics of Deliberation in IndiGolog – From Theory to Implementation. *Annals of Mathematics and Artificial Intelligence*, 41(2–4): 259–299, August 2004.

[96] Francesco Savelli. Existential assertions and quantum levels on the tree of the situation calculus. *Artificial Intelligence*, 170(6):643–652, 2006.

[97] Richard Scherl. Reasoning about the interaction of knowledge, time and concurrent actions in the situation calculus. In *Proceedings of the 18th International Joint Conference on Artificial Intelligence (IJCAI'03)*, pages 1091–1098, 2003.

[98] Richard Scherl and Hector Levesque. Knowledge, Action, and the Frame Problem. *Artificial Intelligence*, 144:1–39, 2003.

[99] S. Schiffel and M Thielscher. Interpreting Golog Programs in Flux. In *Proceedings of the 7th International Symposium on Logical Formalizations of Commonsense Reasoning*, 2005.

[100] S. Schiffel and M. Thielscher. Reconciling Situation Calculus and Fluent Calculus. In *Proceedings of the 21st National Conference on Artificial Intelligence and the 18th Innovative Applications of Artificial Intelligence Conference (AAAI'06/IAAI'06)*, 2006.

[101] Christian Schulte. *Programming Constraint Services.* Doctoral dissertation, Universität des Saarlandes, Naturwissenschaftlich-Technische Fakultät I, Fachrichtung Informatik, Saarbrücken, Germany, 2000.

[102] Christian Schulte. Parallel Search Made Simple. Technical Report TRA9/00, School of Computing, National University of Singapore, 55 Science Drive 2, Singapore 117599, September 2000.

[103] Murray Shanahan. Event calculus planning revisited. In *Proceedings of the 4th European Conference on Planning (ECP'97)*, number 1348 in Lecture Notes in Artificial Intelligence, pages 390–402. Springer, 1997.

[104] Murray Shanahan and Mark Witkowski. High-Level Robot Control Through Logic. In *Proceedings of the 7th International Workshop on Agent Theories, Architectures, and Languages (ATAL2000)*. Springer, 2000.

[105] S. Shapiro and M. Pagnucco. Iterated Belief Change and Exogenous Actions in the Situation Calculus. In *Proceedings of the 16th European Conference on Artificial Intelligence (ECAI'04)*, pages 878–882, 2004.

[106] S. Shapiro, Y. Lespérance, and H. J. Levesque. Specifying Communicative Multi-Agent Systems. In *Agents and Multi-Agent Systems - Formalisms, Methodologies, and Applications*, volume 1441 of *Lecture Notes in Artificial Intelligence*, pages 1–14, 1998.

[107] S. Shapiro, Y. Lesperance, and H. Levesque. Goal Change in the Situation Calculus. *Journal of Logic and Computation*, 17:983–1018, 2007.

[108] Steven Shapiro and Yves Lespérance. Modeling Multiagent Systems with the Cognitive Agents Specification Language — A Feature Interaction Resolution Application. In *Intelligent Agents Volume VII — Proceedings of the 2000 Workshop on Agent Theories, Architectures, and Languages*, pages 244–259. Springer-Verlag, 2001.

[109] Steven Shapiro, Maurice Pagnucco, and Hector J. Levesque. Iterated Belief Change in the Situation Calculus. In *Proceedings of the 7th International Conference on Principles of Knowledge Representation and Reasoning (KR'00)*, pages 527–538, 2000.

[110] Steven Shapiro, Yves Lespérance, and Hector J. Levesque. The Cognitive Agents Specification Language and Verification Environment for Multiagent Systems. In *Proceedings of the 1st International Joint Conference on Autonomous Agents and Multi-Agent Systems (AAMAS'02)*, pages 19–26, 2002.

[111] Tran Cao Son, Chitta Baral, and Le-Chi Tuan. Adding Time and Intervals to Procedural and Hierarchical Control Specifications. In *Proceedings of the 19th National Conference on Artificial Intelligence (AAAI'04)*, pages 92–97, 2004.

[112] M. Tambe. Towards Flexible Teamwork. *Journal of Artificial Intelligence Research*, 7:83–124, 1997.

[113] M. Thielscher. A Unifying Action Calculus. *Artificial Intelligence*, :, 2007. (in submission).

[114] Michael Thielscher. Logic-Based Agents and the Frame Problem: A Case for Progression. In V. Hendricks, editor, *First-Order Logic Revisited: Proceedings of the Conference 75 Years of First Order Logic (FOL75)*, pages 323–336, Berlin, Germany, 2004. Logos.

[115] Michael Thielscher. Introduction to the Fluent Calculus. *Electronic Transations on Artificial Intelligence*, 2:179–192, 1998.

[116] Michael Thielscher. From Situation Calculus to Fluent Calculus: State update axioms as a solution to the inferential frame problem. *Artificial Intelligence*, 111:277–299, 1999.

[117] Johan van Bentham. Modal Logic meets Situation Calculus. Technical Report PP-2007-04, University of Amsterdam, 2007.

[118] Johan van Benthem and Eric Pacuit. The Tree of Knowledge in Action: Towards a Common Perspective. In *Advances in Modal Logic*, volume 6, pages 87–106, 2006.

[119] Johan van Benthem, Jan van Eijck, and Barteld Kooi. Logics of Communication and Change. *Information and Computation*, 204(II):1620–1662, 2006.

[120] Peter Van Roy and Seif Haridi. *Concepts, Techniques, and Models of Computer Programming.* MIT Press, March 2004.

[121] Peter Van Roy and Seif Haridi. Mozart: A Programming System for Agent Applications. In *Proceedings of the International Workshop on Distributed and Internet Programming with Logic and Constraint Languages*, November 1999. (ICLP 99).

[122] Peter Van Roy, Per Brand, Denys Duchier, Seif Haridi, Martin Henz, and Christian Schulte. Logic programming in the context of multiparadigm programming: the Oz experience. *Theory and Practice of Logic Programming*, 3 (6):717–763, 2003.

[123] Stavros Vassos and Hector Levesque. Progression of Situation Calculus Action Theories with Incomplete Information. In *Proceedings of the 20th International Joint Conference on Artificial Intelligence*, pages 2024–2029, 2007.

[124] Stavros Vassos and Hector Levesque. On the Progression of Situation Calculus Basic Action Theories: Resolving a 10-year-old Conjecture. In *Proceedings of the 23rd AAAI Conference on Artificial Intelligence (AAAI'08)*, pages 1004–1009, 2008.

[125] Stavros Vassos, Gerhard Lakemeyer, and Hector J. Levesque. First-Order Strong Progression for Local-Effect Basic Action Theories. In *Proceedings of the 11th International Conference on Principles of Knowledge Representation and Reasoning (KR'08)*, pages 662–671, 2008.