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:*

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

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:*

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

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.:

$$acquire(agt, obj) \neq release(agt, obj)$$
$$acquire(agt_1, obj_1) = acquire(agt_2, obj_2) \rightarrow agt_1 = agt_2 \wedge obj_1 = obj_2$$

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:

$$HasObject(agt, obj, do(a, s)) \equiv a = acquire(agt, obj) \\ \vee \; HasObject(agt, obj, s) \wedge a \neq release(agt, obj)$$

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:

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

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:

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

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

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

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:

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

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:*

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

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:*

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

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:*

$$\mathcal{R}_\mathcal{D}(\varphi_1 \wedge \varphi_2) \stackrel{def}{=} \mathcal{R}_\mathcal{D}(\varphi_1) \wedge \mathcal{R}_\mathcal{D}(\varphi_2)$$

$$\mathcal{R}_\mathcal{D}(\exists x : \varphi) \stackrel{def}{=} \exists x : \mathcal{R}_\mathcal{D}(\varphi)$$

$$\mathcal{R}_\mathcal{D}(\neg\varphi) \stackrel{def}{=} \neg\mathcal{R}_\mathcal{D}(\varphi)$$

$$\mathcal{R}_\mathcal{D}(\alpha(\bar{\nu}, a, \sigma)) \stackrel{def}{=} \mathcal{R}_\mathcal{D}(\Pi_\alpha(\bar{\nu}, a, \sigma))$$

$$\mathcal{R}_\mathcal{D}(F(\bar{\nu}, do(a, \sigma))) \stackrel{def}{=} \Phi_F(\bar{\nu}, a, \sigma)$$

$$\mathcal{R}_\mathcal{D}(F(\bar{\nu}, s)) \stackrel{def}{=} F(\bar{\nu}, s)$$

$$\mathcal{R}_\mathcal{D}(F(\bar{\nu}, S_0)) \stackrel{def}{=} F(\bar{\nu}, S_0)$$

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

Chapter 6

Property Persistence

This chapter develops a new inductive reasoning technique for the situation calculus that can handle certain types of universally-quantified query. As discussed in Chapter 4, for an agent in an asynchronous domain to reason about the world based on its local information, it needs to pose queries that universally quantify over situation terms. Unfortunately such queries cannot be handled using the regression operator, and have thus far been beyond the reach of automated reasoning systems for the situation calculus.

We study a restricted subset of universally-quantified queries that we refer to as *property persistence queries*, introducing an approach to reasoning about them that is similar in spirit to the standard regression operator: transform the query into a form more amenable to automated reasoning. A new meta-operator $\mathcal{P}_{\mathcal{D}}$ is defined such that ϕ persists in s if and only if $\mathcal{P}_{\mathcal{D}}(\phi)$ holds in s. We term the formula generated by this operator the *persistence condition* of ϕ.

The persistence condition is shown to be a fixpoint of applications of the regression operator, which can be calculated using an iterative approximation algorithm. The resulting formula can then be used in combination with standard regression-based reasoning techniques, allowing the inductive component of the reasoning to be "factored out" and approached using a special-purpose reasoning algorithm. The technique is always sound, and is complete in several interesting cases.

Chapter 4 identified a universally-quantified query with which an agent can reason about its own world based on its local view. This query is *not* in a form that that can be handled directly using the persistence condition. However, Chapter 7 will demonstrate how to combine the techniques developed in this chapter with a new formalism for epistemic reasoning, allowing an agent to reason effectively about its own knowledge using a combination of regression and property persistence.

The chapter proceeds as follows: after some more detailed background material on inductive reasoning in the situation calculus in Section 6.1, we formally define the class of property persistence queries in Section 6.2, along with several examples of practical queries that are of this form. Section 6.3 defines the persistence condition operator and demonstrates that it is equivalent to the result of a meta-level fixpoint calculation. Section 6.4 presents a simple iterative algorithm for calculating the persistence condition, and discusses its correctness, completeness, and effectiveness. We conclude with some general discussion in Section 6.5.

6.1 Background

While there is a rich and diverse literature base for the situation calculus, there appears to have been little work on reasoning about universally quantified queries. The work of Reiter [89] shows how to handle such queries manually using an appropriate instantiation of the second-order induction axiom, but makes no mention of automating this reasoning.

Other work considering queries that universally quantify over situations focuses exclusively on verifying state constraints. These are uniform formulae that must hold in every possible situation, a highly specialised form of the more general persistence queries we define in this chapter. The work of Lin and Reiter [66] shows that the induction axiom can be "compiled away" when verifying a state constraint, by means of the following equivalence:

$$\mathcal{D} \models \phi[S_0] \rightarrow (\forall s : S_0 \leq s \rightarrow \phi[s])$$

$$\textit{iff}$$

$$\mathcal{D}_{una} \models \forall s, a : \phi[s] \wedge \mathcal{R}_{\mathcal{D}}(Poss(a, s)) \rightarrow \mathcal{R}_{\mathcal{D}}(\phi[do(a, s)])$$

The set $\mathcal{D}_{una}$ here performs the same role as our background axioms $\mathcal{D}_{bg}$ but contains only the unique names axioms for actions. Verification of a state constraint can thus be reduced to reasoning about a universally quantified uniform formula using only the static background theory, a comparatively straightforward reasoning task which we call *static domain reasoning*. Verification of state constraints was also approached by Bertossi et al. [11], who develop an automatic constraint verification system using an induction theorem prover.

However, there are many issues that are not addressed by work specific to state constraints. What if we are interested in the future of some arbitrary situation σ, rather than only S_0? What if want to restrict future actions according to an arbitrary action description predicate? Can we integrate a method for handling

universally-quantified queries with existing regression techniques? Our treatment of property persistence can provide a concrete basis for these considerations, and is hence significantly more general than this existing work.

Another field that deals with induction over situations is the verification of ConGolog programs. De Giacomo et al. [18] show how to formulate various safety, liveness, and starvation properties of a ConGolog program as fixpoint queries in second-order logic. A preliminary model-checker capable of verifying these properties is described in [42]. Claßen and Lakemeyer [14] develop a logic of ConGolog programs in $\mathcal{ES}$, a variant of the situation calculus based on modal logic. They demonstrate that properties of a program can be verified using an iterative fixpoint computation similar to the one we propose in this chapter.

As we shall see, property persistence queries are equivalent to a particular kind of safety property of a ConGolog program, so our work is in some ways less general than that described above. This means, however, that we can be more specific in our algorithm and approach. These ConGolog verifiers are designed to operate in isolation, while we seek a method of handling universally-quantified queries that can integrate directly with the existing meta-theoretical reasoning machinery of the situation calculus, in particular with the regression operator.

Finally, let us introduce an important property of situations first formally identified by Savelli [96]: that universal quantification over situation terms is equivalent to an infinite conjunction over the *levels* of the tree of situations:

$$\mathcal{D} \models \forall s : \psi(s)$$

$$\text{iff}$$

$$\mathcal{D} \models \bigwedge_{n \in \mathbb{N}} \forall a_1, \ldots, a_n : \psi(do([a_1, \ldots, a_n], S_0))$$

This is a direct consequence of the induction axiom for situations, which restricts situations to be constructed by performing some countable number of actions in the initial situation. While we do not use this result directly in this chapter, it captures an important intuition about situation terms that is fundamental to the operation of our approach.

6.2 Property Persistence Queries

Let us now formally define the kinds of query that will be approached in this chapter. Given some property ϕ and situation σ, a *property persistence query* asks whether

ϕ will hold in all situations in the future of σ:

$$\mathcal{D} \models \forall s : \sigma \sqsubseteq s \rightarrow \phi[s]$$

More generally, one may wish to limit the futures under consideration to those brought about by actions satisfying a certain action description predicate α, which is easily accomplished using the $\leq_\alpha$ macro. We thus have the following definition of a persistence query:

Definition 18 (Property Persistence Query)**.** *Let ϕ be a uniform formula, α an action description predicate, and σ a situation term. Then a property persistence query is a query of the form:*

$$\mathcal{D} \models \forall s : \sigma \leq_\alpha s \rightarrow \phi[s]$$

In words, a persistence query states that "ϕ holds in σ, and assuming all subsequent actions satisfy α, ϕ will continue to hold". For succinctness we will henceforth describe this as "ϕ persists under α". Queries of this form are involved in many useful reasoning tasks, of which the following are a small selection:

Goal Impossibility: Given a goal G, establish that there is no legal situation in which that goal is achieved:

$$\mathcal{D} \models \forall s : S_0 \leq_{Legal} s \rightarrow \neg G(s)$$

Goal Futility: Given a goal G and situation σ, establish that the goal cannot be achieved in any legal future of σ:

$$\mathcal{D} \models \forall s : \sigma \leq_{Legal} s \rightarrow \neg G(s)$$

Note how this differs from goal impossibility: while the agent may have initially been able to achieve its goal, the actions that have subsequently been performed have rendered the goal unachievable. Agents would be well advised to avoid such situations.

Checking State Constraints: Given a state constraint SC, show that the constraint holds in every legal situation:

$$\mathcal{D} \models \forall s : S_0 \leq_{Legal} s \rightarrow SC(s)$$

This can be seen as a variant of goal impossibility, by showing that the constraint can never be violated.

Need for Cooperation: Given an agent agt, goal G and situation σ, establish that no sequence of actions performed by that agent can achieve the goal. Suppose we define $MyAction$ to identify the agent's own actions:

$$MyAction(a, s) \stackrel{\text{def}}{=} actor(a) = agt$$

Then the appropriate query is:

$$\mathcal{D} \models \forall s : \sigma \leq_{MyAction} s \rightarrow \neg G(s)$$

If this is the case, the agent will need to seek cooperation from another agent in order to achieve its goal.

Knowledge with Hidden Actions: An agent reasoning about its own knowledge in asynchronous domains must account for arbitrarily-long sequences of hidden actions. To establish that it knows ϕ, it must establish that ϕ cannot become false through a sequence of hidden actions:

$$\mathcal{D} \models \forall s : \sigma \leq_{Hidden} s \rightarrow \phi[s]$$

This last case is our main motivation for the developments in this chapter, and we will explore the use of property persistence in this context in detail in Chapter 7. The other examples are designed to show that persistence queries are quite a general form of query, and the techniques developed in this chapter thus have application beyond our specific use of them in the remainder of this thesis.

Unfortunately, persistence queries do not meet the criteria for regressable formulae found in Definition 5, since they quantify over situation terms. Such queries therefore cannot be handled using the standard regression operator. Indeed, since universal quantification over situation terms requires the use of a second order induction axiom, current systems needing to answer such queries must resort to second-order theorem proving. This is hardly an attractive prospect for effective automated reasoning.

6.3 The Persistence Condition

To implement practical systems that can perform persistence queries, we clearly need to transform the query into a form suitable for effective automated reasoning. Our approach is to transform a property persistence query at σ into the evaluation of a uniform formula at σ. This transformed query can then be handled effectively using the standard regression operator.

To achieve this we need some transformation of a property ϕ and action description predicate α into a uniform formula $\mathcal{P}_{\mathcal{D}}(\phi, \alpha)$ that is true at precisely the situations in which ϕ persists under α. We call such a formula the *persistence condition* of ϕ under α.

Definition 19 (Persistence Condition)**.** *The persistence condition of ϕ under α, denoted $\mathcal{P}_{\mathcal{D}}(\phi, \alpha)$, is a uniform formula that is equivalent to the persistence of ϕ under α with respect to a basic action theory $\mathcal{D}$ without the initial situation axioms. Formally:*

$$\mathcal{D} - \mathcal{D}_{S_0} \models \forall s : \left(\mathcal{P}_{\mathcal{D}}(\phi, \alpha)[s] \equiv \forall s' : s \leq_\alpha s' \rightarrow \phi[s']\right)$$

Defining $\mathcal{P}_{\mathcal{D}}$ to be independent of the initial world state allows an agent to calculate it regardless of what (if anything) is known about the actual state of the world – after all, an agent may not know all the details of $\mathcal{D}_{S_0}$, and we still want it to be able to use this technique.

This definition alone clearly does not make the task of answering a persistence query any easier, since it gives no indication of how the persistence condition might be calculated in practice. Indeed, we have not yet even shown whether such a formula actually exists. In order to establish these results, we first need to define the weaker notion of a formula *persisting to depth n* in a situation.

Since we wish to establish our technique as a general reasoning mechanism for the situation calculus, we drop the assumption that concurrent actions are in use for the duration of this chapter. Note that nothing in our definitions precludes the use of various situation calculus extensions as described in Section 2.1.4.

Definition 20 (Persistence to Depth 1)**.** *A uniform formula ϕ persists to depth 1 under α in situation s when the formula $\mathcal{P}^1_{\mathcal{D}}(\phi, \alpha)[s]$ holds, as defined by:*

$$\mathcal{P}^1_{\mathcal{D}}(\phi, \alpha) \stackrel{def}{=} \phi^{-1} \wedge \forall a : \mathcal{R}_{\mathcal{D}}(\alpha[a, s])^{-1} \rightarrow \mathcal{R}_{\mathcal{D}}(\phi[do(a, s)])^{-1}$$

Note that $\mathcal{P}^1_{\mathcal{D}}$ is a literal encoding of the requirement "ϕ holds in s and in all its direct successors", using the standard regression operator $\mathcal{R}_{\mathcal{D}}$ and the situation-suppression operator ϕ^{-1} to produce a situation-suppressed uniform formula. With-

out the use of regression, the definition would appear as follows:

$$\mathcal{P}^1_{\mathcal{D}}(\phi, \alpha)[s] \equiv \phi[s] \wedge \forall a : \alpha[a, s] \rightarrow \phi[do(a, s)]$$

Since α is an action description predicate and ϕ is a uniform formula, the expressions $\mathcal{R}_{\mathcal{D}}(\alpha[a, s])^{-1}$ and $\mathcal{R}(\phi[do(a, s)])^{-1}$ are always defined and always produce uniform formulae. Successive applications of $\mathcal{P}^1_{\mathcal{D}}$ can then assert persistence to greater depths:

Definition 21 (Persistence to Depth N)**.** *For any $n \geq 0$, a uniform formula ϕ persists to depth n under α in situation s when the formula $\mathcal{P}^n_{\mathcal{D}}(\phi, \alpha)[s]$ holds, as defined by:*

$$\mathcal{P}^0_{\mathcal{D}}(\phi, \alpha) \stackrel{def}{=} \phi$$
$$\mathcal{P}^n_{\mathcal{D}}(\phi, \alpha) \stackrel{def}{=} \mathcal{P}^1_{\mathcal{D}}(\mathcal{P}^{n-1}_{D}(\phi, \alpha), \alpha)$$

The following theorem confirms that $\mathcal{P}^n_{\mathcal{D}}$ operates according to this intuition – that for any sequence of actions of length $i = 0$ to $i = n$, if each action satisfies α in the situation it is executed in, then ϕ will hold after executing those actions.

Theorem 5. *For any $n \in \mathbb{N}$, $\mathcal{P}^n_{\mathcal{D}}(\phi, \alpha)$ holds in σ iff ϕ holds in σ and in all successors of σ reached by performing at most n actions satisfying α:*

$$\mathcal{D} \models \mathcal{P}^n_{\mathcal{D}}(\phi, \alpha)[\sigma] \equiv$$
$$\bigwedge_{i \leq n} \forall a_1, \ldots, a_i : \left(\bigwedge_{j \leq i} \alpha[a_j, do([a_1, \ldots, a_{j-1}], \sigma)] \rightarrow \phi[do([a_1, \ldots, a_i], \sigma)] \right)$$

Proof Sketch. By induction on the natural numbers. For $n = 0$ we have $\phi[\sigma] \equiv \phi[\sigma]$ by definition. For the inductive case, we expand the definition of $\mathcal{P}^n_{\mathcal{D}}(\phi, \alpha)[\sigma]$ to get the following for the LHS:

$$\mathcal{P}^{n-1}_{\mathcal{D}}(\phi, \alpha)[\sigma] \wedge \forall a : \mathcal{R}_{\mathcal{D}}(\alpha[a, \sigma]) \rightarrow \mathcal{R}_{\mathcal{D}}(\mathcal{P}^{n-1}_{\mathcal{D}}(\phi, \alpha)[do(a, \sigma)])$$

Substituting for $\mathcal{P}^{n-1}_{\mathcal{D}}$ using the inductive hypothesis gives us a conjunction ranging over $i \leq n - 1$, with universally quantified variables $a_1, \ldots, a_i$, and we must establish the $i = n$ case. Pushing this conjunction inside the scope of the $\forall a$ quantifier, we find we can rename $a \Rightarrow a_1$, $a_1 \Rightarrow a_2$ etc to get the required expression. For a detailed proof see Appendix A. □

The $\mathcal{P}^n_{\mathcal{D}}$ operator thus allows us to express the persistence of a formula to any

given depth using a simple syntactic translation based on regression. Intuitively, one would expect $\mathcal{P}_{\mathcal{D}}(\phi, \alpha)$ to be some sort of fixpoint of $\mathcal{P}^1_{\mathcal{D}}(\phi, \alpha)$, since $\mathcal{P}_{\mathcal{D}}(\phi, \alpha)$ must imply persistence up to any depth. Such a fixpoint could then be calculated using standard iterative approximation techniques. The remainder of this section is devoted to verifying this intuition.

We begin by adapting two existing results involving induction from the situation calculus literature, so that they operate with our generalised $\leq_\alpha$ notation and can be based at situations other than S_0.

Proposition 4. *For any action description predicate α, the foundational axioms of the situation calculus entail the following induction principle:*

$$\forall W, s:\ W(s) \wedge \left[\forall a, s':\ \alpha[a, s'] \wedge s \leq_\alpha s' \wedge W(s') \rightarrow W(do(a, s'))\right] \\ \rightarrow \forall s':\ s \leq_\alpha s' \rightarrow W(s')$$

Proof. A trivial adaptation of Theorem 1 in [89]. □

Proposition 5. *For any basic action theory $\mathcal{D}$, uniform formula ϕ and action description predicate α:*

$$\mathcal{D} - \mathcal{D}_{S_0} \models \forall s:\ \phi[s] \rightarrow \left(\forall s':\ s \leq_\alpha s' \rightarrow \phi[s']\right)$$
$$\text{iff}$$
$$\mathcal{D}_{bg} \models \forall s, a:\ \phi[s] \wedge \mathcal{R}_{\mathcal{D}}(\alpha[a, s]) \rightarrow \mathcal{R}_{\mathcal{D}}(\phi[do(a, s)])$$

Proof. A straightforward generalisation of the model-construction proof of Lemma 5 in [66], utilising Proposition 4. □

Proposition 5 will be key in our algorithm for calculating the persistence condition. It allows one to establish the result "if ϕ holds in s, then ϕ persists in s" by using static domain reasoning, a comparatively straightforward reasoning task.

We next formalise some basic relationships between $\mathcal{P}_{\mathcal{D}}$ and $\mathcal{P}^n_{\mathcal{D}}$.

Lemma 2. *Given a basic action theory $\mathcal{D}$, uniform formula ϕ and action description predicate α, then for any n:*

$$\mathcal{D} - \mathcal{D}_{S_0} \models \forall s:\ \left(\forall s':\ s \leq_\alpha s' \rightarrow \phi[s']\right) \equiv \left(\forall s':\ s \leq_\alpha s' \rightarrow \mathcal{P}^n_D(\phi, \alpha)[s']\right)$$

That is, ϕ persists under α iff $\mathcal{P}^n_{\mathcal{D}}[\phi, \alpha]$ persists under α.

Proof. Since $\mathcal{P}^n_{\mathcal{D}}[\phi, \alpha]$ implies ϕ by definition, the *if* direction is trivial. For the *only-if* direction we proceed by induction on n.

For the base case, assume that ϕ persists but $\mathcal{P}^1_{\mathcal{D}}(\phi,\alpha)$ does not, then we must have some s' with $s \leq_\alpha s'$ and $\neg\mathcal{P}^1_{\mathcal{D}}(\phi,\alpha)[s']$. By the definition of $\mathcal{P}^1_{\mathcal{D}}$, this means that:

$$\neg\left(\phi[s'] \wedge \forall a : \alpha[a,s'] \rightarrow \phi[do(a,s')]\right)$$

Since ϕ persists it must hold at s', so there must be some a such that $\alpha[a,s']$ and $\neg\phi[do(a,s')]$. But $s \leq_\alpha do(a,s')$ and so $\phi[do(a,s')]$ must hold by our assumption that ϕ persists, and we have a contradiction.

For the inductive case, assume that $\mathcal{P}^{n-1}_{\mathcal{D}}(\phi,\alpha)$ persists but $\mathcal{P}^n_{\mathcal{D}}(\phi,\alpha)$ does not. By definition we have $\mathcal{P}^n_{\mathcal{D}}(\phi,\alpha) = \mathcal{P}^1_{\mathcal{D}}(\mathcal{P}^{n-1}_{\mathcal{D}}(\phi,\alpha),\phi)$, and we repeat the base case proof using $\phi' = \mathcal{P}^{n-1}_{\mathcal{D}}(\phi,\alpha)$ in place of ϕ to obtain a contradiction. □

Lemma 3. *Given a basic action theory $\mathcal{D}$, uniform formula ϕ and action description predicate α, then for any n:*

$$\mathcal{D} - \mathcal{D}_{S_0} \models \forall s : (\mathcal{P}_{\mathcal{D}}(\phi,\alpha)[s] \rightarrow \mathcal{P}^n_{\mathcal{D}}(\phi,\alpha)[s])$$

Proof. $\mathcal{P}_{\mathcal{D}}(\phi,\alpha)$ implies the persistence of ϕ by definition. If ϕ persists at s, then by Lemma 2 we have that $\mathcal{P}^n_{\mathcal{D}}(\phi,\alpha)$ persists at s . Since the persistence of $\mathcal{P}^n_{\mathcal{D}}(\phi,\alpha)$ at s implies that $\mathcal{P}^n_{\mathcal{D}}(\phi,\alpha)$ holds at s by definition, we have the lemma as desired. □

We are now equipped to prove the major theorem of this chapter: that if $\mathcal{P}^n_{\mathcal{D}}(\phi,\alpha)$ implies $\mathcal{P}^{n+1}_{\mathcal{D}}(\phi,\alpha)$, then $\mathcal{P}^n_{\mathcal{D}}(\phi,\alpha)$ is the persistence condition for ϕ under α.

Theorem 6. *Given a basic action theory $\mathcal{D}$, uniform formula ϕ and action description predicate α, then for any n:*

$$\mathcal{D}_{bg} \models \forall s : \mathcal{P}^n_{\mathcal{D}}(\phi,\alpha)[s] \rightarrow \mathcal{P}^{n+1}_{\mathcal{D}}(\phi,\alpha)[s] \tag{6.1}$$

iff

$$\mathcal{D} - \mathcal{D}_{s_0} \models \forall s : \mathcal{P}^n_{\mathcal{D}}(\phi,\alpha)[s] \equiv \mathcal{P}_{\mathcal{D}}(\phi,\alpha)[s] \tag{6.2}$$

Proof. For the *if* direction, we begin by expanding equation (6.1) using the definition of $\mathcal{P}^1_{\mathcal{D}}$ to get the equivalent form:

$$\mathcal{D}_{bg} \models \forall s : \mathcal{P}^n_{\mathcal{D}}(\phi,\alpha)[s] \rightarrow \mathcal{P}^1_{\mathcal{D}}(\mathcal{P}^n_{\mathcal{D}}(\phi,\alpha),\alpha)[s]$$
$$\mathcal{D}_{bg} \models \forall s : \mathcal{P}^n_{\mathcal{D}}(\phi,\alpha)[s] \rightarrow (\mathcal{P}^n_{\mathcal{D}}(\phi,\alpha)[s] \wedge \forall a : \mathcal{R}_{\mathcal{D}}(\alpha[a,s]) \rightarrow \mathcal{R}_{\mathcal{D}}(\phi[do(a,s)]))$$
$$\mathcal{D}_{bg} \models \forall s,a : \mathcal{P}^n_{\mathcal{D}}(\phi,\alpha)[s] \wedge \forall a : \mathcal{R}_{\mathcal{D}}(\alpha[a,s]) \rightarrow \mathcal{R}_{\mathcal{D}}(\phi[do(a,s)])$$

By Proposition 5, equation (6.1) thus lets us conclude that $\mathcal{P}^n_{\mathcal{D}}(\phi,\alpha)$ persists under α. By Lemma 2 this is equivalent to the persistence of ϕ under α, which is equivalent

to $\mathcal{P}_{\mathcal{D}}(\phi, \alpha)$ by definition, giving:

$$\mathcal{D} - \mathcal{D}_{s_0} \models \forall s : \mathcal{P}^n_{\mathcal{D}}(\phi, \alpha)[s] \rightarrow \mathcal{P}_{\mathcal{D}}(\phi, \alpha)[s]$$

By Lemma 3 this implication is an equivalence, yielding equation (6.2) as required.

The *only if* direction is a straightforward reversal of this reasoning: $\mathcal{P}_{\mathcal{D}}(\phi, \alpha)$ implies the persistence of ϕ, which implies the persistence of $\mathcal{P}^n_{\mathcal{D}}(\phi, \alpha)$, which yields equation (6.1) by Proposition 5. □

Since $\mathcal{D}_{bg} \models \mathcal{P}^{n+1}_{\mathcal{D}}(\phi, \alpha) \rightarrow \mathcal{P}^n_{\mathcal{D}}(\phi, \alpha)$ by definition, equation (6.1) identifies $\mathcal{P}^n_{\mathcal{D}}(\phi, \alpha)$ as a fixpoint of the $\mathcal{P}^1_{\mathcal{D}}$ operator, as our initial intuition suggested. In fact, we can use the constructive proof of Tarski's fixpoint theorem [15] to establish that the persistence condition always exists for a given ϕ and α.

Theorem 7. *Given a uniform formula ϕ and action description predicate α, the persistence condition $\mathcal{P}_{\mathcal{D}}(\phi, \alpha)$ always exists, and is unique up to equivalence under the static background theory $\mathcal{D}_{bg}$.*

Proof. Let L be the subset of the Lindenbaum algebra of the static background theory $\mathcal{D}_{bg}$ containing only sentences uniform in s. L is thus a boolean lattice in which each element is a set of sentences uniform in s that are equivalent under $\mathcal{D}_{bg}$. L is a complete lattice with minimal element the equivalence class of $\bot$ and maximal element the equivalence class of $\top$. Fixing α, $\mathcal{P}^1_{\mathcal{D}}$ is a function whose domain and range are the elements of L.

By definition, we have that $\mathcal{P}^1_{\mathcal{D}}(\phi, \alpha) \rightarrow \phi$, and $\mathcal{P}^1_{\mathcal{D}}$ is thus a *monotone decreasing* function over L. By the constructive proof of Tarski's fixpoint theorem, $\mathcal{P}^1_{\mathcal{D}}$ must have a unique greatest fixpoint less than the equivalence class of ϕ, which can be determined by transfinite iteration of the application of $\mathcal{P}^1_{\mathcal{D}}$. By Theorem 6, this fixpoint is the equivalence class of $\mathcal{P}_{\mathcal{D}}(\phi, \alpha)$ under $\mathcal{D}_{bg}$. □

This theorem legitimates the use of the persistence condition for reasoning about property persistence queries – for any persistence query at situation σ, there is a unique (up to equivalence) corresponding query that is uniform in σ and is thus amenable to standard effective reasoning techniques of the situation calculus.

Of course, it remains to actually calculate the persistence condition for a given ϕ and α. The definition of $\mathcal{P}_{\mathcal{D}}(\phi, \alpha)$ as a fixpoint suggests that it can be calculated by iterative approximation, which we discuss in the next section.

Algorithm 6 Calculate $\mathcal{P}_\mathcal{D}(\phi, \alpha)$

```
pn ⇐ φ
pn1 ⇐ P¹_D(pn, α)
while D_bg ⊭ ∀s : pn[s] → pn1[s] do
  pn ⇐ pn1
  pn1 ⇐ P¹_D[pn, α]
end while
return pn
```

6.4 Calculating $\mathcal{P}_\mathcal{D}$

Since we can easily calculate $\mathcal{P}^n_\mathcal{D}(\phi, \alpha)$ for any n, we have a straightforward algorithm for determining $\mathcal{P}_\mathcal{D}(\phi, \alpha)$: search for an n such that

$$\mathcal{D}_{bg} \models \forall s : \left(\mathcal{P}^n_\mathcal{D}(\phi, \alpha)[s] \rightarrow \mathcal{P}^{n+1}_\mathcal{D}(\phi, \alpha)[s]\right)$$

Since we expect $\mathcal{P}^n_\mathcal{D}(\phi, \alpha)$ to be simpler than $\mathcal{P}^{n+1}_\mathcal{D}(\phi, \alpha)$, we should look for the smallest such n. Algorithm 6 presents an iterative procedure for doing just that.

Note that the calculation of $\mathcal{P}^1_\mathcal{D}(\phi, \alpha)$ is a straightforward syntactic transformation, so we do not present an algorithm for it.

6.4.1 Correctness

If Algorithm 6 terminates, it terminates returning a value of pn for which equation (6.1) holds. By Theorem 6 this value of pn is equivalent to the persistence condition for ϕ under α. The algorithm therefore correctly calculates the persistence condition.

In particular, note that equation (6.1) holds when $\mathcal{P}^n_\mathcal{D}(\phi, \alpha)$ is unsatisfiable for any situation, as it appears in the antecedent of an implication. The algorithm thus correctly returns an unsatisfiable condition (equivalent to $\bot$) when ϕ can never persist under α.

6.4.2 Completeness

Since Theorem 6 is an equivalence, the persistence condition is always the fixpoint of $\mathcal{P}^1_\mathcal{D}$. From Theorem 7 this fixpoint always exists and can be calculated by transfinite iteration. Therefore, the only source of incompleteness in our algorithm will be failure to terminate. Algorithm 6 may fail to terminate for two reasons: the loop condition may never be satisfied, or the first-order logical inference in the loop condition may be undecidable and fail to terminate.

The later case indicates that the background theory $\mathcal{D}_{bg}$ is undecidable. While

this is a concern, it affects more than just our algorithm – any system implemented around such an action theory will be incomplete. With respect to this source of incompleteness, our algorithm is no more incomplete than any larger reasoning system it would form a part of.

The former case is of more direct consequence to our work. Unfortunately, there is no guarantee in general that the fixpoint can be reached via *finite* iteration, which is required for termination of Algorithm 6.

Indeed, it is straightforward to construct a fluent for which the algorithm never terminates: consider a fluent $F(x, s)$ that is affected by a single action that makes it false whenever $F(suc(x), s)$ is false. Letting α be vacuously true, the sequence of iterations produced by our algorithm would be:

$$\begin{gathered}
\mathcal{P}^1_{\mathcal{D}}(F(x,s)) \equiv F(x,s) \wedge F(suc(x),s) \\
\mathcal{P}^2_{\mathcal{D}}(F(x,s)) \equiv F(x,s) \wedge F(suc(x),s) \wedge F(suc(suc(x)),s) \\
\vdots \\
\mathcal{P}^n_{\mathcal{D}}(F(x,s)) \equiv \bigwedge_{i=0}^{i=n} F(suc^i(x),s)
\end{gathered}$$

The persistence condition in this case is clearly:

$$\mathcal{P}_{\mathcal{D}}(F(x,s)) \equiv \forall y : x \leq y \rightarrow F(y,s)$$

While this is equivalent to the infinite conjunction produced as the limit of iteration in our algorithm, it will not be found after any finite number of steps.

As discussed in the proof of Theorem 7, $\mathcal{P}^1_{\mathcal{D}}$ operates over the boolean lattice of equivalence classes of formulae uniform in s, and the theory of fixpoints requires that this lattice be *well-founded* to guarantee termination of an iterative approximation algorithm such as Algorithm 6. We must therefore identify restricted kinds of basic action theory for which this well-foundedness can be guaranteed.

The most obvious case is theories in which the action and object sorts are finite. In such theories the lattice of equivalence classes of formulae uniform in s is finite, and any finite lattice is well-founded. These theories also have the advantage that the static domain reasoning performed by Algorithm 6 can be done using propositional logic, meaning it is decidable and so providing a strong termination guarantee.

Alternately, suppose all successor state axioms and action description predicates have the following restricted form, where the terms in $\bar{y}$ are a subset of the terms in $\bar{x}$ and Φ_F, Π_{ADP} mention no terms other than $\bar{x}$, a and s:

$$F(\bar{x}, do(a, s)) \equiv \bigvee_{i=1}^{n} a = a_i(\bar{y}_i) \wedge \Phi_F(\bar{x}, a, s)$$
$$ADP(\bar{x}, a, s) \equiv \bigvee_{i=1}^{n} a = a_i(\bar{y}) \wedge \Pi_{ADP}(\bar{x}, a, s)$$

Under such theories, applications of $\mathcal{P}^1_\mathcal{D}$ will introduce no new terms into the query, apart from finitely many action terms a_i. The range of $\mathcal{P}^1_\mathcal{D}$ applied to ϕ is then a finite subset of the lattice of equivalence classes of formulae uniform in s, again guaranteed well-foundedness and terminating calculation of $\mathcal{P}_\mathcal{D}$.

Of course this is a very strong restriction on the structure of the theory, as the successor state axioms are not able to contain any quantifiers. It does demonstrate, however, that certain syntactical restrictions on $\mathcal{D}$ are able to guarantee terminating calculation of $\mathcal{P}_\mathcal{D}$. It seems there should be a more general "syntactic well-foundedness" restriction that can be applied to these axioms, but we have not successfully formulated one at this stage.

In a similar vein, suppose that the theory of action is *context free* [67]. In such theories successor state axioms have the following form:

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

The effects of an action are thus independent of the situation it is performed in. Lin and Levesque [64] demonstrate that such theories have a finite state space, again ensuring our algorithm operates over a finite lattice and hence guaranteeing termination. Context free domains are surprisingly expressive; for example, domains described in the style of STRIPS operators are context free.

From a slightly different perspective, suppose that ϕ can never persist under α, so that $\mathcal{P}_\mathcal{D}(\phi, \alpha) \equiv \bot$. Further suppose that $\mathcal{D}$ has the *compactness* property as in standard first-order logic. Then the "quantum levels" of Savelli [96] guarantee that there is a fixed, finite number of actions within which $\neg\phi$ can always be achieved. In this case Algorithm 6 will determine $\mathcal{P}_\mathcal{D}(\phi, \alpha) \equiv \bot$ within finitely many iterations.

It would also be interesting to determine whether known decidable variants of the situation calculus (such as [38]) are able to guarantee termination of the fixpoint construction, or whether more sophisticated fixpoint algorithms can be applied instead of simple iterative approximation. Investigating such algorithms would be a promising avenue for future research.

The important point here is not that we can guarantee completeness in general,

but that we have precisely characterised the inductive reasoning necessary to answer property persistence queries, and shown that it can always be replaced by the evaluation of a uniform formula at the situation in question.

6.4.3 Effectiveness

Our algorithm replaces a single reasoning task based on the full action theory $\mathcal{D}$ with a series of reasoning tasks based on the static background theory $\mathcal{D}_{bg}$. Is this a worthwhile trade-off in practice? The following points weigh strongly in favour of our approach.

First and foremost, we avoid the need for the second-order induction axiom. All the reasoning tasks can be performed using standard first-order reasoning, for which there are high-quality automated provers. Second, the calculation of $\mathcal{P}_{\mathcal{D}}$ performs only *static doing reasoning*, which as discussed in Chapter 2 is a comparatively straightforward task which can be made decidable under certain conditions. Third, $\mathcal{P}_{\mathcal{D}}(\phi, \alpha)[s]$ is in a form amenable to regression, a standard tool for effective reasoning in the situation calculus. Fourth, the persistence condition for a given ϕ and α can be cached and re-used for a series of related queries about different situations, a significant gain in amortised efficiency. Finally, in realistic domains we expect many properties to fail to persist beyond a few situations into the future, meaning that our algorithm will require few iterations in a large number of cases.

Of course, we also inherit the potential disadvantage of the regression operator: the length of $\mathcal{P}_{\mathcal{D}}(\phi, \alpha)$ may be exponential in the length of ϕ. As with regression, our experience has been that this is rarely a problem in practice, and is more than compensated for by the reduced complexity of the resulting reasoning task.

6.4.4 Applications

The persistence condition is readily applicable to the example persistence query problems given in Section 6.2. All of the transformed queries can then be answered using standard regression.

Goal Impossibility: Given a goal G, establish that there is no legal situation in which that goal is satisfied:

$$\mathcal{D} \models \mathcal{P}_{\mathcal{D}}(\neg G, Legal)[S_0]$$

The persistence condition of $\neg G$ with respect to action legality allows goal impossibility to be checked easily.

Goal Futility: Given a goal G and situation σ, establish that the goal cannot be satisfied in any legal future situation from σ:

$$\mathcal{D} \models \mathcal{P}_{\mathcal{D}}(\neg G, Legal)[\sigma]$$

Precisely the same formula is required for checking goal impossibility and goal futility. This highlights the advantage of re-using the persistence condition at multiple situations. Our approach makes it feasible for an agent to check for goal futility each time it considers performing an action, and avoid situations that would make its goals unachievable.

Checking State Constraints: Given a state constraint SC, show that the constraint holds in every legal situation:

$$\mathcal{D} \models \mathcal{P}_{\mathcal{D}}(SC, Legal)[S_0]$$

However, since we want a state constraint to *always* persist, it must satisfy the following equivalence:

$$\mathcal{D}_{bg} \models \phi \equiv \mathcal{P}_{\mathcal{D}}(\phi, Legal)$$

If this equivalence does not hold then $\mathcal{P}_{\mathcal{D}}(\phi, Legal)$ indicates the additional conditions that are necessary to ensure that ϕ persists, which may be used to adjust the action theory to enforce the constraint. This particular application has strong parallels to the approach to state constraints developed by Lin and Reiter [66].

Need for Cooperation: Given an agent agt, goal G and situation σ, establish that no sequence of actions performed by that agent can achieve the goal:

$$\mathcal{D} \models \mathcal{P}_{\mathcal{D}}(\neg G, MyAction)[\sigma]$$

Knowledge with Hidden Actions: In Chapter 7 we will develop a regression rule for knowledge that uses the persistence condition to account for arbitrarily-long sequences of hidden actions. While we defer the details to that chapter, the general form of the rule is:

$$\mathcal{R}_{\mathcal{D}}(\mathbf{Knows}(\phi, do(a, s))) \stackrel{\text{def}}{=} \mathbf{Knows}(\mathcal{R}_{\mathcal{D}}(\mathcal{P}_{\mathcal{D}}(\phi, Hidden), a), s)$$

6.5 Discussion

In this chapter we have developed an algorithm that transforms property persistence queries, a very general and useful class of situation calculus query, to a form that is amenable to standard techniques for effective reasoning in the situation calculus. The algorithm replaces a second-order induction axiom with a meta-level fixpoint calculation based on iterative application of the standard regression operator. It is shown to be correct, and also complete in some interesting cases.

Our approach generalises previous work on universally-quantified queries in several important ways. It can consider sequences of actions satisfying a range of conditions, not just the standard ordering over action possibility, enabling us to treat problems such as need for cooperation and knowledge with hidden actions. It can establish that properties persist in the future of an arbitrary situation, not necessarily the initial situation, enabling us to answer the question of goal futility. The results of calculating the persistence condition can be cached, allowing for example the goal futility question to be efficiently posed on a large number of situations once the persistence condition has been calculated.

Most importantly for the remainder of this thesis, we have *factored out* the inductive reasoning required to answer these queries. Work on increasing the effectiveness of this inductive reasoning, and on guaranteeing a terminating calculation in stronger classes of action theory, can now proceed independently from the development of formalisms that utilise persistence queries. We will henceforth use $\mathcal{P}_{\mathcal{D}}$ as a kind of "black box" operator to formulate regression rules within our framework, dropping the explicit $\mathcal{D}$ subscript as we do for the regression operator.

As noted in Section 6.1, our use of fixpoints in this chapter has much in common with the study of properties of ConGolog programs by [14, 18]. Indeed, a property persistence query is equivalent to a safety query stating that the property ϕ never becomes false during execution of the following program:

$$\delta_{P\alpha} \stackrel{\text{def}}{=} (\pi(a,\, \alpha[a]?\,;\, a))^*$$

Formally:

$$\mathcal{D} \models \forall s : \sigma \leq_\alpha s \rightarrow \phi[s]$$

iff

$$\mathcal{D} \cup \mathcal{D}_{golog} \models \forall s, \delta : Trans^*(\delta_{P\alpha}, \sigma, \delta, s) \rightarrow \phi[s]$$

Since we intend to use persistence queries as part of a larger reasoning apparatus, rather than as a stand-alone query, we cannot directly leverage the existing work on verifying ConGolog programs. However, given the similarity between the approaches, we are confident that advances in reasoning effectively about ConGolog programs will also advance our ability to effectively answer persistence queries.

This chapter has thus significantly increased the scope of queries that can be posed when building systems upon the situation calculus. In the coming chapters, the persistence condition operator will allow us to factor out certain inductive aspects of reasoning, treating them as separate, well-defined components of the overall reasoning machinery.

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, Yves Lespérance, 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.