COMMON KNOWLEDGE, HIDDEN ACTIONS, AND JOINT EXECUTION IN THE SITUATION CALCULUS

Ryan Francis Kelly

Submitted in total fulfillment of the requirements of the degree of Doctor of Philosophy

October 2008

Department of Computer Science and Software Engineering

The University of Melbourne

Abstract

This thesis develops several powerful extensions to the situation calculus, enabling it to reasoning about and planning for teams of agents in rich multi-agent domains. Driven by the desire to cooperative plan and perform the execution of shared program, we add support for: hidden actions, common knowledge, joint executions.

Declaration

This is to certify that:

(i) the thesis comprises only my original work towards the PhD except where indicated in the Preface,

(ii) due acknowledgment has been made in the text to all other material used,

(iii) the thesis is less than 100,000 words in length, exclusive of tables, maps, bibliographies, and appendices.

Ryan Francis Kelly

Preface

During the course of this project, a number of public presentations have been made which are based on the work presented in this thesis. They are listed here for reference.

- Ryan F. Kelly and Adrian R. Pearce. Towards high-level programming for distributed problem solving. In *Proc. IEEE/WIC/ACM International Conference on Intelligent Agent Technology*, pages 490–497, 2006
- Ryan F. Kelly and Adrian R. Pearce. Property persistence in the situation calculus. In *Proc. IJCAI'07*, pages 1948–1953, 2007
- Ryan F. Kelly and Adrian R. Pearce. Knowledge and observations in the situation calculus. In *Proc. AAMAS '07*, pages 841–843, 2007
- Ryan F. Kelly and Adrian R. Pearce. Complex epistemic modalities in the situation calculus. In *Proc. of the 11th International Conference on Principles of Knowledge Representation and Reasoning*, 2008

Acknowledgments

Lozz, Adrian, Mum & Dad, Office-mates, Sebastian et al, APA, pets etc...

Contents

List of Tables

List of Figures

Chapter 1

Introduction

This thesis develops techniques for the cooperative execution of a shared program by a team of agents in rich multi-agent domains. To make this more concrete, let us begin with a short motivating example which will be used throughout the thesis:

> You have just taken possession of a team of robotic chefs. It is your task to program the chefs to prepare a delicious meal.

- motivating example: cooking agents
- introduce HLP paradigm, briefly argue in its favour
- major limitation: focused on single-agent systems
- the "MIndiGolog Vision": cooperative execution of a HLP
- situation calculus needs extending to represent rich MA domains
- achievements in this thesis:
  - MIndiGolog: HLP semantics suitable for multi-agent domains
  - New reasoning technique for univsersally quantified queries
  - Robustly multi-agent account of knowledge, common knowledge
  - Semantics and techniques for cooperative planning of a legal execution
  - Implementations in Oz with distributed execution planning

Chapter 2

Background

- Quickly highlight difference between open multi-agent systems and distributed problem solving - explain what we're *not* interested in.
- Overview of DPS formalisms, task representation (HTN, TAEMS, etc...)
- Briefly introduce HLP approach, discuss advantages.
- Highlight success of Readylog team, but discuss limitations.
- Overview of coordination techniques, esp. social laws for coordination without communication
- Overview of distributed planning techniques [6]

2.1 The Situation Calculus

The situation calculus is a formalism for reasoning about action and change based on first-order logic. A solid understanding of formal logic is assumed throughout this thesis; see [8] for a comprehensive treatment. It was first introduced by McCarthy and Hayes [20] and has since been significantly expanded and formalized [28, 27, 15]. The notation and conventions used in this thesis are derived from [15].

The situation calculus is built on three fundamental notions:

- An ACTION is an instantaneous event that causes the state of the world to change
- A SITUATION is a history of all the actions that have occurred in the world, thus determining the state of the world
- A FLUENT is a particular aspect of the state of the world

Actions and situations are represented by function terms in the logic, while fluents are predicates or functions that are restricted to taking a situation term as their final argument. These notions will be formally defined in subsequent paragraphs.

As originally conceived, the situation calculus describes domains in which there is a single agent who has complete control over the world and can perform only a single action at a time. To provide a formalism expressive enough for the righ multi-agent domains targeted in this thesis, several important extensions to the situation calculus must be incorporated:

- Multiple agents acting in the world, by having the first argument of each action term identify the agent performing the action as in [35, 36].
- Multiple actions occurring at the same instant, by using sets of individual actions to build up situation terms as in [19, 26, 29]
- Actions with a finite duration, by breaking them into explicit start and end actions as in [26]
- An explicit notion of time, as in [26, 29]

The language of the situation calculus, $\mathcal{L}_{sitcalc}$, is a language of multi-sorted second-order logic with equality.

- Basic Sitcalc: origins with McCarthy [20], formalizations by Reiter et al [30, 27, 15]
- incorporate [24, 23] and the unfortunately unpublished [25]
- Our own customizations, e.g. "Action Description Predicates" [12]
- Concurrent Actions, Continuous Time, Natural Actions [26, 29]
- Reasoning (Regression, Decidable Fragments)
- Related approaches (Fluent Calculus, Event Calculus)
  - justify focusing on sitcalc by highlighting deep links between the formalisms
- Briefly highlight things we *don't* consider - nondeterministic/probabilistic actions, decision theoretic aspects, etc - and explain why.
- maybe also "stratified definitions" from [26]

2.2 High-Level Program Execution

- Basic Golog, operators, semantics [17]
- Highlight the vast body of similar approaches, e.g. Dynamic Logic
- So why golog? - full power of sitcalc, answer extraction for planning.
- Discuss idea of 'Legal Execution' in depth

- ConGolog [3]. Also some discussion of related semantics, such as CCS
- IndiGolog - highlight online/offline distinction and connections to coordination and planning [4]
- TConGolog - highlight deficiencies [1]
- Other Gologs (DTGolog, HTNGolog, GTGolog)
  - for completeness only, we have already established that we aren't operating in domains appropriate for them
  - But, take time to highlight rich cross-pollination between works, and that our results similarly have potential for integration with these approaches

2.3 Epistemic Reasoning

- Knowledge, Distributed Knowledge, Common Knowledge [9, 7]
- Importance of Common Knowledge for Coordination
- Knowledge in the Situation Calculus
  - Reasoning about Knowledge [21, 33]
  - Use in a multi-agent setting (??what's the best reference)
  - Concurrency and time [32]
  - Shortcomings of multi-agent extensions
  - Decidable fragments, weakening for computational efficiency
- Epistemic feasibility of plans: [31, 14]

2.4 The Oz Programming Language

- Basic introduction [38]
- Oz for Logic Programming [37]
- Distributed Logi Programming: Parallel Search [34]
- include some simple example programs

Chapter 3

Multi-Agent IndiGolog

3.1 MIndiGolog: Multi-Agent IndiGolog

- IndiGolog is unsuited to specifying team behavior
- We add: True Concurrency, Continuous Time, Natural Actions
- Explanation of why each is desirable
- Example programs from the cooking domain

3.2 Continuous Time

- Highlight changes to the semantics
- Explain use of rational constraint solver

3.3 True Concurrency

- Safety restrictions: actions must be possible, no skipping of actions
- Show changes to concurrency operator
- Discuss why other operators (e.g. prioritized concurrency) remain unchanged
- Ways to maximize concurrency

3.4 Natural Actions

- Show changes to atomic action operator
- Show changes to test operator

- Contrast with handling of exogenous actions in ConGolog
- Discuss ways to ensure that online execution consumes natural actions in a greedy manner

3.5 Implementation in Oz

- discuss basics of the software architecture, refer to appendix for details
- clauses encoding Trans, Final, TransStar

3.6 Executing the Program

- Show an execution from the example program
- Discuss coordination-without-communication in the style of Readylog
- Shortcoming: lots of duplication of effort

3.7 Distributed Execution Planning

- Planning is a logic program, so we can use existing techniques
- Details of parallel search setup in Oz
- How to coordinate the search operator with multiple agents: appoint a controller for the search

3.8 Limitations

- All actions known to all agents.
- No support for sensing actions (e.g. is cake cooked?)
- DKnows = Knows = EKnows = CKnows
- Basically signpost the rest of the thesis

Chapter 4

Property Persistence

4.1 Effective Reasoning

- Careful qualification of what we mean by "Effective"
- Review basics of reasoning in more detail, esp. regression.
- SitCalc as a re-writing system
- reasoning in the "fluent domain":

$$\mathcal{D}_{una} \models \forall s.\phi(\bar{x}, s) \quad iff \quad \mathcal{D}_{una} \models \phi(\bar{x})$$

- using typing of functions to guarantee a finite herbrand universe ([18], pp69) and therefore decidability

4.2 Property Persistence

- Formal definition
- Examples of why it's important
- Why it cant be done using standard regression

4.3 The Persistence Condition

- Definition of $\mathcal{P}$, $\mathcal{P}^1$ operators
- Proof that $\mathcal{P}$ is a least-fixed-point
- Justification that it's an "effective" technique
- Techniques for ensuring completness

4.4 Calculating $\mathcal{P}$

- Naive algorithm: definition, shortcomings
- Algorithm based on explicit effect axioms
- Handling advanced features: interacting effects, natural actions
- TODO: interacting preconditions and effects

Chapter 5

Knowledge

5.1 Hidden Actions

- Expand discussion from Background section
- Define notion of observations, observation history

5.2 The Knowledge Fluent

- Define how K should behave, in terms of observation histories
- Show S.S.A. for K, prove that it behaves as required
- Show various axiomatizations for $Obs()$, prove that they capture existing accounts of knowledge

5.3 Regressing Knowledge Queries

- Expand on proof present in conference paper
- Discuss intuitive appeal of the formulation

5.4 Example

- Lift example from journal paper, or use a new one based in the cooking domain?

5.5 Approximate Epistemic Reasoning

- Restrict formulae allowed inside **Knows** to atomic literals, in style of [5].
- Leverage encoding develope by [22]

Chapter 6

Group Knowledge

6.1 Common Knowledge

- Common Knowledge as a fixed-point, and why it's a bad idea
- Impossibility of a direct regression rule with only common knowledge

6.2 LCC

- Discuss main ideas of LCC
- Point of limitations with respect to sitcalc

6.3 Epistemic Path Language

- Lift material from journal paper

6.4 Syncrhonous Epistemic Fluent

- Lift material from journal paper

6.5 Introducing Hidden Actions

- Lift material from journal paper

6.6 Example

- Lift material from journal paper

6.7 Distributed Knowledge

- Sharing observation histories
- Approximations (e.g. ”someone knows”)

Chapter 7

Planning and Program Execution

7.1 What is Planning?

- In this setting, it is the process of *resolving nondeterminism*
- Planning should produce a program for the agent to follow, that does not itself require deliberation to perform. [16, 31]
- Must account for outcomes of sensing actions - some sort of branch/case statement
- Must be epistemically feasible (show formalisation from [31])

7.2 What is Team Planning?

- Produce a program for each agent
- Programs must include the necessary coordination actions/conditions
- We will need explicit syntax for synchronisation - "waiting for an observation"
- Any interleaving of the programs must be a valid legal execution
- similar idea in strips: [2]

7.3 Resolving Non-Determinism

- Hard Problem: removing only as much non-determinism as necessary
- Easier Problem: re-inserting non-determinism where possible
- Idea: plan by first *completely* resolving nondeterminism, but maintain auxiliary information that allows actions to be nondeterministically interleaved where a legal execution is maintained.

7.4 Joint Executions

- Based on *prime event structures*, a formalism for partially-ordered events
- Ensure that actions are ordered where necesary, but unordered where possible
- Contain branching based on action outcomes
- Can be built up an action at a time, much like standard situation terms
- Import definitions, theories etc from conference paper

7.5 Distributed Planning

- To obtain coordination without communication, must use common knowledge.
- Each agent can reason about CKnows(IsJointExec(δ,δ_N)) to come up with a common plan, which they can then execute
- Or, they can communicate to share the planning workload by using distributed knowledge DKnows(IsJointExec(δ,δ_N))

7.6 Coordination using Social Laws

- Use an ordering to specify which actions are prefered
- Define the notion of a "safe" action
- Perform a joint action when its safety is common knowledge among actors
- Communicating to determine safety of actions
- ?? Ways to increase efficiency of this approach ??

7.7 Managing the Search Operator

- beginplan() and endplan() actions observable by all
- planning in the face of change - "restart" actions incorporated into planning procedure

Chapter 8

Implementation

8.1 Problems with Naive Implementation

- Just representing formulae as terms quickly leads to problems:
- Exponential Growth of Formulae, compounded by regression

8.2 First-Order Shannon Graphs

- Structure Sharing
- Regression of a Shannon Graph
- Experimental data showing memory savings over naive approach
- Simplification, Theorem Proving
- Re-using results from previous proofs

8.3 The Planning Loop

8.4 Distributed Planning

8.5 TODO

- Using constraint solvers to reason about time ("Theory Reasoning")

Chapter 9

Conclusion

9.1 Achievements

- Refer back to introduction

9.2 Further Work

- Computational Efficiency - decidable fragments, belief instead of knowledge.

Appendix A

Code

full code listing goes in the appendix

Appendix B

Proofs

full proof details go in the appendix

References

[1] Jorge Baier and Javier Pinto. Integrating true concurrency into the robot programming language GOLOG. In *XIX Int. Conf. of the Chilean C.S. Society*, 1999.

[2] C. Boutilier and R. Brafman. Partial-order planning with concurrent interacting actions. *Journal of Artificial Intelligence Research*, 14:105–136, 2001.

[3] Giuseppe De Giacomo, Yves Lespérance, and Hector Levesque. ConGolog, a concurrent programming language based on the situation calculus. *AI*, 121(1-2):109–169, 2000.

[4] 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.

[5] Robert Demolombe and Maria del Pilar Pozos Parra. A simple and tractable extension of situation calculus to epistemic logic. In *ISMIS '00: Proceedings of the 12th International Symposium on Foundations of Intelligent Systems*, pages 515–524, London, UK, 2000. Springer-Verlag.

[6] M. desJardins, E. Durfee, C. Ortiz, and M. Wolverton. A survey of research in distributed, continual planning, 1999.

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

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

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

[10] Ryan F. Kelly and Adrian R. Pearce. Towards high-level programming for distributed problem solving. In *Proc. IEEE/WIC/ACM International Conference on Intelligent Agent Technology*, pages 490–497, 2006.

[11] Ryan F. Kelly and Adrian R. Pearce. Knowledge and observations in the situation calculus. In *Proc. AAMAS '07*, pages 841–843, 2007.

[12] Ryan F. Kelly and Adrian R. Pearce. Property persistence in the situation calculus. In *Proc. IJCAI'07*, pages 1948–1953, 2007.

[13] Ryan F. Kelly and Adrian R. Pearce. Complex epistemic modalities in the situation calculus. In *Proc. of the 11th International Conference on Principles of Knowledge Representation and Reasoning*, 2008.

[14] Yves Lespérance. On the epistemic feasibility of plans in multiagent systems specifications. In *Proc. 8th International Workshop on Agent Theories, Architectures, and Languages*, volume 2333 of *LNAI*, pages 69–85, Seattle, USA, August 2001.

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

[16] 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.

[17] 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.

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

[19] F. Lin and Y. Shoham. Concurrent actions in the situation calculus. In *Proc. of the National Conference on Artificial Intelligence*, 1992.

[20] 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.

[21] Robert C. Moore. Reasoning about knowledge and action. Technical Note 191, SRI International, October 1980.

[22] Ron Petrick and Hector Levesque. Knowledge equivalence in combined action theories. In *Proceedings of KR'02*, 2002.

[23] Javier Pinto. Concurrent actions and interacting effects. In *KR*, pages 292–303, 1998.

[24] Javier Pinto. Compiling ramification constraints into effect axioms. *Computational Intelligence*, 15:280–307, 1999.

[25] Javier Pinto. Concurrency and action interaction. http://www.scs.carleton.ca/ bertossi/javier/papers/pinto00.ps.gz, 2000.

[26] Javier A. Pinto. *Temporal Reasoning in the Situation Calculus*. PhD thesis, University of Toronto, Toronto, 1994.

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

[28] 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, San Diego, CA, USA, 1991. Academic Press Professional, Inc.

[29] Ray Reiter. Natural actions, concurrency and continuous time in the situation calculus. In *Proc. KR'96*, pages 2–13. Morgan Kaufmann, 1996.

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

[31] 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. Previous version appeared in Proc. of KR-2002.

[32] R. B. Scherl. Reasoning about the interaction of knowledge, time and concurrent actions in the situation calculus. In *Proc. IJCAI'03*, pages 1091–1098, 2003.

[33] Richard Scherl and Hector Levesque. Knowledge, action, and the frame problem. *Artificial Intelligence*, 144:1–39, 2003.

[34] 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. To appear.

[35] S. Shapiro, Y. Lespérance, and H. J. Levesque. Specifying communicative multi-agent systems. *Lecture Notes in Computer Science*, 1441:1–14, 1998.

[36] Steven Shapiro and Yves Lespérance. Modeling multiagent systems with the cognitive agents specification language — a feature interaction resolution application. In *Intelligent Agents Volume VII — Proc. of the 2000 Workshop on Agent Theories, Architectures, and Languages*, pages 244–259. Springer-Verlag, 2001.

[37] 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.

[38] Peter Van Roy and Seif Haridi. Mozart: A programming system for agent applications. In *Int. Workshop on Distributed and Internet Programming with Logic and Constraint Languages*, November 1999. (ICLP 99).