Project no.:	**027657**
Project full title:	**Perception, Action & Cognition through learning of Object-Action Complexes**
Project Acronym:	**PACO-PLUS**
Deliverable no.:	**D4.3.1**
Title of the deliverable:	**Mapping OACs to Grounded Symbolic LDEC Plan Representations**

Contractual Date of Delivery to the CEC:	31 January 2007		
Actual Date of Delivery to the CEC:	30 January 2007		
Organisation name of lead contractor for this deliverable:	UEDIN		
Author(s): Christopher Geib, Ronald Petrick, and Mark Steedman			
Participant(s): UEDIN, UL, BCCN, AAU			
Work package contributing to the deliverable:	WP4, WP5		
Nature:	R		
Version:	Final		
Total number of pages:	46		
Start date of project:	1st Feb. 2006	**Duration:**	48 month

Project co-funded by the European Commission within the Sixth Framework Programme (2002–2006)

Dissemination Level		
PU	Public	X
PP	Restricted to other programme participants (including the Commission Services)	
RE	Restricted to a group specified by the consortium (including the Commission Services)	
CO	Confidential, only for members of the consortium (including the Commission Services)	

Abstract:

The core focus of WP4 is the formalization of Object Action Complexes (OACs). As part of WP4.3 (Action Description Language) this deliverable reports on the translation of OACs into explicit symbolic representations, using a formal language like the Linear Dynamic Event Calculus (LDEC), and their applicability in constructing a knowledge base that supports high-level planning and plan recognition. This deliverable includes a number of attached papers that make very specific proposals for the formalization of OACs in LDEC, as well as methods for learning these representations.

Keyword list: Object Action Complexes (OACs), LDEC, planning, plan recognition, symbolic representation

Table of Contents

1. Executive Summary

The core focus of WP4 is the formalization of Object Action Complexes (OACs). More specifically, as part of WP4.3 (Action Description Language) this deliverable reports on the translation of OACs into explicit symbolic representations, using a formal language like the Linear Dynamic Event Calculus (LDEC), and their applicability in constructing a knowledge base that supports high-level planning and plan recognition. While it is still early in the project, and we are in the process of achieving consensus on some of the lower level details of Object Action Complexes, there are a number of important themes that we have identified concerning OACs. Moreover, these insights have resulted from discussions with other workpackage groups that we believe couldn't have come about from any one group pursuing these ideas in a vacuum.

Foundationally, we see OACs as knowledge structures that encapsulate

- both an object and the actions it affords, and
- the situational features that suggest the efficacy of each affordance.

As such, we believe that OACs not only provide a rich theoretical foundation but also provide a natural integration of ideas from multiple disciplines including: high-level AI planning using LDEC action representations (WP4, WP5); low-level real time vision (WP2), robotic action, and object representations (WP1, WP2); psychology and the Theory of Event Coding (TEC)[4]; and learning technologies for building these representations (WP6).

Critically, we believe that these structures cross the levels of the cognitive hierarchy of our agent, making use of representational information that includes: low-level vision systems, robotic control algorithms, high-level action and planning representations, all the way to representations usually reserved for plan recognition, language, and other kinds of abstract reasoning.

Our proposed formalization grounds OACs in the LDEC language. LDEC [8, 9] is a logical formalism that combines the insights of the Event Calculus [5] and the STRIPS planner [1], together with Dynamic and Linear Logics (e.g., as in [2], [3], and others). The use of LDEC as a high-level representation language follows in the tradition of logical languages inspired by the situation calculus [6]. LDEC has the added advantage, however, that it incorporates into its semantics a STRIPS-style treatment of fluent change, making it a suitable language for modelling planning domains, and the basic procedure underlying the planning process itself.

In this deliverable we have attached a number of additional documents that cover our current thinking on these topics in a more detailed manner. In particular, we have made very specific proposals for the formalization of OACs in LDEC as well as methods for learning these representations. Here we very briefly sketch their relation to this work package and its deliverables, and make links to the specific contributions of each paper.

[A] *(Presented at Humanoids-06 in Genoa, Italy)* This paper sketches the relationship between lower-level robot control and high-level robot planning. As such, it draws together a number of diverse ideas, provides a much more formal definition of OACs in terms of an LDEC representation, and makes proposals for how one could learn such a representation using cognitively plausible mechanisms.

[B] *(Technical Report)* This document is an evolving specification for the interfaces required for an actual artifact of the type described in the Genoa paper. This document was produced with the anticipation of implementing its suggestions for the Paco-Plus demos. As such it leverages the existing hardware platforms and existing software of multiple of the Paco-Plus teams to produce a working robot system based on OACs.

[C] *(Submitted to AAAI-07)* This paper presents a formal treatment of LDEC and its relationship to STRIPS-style planning, under conditions of complete knowledge and in the presence of incomplete information and sensing. Our current approach to high-level plan generation takes advantage of this close correspondence by building on the extended-STRIPS planner PKS [7], which is capable of constructing conditional plans.

[D] *(Presented at IJCAI-07 in Hyderabad, India)* This paper present a formal discussion of the relation of plan recognition and language and suggests ways in which they could be drawn closer together. We see this as foundational work aimed at bringing together a number of different representations that are used for different cognitive tasks and must be unified to produce a more complete cognitive architecture.

Together, these papers report a number of significant developments:

- OACs grounded from instantiated actions in robot control-space can be used as an interface between the very different representation languages used for robot control and AI planning.
- Information leveraged from an early exploration/robot vision system can be used to induce a high-level representation that abstracts the robot-level state space, leading to a division of tasks that can be performed between the two levels of representation.
- OACs can be embodied in an Associative Net and learned by very simple machine-learning algorithms.
- LDEC serves as a suitable language for modelling both complete and incomplete information, providing a representation of ordinary physical actions as well as sensing actions.
- Our OAC + LDEC model provides a complete base for planning (utilizing existing planning technology), and progress towards plan recognition as far as possible in advance of robot perceptual capabilities.

This work package has also heavily informed and influenced other work packages including WP2,WP3, WP5, and WP7. Formalization of the OAC concept and its associated machinery both for learning and reasoning has helped to establish interfaces to other components as well as suggest learning and control paradigms in our discussions with other work packages.

2. Attached Papers

[A] **Object Action Complexes as an Interface for Planning and Robot Control**
Christopher Geib, Kira Mourão, Ronald Petrick, Nico Pugeault, Mark Steedman,
Norbert Krueger, and Florentin Wörgötter
Published in the Proceedings of the Humanoids-06 Workshop: Toward Cognitive Humanoid Robots, 2006

Abstract: Much prior work in integrating high-level artificial intelligence planning technology with low-level robotic control has foundered on the significant representational differences between these two areas of research. We discuss a proposed solution to this representational discontinuity in the form of object-action complexes (OACs). The pairing of actions and objects in a single interface representation captures the needs of both reasoning levels, and will enable machine learning of high-level action representations from low-level control representations.

[B] **Paco-Plus Design Documentation for Integration of Robot Control and AI Planning**
Christopher Geib, Ronald Petrick, Kira Mourão, Nicolas Pugeault, Mark Steedman,
Pascal Haazebroek, Norbert Krueger, Dirk Kraft, and Florentin Wörgötter
Technical report

Abstract: This document was started as a way to capture the conclusions of the discussions between the authors, for specific proposals about how to interface the high-level "AI Planning layer" and the "Robot control layer". This document captures our current conclusions and the implications for the interface between these two modules.

[C] **Representing Knowledge and Sensing in the Linear Dynamic Event Calculus for Knowledge-Level Planning**
Ronald Petrick and Mark Steedman
Submitted to the AAAI Conference on Artificial Intelligence (AAAI-07)

Abstract: The Linear Dynamic Event Calculus (LDEC) is a logical language for reasoning about actions and change. One of the novel features of this formalism is its close relationship with STRIPS, and its STRIPS-style treatment of fluent change for addressing the frame problem. In this paper we extend the LDEC representation to incorporate knowledge and sensing actions using the idea of "knowledge fluents"—fluents that model the agent's knowledge of ordinary world-level fluents. Using this representation, we show how to transform LDEC actions into STRIPS-like planning operators usable by the knowledge-level contingent planner PKS.

[D] **On Natural Language Processing and Plan Recognition**
Christopher Geib and Mark Steedman
Published in the Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI-07), pages 1612–1617, 2007

Abstract: The research areas of plan recognition and natural language parsing share many common features and even algorithms. However, the dialog between these two disciplines has not been effective. Specifically, significant recent results in parsing mildly context sensitive grammars have not been leveraged in the state of the art plan recognition systems. This paper will outline the relations between natural language processing(NLP) and plan recognition(PR), argue that each of them can effectively inform the other, and then focus on key recent research results in NLP and argue for their applicability to PR.

References

[1] Richard E. Fikes and Nils J. Nilsson. STRIPS: A new approach to the application of theorem proving to problem solving. *Artificial Intelligence*, 2:189–208, 1971.

[2] J.-Y. Girard. Linear logic. *Theoretical Computer Science*, 50:1–102, 1987.

[3] D. Harel. Dynamic logic. In D. Gabbay and F. Guenthner, editors, *Handbook of Philosophical Logic, volume II*, pages 497–604. Reidel, Dordrecht, 1984.

[4] Bernhard Hommel, Jochen Müsseler, Gisa Aschersleben, and Wolfgang Prinz. The theory of event coding (TEC): A framework for perception and action planning. *Behavioral and Brain Sciences*, 24:849–937, 2001.

[5] R. Kowalski and M. Sergot. A logic-based calculus of events. *New Generation Computing*, 4:67–95, 1986.

[6] John McCarthy and Patrick J. Hayes. Some philosophical problems from the standpoint of artificial intelligence. *Machine Intelligence*, 4:463–502, 1969.

[7] Ronald P. A. Petrick and Fahiem Bacchus. A knowledge-based approach to planning with incomplete information and sensing. In *Proc. of AIPS-2002*, pages 212–221. AAAI Press, 2002.

[8] Mark Steedman. Temporality. In Johan van Benthem and Alice ter Meulen, editors, *Handbook of Logic and Language*, pages 895–938. North Holland/Elsevier, Amsterdam, 1997.

[9] Mark Steedman. Plans, affordances, and combinatory grammar. *Linguistics and Philosophy*, 25:723–753, 2002.

Appendix A

Object Action Complexes as an Interface for Planning and Robot Control

Christopher Geib, Kira Mourão, Ronald Petrick, Nico Pugeault, and Mark Steedman

School of Informatics
University of Edinburgh
Edinburgh EH8 9LW, Scotland

Norbert Krueger

The Maersk Mc-Kinney Moller Institute
University of Southern Denmark
DK-5230 Odense M, Denmark

Florentin Wörgötter

Institute for Informatics
University of Göttingen
37083 Göttingen, Germany

Abstract

Much prior work in integrating high-level artificial intelligence planning technology with low-level robotic control has foundered on the significant representational differences between these two areas of research. We discuss a proposed solution to this representational discontinuity in the form of object-action complexes (OACs). The pairing of actions and objects in a single interface representation captures the needs of both reasoning levels, and will enable machine learning of high-level action representations from low-level control representations.

1 Introduction and Background

The different representations that are effective for continuous control of robotic systems and the discrete symbolic AI presents a significant challenge for integrating AI planning research and robotics. These areas of research should be able to inform one another. However, in practice, many collaborations have foundered on the representational differences. In this paper, we propose the use of object-action complexes[7] to address the representational difference between these reasoning components.

The representations used in the robotics community can be generally characterized as vectors of continuous values. These vectors may be used to represent absolute points in three dimensional space, relative points in space, joint angles, force vectors, and even world-level properties that require real-valued models [12]. Such representations allow system builders to succinctly specify robot behavior since most if not all, of the computations for robotic control are effectively captured as continuous transforms of continuous vectors over time. AI representations, on the other hand, have focused on discrete symbolic representations of objects and actions, usually using propositional or first-order logics. Such representations typically focus on modeling the high-level conceptual state changes that result from action execution, rather than the low-level continuous details of action execution.

Neither of the representational systems alone cover the requirements for controlling deliberate action, however, both levels seem to be required to produce human level behavioral control. Our objective is to propose an interface representation that will both allow the effective exchange of information between these two levels and the learning of high level action representations on the basis of the information provided by the robotic control system.

Any such representation must provide clear semantics, and be easily manipulable at both levels. Further it must leverage the respective strengths of the two representation levels. In particular, the robotic control system's access to the actual physical state of the world through its sensors and effectors is essential to learning the actions the planning system must reason about. Each low-level action executed by the robot offers the opportunity to observe a small instantiated fragment of the state transition function that the AI action representations must capture. Therefore, we propose that the robotic control system provide fully instantiated fragments of the planning domains state transition function, that is captured during low-level execution, to the high-level AI system to enable the learning of abstract action representations. We will call such a fragment an *instantiated state transition fragment (ISTF)*, and define it to be a situated pairing of an object and an action that captures a small, but fully instantiated, fragment of the planning domain's state transition function. The process of learning domain invariants from repeated, reproducible instances of very similar ISTFs will result in generalizations over such instances that we will call *object-action complexes (OACs)*. To see how this is done, the rest of this paper will first discuss a detailed view of a robot control system, then we will discuss an AI planning level description of the same domain. We will then more formally define ISTFs and OACS, show how ISTFs can be produced by the robot control system, and how OACs relate to the AI planning level description. We will then discuss the learning of OACs on the basis of ISTFs.

To do all this, we require a particular domain for the robot to interact with. Imagine a relatively standard but simple robot control scenario illustrated in Figure 1. It consists of an arm with a gripper, a table with two light colored cubes and one dark colored cube. The robot has the task of placing the cubes into a box, also located on the table. We will also assume the robot is provided with a camera to view the objects in the domain. However, at the initial stage, the system does not have any knowledge of those objects. The only initial world knowledge available to the system is provided by the vision module, and the hard-coded action reflexes that this visual input can elicit.

2 Vision-based Reflex Driven Discovery of Objects and Affordances

We assume a vision front-end based on an *Early Cognitive Vision* framework (see [8]) that provides a scene representation composed of local 3D edge descriptors that outline the visible contours of the scene [15]. Because the system lacks knowledge of the objects that make up the scene, this visual world representation is *unsegmented*: descriptors that belongs to one of the objects in the scene are not explicitly distinct from the ones belonging to another object, or to the background (this is marked by question marks in Figure 1-2). This segmentation problem has been largely addressed in the literature [16, 11, 3]. However, while these segmentation methods are purely vision-based and do not require of the agent to interact with the scene they are unsatisfying for our purpose because they assume certain qualities from the objects in order to segment them: e.g., constant color or texture, moving objects, etc.

Instead we will approach the problem from another angle: we will assume that the agent is endowed with a basic reflex action [1] (Figure 1-3) that is elicited directly by specific visual feature combinations in the unsegmented world representation. The outcome of these reflexes will allow the agent to gather further knowledge about the scene. This information will be used to segment the visual world into objects and identify their affordances.

We will only consider a single kind of reflex here: the agent tries to grasp any planar surface in the scene.[1] The likely locations of such planar surfaces are inferred from the presence of a coplanar pair of edges in the unsegmented visual world. This type of reflex action is described in [1]. Every time the

[1]Note that other kind of reflex actions could be devised to enable other basic actions than grasping.

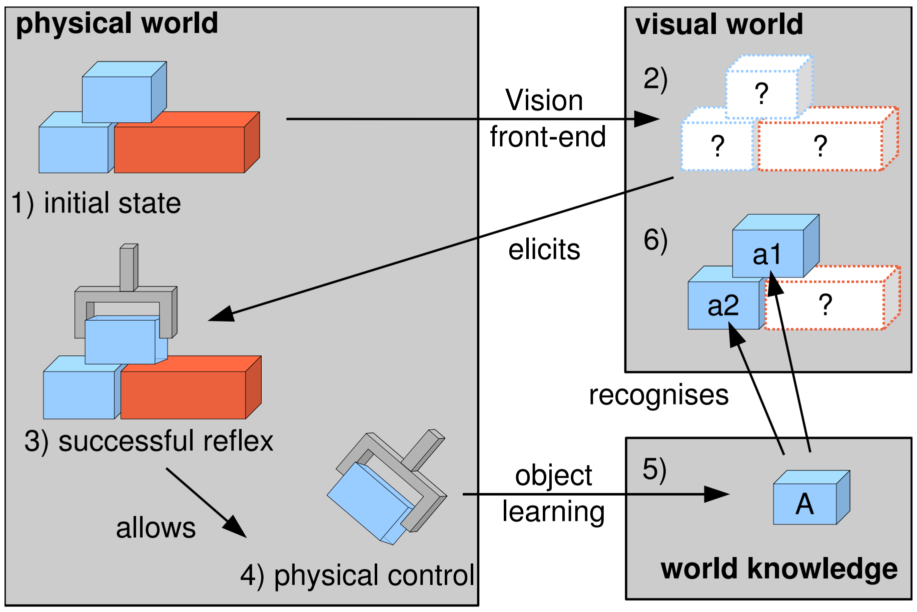

Figure 1: Illustration of how object classes are discovered from basic uninformed reflex actions.

agent executes such a reflex, haptic information allows the system to evaluate the outcome: either the grasp was successful and the gripper is holding something, or it failed and the gripper closed on thin air. A failed attempt drives the agent to reconsider its original assumption (the presence of a graspable plane at this location in the scene), whereas a successful attempt confirms the feasibility of this reflex. Moreover, once a successful grasp has been performed, the agent has gained physical control over some part of the scene (i.e. the object grasped, Figure 1-4). If we assume that we know the full kinematics of the robot's arm (which is true for an industrial robot), it is then possible to segment the grasped object from the rest of the visual world as it is the only part that moves synchronously with the arm of the robot. At this point a new "object" relevant for the higher level planning model is "born".

Having physical control of an object allows the agent to segment it and to visually inspect it under a variety of viewpoints and construct an internal representation of the full 3D shape of the object (see [9]). This shape can then be stored as the description of newly discovered class **A** (Figure 1-5) that affords **grasp-reflex-A** encoding the initial reflex that "discovered" the object.

The object held in the gripper is the first instance **a1** of the class **A**. The agent can use its new knowledge of class **A** to reconsider its interpretation of the scene: using a simple object recognition process (based on the full 3D representation of the class), all other instances (e.g., in our example **a2**) of the class in the scene are identified and segmented from the unknown visual world.

Thus through a reflex-based exploration of the unknown visual world object classes can be discovered by the system until it achieves an informed, fully segmented representation of the world, where all objects are instances of symbolic classes and carry basic affordances.

To distinguish the specific successful instances of the robot's reflexes, we will refer to the specific instance of the reflex that was successful for the object as a particular *motor program*. Note that such motor programs are defined relative to a portion of an object, in our example, the surface that was grasped. We will extend this by assuming *all* motor programs can be defined relative to some object.

The early cognitive vision system [15], the grasping reflex [1] as well as the accumulation mechanism [9] that together provides a segmentation of the local feature descriptors into independent objects currently exist in one integrated system that we will use as a foundation for this architecture.

3 Representing AI Planning Actions

As we have noted, we can also model this robot domain scenario using a formal AI representation. In this case, we will formalize the robot domain using the Linear Dynamic Event Calculus (LDEC) [18, 19], a logical language that combines aspects of the situation calculus with linear and dynamic logics, to model dynamically-changing worlds[10, 6, 5].

Our LDEC representation will define the following actions.

Definition 1 *High-Level Domain Actions*

- *grasp*(x) *– move the gripper to pick up object x,*
- *ungrasp*(x) *– release the object x in the gripper,*
- *moveEmptyGripperTo*(ℓ) *– move an empty gripper to the specified location ℓ,*
- *moveFullGripperTo*(ℓ) *– move a full gripper to the specified location ℓ.*

These actions represent higher level counterparts of some of the motor programs available to the robot controller, but already these actions incorporate elements of the state of the world that are not part of robotic control representations of actions. For instance, *ungrasp* models an action that is quite similar to a motor program that performs this operation. Actions like *moveEmptyGripperTo* and *moveFullGripperTo*, on the other hand, are much more abstract and encode information about the state of the world (i.e. the gripper is empty or full). Note that in this case the actions partition the low-level "move gripper" motor-programs into two separate actions that, as we will see, can more readily be learned from the available ISTFs. This representation will also allow us to bypass the learning of the conditional effects[13] of such actions.

Our LDEC representation will also include a number of high-level properties.

Definition 2 *High-Level Domain Properties*

- *graspable*(x) *– a predicate that indicates whether an object x is graspable or not,*
- *gripperLoc* $= \ell$ *– a function that indicates the current location of the gripper is ℓ,*
- *objInGripper* $= x$ *– a function that indicates the object in the gripper is x; x is* nil *if the gripper is empty,*
- *objLoc*$(x) = \ell$ *– a function that indicates the location of object x is ℓ.*

Finally, we also specify a set of "exogenous" domain properties.

Definition 3 *Exogenous Domain Properties*

- *over*$(x) = \ell$ *– a function that returns a location ℓ over the object x,*
- *locOnTable*$(\ell_1) = \ell_2$ *– a function that returns a location ℓ_2 relative to the table (e.g., on the table or in a box) for another location ℓ_1 above the table.*

Like the properties in Definition 2, the exogenous properties model high-level features of the domain. However, unlike domain properties that are directly tracked by the high-level AI model; exogenous properties are information provided to the high-level AI system by some external (possibly lower level) source. (We will say more about exogenous properties in Section 6.)

Using these actions and properties we can write LDEC axioms that capture the dynamics of the robot scenario described in Table 1). Action precondition axioms describe the properties that must hold of the world to apply a given action (i.e., affordances), while the effect axioms characterize what changes

Table 1: LDEC Axiomatization of High-Level Domain Actions

LDEC Action Precondition Axioms

$$\mathit{objInGripper} = \mathit{nil} \wedge \mathit{graspable}(x) \Rightarrow \mathit{affords}(\mathit{grasp}(x))$$
$$\mathit{objInGripper} = x \wedge x \neq \mathit{nil} \Rightarrow \mathit{affords}(\mathit{ungrasp}(x))$$
$$\mathit{objInGripper} = \mathit{nil} \Rightarrow \mathit{affords}(\mathit{moveEmptyGripperTo}(\ell))$$
$$\mathit{objInGripper} = x \wedge x \neq \mathit{nil} \Rightarrow \mathit{affords}(\mathit{moveFullGripperTo}(\ell))$$

LDEC Effect Axioms

$$\{\mathit{affords}(\mathit{grasp}(x))\} \multimap$$
$$[\mathit{grasp}(x)]\, \mathit{objInGripper} = x \wedge \mathit{gripperLoc} = \mathit{objLoc}(x)$$
$$\{\mathit{affords}(\mathit{ungrasp}(x))\} \multimap$$
$$[\mathit{ungrasp}(x)]\, \mathit{objInGripper} = \mathit{nil} \wedge \mathit{objLoc}(x) = \mathit{locOnTable}(\mathit{objLoc}(x))$$
$$\{\mathit{affords}(\mathit{moveEmptyGripperTo}(\ell))\} \multimap$$
$$[\mathit{moveEmptyGripperTo}(\ell)]\, \mathit{gripperLoc} = \ell$$
$$\{\mathit{affords}(\mathit{moveFullGripperTo}(\ell))\} \multimap$$
$$[\mathit{moveFullGripperTo}(\ell)]\, \mathit{gripperLoc} = \ell \wedge \mathit{objLoc}(\mathit{objInGripper}) = \ell$$

as a result of the action. These axioms also encode the STRIPS assumption: fluents that aren't directly affected by an action are assumed to remain unchanged by that action [4].

We note our LDEC axiomatization is readily able to accommodate the *indexical*, or relative information. For example, an instantiated function like $\mathit{over}(\mathit{box}1)$ represents a form of indexical knowledge, rather than a piece of definite information like the coordinates of the box in three dimensional space. Moreover, our LDEC axiomatization can model spatial relationships expressed with respect to objects. For instance, $\mathit{moveFullGripperTo}(\mathit{over}(\mathit{box1}))$ can represent an action instance that moves the object in the gripper to a location "over *box1*"

Intuitively, the information encoded in a collection of LDEC axioms captures a generalization of the information in a larger set of ISTFs. The action precondition axioms capture information from the initial state of an ISTF and the action executed, while the effect axioms capture the generalities for the initial state to final state mappings from an ISTFs. As such we believe they can be learned from the ISTFs.

It is easy to show that this representation supports high-level planning. For instance, with these axioms it is trivial for an AI planner to construct the following simple plan:

$$[\mathit{grasp}(\mathit{obj}1); \mathit{moveFullGripperTo}(\mathit{over}(\mathit{box}1)); \mathit{ungrasp}(\mathit{obj}1)]\,,$$

to put an object $\mathit{obj}1$ into $\mathit{box}1$, from a state in which the robot's gripper is empty. However, building even this sort of simple plan from first principles is well beyond the capability of the robot controller alone.

So far we have shown that a low level robot controller is capable of producing ISTFs for a domain, we have shown a way an AI level planner could formalize the same domain, and we have shown the necessity of using the AI planner with the robot controller to produce high level behavior. In the remainder of the paper we will outline a process whereby we can learn the AI level representation from the ISTFs produced by the robot controller.

4 Bridging Robot Control and Planning with ISTFs and OACs

With these two views of the problem in hand, we now, consider how we can bridge the two representational levels. We see that we can obtain a wealth of object-centric information each time the robotic system successfully grasps an object: the object grasped, the type of grasping reflex used, the relative position of the gripper, the fact that the object has been effectively grasped and is now in the gripper instead of being on the table, etc. This association of before and after states of a particular "grasp" motor program with a specific domain object meets our definition of an ISTF. It completely describes a fragment of the planning domain's transition function.

We more formally define an ISTF as a tuple $\left\langle s_i, mp_j, Obj_{mp_i}, s_{i+1}\right\rangle$ comprised of the initial sensed state of the world s_i, a motor program instance mp_j, the whole object containing the component the motor program was defined relative to Obj_{mp_i}, and the state that results from executing the motor program s_{i+1}. Keep in mind that the state representations for this ISTF contain all of the information the robot has about the two states of the world. Some of which may be relevant some of which may be completely irrelevant to the outcome of the action.

It will be the task of the learning module to abstract away this irrelevant information from the ISTFs to produce OACs that contain only the relevant instantiated information needed to effectively predict the applicability of the action and the likely effects of the action. This is only possible if the system is provided with multiple encounters with reproducible ISTFs. Thus as the system repeatedly interacts with the world it is presented with multiple very similar ISTFs which it generalizes into OACs, thereby learning a representation that is not unlike the one we specified in the previous section.

On this basis, we define an OAC as a generalized ISTF tuple: $\left\langle S_i, MP_j, Obj_k, S_{i+1}\right\rangle$ comprised of two abstracted states (S_i and S_{i+1}) a set of motor programs MP_j, and an object class Obj_k. The initial state of the world, S_i, is abstracted to contain only those properties that are necessary for any of the set of motor-programs in MP_j when acting on an object of class Obj_k to result in an state that is satisfied but the abstracted state definition S_{i+1}. Thus such an OAC contains all of the information found in our initial LDEC definitions for this domain.

Given the parallels to LDEC representations how are OACs different? The answer to this is, a very subtle point. OACs constrain the kinds of LDEC rules that can be learned. First OACs distribute information in a subtly different manner than LDEC rules. An OAC contains information normally found in two different parts of the LDEC representation. By bringing together information found in precondition rules with the effect rules and the object in question they allow learning to take place that previously couldn't have been accomplished. Second the heavy use of the object and the object centeredness of OACS produce LDEC representations that easily lend themselves to a simple forward looking planning algorithm that is heavily directed by the affordances of the available objects. Third and finally the use of OACs constrains the LDEC representations to a simple form of axioms that are easier to learn. For example, without more complex machinery, OACs induced from ISTFs are not able to create action representations with conditional effects. Learning such conditional effects of actions is a significant problem for other approaches.

5 Learning Action Representations

The ability of a low level robotic control system to identify world-level objects only takes us part of the way to kind of representation we have just described. We must learn from the ISTFs coherent, high-level actions. Our current proposal for learning such action representations involves the use of Willshaw nets or Associative Nets(AN).

ANs were first was described in [22, 21] following early work by [20] and [2] extended by [17] and [14]. They illustrate three basic properties which are characteristic of mechanisms involved in phenomena of human memory and attention: 1) non-localized storage ("Distributivity"), 2) recovery of complete stored patterns from partial or noisy input ("Graceful Degradation"), and 3) effective functioning even in the face of damage ("Holographic Memory").

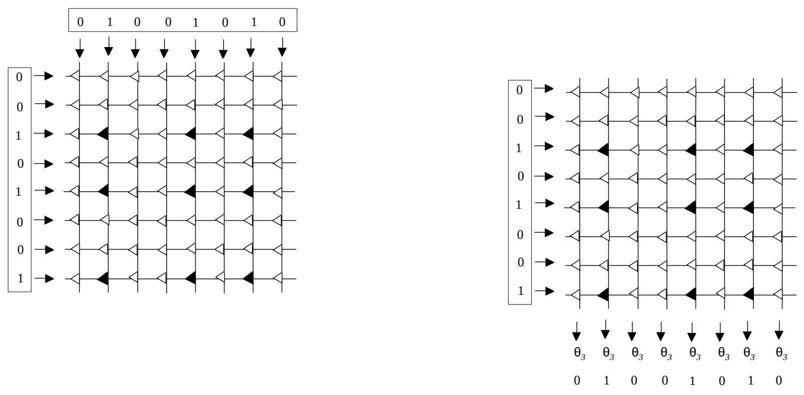
0 1 0 0 1 0 1 0
0 0 1 0 1 0 0 1
0 0 1 0 1 0 0 1
θ_3 θ_3 θ_3 θ_3 θ_3 θ_3 θ_3 θ_3
0 1 0 0 1 0 1 0

Figure 2: Hetero-associative net: Storage and Retrieval

ANs associate pairs of input and output vectors using a grid of horizontal input lines and vertical output lines with binary switches (triangles) at the intersections (Figure 2). To store an association between the input vector and the output vector, switches are turned on (black triangles) at the intersection of lines which correspond to a 1 in both input and output patterns.

To retrieve the associate of the input, a signal is sent down each input line corresponding to a 1 in the input. When this signal encounters an "on" switch, it increments the signal on the corresponding output line by one. The output lines are then thresholded at a level corresponding to the number of "on" bits in the input. If we store an input pattern with itself as output (an *auto-associative net*), ANs can be used to complete partial patterns, as needed to recall perceptually non-evident properties of objects, such as the fact that the red cube on the table affords grasping. This is exactly the information that is encoded in action precondition axioms. Further it is worthwhile to notice that all of the information needed for this AN is available in each new instance of an ISTF. In this case, the input and output patterns for the AN are the same: the initial state, action, and object for a cluster of reproducable ISTFs observed in the course of interacting with the world. We thereby use repeated presentations of very similar ISTFs (clustered by action and object) to train auto-associative ANs to effectively store and retrieve associations between the LDEC action precondition axioms and the property of *affording* such LDEC operators.

Now consider the LDEC style effect axioms. Rather than using an auto-associative net we can use a hetero-associative network for this task. In this case, we again use the initial state, action, and object as the input pattern from each ISTF, however as the output pattern we use the resulting state from the ISTF. This will allow us to learn and retrieve the state-change transitions associated with LDEC operators, with states represented as sparse vectors of relevant facts or propositions.

Thus, we hypothesize that such associations can be learned in ANs using repeated presenations of reproducable ISTFs using the Perceptron Learning Algorithm (PLA). We replace the binary AN switches with continuous valued switches and use multiple ISTFs that have the same action, object, and resulting state and the PLA to adjust the weights on the relevant switches. We believe that such an approach can learn consistent state changes or actions, and learn the association between preconditions and associated affordances.

More specifically, in the envisioned scenario, as the robot controller explores the world, successful grasps will produce ISTFs. On the basis of multiple reproducable experiences of particular ISTFs we can learn the instantiated versions of the precondition axioms and the effect axioms for the robots actions. The resulting state in each ISTF will vary only in terms of the object-type grasped and the grippers pose. Further, the invariants can be learned as a basis for classifying the world into object classes and action types. As we have discussed, identifiers for actions-types can then be associated with the input conditions for the action via an auto-associative net. Such affordances are added by adding new input and output lines to the net for the new affordance, and using the existing learning algorithm.

This network can be presented with a possibly incomplete set of properties representing the current state of the world, and used to retrieve a complete model of the world state, including non-perceptually available associates including the affordances and object classes.(Figure 3) For ease of exposition, in

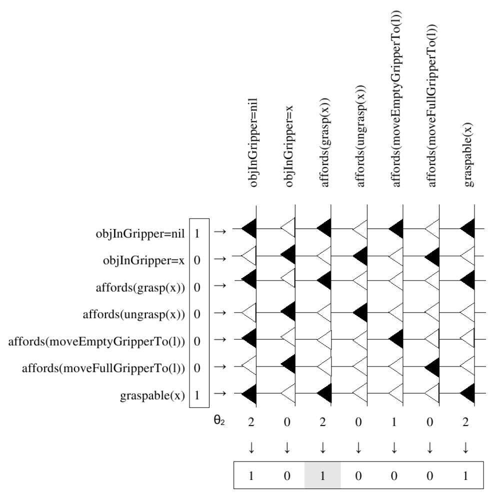

Figure 3: Retrieval of *affords*(*grasp*(x)) from *objInGripper* = *nil* ∧ *graspable*(x) in the loaded auto-associative net

this and the following figures we will continue to show weights of 0 and 1. The full pattern including affordances can then be input to the other hetero-associative net, and used to retrieve the effects of carrying out particular actions. (Figure 4).

If the output states and affordances are the same following two different grasp actions for a particular input state, then clearly the effects (as far as the learner and planner are concerned) of the two grasps are the same for that input. If the effects are the same for all inputs then the grasps are equivalent and can be collapsed together. We discuss this next.

5.1 Learning Multiple Grasp Actions

Recall from our discussion of the high-level action *grasp* that at the lower level there may in fact be many low-level grasps available to the robot at any point. While many of these grasping actions may have effects that are indistinguishable from one another, there will also be grasping actions that result in very different effects. Given this, and our desire to avoid the difficulties of learning actions with conditional effects, it becomes clear that we will need multiple grasp actions at the higher level of abstraction. To distinguish these actions and their effects during planning and learning we will introduce multiple predicates indicating “graspability” by particular motor programs.

Our learning process now operates as follows: when an object is “born” at the lower level of representation (See Section 2), the message for the addition of the object (e.g., *obj*23) should include an identifier for the specific action that was executed (e.g., *grasp*28, *grasp*95, etc.) as well as asserting the existence of a new predicate indicating the object has that action as an affordance (e.g., *affords*(*grasp*28(*obj*23))).[2] This predicate is added to the AN and can be used for learning.

We make the strong assumption that the invariants of the domain map onto the input units of the associative network, which we assume in animals have evolved to this end and for the robot must be built in, are such as to ensure that when distinct low-level motor programs are indistinguishable at the higher level of abstraction, they will automatically be classified as instances of a single action.

6 Using Learned Abstract Action Representations

We have described a process that results in learning abstracted action representations that should be close to the LDEC representations we have sketched for this domain. However, by abstracting the actions in

[2]Although we only consider grasping actions, we assume other actions, such as pushing, also result in the “birth” of an object-affordance complex.

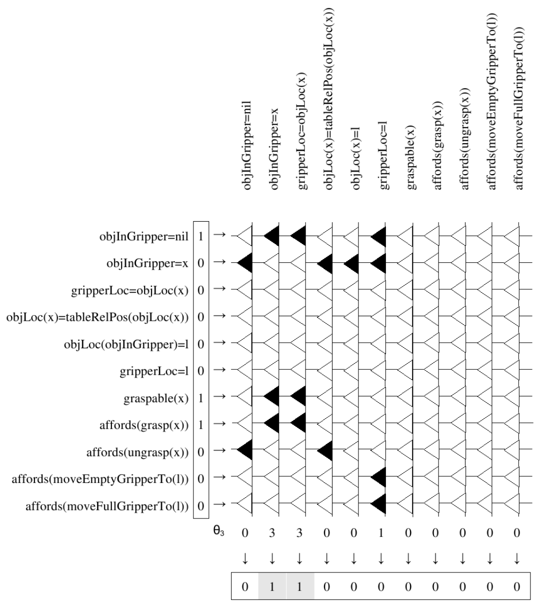

Figure 4: Retrieval of effect *grasp*(x) from the hetero-associative net

this way there remains a number of open concerns we must address.

Using Learned Action Knowledge with New Objects All new objects are initially associated with "new" actions. Our problem is to associate a previously unseen motor-program object pair with an existing high-level action or to mark it as a new action that must be learned at the high level.

Using Learned Action Knowledge for Execution It will be necessary to convert our learned abstract actions to specific motor programs for execution. Keeping the list of the motor program-object pairs abstracted by each high-level action should address this issue. Since all abstracted pairs for a given action should be equivalent, we suggest selecting any one that matches the object bound in the high level plan.

Learning Exogenous Domain Properties Although we have described a process for learning certain domain properties, the question remains as to how we will learn the exogenous properties given in Definition 3. For the present we simply assume the presence of *over* as an exogenous domain property that is computed by a lower level function.

7 Conclusion

This paper has argued that object-action complexes (OACs) grounded from instantiated actions in robot control-space, can be used as an interface between the very different representation languages of robot control and AI planning. We have shown that OACS can be embodied in an Associative Net, and that they can be learned by a very simple machine-learning algorithm. Almost all of these claims are unproven but we offer them as defining a research program that we shall be pursuing in the coming years in order to combine existing robot platforms and existing planners based on LDEC and other situation/event calculi.

References

[1] Daniel Aarno, Johan Sommerfield, Danica Kragic, Nicolas Pugeault, Sinan Kalkan, Florentin Wörgötter, Dirk Kraft, and Norbert Krüger. Integration of elementary grasping actions and second

order 3d feature relations for early reactive grasping. *submitted to 2006 IEEE-RAS International Conference on Humanoid Robots*, submitted.

[2] John Anderson. A memory storage model utilizing spatial correlation functions. *Kybernetik*, 5:113–119, 1968.

[3] Yining Deng and B.S. Manjunath. Unsupervised segmentation of color-texture regions in images and videos. *IEEE transactions on Pattern Analysis and Machine Intelligence*, 2001.

[4] Richard E. Fikes and Nils J. Nilsson. STRIPS: A new approach to the application of theorem proving to problem solving. *Artificial Intelligence*, 2:189–208, 1971.

[5] J.-Y. Girard. Linear logic. *Theoretical Computer Science*, 50:1–102, 1987.

[6] D. Harel. Dynamic logic. In D. Gabbay and F. Guenthner, editors, *Handbook of Philosophical Logic, volume II*, pages 497–604. Reidel, Dordrecht, 1984.

[7] B. Hommel, J. Müsseler, G. Aschersleben, and W. Prinz. The theory of event coding (tec): A framework for perception and action planning. *Behavioral and Brain Sciences*, 24:849–878, 2001.

[8] N. Krüger, M. Van Hulle, and F. Wörgötter. Ecovision: Challenges in early-cognitive vision. *International Journal of Computer Vision*, 2006.

[9] Norbert Krüger, Markus Ackermann, and Gerald Sommer. Accumulation of object representations utilizing interaction of robot action and perception. *Knowledge Based Systems*, 15:111–118, 2002.

[10] John McCarthy and Patrick J. Hayes. Some philosophical problems from the standpoint of artificial intelligence. *Machine Intelligence*, 4:463–502, 1969.

[11] Fabrice Moscheni, Sushil Bhattacharjee, and Murat Kunt. Spatiotemporal segmentation and based on region merging. *IEEE transactions on Pattern Analysis and Machine Intelligence*, 1998.

[12] R.M. Murray, Z. Li, and S.S. Sastry. *A mathematical introduction to Robotic Manipulation.* CRC Press, 1994.

[13] Edwin P. D. Pednault. ADL: Exploring the middle ground between STRIPS and the situation calculus. In Ronald J. Brachman, Hector J. Levesque, and Raymond Reiter, editors, *Proceedings of the First International Conference on Principles of Knowledge Reprensentation and Reasoning (KR-89*, pages 324–332, San Mateo, CA, 1989. Morgan Kaufmann Publishers.

[14] Tony Plate. Holographic reduced representations: Convolution algebra for compositional distributed representations. In *Proceedings of the 12th International Joint Conference on Artificial Intelligence, San Mateo CA*, pages 30–35, San Francisco, CA, 1991. Morgan Kaufmann.

[15] N. Pugeault, F. Wörgötter, , and N. Krüger. Multi-modal scene reconstruction using perceptual grouping constraints. In *Proceedings of the 5th IEEE Computer Society Workshop on Perceptual Organization in Computer Vision, New York City June 22, 2006 (in conjunction with IEEE CVPR 2006)*, 2006.

[16] Jianbo Shi and Jitendra Malik. Motion segmentation and tracking using normalized cuts. In *ICCV*, pages 1154–1160, 1998.

[17] Friedrich T. Sommer and Günther Palm. Bidirectional retrieval from associative memory. In Michael I. Jordan, Michael J. Kearns, and Sara A. Solla, editors, *Advances in Neural Information Processing Systems*, volume 10. The MIT Press, 1998.

[18] Mark Steedman. Temporality. In Johan van Benthem and Alice ter Meulen, editors, *Handbook of Logic and Language*, pages 895–938. North Holland/Elsevier, Amsterdam, 1997.

[19] Mark Steedman. Plans, affordances, and combinatory grammar. *Linguistics and Philosophy*, 25:723–753, 2002.

[20] K Steinbuch. Die lernmatrix. *Kybernetik*, 1:36–45, 1961.

[21] David Willshaw. Holography, association and induction. In Geoffrey Hinton and James Anderson, editors, *Parallel Models of Associative Memory*, pages 83–104. Erlbaum, Hillsdale, NJ, 1981.

[22] David Willshaw, Peter Buneman, and Christopher Longuet-Higgins. Non-holographic associative memory. *Nature*, 222:960–962, 1969.

Appendix B

Paco-Plus Design Documentation for Integration of Robot Control and AI Planning

Christopher Geib, Ronald Petrick, Kira Mourão, Nicolas Pugeault, and Mark Steedman
University of Edinburgh

Pascal Haazebroek
Leiden University

Norbert Krueger, Dirk Kraft
Aalborg University

Florentin Wörgötter
University of Göttingen, Bernstein Center for Computational Neuroscience

15 December 2006

1 Overview

The following document was started as a way to capture the conclusions of the discussions between the authors, for specific proposals about how to interface the high-level "AI Planning layer" and the "Robot control layer". We believe that it most properly should be seen as an evolving design document specifying the conclusions that we came to and the implications for the interface between these two modules.

1.1 Document history

Rev. 1: Results of October 9th–11th meeting in Edinburgh
Rev. 2: Extended and modified as a result of November 23rd–24th meeting in Goettingen.

1.2 Objective

Our overall objectives are to produce a principled interface for the interaction of an AI level planning component with a lower level robotic control component with three requirements:

1. use of Object Action Complexes (OACs) to define and constrain the interactions
2. enable the discovery of objects, actions, and properties by the robot component.
3. enable the learning of AI planning level representations for the objects, actions, and properties on the basis of robotic level observations

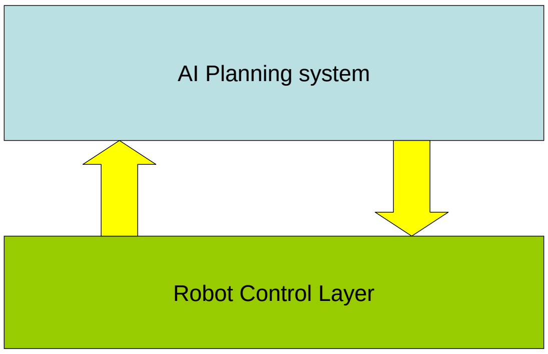

Figure 1: Very high-level system architecture

1.3 Basic Assumptions

We have been assuming a two layered architecture wherein the robot controller is responsible for operating and controlling "continuous" or real valued sensors and actuators and reporting up to the AI planning level discrete (possibly thresholded) state information. When possible, the AI planner will submit sequences of actions to be executed to the robot controller in order to achieve high-level goals.

This has the basic impact of requiring information to flow both from the robot controller to the AI Planning system and from the planning system back to the robot controller. In Figure 1 we see the flow of information between these two components. The purpose of much of this document is to define the nature of that communication and changes in control that accompany them.

2 Issues for the Interface

During our discussions we identified a number of issues that any interface between the AI planner and Robot controller must address. In this section we identify those issues and sketch any agreed upon solutions.

2.1 System Level control

As with any multi-layer control system where there are multiple controllers that can suggest actions for execution there is an issue about adjudicating between the suggested actions. Effectively the question is: when is each control algorithm in charge of determining the next action for the system as a whole to execute.

As will be clear when we discuss a proposed example of the system's execution, one of the primary objectives of the system is to acquire more information about the state of the world, the objects in it and the actions that can be performed on them. This has the following implications. Most of the actions called for by the lower level robot controller are reflexive and exploratory in nature. Most of the actions called for by the AI planning level are goal directed and require richer models that must be learned from the lower level. As a result the interface is designed such that while the AI planner has sufficient information to suggest plans to achieve goals it is allowed to hand actions to the lower level for execution. When the goals of the AI system have been achieved or the system doesn't have enough information about the state of the world the robot controller's exploratory behaviours are invoked. We will see this in much more detail when we discuss an example of the system working.

2.2 Exogenous Domain Predicates

We define *exogenous domain predicates* as domain information required by the AI level about the state of the world that cannot be determined by AI system reasoning. Such predicates represent critical state information needed for planning, for instance for reasoning about the relative positions of objects. An example of this would be the predicate “over” as in “over(object23, object98)”. It has been generally agreed that the lower level robot controller is capable of providing this information. Such predicates may fall into two distinct classes.

1. **Computed:** A simple definition of the predicate “over” would simply be the projection upwards of the bounding box for the object. This is easily derivable from the objects visual features, however may not cover more detailed action needs.
2. **Learned:** Defining “over the box for the purpose of dropping something into the box” may be a predicate that requires a more active learning process. This may include experimentation.

This means that in the AI level representation we will have more “prepositions” including prepositions that are task specific: for example, “over-for-grasp23” and “over-for-drop-into”. This is consistent with linguistic evidence about interpreting prepositions within context.

2.3 Monitoring of Lower Level Action Execution

As part of such an architecture it is important for the AI level to be able to monitor lower level action execution and, if the plan is failing to meet its goals, to be able to stop the execution. To meet this requirement we place the following requirement on the AI planner: as the world state is updated at the higher level, the AI level is required to monitor these conditions and issue a “halt action” command if the action is not proceeding according to expectations.

2.4 Agreement not to use Specific Locations

Testing equality of locations is very hard to do at the robot level and not something we should ever expect will be available from the low level. This strengthens the idea that almost all work done at the AI level for planning should be done relative to specific object identifiers rather than positions in an *N* dimensional space.

2.5 Preparatory Actions

There is a necessity for including actions in the AI level plan for “grasp setup actions”. Robotic path planning is very complex and we need to make a nod in the direction of acknowledging that. At this point we believe this will be little (or possibly nothing) more than a stub action. But recognizing the problem is significant. We note that learning such preparatory actions may present a much more significant challenge for the AI level, since it is not as obvious what if any perceptual change in the world would be relevant for the learning of this action. But this may be consistent with the child development evidence.

2.6 Division of Representation

Different representations and the corresponding operations that generate/manage these representations need to be separated into two levels: the lower vision/robot level and the higher AI planning level. We believe that the lower level should manage information concerning visual features, relations, and untested reflexes. Class information and newly born objects would be generated at the lower-level through the operation of the vision system.

The information provided by the lower level would be used to induce the high-level representations. The high level would operate with unique identifiers (possibly generated at the higher level on the request of the lower level) that would be used to index lower-level objects, classes, and motor programs/reflexes. The high level would have no direct access to local visual features and relations, and would avoid reasoning about continuous values. Plans generated at the high level should therefore contain appropriate “links” to the required low-level entities using the unique identifiers.

2.7 Example of Pulling Objects into the Workspace

There are a number of interesting problems that clearly motivate the need for integrating AI level planning and lower level robot planning. For example, consider the case of pulling an object that is currently unreachable, into the working space. This is a textbook example of where AI level planning can be used to make objects more accessible. The robot system is unable to reach the object initially, however it is possible to invoke the AI planner to build a plan to pull the object into the work space to allow for a better grasping action.

The significant caveat is that we need to be sure to be able to define more exogenous domain predicates that will allow us to do the planning in a discrete space, rather than attempting to do “geometric reasoning” at the AI planning level. For example, the addition of “not in reach” as a perceptual primitive would be needed for this example, and could be provided by the robot level.

3 Demo 1 Example

As a design document we will not include here a detailed discussion of the component technologies that go into this system. Instead we suggest reading [1, 2, 3] for details of how the AI planning system and robot control system work. In this section we will provide a very simple sketch of an imagined interaction between these components to achieve the end of learning about the world, culminating in clearing a table of a number of coffee mugs.

We imagine our AI planning system and robot arm equipped with a vision system confronted with a table that has on it a number of objects:

1. three identical cups,
2. three identical plates, and
3. a box the cups and plates should be put into.

We do not assume the system has knowledge of the cups or plates or their affordances, however, we do assume that the system knows about the box and its affordances, and that the AI system has the goal of putting all objects on the table into the box.

We envision the following scenario:

1. Since the AI planner does not know about the objects and their affordances, it invokes the robot level reflexive exploration behaviour.
2. The robot controller experiments with choosing pairs of coplanar points and attempting to grasp them to see if they define an object. This task repeats until the robot manages to effectively grasp an object. For the first case we assume it is one of the cups.
3. By moving the object and subtracting out its gripper the robot forms a model of the cup, and can then place it back on the table.
4. As a result of this exploration the robot controller reports to the AI planner the “birth of an object”. As part of this it reports: a UID for the class of objects, a UID for each instance of the object in the visual field (in this case one for each of the three cups), and a UID for the associated affordance of the object (namely the single successful grasping instance just used).

5. On the basis of this new information the AI planning system now knows that its goal of having a clear table is not satisfied and it attempts to achieve it. Given the new objects and their grasping affordances the planner is able to build a plan for putting the three cups in the box and sends that plan to the robot controller for execution.

6. The robot controller executes the plan, sending reports of state changes back to the AI system (to enable it to monitor the plan for progress). We assume the plan works perfectly, placing all of the cups in the box.

7. The AI system now believes that its goal of having a clear table has been met (since it knows nothing about the plates) and again invokes the robot controller's reflex driven exploration behaviour looping back to the first step.

In discussing this type of demo we have made the commitment that the AI planner would not ask the low-level system to execute any action that it hasn't done before (at least as far as identified classes of objects are concerned). However it is worth noting that the large amount of information we are assuming is already present in the AI planning system. For example, in this document (though not in other places [1]) we have glossed over the question of how the system actually learns the high-level representations. Such learning requires previous knowledge of exogenous domain predicates, classes of actions and much other knowledge that we are assuming we have access to for this first demo.

4 List of Robot Primitives

The following is a proposed list of the primitive robot reflexes that are initially available in the system. Note that the names given in this list may not correspond to the names given to these primitives in the robot system, but instead capture the major classes of reflex behaviours available from the robot system.

- **moveGripper:** This corresponds to the general movement of the robots end effector to a specific location in the domain. It will underlie two of the AI level actions, *moveEmptyGripper* and *moveFullGripper*.
- **grasp1, grasp2, . . . , graspX:** This set forms the low-level reflex for grasping based on a pair of features within the visual field. This will be the foundation for the AI system's ability to learn generalized grasping.
- **ungrasp1, ungrasp2:** This set are the reflexes that are used to release an object that has been grasped.

These may of course be extended with other low-level primitives.

5 Definition of Domain Terms

To be more specific in our discussions, we define Object, ObjectType, Pose, Location, and Affordance.

Definition 1 Proposed terms that will be used in following definitions

1. **Object:** a unique object identifier. That is a unique identifier of an object instance in the world model that is shared between the AI planner and the Robot/vision system.

2. **ObjectType:** a unique identifier denoting the type or class of a particular object instance. Each Object has a single ObjectType.

3. **Pose:** an identifier that denotes the current orientation of an object. The set of all available poses is an enumerated type, e.g., $\{1, 2, 3, \ldots, \mathit{upright}\}$

4. **Location:** a unique location identifier. That is a unique identifier used to refer to a location in the robot/vision systems world model.
5. **Affordance:** a unique affordance identifier. That is a unique identifier of an affordance that can be re-executed by the Robot/vision system.

6 Functions that map objects to locations

We are already aware of a number of specialized functions that will be required for the AI system to map from Objects to Locations for use by the robot system. These functions will have to be learned by the system and this must take place both at the robot level (training a perceptual primitive to send a message to the AI system when they are true) and at the AI level (learning when they are important for describing the preconditions or effects of actions).

- **into:** This function maps from an object that is concave to a point inside the object. This function is assumed to produce a location such that if an object in the gripper is released from the location, the object can be considered "carefully placed into the object". Note that if the object is not known to be concave the function throws an exception?
- **onto:** This function maps from an object to an arbitrary point that is wholly on the object. This function is assumed to produce a location such that if an object in the gripper is released from the location, the object can be considered "carefully placed on the object".
- **tograsp1, tograsp2, . . . , tograspX:** For each grasp affordance that is learned we introduce a function that will map from an object that will be grasped using that affordance to a location that sets up the grasp.

7 Messages from the Robot: Predicates that Define the World State at the AI Planning Level

In order for the AI Planning system to maintain a world model for planning the robot/vision system must send messages to the AI planning system about any significant perceptual changes to the world. Such messages should be "pushed" to the AI system to allow asynchronous update of the world model even possibly during the execution of an action requested by the AI planning system.

The following table specifies the current complete set of the messages the robot/vision system must be able to send for this purpose. Note that we distinguish the messages that contain meta-information (the introduction of new objects or affordances to the world model) from the messages that update the state of a predicate in the world model.

Predicate Def	Example Use	Descriptive Note
$in(Object_{container}, Object_{contained})$	$in(box1, obj1)$	Captures locations of objects
$on(Object_{supporting}, Object_{supported})$	$on(table1, obj1)$	Captures locations of objects
$ingripper(Object)$	$ingripper(obj1)$	Captures locations of objects
$pose(Object, Pose)$	$pose(obj1, upright)$	Captures the pose of the object
$gripperempty$	$gripperempty$	Captures the state of the gripper when empty
$gripperat(Location)$	$gripperat(tograsp23(obj1))$	Captures the location of the gripper
Message Def	**Example Use**	**Descriptive Note**
$newobj(Object)$	$newobj(obj1)$	Introduces a new object.
$newaff(ObjectType, Affordance)$	$newaff(objtype1, grasp28)$	Introduces a new affordance for an object.

Figure 2: Messages sent by the Robot to the AI Planning Level

8 Messages from the AI Planning Level: Action Requests from the Planner

The AI planner requests actions for execution. Each such action has specific action effects that should be visible in the world model after a successful execution. The following table provides the specification for the action, an example of its use, and the expected change the robot level will report back to the AI level if the action is successful.

Action Def	Example Use	Successful Execution Result
moveEmptyGripper(*Location*)	*moveEmptyGripper*(*tograsp23*(*obj*1))	*gripperat*(*tograsp23*(*obj*1))
graspI(*Object*)	*grasp23*(*obj*1)	*ingripper*(*obj*1)
moveFullGripper(*Location*)	*moveFullGripper*(*into*(*box*1))	*in*(*box*1, *obj*1)
drop(*Object*)	*drop*(*obj*1)	*gripperempty*
beginexploration	*beginexploration*	*N/A*

Figure 3: Possible AI Planning Level Action Requests

We note that *beginexploration* is special in the sense that it is a meta-level operation that initiates a process at the robot level, without any direct execution results.

9 Tasks for AI Planning Level

The following are the major tasks the AI Planning Level team needs to complete.

1. **Encode Planning Domain as Specified:** This includes encoding the planning problem as specified above with the given domain primitives and actions. This will result in a planner capable of building plans for clearing the table of discovered objects. Note that for our example domain all that will be missing for the system is the specific action/affordance and object instance information that is needed to produce the plan. This will be provided by the robot controller.

2. **Build High-Level Planning System Architecture:** This includes creating an infrastructure that builds plans, submits plans for execution, verifies the resulting state is consistent with its expectations and if so submits the next action in the plan.

3. **Build Truth Maintenance System for High-Level World Model:** Since the robot system is responsible for the assertion of some facts that have multiple effects on the world model at the AI level, we are responsible for building a small system to effectively update the AI world model.

4. **OPTIONAL: Interface to GraspIt Software:** If the interface to the GraspIt software is identical to the interface that will be needed for the Robot then we will make any small changes necessary for the testing of the AI Planning system within this simulation environment. To the degree that the interface would be significantly different and require significant extra code we are not sure that we see the value of the effort in such an integration.

10 Tasks for Robot Level

The following are the major tasks the Robot/Vision team needs to complete. [this needs real work]

1. **Smoothing of percepts to remove irrelevant discontinuities:**

2. **Construction of perceptual functions**

3. **Wrapper for execution of requested tasks**

4. **Others?**

11 Known "Open" Issues that will Require Addressing Later

During our discussions we identified a number of issues that we will need to return to address later. These include:

1. **Dropping and "onto" vs push and "into"**
2. **Pushing and pulling actions**
3. **Objects "disappearing" from the perceptual system**

References

[1] C. W. Geib, K. Mourão, R. Petrick, N. Pugeault, M. Steedman, N. Krueger, and F. Wörgötter. Object action complexes as an interface for planning and robot control. In *Proceedings of the Humanoids-06 Workshop - Toward Cognitive Humanoid Robots*, 2006.

[2] R. P. A. Petrick and F. Bacchus. A knowledge-based approach to planning with incomplete information and sensing. In *Proceedings of the Sixth International Conference on Artificial Intelligence Planning and Scheduling (AIPS-2002)*, pages 212–221. AAAI Press, 2002.

[3] R. P. A. Petrick and F. Bacchus. Extending the knowledge-based approach to planning with incomplete information and sensing. In *Proceedings of the 14th International Conference on Automated Planning and Scheduling (ICAPS-04)*, pages 2–11. AAAI Press, 2004.

Appendix C

Representing knowledge and sensing in the Linear Dynamic Event Calculus for knowledge-level planning

Ronald P. A. Petrick and Mark Steedman

School of Informatics
University of Edinburgh
Edinburgh EH8 9LW, Scotland, UK
{rpetrick, steedman}@inf.ed.ac.uk

Abstract

The Linear Dynamic Event Calculus (LDEC) is a logical language for reasoning about actions and change. One of the novel features of this formalism is its close relationship with STRIPS, and its STRIPS-style treatment of fluent change for addressing the frame problem. In this paper we extend the LDEC representation to incorporate knowledge and sensing actions using the idea of "knowledge fluents"—fluents that model the agent's knowledge of ordinary world-level fluents. Using this representation, we show how to transform LDEC actions into STRIPS-like planning operators usable by the knowledge-level contingent planner PKS.

1 Introduction

The construction of practical planning systems can often benefit from the insights provided by formal representations based on logical languages. Since the core task of a planner is to reason about actions and change, formal approaches that facilitate efficient reasoning can aid in the design of effective planning techniques.

In this paper we focus on a variant of the situation calculus [9] called the Linear Dynamic Event Calculus (LDEC) [17, 18], a logical formalism for modelling dynamically-changing worlds. The importance of LDEC lies in its close connections to STRIPS [4]. In particular, this representation motivates a simple forward-chaining approach to planning using STRIPS-style action operators, making it a natural choice for representing classical planning domains.

One of LDEC's weaknesses, in its original form, is that it lacks an obvious way to represent knowledge and sensing actions. While classical planning domains typically assume an agent has complete knowledge, such assumptions are not always reasonable. Reasoning about sensing as a form of action, however, requires the ability to reason effectively about knowledge and is often a prerequisite for modelling complex domains, such as those that require contingent planning.

Conceptually, the problem of modelling knowledge and action has been extensively studied and many approaches share a common treatment of knowledge, namely that reasoning about knowledge reduces to reasoning about an accessibility relation over a set of possible worlds. From a computational point of view, working directly with such a representation is problematic since establishing that n propositional formulae are known (a task that planners often perform) may require reasoning about 2^n possible worlds.

We propose an extension to LDEC that avoids the possible-worlds account of knowledge. Instead, we adapt an existing approach from the situation calculus that represents knowledge using "knowledge fluents," i.e., fluents that directly model the agent's knowledge of ordinary world-level properties [3]. Although this approach is representationally less expressive than the possible-worlds model, it has the advantage that reasoning about knowledge reduces to the problem of reasoning about ordinary fluent change and, thus, can be made quite efficient in practice.

We will also take advantage of the insights provided by *knowledge-level planning*, which has been shown to be an effective alternative in certain domains to planners that use world-level representations (e.g., [1], [19], plus others). In particular, we show how to syntactically transform LDEC domains into planning operators usable by the contingent planner PKS (Planning with Knowledge and Sensing) [12]. Since PKS is itself built on an extended-STRIPS representation, we can also interpret our new LDEC domains in terms of STRIPS-style action operators, and use LDEC as a practical formalism for modelling conditional planning domains.

This paper is organized as follows. First, we briefly describe the LDEC formalism and demonstrate the relationship between LDEC and STRIPS. We then introduce the notion of knowledge into LDEC and use it to model an interesting range of sensing actions. Using these new additions, we show how they relate to the extended-STRIPS representation used by the PKS planner. Finally, we give some insights into an intended application of LDEC, and describe some future extensions of our work.

2 Linear Dynamic Event Calculus (LDEC)

The Linear Dynamic Event Calculus (LDEC) [17, 18] is a logical formalism that combines the insights of the Event Calculus of [8], itself a descendant of the Situation Calculus [9], and the STRIPS planner of [4], together with the Dynamic and Linear Logics developed by [6], [7], and others.

The particular dynamic logic that we work with here exclusively uses the deterministic "necessity" modality $[\alpha]$. For instance, if a program α computes a function f over the integers, then an expression like "$n \geq 0 \Rightarrow [\alpha](y = f(n))$" indicates that "in any situation in which $n \geq 0$, after every execution of α that terminates, $y = f(n)$." We can think of this modality as defining a logic whose models are Kripke diagrams. Accessibility between situations is represented by events defined in terms of the conditions which must hold before an event can occur (e.g., "$n \geq 0$"), and the consequences of the event that hold as a result (e.g., "$y = f(n)$").

This logic also includes the dynamic axiom

$$[\alpha][\beta]\phi \Rightarrow [\alpha;\beta]\phi,$$

which defines the *sequence* operator ";" as a composition operation over events. Like other dynamic logics, LDEC does not use explicit situation terms to denote the state-dependent values of domain properties. Instead, it uses the sequence operator to chain together finite sequences of actions. For instance, $[\alpha_1, \alpha_2, \ldots, \alpha_n]$ denotes a sequence of n actions and $[\alpha_1, \alpha_2, \ldots, \alpha_n]\phi$ indicates that ϕ must necessarily hold after every execution of this action sequence.

LDEC also mixes two forms of logical implication, which contributes to its representational power. Besides standard (or intuitionistic) implication $\Rightarrow$, LDEC follows [2] and others in using *linear* logical implication, denoted by the symbol $\multimap$. It is this second form of implication that leads to a solution to the *frame problem* [9], as we'll see below.

An LDEC *domain* is formally described by a collection of axioms. *Actions* (or *events*) provide the sole means of change in the world, and affect the *fluents* (i.e., properties) of the domain. We define an LDEC domain as follows.

Definition 1 LDEC domains are described by the axioms:

1. For each action α, an *action precondition axiom*:

$$L_1 \wedge L_2 \wedge \ldots \wedge L_k \Rightarrow \mathit{affords}(\alpha),$$

   where each L_i is a fluent literal (a fluent or its negation).

2. For each action α, an *effect axiom* of the form:

$$\{\mathit{affords}(\alpha)\} \wedge \phi \multimap [\alpha]\psi \wedge \gamma,$$

where ϕ has the form $C_1 \wedge C_2 \wedge \ldots \wedge C_l$, ψ has the form $\neg D_1 \wedge \neg D_2 \wedge \ldots \wedge \neg D_m$, and γ has the form $F_1 \wedge F_2 \wedge \ldots \wedge F_n$. Each C_i, D_i, and F_i is a unique fluent. Also, the C_is must appear syntactically identical in the corresponding action precondition axiom for α.

3. A collection of *initial situation axioms* of the form:

$$L_1 \wedge L_2 \wedge \ldots \wedge L_p,$$

where each L_i is a ground fluent literal.

Definition 1 enforces certain syntactic restrictions to the axioms we permit, to accommodate the representational limitations inherent in many planners. For instance, actions and fluents may be parameterized, however, we restrict all fluent arguments to only contain variables that also appear in the corresponding α term of the same axiom. (Free variables are treated as being universally quantified.) Moreover, we assume that whenever an action α appears in an axiom, it always has the same parameters.

Action precondition axioms specify the conditions that *afford* a particular action. For instance, axiom (1) in Table 1 specifies that in order to pick up an object x, an agent's hand must be empty and the object x must be on the table.

We also define a transitivity property for *affords*:

$$\mathit{affords}(\alpha) \wedge [\alpha]\mathit{affords}(\beta) \Rightarrow \mathit{affords}(\alpha; \beta).$$

This rule says that any situation which affords an action α, and after performing α results in a situation which affords action β, is a situation which also affords α *then* β. Thus, we can also reason about the possibility of action sequences.

Logical languages like LDEC must also address the *frame problem* [9], which arises since actions typically affect just a few properties among a large set of properties that constitutes the state of the world. As a result, formal representations must not only capture the changes caused by action effects on the properties of the world, but also the "non-effects" of actions on those properties that remain unchanged.

Effect axioms use linear implication to build certain "update rules" directly into the LDEC representation. In particular, when an effect axiom is applied, the fluents in the antecedent (the C_is) are treated as consumable resources that are "replaced" by the fluents in the consequent (the $\neg D_i$s and the F_is).[1] A formula contained in $\{\cdot\}$ indicates that it is a non-consumable resource. All other fluents remain unchanged. Thus, linear implication lets us avoid having to include explicit frame axioms in our LDEC domains. Such rules are instead built into the background proof theory concerning linear implication (see [17, 18]). For instance, axiom (3) in Table 1 indicates that if a situation affords picking up x, and the agent's hand is empty and x is on the table, then after performing the action the agent will be holding x, the agent's hand will cease to be empty, and x will no longer be on the table. All other world properties are unaffected.

Finally, initial situation axioms complete a domain description by defining the initial status of particular fluents.

3 LDEC and classical STRIPS planning

The axioms in Definition 1 provide a straightforward way of generating *plans*, by finding an action sequence afforded by the initial situation that satisfies a given goal formula.

Example 1 Consider the LDEC domain described by the axioms in Table 1, and say that we seek a constructive proof to establish $\mathit{affords}(\alpha) \wedge [\alpha]\mathit{inBox}(o1, b1)$, i.e., a "possible" action sequence α whose

[1]We treat consumed fluents as being made false and we require that these fluents be part of a corresponding action precondition axiom. In general, linear implication lets us model actions with resource constraints, however, we do not focus on such actions here.

execution results in $o1$ being in box b_1. Since *handempty* and *onTable*$(o1)$ hold by the initial situation axiom (5), the situation affords *pickup*$(o1)$ by axiom (1). Applying this action's effects in axiom (3) causes these two facts to be replaced by *holding*$(o1)$ in the resulting situation. This new fact, together with *empty*$(b1)$ (from (5)) affords *dropInBox*$(b1)$ by axiom (2). The result of this action in axiom (4) causes these two facts to be invalidated, but brings about *handempty* and *inBox*$(o1, b1)$, satisfying the goal. Thus, one solution is given by the plan $\alpha = [\textit{pickup}(o1); \textit{dropInBox}(o1, b1)]$. $\square$

One of the novel features of LDEC is that we can interpret its use of linear implication in terms of operations that update a STRIPS database [4]. In STRIPS, a database $\mathcal{D}$ represents the world state. $\mathcal{D}$ is also a *closed world* database that let us use negation as failure to determine the truth of a formula ϕ: ϕ is true if $\phi \in \mathcal{D}$ and $\neg\phi$ is true if $\phi \notin \mathcal{D}$. Action preconditions and goals are established by simply querying $\mathcal{D}$ in this way to verify their truth. The effects of an action α are modelled by two sets of updates that, when applied to $\mathcal{D}$, produce a new database representing the state resulting from the execution of α: formulae added to $\mathcal{D}$ indicate the properties that α makes true, while formulae deleted from $\mathcal{D}$ indicate the properties that α makes false. All other properties remain unchanged.

As we have seen, linear implication combined with the necessity operator performs state-changing operations similar to those of STRIPS. The frame assumptions built into the background theory for linear implication also parallel STRIPS's treatment of properties that are unchanged by action. This resemblance lets us define a simple syntactic transformation for converting LDEC actions into STRIPS operators. Assuming all LDEC fluents and action names have STRIPS counterparts of the same syntactic form and arguments, we use X^S to denote the STRIPS form of an LDEC expression X in the following construction:

Transformation 1 For each LDEC action α, a corresponding STRIPS operator α^S is defined as follows:

1. The preconditions for α^S are given by $L_1^S, \ldots, L_k^S$, for the $L_i s$ in an action precondition axiom for α,
2. The delete list for α^S is given by $C_1^S, \ldots, C_l^S$ and $D_1^S, \ldots, D_m^S$, for the C_is, D_is in an effect axiom for α,
3. The add list for α^S is given by $F_1^S, \ldots, F_n^S$, for the F_is in an effect axiom for α.

Space prohibits us from including a proof of the soundness of Transformation 1, however, we note that the constituent parts of each LDEC effect axiom are extracted and converted into corresponding STRIPS add and delete lists. Effects that cause fluents to be negated, such as the $D_i s$ in the effect axioms, are represented by delete operations.

For example, if we apply Transformation 1 to the axioms in Table 1 we generate the STRIPS operators shown in Table 2. If we apply *pickup*$(o1)$ to an initial database denoted by $\mathcal{D} = \{\textit{box}(b1), \textit{handempty}, \textit{empty}(b1), \textit{onTable}(o1)\}$, we bring about the database $\mathcal{D}' = \{\textit{box}(b1), \textit{holding}(o1), \textit{empty}(b1)\}$. Applying *dropInBox*$(o1, b1)$ then produces the database $\mathcal{D}'' = \{\textit{box}(b1), \textit{inBox}(o1, b1), \textit{handempty}\}$, which achieves the goal of putting $o1$ in box $b1$, and matches our LDEC conclusions in Example 1.

Given suitable search control, we could employ a simple forward-chaining mechanism to automate the process of generating plans using the LDEC rules, or by first transforming them into STRIPS operators. Whichever form we choose, the existing LDEC representation limits us to modelling actions from *classical planning domains*, characterized by a restrictive assumption of complete knowledge.

4 Representing knowledge in LDEC

The assumption of complete knowledge is not always realistic. Agents often need to operate with incomplete information, and must sense the world to gather additional information. Reasoning about sensing

$$\begin{aligned}
&\textit{handempty} \wedge \textit{onTable}(x) \Rightarrow \textit{affords}(\textit{pickup}(x)), && (1)\\
&\textit{holding}(x) \wedge \textit{box}(y) \Rightarrow \textit{affords}(\textit{dropInBox}(x,y)), && (2)\\
&\{\textit{affords}(\textit{pickup}(x))\} \wedge \textit{handempty} \wedge \textit{onTable}(x) && \\
&\qquad \multimap [\textit{pickup}(x)]\textit{holding}(x), && (3)\\
&\{\textit{affords}(\textit{dropInBox}(x,y))\} \wedge \textit{holding}(x) \wedge \textit{empty}(y) && \\
&\qquad \multimap [\textit{dropInBox}(x,y)]\textit{inBox}(x,y) \wedge \textit{handempty}, && (4)\\
&\textit{box}(b1) \wedge \textit{handempty} \wedge \textit{empty}(b1) \wedge \textit{onTable}(o1). && (5)
\end{aligned}$$

Table 1: LDEC domain axioms

Action	Preconditions	Add list	Delete list
$\textit{pickup}(x)$	$\textit{handempty}$ $\textit{onTable}(x)$	$\textit{holding}(x)$	$\textit{handempty}$ $\textit{onTable}(x)$
$\textit{dropInBox}(x,y)$	$\textit{holding}(x)$ $\textit{box}(y)$	$\textit{inBox}(x,y)$ $\textit{handempty}$	$\textit{holding}(x)$ $\textit{empty}(y)$

Table 2: STRIPS operators produced by Transformation 1

requires the ability to reason about *knowledge*, however, which lets us distinguish between what is true in the world and what is known about the world.

The model of knowledge we introduce in LDEC makes use of *knowledge fluents*: fluents that directly track the agent's knowledge of ordinary world-level fluents [3]. For each ordinary fluent F, we associate with it a pair of knowledge fluents, KF and $K\neg F$, that have the same arguments as F. Intuitively, KF means "F is known to be true," and $K\neg F$ means "F is known to be false." $\neg KF$ and $\neg K\neg F$ together mean "F is unknown." Each pair of knowledge fluents therefore provides an explicit representation of the agent's knowledge of the underlying fluent F.[2] Using such fluents we can model the effects of sensing actions, and the changes that physical actions make, not only to the world state, but also to the agent's knowledge state.

Our approach contrasts the more standard approaches to representing knowledge that use sets of possible worlds (e.g., see [10], [14]). Reasoning about knowledge in these accounts reduces to the problem of reasoning about an accessibility relation over a set of possible worlds, each of which models a different configuration of the fluents. Such reasoning can be computationally expensive, however, since determining whether n fluents are known may require reasoning about 2^n possible worlds. With knowledge fluents, the problem reduces to determining if particular knowledge fluents are true or false. Thus, we need only reason about $2n$ knowledge fluents and n ordinary fluents. The main drawback with this approach is its expressiveness: we are restricted to simple assertions about relational knowledge.

Knowledge fluents are not a new idea. [3] introduces knowledge fluents into the situation calculus. [13] builds on this formalism to transform standard possible world accounts of knowledge into knowledge fluent accounts. [15] and [16] (among others) also represent definite knowledge using sets of fluents that are known to be true or known to be false.

LDEC can also support other representations of knowledge, including standard modal logics or possible world account. We choose to use knowledge fluents, however, since they provide a means of directly modelling knowledge as ordinary fluents, and they evolve as ordinary fluents do when actions are applied. Thus, we can work with simple first-order predicates without introducing new modal operators or reasoning about accessible possible worlds. Since planning systems often restrict their representation languages, we do not view the expressive limitations of knowledge fluents as being altogether problem-

[2]We only focus on representing *knowledge*, where known fluents must also be true in the world. This approach can also model *belief*, where an agent's beliefs need not be true in the world.

atic, and we can take advantage of existing approaches that work with similar constructs.

Knowledge fluents can appear wherever ordinary fluents can in our LDEC domains, however, we will consider particular effect axioms for our sensing and physical actions.

4.1 Knowledge-producing actions

For simplicity, we will only consider *knowledge-producing* or *sensing* actions that change the agent's knowledge state, rather than the world state. Thus, sensing actions will only affect changes to the knowledge fluents, but these changes will depend on the states of the ordinary world-level fluents.

Definition 2 A sensing action α is defined by n pairs of LDEC axioms, where each axiom pair i has the form:

$$\{\mathit{affords}(\alpha) \land \phi_i \land F_i\} \multimap [\alpha]KF_i \land \neg K\neg F_i,$$
$$\{\mathit{affords}(\alpha) \land \phi_i \land \neg F_i\} \multimap [\alpha]K\neg F_i \land \neg KF_i.$$

Each F_i is an ordinary fluent with corresponding knowledge fluents KF_i and $K\neg F_i$, $\phi_1 := \top$, and $\phi_i := F_{i-1} \land \phi_{i-1}$ for each $i = 2, \dots, n$.

In each pair i of axioms for a sensing action α, the status of F_i is (conditionally) sensed. This is done by associating the truth of F_i with an appropriate change to the status of KF_i and $K\neg F_i$. Note that F_i appears as a non-consumable resource since its status is unchanged by α. The fluents and actions that appear in Definition 2 can also be parameterized, provided the same set of arguments always appears for each F_i, KF_i, and $K\neg F_i$, across the n pairs of axioms for α.

Definition 2 lets us to model many interesting types of sensory effects, three of which we illustrate below: (i) actions that sense the truth value of a particular instance of some fluent, (ii) actions that sense the truth value of every instance of some fluent, and (iii) actions that sense the truth of a fluent and then conditionally sense the truth of other fluents, depending on how the initial sensing turns out. (We permit finite "chains" of such conditional sensing.)

Example 2 Recall the box scenario from Example 1, and say that an agent has two sensing actions: *sense*$_1$ and *sense*$_2$. *sense*$_1$ tells the agent whether or not the box is empty. *sense*$_2$ is much more complex, informing the agent of the objects in the box and whether or not each of these objects is fragile. We encode the effects of these actions with the axioms:

$$\{\mathit{affords}(\mathit{sense}_1) \land \mathit{empty}\} \multimap [\mathit{sense}_1]\mathit{Kempty} \land \neg K\neg\mathit{empty} \tag{6}$$
$$\{\mathit{affords}(\mathit{sense}_1) \land \neg\mathit{empty}\} \multimap [\mathit{sense}_1]K\neg\mathit{empty} \land \neg\mathit{Kempty} \tag{7}$$
$$\{\mathit{affords}(\mathit{sense}_2) \land \mathit{inBox}(x)\} \multimap \\ [\mathit{sense}_2]\mathit{KinBox}(x) \land \neg K\neg\mathit{inBox}(x), \tag{8}$$
$$\{\mathit{affords}(\mathit{sense}_2) \land \neg\mathit{inBox}(x)\} \multimap \\ [\mathit{sense}_2]K\neg\mathit{inBox}(x) \land \neg\mathit{KinBox}(x), \tag{9}$$
$$\{\mathit{affords}(\mathit{sense}_2) \land \mathit{inBox}(x) \land \mathit{fragile}(x)\} \multimap \\ [\mathit{sense}_2]\mathit{Kfragile}(x) \land \neg K\neg\mathit{fragile}(x), \tag{10}$$
$$\{\mathit{affords}(\mathit{sense}_2) \land \mathit{inBox}(x) \land \neg\mathit{fragile}(x)\} \multimap \\ [\mathit{sense}_2]K\neg\mathit{fragile}(x) \land \neg\mathit{Kfragile}(x). \tag{11}$$

Axioms (6) and (7) describe a simple binary sensor: if *empty* is true then executing *sense*$_1$ means that *Kempty* will also become true (and *K¬empty* becomes false), i.e., the agent will know that the box is empty. Similarly, if *¬empty* is true then *K¬empty* becomes true (and *Kempty* becomes false) after executing *sense*$_1$. In other words, after executing *sense*$_1$ the agent will *know whether* the box is empty or not.

Axioms (8) and (9) encode a universal effect for *sense*$_2$: the agent comes to know whether *inBox*(x) is true or not for every instantiation of x. Thus, the sensors provide a form of *local* closed world information: the agent has complete information about the objects in the box, but other aspects of the agent's knowledge may still be incomplete.

Axioms (10) and (11) encode a conditional sensing effect for *sense*$_2$: the agent comes to know whether *fragile*(x) is true or not for those instantiations of x for which *inBox*(x) is true. Thus, it senses whether or not the objects in the box are fragile. □

4.2 Knowledge-level effects of physical actions

Unlike sensing actions, ordinary "physical" actions change the state of the world by altering ordinary fluents. If an action's effects are known to the agent then changes to ordinary fluents must also bring about changes to the corresponding knowledge fluents. To capture such effects in our LDEC axioms we will generate the knowledge-level effects of physical actions directly from the ordinary effect axioms, and augment these axioms with the new effects.

Transformation 2 For each ordinary LDEC effect axiom in a domain, add the following conjuncts to the consequent of the linear implication:

1. For each C_i in the antecedent, add $K\neg C_i$ and $\neg KC_i$,
2. For each D_i in the consequent, add $K\neg D_i$ and $\neg KD_i$,
3. For each F_i in the consequent, add KF_i and $\neg K\neg F_i$.

The intuition behind this transformation is that an agent will come to know the effects of a physical action when it is applied. Thus, if an action makes F true, we must also ensure that KF is made true and $K\neg F$ is made false. Similarly, if an action makes F false, we must make $K\neg F$ true and KF false. The end result is a new effect axiom with both world-level and knowledge-level effects grouped together.

Example 3 Applying Transformation 2 to axiom (3) of Table 1 produces the following new effect axiom:

$$\begin{aligned}&\{\mathit{affords}(\mathit{pickup}(x))\} \wedge \mathit{handempty} \wedge \mathit{onTable}(x) \multimap \\ &\quad [\mathit{pickup}(x)]\mathit{holding}(x) \wedge K\neg\mathit{handempty} \wedge \\ &\quad \neg K\mathit{handempty} \wedge K\neg\mathit{onTable}(x) \wedge \neg K\mathit{onTable}(x) \wedge \\ &\quad K\mathit{holding}(x) \wedge \neg K\neg\mathit{holding}(x).\end{aligned}$$

After executing *pickup*(x) the agent will be holding x, its hand will not be empty, and x will no longer be on the table. The agent will also *know* this information to be true. □

4.3 Knowledge-reducing actions

The final type of action we briefly mention is the dual of knowledge-producing actions, namely *knowledge-reducing* actions. As with sensing, these actions only affect knowledge fluents but, unlike sensing, they produce a knowledge loss effect by making pairs of knowledge fluents *false*. E.g.,

$$\{\mathit{affords}(\mathit{forget})\} \multimap [\mathit{forget}]\neg KF \wedge \neg K\neg F,$$

describes an action, *forget*, that causes knowledge of F to become "unknown" by making both KF and $K\neg F$ false. F itself is unchanged. We do not discuss these actions in detail here, however, we encode such effects as above, by appending pairs of negated knowledge fluents as conjuncts to the consequent of our transformed LDEC effect axioms.

Action	Preconditions	Effects
$\textit{pickup}(x)$	$K(\textit{handempty})$ $K(\textit{onTable}(x))$	$\textit{add}(K_f, \textit{holding}(x))$ $\textit{add}(K_f, \neg\textit{handempty})$ $\textit{add}(K_f, \neg\textit{onTable}(x))$
$\textit{sense}_2$		$\textit{add}(K_w, \textit{inBox}(x))$ $\textit{add}(K_w, \textit{inBox}(x) \wedge \textit{fragile}(x))$
forget		$\textit{del}(K_f, F), \textit{del}(K_f, \neg F)$

Table 3: PKS actions produced by Transformation 3

5 LDEC and knowledge-level planning

Knowledge-extended LDEC domains present some problems for planning, since LDEC axioms can include references to knowledge-level *and* world-level fluents. Working with both types of fluents is difficult for "offline" planners, since such planners have no direct access to the world state, but must consult their internal representations, which may not be complete. This is particularly problematic for sensing actions due to their context-dependent nature: the state of the world-level fluent being sensed determines the resulting state of the corresponding knowledge fluents.

Our solution is to construct action operators that do not directly access the world state, by extracting the "knowledge-level" effects from the LDEC axioms. To do this we make use of the knowledge-level representation provided by the planner PKS. The resulting operators we construct will be similar to STRIPS operators and will support a natural forward-chaining approach to planning.

PKS (Planning with Knowledge and Sensing) is a contingent planner that constructs plans in the presence of incomplete information and sensing [12]. PKS generalizes the STRIPS representation by using a set of databases to represent the agent's knowledge, instead of a single database to represent the state of the world. Each database models a particular type of knowledge (that can be formally understood in terms of formulae of a first-order modal logic of knowledge). We focus on two of these databases here: K_f and K_w.

$\mathbf{K_f}$: This database is like the STRIPS database $\mathcal{D}$, except it can contain both positive and negative facts, and the closed world assumption is not applied. K_f typically models the effects of physical actions and can include any ground fluent literal L, where $L \in K_f$ means "the agent knows L."

$\mathbf{K_w}$: This database represents the plan-time effects of sensing actions. If an action senses a fluent F, then at plan time all the agent knows is that *after* it has executed the action it will know F or know $\neg F$. At plan time, the agent cannot resolve this disjunction and the actual value of F remains unknown. Hence, $\phi \in K_w$ means that the agent "knows whether ϕ," that is, it either knows ϕ or knows $\neg\phi$.

PKS also provides a primitive query language for answering questions about what it does, or does not, know or "know whether." We will only use one query, $K(\phi)$: is ϕ known to be true? An inference algorithm evaluates queries by checking database contents, taking into consideration the interaction between different types of knowledge. PKS uses primitive queries to represent action preconditions and goals.

PKS action effects are modelled as updates to the agent's knowledge state, rather than the world state, using sets of STRIPS-style "add" and "delete" operations that modify the contents of the databases. E.g., $\textit{add}(K_f, \neg\phi)$ would add $\neg\phi$ to K_f, and $\textit{del}(K_w, \phi)$ would remove ϕ from K_w.

Using these databases, queries, and update operations, we can transform our knowledge-extended LDEC domains into PKS action operators, by compiling action preconditions into K queries, the effects of physical actions into K_f updates, and the effects of sensing actions into K_w updates.

Transformation 3 For each LDEC action α, a corresponding PKS operator is defined as follows:

1. For each knowledge fluent KL (similarly, $\neg KL$) in the precondition axiom for α, add a PKS precondition: $K(L)$ (similarly, $\neg K(L)$),

2. If α is a physical action: for each knowledge fluent KL (similarly, $\neg KL$) in the transformed effect axiom for α, add a PKS effect: $\mathit{add}(K_f, L)$ (similarly, $\mathit{del}(K_f, L)$),

3. If α is a sensing action: for each pair i of effect axioms for α, add the PKS effect: $\mathit{add}(K_w, \phi_i \wedge F_i)$.

To ensure Transformation 3 is sound, we require LDEC precondition axioms that only mention knowledge fluents, since PKS action preconditions are formed from knowledge-level queries like $K(\phi)$. We also require PKS's inference algorithm to be complete with respect to evaluating such queries. Although we have yet to establish this second requirement in general, it has not posed a problem for us in practice.

Table 3 shows the application of Transformation 3 to some of the actions we defined earlier. The effects of $\mathit{pickup}(x)$ are modelled as updates to K_f alone, since it is a physical action. (We note that PKS lets us combine the addition of a fluent and the deletion of its negation into a single *add* operation, which is reflected in Table 3.) The sensing action sense_2, on the other hand, is modelled by updating K_w. In this case, the effects encoded by pairs of LDEC axioms are combined into single *add* updates to K_w. Finally, the *forget* action is modelled by a pair of updates that remove all information about F from K_f.

Modelling actions as database updates lets PKS employ an efficient, forward-chaining approach to finding plans at the knowledge level, by progressing databases in a STRIPS-like manner [12]. Physical actions update PKS's knowledge of world-level fluents. For instance, applying the action $\mathit{pickup}(o1)$ to $K_f = \{\mathit{handempty}, \mathit{onTable}(o1)\}$ results in $K_f = \{\neg \mathit{handempty}, \neg \mathit{onTable}(o1), \mathit{holding}(o1)\}$. Sensing actions update PKS's know-whether knowledge, which plays an important role in building contingent plans. For instance, sense_2 adds $\mathit{inBox}(x)$ to K_w which lets PKS build plan "branches" for the two outcomes of this information, given any instantiation c of x: along one branch $\mathit{inBox}(c)$ is assumed to be true, while along the another branch $\neg \mathit{inBox}(c)$ is assumed to be true. Planning can then continue along each branch, using this new knowledge.

Our ability to transform the extended LDEC axioms into PKS operators provides us with a way of planning with our LDEC domains. Moreover, we preserve a STRIPS-style "state-update" interpretation for LDEC, and the forward-chaining model of planning this formalism induces.

6 Discussion and conclusions

This work forms part of a larger project called Paco-Plus that explores agent perception, action, and cognition using *object-action complexes*, or instantiated "fragments" of the state-action transition space.[3] As part of this work we are interested in learning high-level, symbolic representations that are abstracted away from a low-level robot/vision system. LDEC fits into this picture as a formal language for encoding the domain axioms that arise in this setting. Moreover, we see LDEC axioms as the kinds of rules we can realistically learn, using existing machine learning techniques, from a robot system that has only primitive actions available to it for observing and acting in the world [5].

The LDEC extensions we describe here are just a first step. We would also like to use LDEC for modelling indexical knowledge, and extend the planner to construct plans with loops. We also hope to leverage existing insights on knowledge-level planning, in particular, recent work based on [13] that investigates the relationship between knowledge fluent theories and standard possible world representations of knowledge. We have already extended LDEC to support more complex ADL-style actions with context-dependent effects [11], but have not discussed this addition here. (Such effects aren't required for our initial learning tasks.)

Finally, we have implemented the transformations described in this paper for compiling LDEC axioms into PKS operators, and are exploring the feasibility of using PKS in a robotics setting for high-level planning. (The examples in this paper are motivated by a proposed test domain.) Preliminary results using offline simulations look promising, and we remain positive about the utility of LDEC and PKS.

[3]See `www.paco-plus.org` for a description of this project.

References

[1] P. Bertoli, A. Cimatti, M. Roveri, and P. Traverso. Planning in nondeterministic domains under partial observability via symbolic model checking. In *Proc. of IJCAI-2001*, pages 473–478, 2001.

[2] W. Bibel, L. F. del Cerro, and A. Herzig. Plan generation by linear proofs: on semantics. In *German Workshop on Artificial Intelligence (GWAI'89)*, volume 216 of Informatik-Fachberichte, Berlin, 1989. Springer Verlag.

[3] Robert Demolombe and Maria del Pilar Pozos Parra. A simple and tractable extension of situation calculus to epistemic logic. In *Proc. of ISMIS-2000*, pages 515–524, 2000.

[4] Richard E. Fikes and Nils J. Nilsson. STRIPS: A new approach to the application of theorem proving to problem solving. *Artificial Intelligence*, 2:189–208, 1971.

[5] Christopher Geib, Kira Mourão, Ron Petrick, Nico Pugeault, Mark Steedman, Norbert Krueger, and Florentin Wörgötter. Object action complexes as an interface for planning and robot control. In *Proc. of the Humanoids-06 Workshop - Toward Cognitive Humanoid Robots*, 2006.

[6] J.-Y. Girard. Linear logic. *Theoretical Computer Science*, 50:1–102, 1987.

[7] D. Harel. Dynamic logic. In D. Gabbay and F. Guenthner, editors, *Handbook of Philosophical Logic, volume II*, pages 497–604. Reidel, Dordrecht, 1984.

[8] R. Kowalski and M. Sergot. A logic-based calculus of events. *New Generation Computing*, 4:67–95, 1986.

[9] John McCarthy and Patrick J. Hayes. Some philosophical problems from the standpoint of artificial intelligence. *Machine Intelligence*, 4:463–502, 1969.

[10] Robert C. Moore. A formal theory of knowledge and action. In Jerry R. Hobbs and Robert C. Moore, editors, *Formal Theories of the Commonsense World*, pages 319–358. Ablex Publishing, 1985.

[11] Edwin P. D. Pednault. ADL: Exploring the middle ground between STRIPS and the situation calculus. In *Proc. of KR-89*, pages 324–332. Morgan Kaufmann Publishers, 1989.

[12] Ronald P. A. Petrick and Fahiem Bacchus. A knowledge-based approach to planning with incomplete information and sensing. In *Proc. of AIPS-2002*, pages 212–221. AAAI Press, 2002.

[13] Ronald P. A. Petrick and Hector J. Levesque. Knowledge equivalence in combined action theories. In *Proc. of KR-2002*, pages 303–314. Morgan Kaufmann Publishers, 2002.

[14] Richard B. Scherl and Hector J. Levesque. Knowledge, action, and the frame problem. *Artificial Intelligence*, 144(1–2):1–39, 2003.

[15] Tran Cao Son and Chitta Baral. Formalizing sensing actions – a transition function based approach. *Artificial Intelligence*, 125(1–2):19–91, 2001.

[16] Mikhail Soutchanski. A correspondence between two different solutions to the projection task with sensing. *Commonsense-01*, 2001.

[17] Mark Steedman. Temporality. In Johan van Benthem and Alice ter Meulen, editors, *Handbook of Logic and Language*, pages 895–938. North Holland/Elsevier, Amsterdam, 1997.

[18] Mark Steedman. Plans, affordances, and combinatory grammar. *Linguistics and Philosophy*, 25:723–753, 2002.

[19] Daniel S. Weld, Corin R. Anderson, and David E. Smith. Extending graphplan to handle uncertainty & sensing actions. In *Proc. of AAAI-98*, pages 897–904. AAAI Press, 1998.

Appendix D

On Natural Language Processing and Plan Recognition

Christopher W. Geib and Mark Steedman

School of Informatics
University of Edinburgh
Edinburgh EH8 9LW, Scotland
cgeib@inf.ed.ac.uk

Abstract

The research areas of plan recognition and natural language parsing share many common features and even algorithms. However, the dialog between these two disciplines has not been effective. Specifically, significant recent results in parsing mildly context sensitive grammars have not been leveraged in the state of the art plan recognition systems. This paper will outline the relations between natural language processing(NLP) and plan recognition(PR), argue that each of them can effectively inform the other, and then focus on key recent research results in NLP and argue for their applicability to PR.

1 Introduction

Without performing a careful literature search one could easily imagine that the fields of *Plan Recognition*(PR) and *Natural Language Processing*(NLP) are two separate fields that have little in common. There are few papers in either discipline that directly cite work done in the other. While there are exceptions,[7, 4, 21, 24], even these papers often are only citing NLP in passing and not making use of recent research results.

Interestingly, many researchers do see these two areas as very related, but are still not taking the recent lessons learned in one area and applying them to the other. In an effort to rectify this lack, this paper will outline the commonalities between PR and NLP, argue why the results from each of these research areas should be used to inform the other, and then outline some recent research results that could inform a unified view of these two tasks.

2 Commonalities

In this section we will sketch the similarities at the surface and algorithmic levels between PR and NLP before more formally drawing their representations together in the Section 3. We will start this process by laying out some terminology so that we can see the common parts of NLP and PR.

Both PR and NLP take as input a set of *observations*. In PR these are observations of action executions and in NLP these are individual words or utterances. In both cases, the observations are used to create a higher level structure. In NLP these higher level structures may be parse trees [9] or logical forms [5]. In PR they are usually a hierarchical plan structure[18, 17, 10] or at least a high level root goal[14]. In either case, both NLP and PR construct a higher level knowledge structure that relates the meanings of each of the individual observations to a meaning for the collection of observations as a whole.

For the purposes of this discussion it will aid us to abstract away from the specific details of the higher level structure that is built by this process. To simplify this discussion we will talk about these systems as if they were creating an hierarchical data structure that captures the meaning of the collection of observations. We will use the PR terminology and call this structure an *explanation* and following the NLP terminology call the process of producing a single explanation *parsing*.

In order to parse a set of observations into an explanation both PR and NLP must specify the patterns of observations they are willing to accept or the rules that govern how the observations can be combined. In PR this specification is done in the form of a library of plans, while in NLP this is done through a grammar. In Section 3 we will argue that there is no significant distinction between PR plan libraries and NLP grammars. Therefore, in this paper we will call all such specifications of the rules for acceptable combination of observations *grammars*.

With this terminology in place, we can now describe both NLP and PR as taking in as inputs a set of observations and a grammar specifying the acceptable sets of observations. Both NLP and PR then parse these observations to produce explanations that organize the observations into a structured representation of the meaning of the collection.

Given this level similarity, it is not surprising that grammars in both NLP and PR can result in multiple explanations for a given set of observations. However, it is of interest that in both disciplines this ambiguity has been resolved using very similar probabilistic methods. In both areas, the state of the art methods are based on weighted model counting. These systems build the set of possible explanations and establish a probability distribution over the set in order to determine the most likely explanation.

The work in NLP often uses probability models derived from an annotated corpus of text[8] while the probability models from PR have been based on Markov models of the world dynamics [6] or probabilistic models of plan execution [10]. While space prohibits a full exposition of these very different probability models, it is still telling that a weighted model counting method is the state of the art in both fields.

Beyond these surface and algorithmic similarities there are psycholinguistic reasons for believing that PR and NLP are very closely tied process that should inform one another. For example, consider indirect speech acts like asking someone “Do you know what time it is?” To correctly understand and respond to this question requires both NLP and PR.

Correctly responding requires not merely parsing the sentence to understand that it is a request about ones ability to provide a piece of information. It also requires recognizing that asking the question of someone else is the first step in a two part plan for finding out a piece of information by asking someone else. PR allows one to conclude that if someone is following this plan they most likely have the goal of knowing the piece of information (the current time in this case) and that providing the desired information will be more helpful than answering the literal question asked.

Given the similarities between the two areas, it seems reasonable that work in one area should inform the other. However important results in each area are not being leveraged in the other community. In the next section we will more formally specify the relation between these two areas to help researchers take advantage of the results in both areas.

3 Plans as Grammars

Our argument that PR and NLP should inform one another would be significantly strengthened if we could show, as we have asserted above, that the plan libraries used by PR systems are equivalent to the grammars used by NLP systems. In the following section we will show the parallels between these two constructs and a mapping between them.

Almost all PR work has been done on traditional hierarchical plans.[1] While much of the work in plan recognition has not provided formal specifications for their plan representations they can all generally be seen as special cases of Hierarchical Task Networks (HTN) as defined in [11].

According to Ghallab the actions of an HTN domain are defined as either *operators* or *methods*. An operator corresponds to an action that can be executed in the world. Following Ghallab we will define them as a triple $(n, add-list, delete-list)$ where n is the name of the operator, $add-list$ is a list

[1]See [6] for an exception that works on hierarchical Markov models

of predicates that are made true or added to the world by the operator, and $delete - list$ is the set of predicates that are made false or deleted from the world by the operator.

A method on the other hand represents a higher level action and is represented as a 4-tuple $(name, T, \{st_0, ..., st_n\}, C)$ such that $name$ is a unique identifier for the method, T names the higher level action this method decomposes, and $\{st_0, ..., st_n\}$ identifies the set of sub-tasks that must be performed for the higher level task to be performed. Finally, C represents a set of ordering constraints that have to hold between the subtasks for the method to be effective.

We will draw a parallel between HTNs and context free grammars (CFGs). Following Aho and Ullman[1] we define a CFG, G, as a 4-tuple $G = (N, \Sigma, P, S)$ where

- N is a finite set of *nonterminal symbols*,
- Σ is a finite set of *terminal symbols* disjoint from N,
- P is a set of *production rules* that have the form $n \rightarrow \omega$ where $n \in N$ and $\omega \in (\Sigma \cup N)^*$, and
- S is a distinguished $S \in N$ that is the start symbol.

Given these definitions, we would like to map the plans represented as an HTN into an equivalent CFG. We first consider the case of a collection of HTN plans that are totally ordered. That is, we assume that for every method definition the constraints on the subtasks $st_0, ..., st_n$ define a total ordering over the subtasks. Without loss of generality, we assume that the subtasks' subscripts represent this ordering.

To encode the HTN as a CFG, we first consider the operators. The processing for these is quite simple. We identify the names of each operator as a terminal symbols in our new grammar, and attach the add and delete lists to the non-terminal as features. Next we consider mapping the method definitions into productions within the grammar.

Given a totally ordered method definition, we can add the task to be decomposed to the set of non-terminal symbols. Then we define a new production rule with this task its left hand side. We then define the right hand side of the rule as the ordered set of subtasks. Thus, the method definition $(name, T, \{st_0, ..., st_n\}, C)$ is rewritten as the CFG production rule: $T \rightarrow st_0, ..., st_n$ and T is added to the set of non-terminals.

For example, consider the very simple HTN method, $m1$ for acquiring shoes:

$$\begin{aligned}(m1, \quad & acquire(shoes),\\ & \{goto(store), choose(shoes), buy(shoes)\},\\ & \{(1 \prec 2), (2 \prec 3)\})\end{aligned}$$

where the constraints $(1 \prec 2)$ indicates the task $goto(store)$ must precede the task $choose(shoes)$ and $(2 \prec 3)$ indicates that $choose(shoes)$ must precede $buy(shoes)$. This is very easily captured with the CFG production:

$$\begin{aligned}& acquire(shoes) \rightarrow\\ & \quad goto(store), choose(shoes), buy(shoes)\end{aligned}$$

This process of converting each method definition into a production rule and adding the task to be decomposed to the set of non-terminals is repeated for every method in the HTN to produce the CFG for the plans. Now we turn to the question of partial ordering.

Limited cases of partial orderness could be handled in CFGs by expanding the grammar with a production rules for each possible ordering. However, as the NLP community has realized this can result in an unacceptable increase in the size of the grammar, and the related runtime of the parsing algorithm[3].

So instead, to address this, the NLP community has produced a number of different grammar formalisms that allow the grammar to separately express decomposition and ordering. This includes the work of Shieber on ID/LP grammars [22], Nederhof on poms-CFGs [20], and Hoffman[13] and Baldridge [2] on partial orderness in Combinatory Catagorial Grammars. All of of these are attempts to include partial orderness within the grammar formalism (and parsing mechanism) without the exponential increase in the grammar size and runtime. Since each of these formalisms use very different representations, rather than presenting examples, we refer the reader to the cited papers. It suffices to say that these grammar formalisms introduce notational additions to denote partial orderness within the production rules and to explicitly specify the ordering relations that are required in each production. These formalisms can be used to capture HTN plan domains that require partial ordering.

It should be clear from this exposition that the grammar formalisms found in the NLP literature are sufficient to cover the method definitions found in most if not all of the PR literature. However, to the best of our knowledge no one has used any of the relatively recent grammar formalisms and their associated parsing machinery for plan recognition. Making use of these grammatical formalisms would also allow the use of their associated formal complexity results as well, something that has often been lacking in the work in PR.

Thus, we propose that NLP and PR could be unified by the use of the same underlying grammatical formalisms for representing the constraints on observations, and using a common parsing mechanism. In the case of probabilistic NLP and PR systems, we believe these systems may need to retaining separate methods for computing their probability distributions, however the parsing of observations into explanations could share a common framework. In the next section we will advocate a specific class of grammars for this task.

4 New Grammar Formalisms for PR

Given that researchers in NLP have been working on the close relationship between grammars, parsers, and language expressiveness it shouldn't be a surprise that results from this work could inform the work in PR. There are some classes of grammars that are too computationally expensive to parse for real world application. For example, the well known complexity results for parsing context sensitive grammars (CSGs) have all but ruled them out for NLP work. Likewise we expect poor performance for applications of CSGs to PR. Unfortunately, PR researchers have used these results as a motivation to build their own algorithms for parsing, often without even considering the limitations of the existing parsing algorithms. Examples include graph covering[18] and Bayes nets[6], that trade one np-hard problem for another. What has been largely ignored by the PR community is the NLP work in extending context free grammars and their efficient parsing algorithms.

Recent work in NLP has expanded the language hierarchy with grammars that have a complexity that falls between context free and context sensitive. Examples, include IL/LP Grammars[22], Tree Adjunction Grammars(TAG)[15], and Combinatory Catagorial Grammars(CCG)[23, 12, 8]. These "mildly context sensitive grammars"(MCSGs) have a number of properties that make them attractive for NLP including greater expressiveness than CFGs but still having polynomial algorithms for parsing. These properties also make them attractive for adoption by the PR community.

While these grammars are of scientific interest, we should justify their use, since it is not clear that PR requires grammars that are more expressive than CFGs. Such a claim would rest on the empirical need for plans that are not context free. If nothing more than a CFG is needed for PR, then a well known parsing algorithm like CKY that has a cubic complexity seems to be the obvious choices for application to PR. However, if there are PR problems that require recognizing plans that are not within the class of CFG plans, this would provide a convincing argument that PR requires a grammar that is not context free. In the following we will provide just such an example. While there are a number of different classes of MCSGs with different expressiveness results, and the exploration of all of them may prove useful for PR research, we will focus on the subclass of MCSGs that includes CCGs and TAGs called Linear Index Grammars(LIG).

Steedman[23] has argued convincingly that CCGs and other LIGs are able to capture phenomena beyond CFGs that are essential to real world language use. Given the parallels we have already demonstrated between NLP and PR, we argue that if this class is necessary for NLP it shouldn't be surprising to us if this class of grammars captured essential phenomena in PR as well. In this light, Steedman shows that CCGs provide for *crossing dependencies* in NLP, a critical extension that context free grammars cannot capture. Likewise, if we find that such crossing dependencies are necessary for recognizing plans we would have a strong argument that PR requires a grammar that is in the MCSG family.

While a full discussion of the handling of crossing dependencies in CCGs is beyond the scope of this paper, it will be helpful to understand their basic structure in order to identify them in PR contexts. Crossing dependencies occur when the words that make up a constituent (like a relative clause) are interleaved in the sentence with the elements of a different constituent. Steedman[23] has argued that a particularly strong example of the naturalness of these constructs are Dutch verbs like *proberen* 'to try' which allow a number of scrambled word orders that that are outside of the expressiveness of CFGs.

For example the translation of the phrase "... because I try to teach Jan to sing the song." has four possible acceptable orderings [1 - 4] and a fifth that is more questionable.

1. . . . omdat ik$_1$ Jan$_2$ het lied$_3$ probeer$_1$ te leren$_2$ zingen$_3$.
   . . . because I Jan the song try to teach to sing.
2. . . . omdat ik$_1$ probeer$_1$ Jan$_2$ het lied$_3$ te leren$_2$ zingen$_3$.
3. . . . omdat ik$_1$ probeer$_1$ Jan$_2$ te leren$_2$ het lied$_3$ te zingen$_3$.
4. . . . omdat ik$_1$ Jan$_2$ probeer$_1$ te leren$_2$ het lied$_3$ te zingen$_3$.
5. ?. . . omdat ik$_1$ Jan probeer$_1$ het lied te leren zingen

The subscripts are included to show the correspondence of the noun phrases to the verbs. For example in the first ordering the noun phrases are all introduced first followed by their verbs in the same order as their nouns. This produces the maximally crossed ordering for this sentence.

The realization of these kinds of crossed dependencies in a PR context is relatively straightforward. Its important to keep in mind the mapping that we are using between traditional language grammars and planning grammars will mean that dependencies in PR are not the same as in NLP. In NLP dependencies are features like gender, number or tense that must agree between different words within the sentence. In the PR context, dependencies are equivalent to causal links in traditional nonlinear planning[19]. That is, they are states of the world that are produced by one action and consumed by another. Therefore, a plan with a crossing dependency would have the causal structure shown in Figure 1 where in $act1$ is found to produce the preconditions for actions $act2$ and $act3$ which each produce a precondition for $act4$. Such a structure requires that two different conditions be created and preserved across two different actions for their use. Note that while the actions are only partially ordered, there is no linearizion of them that will remove the crossing dependency. That is, $act2$ and $act3$ can be reordered but this will not remove the crossing dependency.

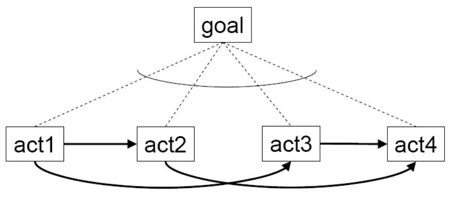

Figure 1: An abstract plan with a crossed dependency structure

The argument for the necessity of MCSGs for planning rests on real world examples of plans with this structure. Being able to describe what such a plan looks like is not compelling if they never occur in PR problem domains. Fortunately, examples of plans with this structure are relatively common. Consider recognizing the activities of a bank robber that has both his gun and ski-mask in a duffel bag and his goal is to rob a bank. He must open the bag, put on the mask and pick up the gun and enter the bank. Figure 2 shows this plan. This plan has exactly the same crossed dependency structure shown in Figure 1.

Note, that we could make this plan much more complex with out effecting the result. Actions could be added before opening the bag, after entering the bank, and even between putting on the ski-mask and picking up the gun so long as the critical causal links are not violated. The presence of plans with this

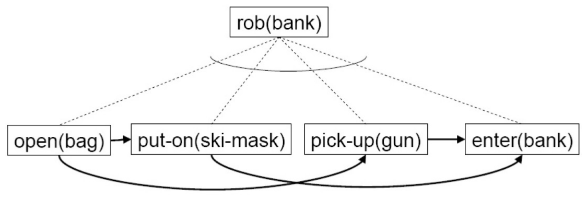

Figure 2: An example plan with crossing dependency structure

structure and our desire to recognize such plans gives us a strong reason to look at the grammars that fall this class as a grammatical formalism for PR.

4.1 Why MCSGs?

Joshi[16] first formally defined the class of MCSGs as those grammars that share four properties that are relevant for NLP:

- The class of languages included covers all context free languages.
- The languages in the class are polynomially parsable.
- The languages in the class only capture certain types of dependencies including nested (non-crossing) and crossed dependencies.
- The languages in the class have the *constant growth property* which requires that if all of the sentences in the language are sorted according to their length then any two consecutive sentences do not differ in their length by more than a constant factor determined by the grammar.

This set of properties are also relevant for defining the class of grammars that would work well for PR. We will argue for each of them in order.

First, we have just demonstrated the need for grammars that are more than context free for PR. Second, clearly polynomial parsing is desirable for PR. In order to use these algorithms in real world applications they will need to be extended to consider multiple possible interleaved goals and to handle partially observable domains[10]. If a single goal can't be parsed in polynomial time what hope do we have for efficient algorithms for the needed extensions? Further, PR is needed in a great many applications that will not tolerate algorithms with greater complexity. For example, assistive systems are not useful if their advice comes to late.

Third, Plans do have structure that is captured in dependency structures. Therefore, it seems natural to try to restrict the grammar for plans to the kinds of dependency structures that are actually used. Whether or not the dependency restrictions that are consistent with NLP are the same set for PR is largely an empirical question. We have already seen evidence of crossing dependencies that required us to abandon CFGs in favor of MCSG's. While nested and crossing dependencies in the abstract can cover all the kinds of dependencies needed in planning, different MCSGs place different restrictions on the allowable depth of crossings and the number of nesting. This will have a significant impact on the expressiveness of a particular MCSG and its applicability to PR.

Fourth and finally, the requirement of the constant growth rate may be the hardest to understand. Intuitively in the PR domain this means that if there is a plan of length n then there is another plan of length at most n+K where K is a constant for the specific domain. For example this rules out that the length of the next plan is a function of the length of the previous plan or some other external feature. Note this says nothing about the goals that are achieved by the plans. The plan of length n and length n+K may achieve very different goals but they are both acceptable plans within the grammar. This speaks to the intuition that given a plan one should be able to add a small fixed number of actions to the plan and get another plan. Again this seems to be the kind of property one expects to see in a PR domain and therefore in a plan grammar.

Now, while we believe we have made a strong argument for the use of MCSGs for PR, this is not the final word on the question. While we have presented an argument that we need at least the expressiveness of LIGs, it may be the case that still more powerful grammar formalisms are needed. The most promising method for proving such a result would require finding plans with dependency structures that are not in MCSG that our PR systems need to recognize. Thus, determining if MCSGs are sufficient for PR is an open research question for the community.

While there are well known languages that are not in MCSG it is difficult to see their relevance to planning domains. For example the language $\{a^{2^n}\}$, that is the language where in the length of any sentence of the language is a power of two, is not MCSG as it fails the constant growth requirement. It is possible to imagine contrived examples where this would be relevant for PR (Perhaps as part of some kind of athletic training regime, we want to recognize cases where in someone has run around the track a number of times that is a power of two.) However, this certainly seems anomalous and most likely should be dealt with by reasoning that falls outside of the grammar, like a counter and a simple test.

5 Conclusions

There are close ties between the process of natural language processing and plan recognition. This relation should allow these two processes to inform each other and allow the transfer of research results from one area to the other. However, much recent work in both fields has gone unnoticed by researchers in the other field. This paper begins the process of sharing these results, describing the isomorphism between the grammatical formalisms from NLP and plan representations for PR, arguing that like NLP, PR will require a grammatical formalism in the mildly context sensitive family, and finally that NLP and PR can form a common underlying task that can usefully be explored together.

Acknowledgments

The work described in this paper was conducted within the EU Cognitive Systems project PACO-PLUS (FP6-2004-IST-4-027657) funded by the European Commission.

References

[1] Alfred V. Aho and Jeffrey D. Ullman. *Foundations of Computer Science*. W.H. Freeman/Computer Science Press, New York, NY, 1992.

[2] Jason Baldridge. *Lexically Specified Derivational Control in Combinatory Categorial Grammar*. PhD thesis, University of Edinburgh, 2002.

[3] G. Edward Barton. On the complexity of id/lp parsing. *Computational Linguistics*, 11(4):205–218, 1985.

[4] Nate Blaylock and James Allen. Corpus-based statistical goal recognition. In *Proceedings of the 18th International Joint Conference on Artificial Intelligence*, pages 1303–1308, 2003.

[5] Johan Bos, Stephen Clark, Mark Steedman, James Curran, and Julia Hockenmaier. Wide-coverage semantic representations from a ccg parser. In *Proceedings of the 20th International Conference on Computational Linguistics (COLING '04)*, 2004.

[6] Hung H. Bui, Svetha Venkatesh, and Geoff West. Policy recognition in the abstract hidden markov model. In *Technical Report 4/2000 School of Computer Science, Curtin University of Technology*, 2002.

[7] Sandra Carberry. *Plan Recognition in Natural Language Dialogue*. ACL-MIT Press Series in Natural Language Processing. MIT Press, 1990.

[8] Stephen Clark and James Curran. Parsing the wsj using ccg and log-linear models. In *Proceedings of the 42nd Annual Meeting of the Association for Computational Linguistics (ACL-04)*, 2004.

[9] Michael Collins. Three generative, lexicalized models for statistical parsing. In *ACL '97: Proceedings of the 35th Annual Meeting of the Association for Computational Linguistics*, 1997.

[10] Christopher W. Geib and Robert P. Goldman. Recognizing plan/goal abandonment. In *Proceedings of IJCAI 2003*, 2003.

[11] Malik Ghallab, Dana Nau, and Paolo Traverso. *Automated Planning: Theory and Practice*. Morgan Kaufmann, 2004.

[12] Julia Hockenmaier. *Data and Models for Statistical Parsing with Combinatory Catagorial Grammar*. PhD thesis, University of Edinburgh, 2003.

[13] Beryl Hoffman. Integrating 'free' word order syntax and information structure. In *Proceedings of the 1995 Conference of the European Chapter of Association for Computational Linguistics*, pages 245–252, 1995.

[14] Eric Horvitz, Jack Breese, David Heckerman, David Hovel, and Koos Rommelse. The lumiere project: Bayesian user modeling for inferring the goals and needs of software users. In *Proceedings of the 14th Conference on Uncertainty in Artificial Intelligence*, 1998.

[15] A. Joshi and Y. Schabes. Tree-adjoining grammars. In *Handbook of Formal Languages, Vol. 3*, pages 69–124. Springer Verlag, 1997.

[16] Aravind Joshi. How much context-sensitivity is necessary for characterizing structural descriptions - tree adjoining grammars. In *Natural Language Processing - Theoretical, Computational, and Psychological Perspective*, pages 206–250. Cambridge University Press, 1985.

[17] G. Kaminka, D.V. Pynadath, and M. Tambe. Monitoring deployed agent teams. In *Proceedings of the International Conference on Autonomous Agents*, pages 308–315, 2001.

[18] Henry Kautz and James F. Allen. Generalized plan recognition. In *Proceedings of the Conference of the American Association of Artificial Intelligence 1986*, pages 32–38, 1986.

[19] David McAllester and David Rosenblitt. Systematic nonlinear planning. In *Proceedings of the Conference of the American Association of Artificial Intelligence (1991)*, pages 634–639, 1991.

[20] Mark-Jan Nederhof, Giorgio Satta, and Stuart M. Shieber. Partially ordered multiset context-free grammars and ID/LP parsing. In *Proceedings of the Eighth International Workshop on Parsing Technologies*, pages 171–182, Nancy, France, April 2003.

[21] David Pynadath and Michael Wellman. Probabilistic state-dependent grammars for plan recognition. In *Proceedings of the Conference on Uncertainty in Artificial Intelligence (UAI-'00)*, pages 507–514, 2000.

[22] Stuart M. Shieber. Direct parsing of ID/LP grammars. *Linguistics and Philosophy*, 7(2):135–154, 1984.

[23] Mark Steedman. *The Syntactic Process*. MIT Press, 2000.

[24] Marc Vilain. Deduction as parsing. In *Proceedings of the Conference of the American Association of Artificial Intelligence (1991)*, pages 464–470, 1991.