AUTOMATED EXTRACTION OF ACTION SEMANTICS FOR EMBODIED VIRTUAL AGENTS USING TEXTUAL KNOWLEDGE BASES

by

J. Timothy Balint

A Dissertation
Submitted to the
Graduate Faculty
of
George Mason University
In Partial fulfillment of
The Requirements for the Degree
of
Doctor of Philosophy
Computer Science

Committee:

_____	Dr. Jan M Allbeck, Dissertation Director
_____	Dr. Dana Richards, Committee Member
_____	Dr. Michael Hieb, Committee Member
_____	Prof. Seth Hudson, Committee Member
_____	Dr. Yotam Gingold, Committee Member
_____	Dr. Sanjeev Setia, Department Chair
_____	Dr. Kenneth S. Ball, Dean, The Volgenau School of Engineering

Date: _____ Summer Semester 2017
George Mason University
Fairfax, VA

Automated Extraction of Action Semantics for Embodied Virtual Agent using Textual Knowledge Bases

A dissertation submitted in partial fulfillment of the requirements for the degree of
Doctor of Philosophy at George Mason University

By

J. Timothy Balint
Master of Science
George Mason University, 2014
Bachelor of Science
Roanoke College, 2011

Director: Dr. Jan M Allbeck, Professor
Department of Computer Science

Summer Semester 2017
George Mason University
Fairfax, VA

Dedication

I dedicate this dissertation to my father.
I dedicate this dissertation to my family.

Acknowledgments

I would like to thank my advisor and Dissertation Chair, Dr. Jan M. Allbeck, for instructing me on how to research and molding me into the researcher I am today.
I would like to thank my committee members, Dr. Dana Richards, Dr. Micheal Hieb, Seth Hudson and Dr. Yotam Gingold for their continued support and insight for this work.
I would like to acknowledge the undergraduate researchers who assisted in this work: Cameron Pelkey, Shibble Duman and Nasrin Noor Ahmad. There support with data collection proved invaluable for this thesis.
I would like to thank Lubaba Tasneem, who created some of the virtual agents seen in this thesis.
I would like to thank Dr. Lisa Guo, for her invaluable donation of hard drives to store ALETs parsed data on.
I would like to thank the mentors I had over two summer internships, Dr. Leslie Blaha and Dr. Dustin Arendt. They expanded my breadth of research, which ultimately strengthened this work.
I would like to thank the Intelligent Virtual Agents 2015 Doctoral Consortium Committee, Specifically, Dr. Dirk Heylen, Dr. Joost Broekens, Dr. Catherine Pelachaud, Dr. Hannis Vilhjalmsson and Dr. Khiet Truong for their comments and suggestions on the focus and direction of this thesis.
I would like to acknowledge the other members of the Games and Intelligent Animation Labs, Weizi Li and John Mooney for their collaboration and brotherhood during my research career at George Mason University.
I would like to thank the professors at Roanoke College who kindled the flame of discovery within me. Specifically, I would like to thank Dr. Durrell Bouchard, Dr. Matthew Fleenor, Dr. Richard Grant and Dr. Rama Balasubramanian, who showed me not only the value of research and exploration, but of interdisciplinary discovery.
I would like to thank my family, especially my mother Maryann Balint and sister Dianna Balint, for encouraging me throughout my entire life to keep striving for excellence.
I would like to thank my fiance Jessica Randall, without whose love, care, support and editing abilities, I would have burnt out and not been able to finish this thesis.
I would like to thank my pride members, especially Sarah Alhbrand and Dr. Maryam Bandari who kept me focused on this thesis on a daily basis. I would also like to thank my other pride members, specifically Starr Marberry, Kelly McCormack, Lindsey Kravitz, Lexie Reimer, Derek Butler, Alexis Hopkins and Rachel Desando for their love and encouragement throughout my academic and personal life.
I would like to thank those who participated in Panda Tuesday, for providing a known and well needed break during the week.
I would like to thank the members of the other labs that I had shared a space with.
I would like to thank the friends I have met in the virtual agent community for adding an element of fun and excitement to an already fascinating topic.
I would like to acknowledge those friends I have not mentioned, for keeping my spirits high.
Finally, I would like to thank all those that have had a positive or negative impact on my life, without which I would not be the person that I am today.

Table of Contents

Page

List of Tables ix
List of Figures xii
Abstract xvi
1 Introduction 1
1.1 The Need for a Consistent Action Set 3
1.2 Contributions of this Work 4
1.2.1 Organization of Thesis 5
1.3 Related Work 5
1.3.1 Virtual Character Animation 5
1.3.2 Semantics for Simulations and Animations 7
1.3.3 Generating Meta-Data 9
1.3.4 Agent Programming Languages 12
1.3.5 Hierarchical Task Networks 14
1.4 The Parameterized Action Representation 15
2 Defining the Parameterized Action Representation 17
2.1 Agent Languages 18
2.1.1 Representations of Actions for Virtual Characters 18
2.2 The Parameterized Action Representation Defined 22
2.2.1 A Note on Symbolic Notation 22
2.3 Definition of Semantics $\mathbb{S}$ 23
2.4 Definition of Objects $\mathbb{OBJ}$ 25
2.5 Definition of Conditions and Assertions 29
2.6 Definition of Actions 30
2.6.1 Execution of an Action 33
2.6.2 Roles **R** 35
2.6.3 Condition and Assertions 38
2.6.4 Semantics of Actions **S** 41
2.7 Conjunctives 42
2.8 Explanation of Tasks **T** 45

2.8.1 Annotation . . . 47
2.8.2 Task Grammar and Representations . . . 48
2.8.3 Measure for Behavior Similarity . . . 51
2.9 Case Study . . . 54
2.10 Conclusions . . . 63
2.11 Summary of Key Components . . . 65
2.11.1 Symbols for the Language . . . 65
2.11.2 Key Functions . . . 66
3 Considerations on Action Organization . . . 67
3.1 Introduction . . . 67
3.2 Parent-Child Action Relationships . . . 69
3.2.1 Task Inheritance . . . 70
3.2.2 Semantic Inheritance . . . 71
3.2.3 Role Inheritance . . . 74
3.2.4 Condition-Assertion Inheritance . . . 74
3.3 An Application-based Measurement for FIDAG Organization . . . 75
3.3.1 Narrative Planning and Behavior Selection Algorithms . . . 76
3.3.2 Data Compression . . . 79
3.4 Experimentation . . . 81
3.5 A Quick Note on Pragmatics . . . 85
3.6 Conclusions . . . 86
4 Automated Generation of Action Hierarchies . . . 87
4.1 Data Sources for Automated Generation . . . 89
4.2 Using WordNet to Generate Ontologies . . . 90
4.2.1 A Multi-sense Method to determine the Sense of an Action . . . 90
4.2.2 Hypernym Tree Generation . . . 94
4.2.3 Confidence Based Symbiotic Generation . . . 94
4.3 Using Continuous Bag of Words to Generate Hierarchies . . . 97
4.3.1 Creating a Hierarchy using Word Vectors . . . 99
4.4 Automated Generation of FIDAGs . . . 101
4.5 Experimentation . . . 103
4.6 Conclusions . . . 117
5 Automated Generation of Action Semantics . . . 118
5.1 Connecting Objects to Actions using FrameNet . . . 121
5.2 Agents Learning their Environment through Text (ALET) . . . 122
5.2.1 Dependency Grammar Parsing . . . 123

5.2.2 Co-occurance Connections . . . 124
5.2.3 Operational Information Population . . . 126
5.3 Experimentation . . . 128
5.3.1 Datasets used in Experiment . . . 129
5.3.2 Connecting Objects to FrameNet . . . 132
5.3.3 ALET Datasets . . . 133
5.3.4 Analysis of Role Connection and ALET . . . 134
5.3.5 Analysis of Adjectives and Adverbs in Dataset . . . 140
5.3.6 Demonstration . . . 147
5.4 Conclusions . . . 148
6 Conclusions and Future Work . . . 150
6.1 Summary and Conclusions . . . 150
6.1.1 Key Contributions . . . 150
6.2 Lessons Learned . . . 151
6.3 Future work . . . 153
A PARS used in Case Study . . . 157
A.1 Object Designation . . . 157
A.2 Designation of Actions . . . 159
B Keywords used in disambiguation tests . . . 167
B.1 Keyword List . . . 167
B.1.1 Smartbody Definition Keywords . . . 167
B.1.2 Smartbody Synonym Keywords . . . 168
B.1.3 CMU Definition Keywords . . . 169
B.1.4 CMU Synonym Keywords . . . 172
B.1.5 BLS Definition List . . . 174
B.1.6 BLS Synonym List . . . 177
B.2 Object Leafs for Experimentation Section . . . 179
C Automated Semantics Data . . . 185
C.1 Objects used in our Analysis and their Associated Synsets . . . 185
C.1.1 ModelNet . . . 185
C.1.2 CalTech 101 . . . 216
C.1.3 Office . . . 221
C.1.4 Pub . . . 225
C.2 Actions and Frames used in our Analysis . . . 228
C.2.1 SmartBody . . . 228
C.2.2 CMU . . . 229

C.2.3 The ICS Action Database . . . 232
C.2.4 The Human Motion Database . . . 233
C.2.5 The American Time Use Survey . . . 235
C.3 The Designation of Functional Elements for Each Frame . . . 238
C.3.1 ICT SmartBody . . . 238
C.3.2 CMU . . . 242
C.3.3 ICS Action Database . . . 252
C.3.4 Human Motion Database . . . 256
C.3.5 BLS . . . 264
Bibliography . . . 269
D.1 Education . . . 280
D.2 Awards, Grants, and Professional Memberships . . . 280
D.3 Appointments . . . 280
D.4 Journal Papers and Book Chapters . . . 281
D.5 Conference Papers . . . 281
D.6 Workshop Papers and Posters . . . 282
D.7 Presentations and Invited Talks . . . 282
D.8 Service . . . 283

List of Tables

Table		Page
1.1	A list of different names to connect action and objects in a virtual simulation. Different meanings of the same word have been broken up into separate entries.	8
2.1	Set components of PAR	31
2.2	The minimum and maximum distance actions based on roles for both exact and similar measurements. Actions that differ between similar and exact are shown in bold.	55
2.3	The number of ties for each action, using both similarity measure and exact measure.	57
2.4	The action *DistractGuard* from Shoulson et al. [1]	58
2.5	The action from Table 2.4, written out as a PAR.	59
2.6	The minimum and maximum cost actions when comparing tasks using an exact measurement. The task measurement is shown using both role cost and non-role cost.	61
2.7	The minimum and maximum cost actions when comparing tasks using Wu-Palmer similarity measurement. The task measurement is shown using both role cost and non-role cost.	63
4.1	An example of the polysemous word *Cook* taken from WordNet. Only the verb senses are shown.	90
4.2	Statistics for single parent word vectors on each action set used in our tests. The number of actions are the base number of actions and the total items in the hierarchy include generalizations. The % for actions with frames is calculated from the base number of actions.	110
4.3	The average number of parents found when generating FIDAGS, using either a single threshold or using the highest action and a threshold.	112
4.4	Statistics on each action set used in our tests. The number of actions are the base number of actions and the total items in the hierarchy include generalizations. The % for actions with frames is calculated from the base number of actions.	115
5.1	Functional Elements for the Frame *Cook*, specifically *Cooking Creation*. The first three are operational information $\mathbf{R} \in \mathbb{ACT}$, while the last two are NIFI $\mathbf{S} \in \mathbb{ACT}$.	122
5.2	A sample of the matrix *ND* for the object *Food.n.01*, along with its counts.	125

5.3 A sample of the matrix *VD* for the action *Cook.v.01*, along with its counts. 126
5.4 Statistics on each object set used in our tests. The total number of items in the hierarchy includes objects that are generalizations of other objects. 129
5.5 Statistics on each action set used in our tests. The number of actions are the base number of actions and the total items in the hierarchy include generalizations. The % for actions with frames is calculated from the base number of actions. 131
5.6 The average accuracy for detection $\mathbf{R} \in \mathbb{ACT}$ in ALET using Wikipedia as a data-source 134
5.7 The average accuracy for detection $\mathbf{R} \in \mathbb{ACT}$ in ALET using the Billion Word News Corpus as a data-source 135
5.8 The average accuracy for detection $\mathbf{R} \in \mathbb{ACT}$ in using Pelkey and Allbeck's Method 135
5.9 The average accuracy for detection $\mathbf{S} \in \mathbb{ACT}$ using ALET with Wikipedia 137
5.10 The average accuracy for detection $\mathbf{S} \in \mathbb{ACT}$ using ALET with the Billion Word News Corpora 138
5.11 The average accuracy for identifying functional elements as either operational information or NIFI. Errors are shown as one standard deviation. A single factor ANOVA found significance between tests, with $\rho < 0.001$. 139
5.12 The average accuracy for identifying functional elements only as operational information. Errors are shown as one standard deviation. A single factor ANOVA found significance between tests, with $\rho < 0.001$. 140
A.1 Action Number 1:Hide 159
A.2 Action Number 5:Lock 159
A.3 Action Number 10:EscapeCell 160
A.4 Action Number 12:Press 160
A.5 Action Number 16:Guard 160
A.6 Action Number 19:Trap 160
A.7 Action Number 20:TrapGuardsAlarm 161
A.8 Action Number 21:TrapGuards 162
A.9 Action Number 24:Draw 162
A.10 Action Number 26:Daze 162
A.11 Action Number 35:Call 163
A.12 Action Number 38:Approach 163
A.13 Action Number 40:Give 163
A.14 Action Number 41:Exchange 163
A.15 Action Number 43:Open 164

A.16 Action Number 44:Unlock 164
A.17 Action Number 45:Take 164
A.18 Action Number 47:StealKey 164
A.19 Action Number 54:SoundAlarm 165
A.20 Action Number 55:Close 165
A.21 Action Number 62:DistractGuard 165
A.22 The actions used to create a generalization set for the actions in the case study. . . . 166

List of Figures

Figure		Page
1.1	Two frames of a keyframe animation using a bone hierarchy.	6
1.2	A sample (a) interactional information and the equivalent (b) operational information for the action *sit* and two objects.	9
1.3	A virtual environment displaying all actions that can be used on each object. The number of actions used in this data-set is approximately sixty. There is a large amount of overlap for actions due to objects being of a similar template. A virtual human deciding on an action would examine all or a subset of the actions attached to all or a subset of the objects, which would require examining several redundant actions.	10
1.4	A sample task network decomposition for the action *PickUp*, which can be broken up into several smaller actions.	14
1.5	A sample action hierarchy in PAR.	16
2.1	A high-level specification for *Walk* in (a) STRIPS format and (b) hierarchical task network (HTN) format.	19
2.2	A high-level specification of *Walk* in Event format.	20
2.3	A high-level specification of *Walk* in PAR format	21
2.4	A set of sample objects connected to their generalizations. Note that in PAR, agents are also considered as objects	26
2.5	Two example action sets with generalizations. (a) Actions that are single entities. (b) Actions with multiple generalizations.	32
2.6	The processing steps for executing an action in PAR. This is a flowchart of Algorithm 1	36
2.7	A blend space between three animations in a single action. This one dimensional blend is controlled by a speed parameter	43
2.8	A graphical depiction of the three sequential connectors, (a) AND,(b) OR, and (c) GATHER.	44
2.9	A graphical depiction of two parallel connectors.	45

2.10 The task for the action *SoundAlarm* with its associated roles, shown as annotations of each leaf node in the task. This task was converted from [1] and can be found in Appendix A. 48
2.11 Three representations for the task Exchange, adapted from [1] using their name. (a) The task as a tree. (b) A post-order representation of the task for *Exchange*. (c) Our processed representation of the task for *Exchange* 52
2.12 Entailment groupings for all actions in the case study when only comparing roles, using (a) similarity metric or (b) exact comparison. This is a visual explanation of Table 2.2 56
2.13 Entailment groupings for all actions in the case study when no information about generalization is used. Red action names signifies changes between (a) and (b). . . 60
2.14 Entailment groupings for all actions in the case study when generalizations are used. Red action names signifies changes between (a) and (b). 64
3.1 An action set of three actions, where one (*Travel*) is the parent of the others. 68
3.2 A motion blend of several animation, including (a)*Walk* and (b)*Run* 72
3.3 An action hierarchy with (a) semantics on the action and (b) semantics after being combined 73
3.4 The compression size for both Roles and Assertions for our action sets 82
3.5 The time to compute a reasoning step of Normoyle et al. vs. the method of organizing a hierarchy. 83
3.6 The time to compute a narrative plan of Kartal et al. vs. the method of organizing a hierarchy. 84
4.1 System Overview 89
4.2 The Multi-pass technique's testing conditions. Each level determines word senses with the most precise methods higher up in the pyramid. Techniques lower in the pyramid are less precise but have greater coverage. 91
4.3 A node in an action hierarchy being added (b) matches a node in an existing tree in (a). The hierarchies are merged at the common node and the ancestors of (b)'s matching node are removed (c). 95
4.4 An overview of how our confidence-based method disambiguates action candidates. 97
4.5 An example bag of words, and the word vector from the sentence "jane eats". . . . 98

4.6 A two component PCA-based word vector representation of the CMU data-set. (a) The actions are shown with their two components. (b) DBScan measures the distance between actions, connecting any neighbors based on their distance. (c) The final clusters are combined based on the connected neighborhoods. 100
4.7 A Sample FIDAG generated for SaccadeSpeak using a hypernym multi-sense method. 102
4.8 A Sample FIDAG generated for SaccadeSpeak using the word-vector tree building model. 103
4.9 The percent of found (a) and correct (b) senses for our ten sample test. Error bars represent one standard deviation. A single factor ANOVA analysis provided a negligible p-value for both figures, with a Tukey-Kramer test showing significant difference between our method and the word only, definition only, and word with path methods for CMU's data set, and word only and definition only for Smartbody's data set. A Tukey-Kramer test shows significant differences between all methods and the multi-path method for the BLS data set. 106
4.10 The percent matched vs. dataset when using keywords picked by a system compared to those generated from text corpora. Error bars are given as one standard deviation. 108
4.11 The precision-recall graph obtained in testing the confidence-based method and the multi-sieve method on our action data-sets over all three data-sets. 109
4.12 The percent matched vs. dataset when all keywords matched over the threshold vs. using the highest matched over the threshold. 111
4.13 The average percentage of parents with completely compatible (a) semantics and (b) roles for both an above-all threshold and a highest threshold. 113
4.14 The average percentage of (a) semantics and (b) roles that are the same across all parents. 114
4.15 The percent of Frames found for each multiple parent tree building method. A single factor ANOVA shows no difference between the three methods, with $p = 0.1$ . . . 116
5.1 An overview of the basic techniques for populating action roles and semantics . . . 120
5.2 A pictorial overview of the connection step. The object parameters of actions are linked to object types in the object ontology. 121
5.3 A graphical representation of dependency parsing on text. 124
5.4 Two sample object rich virtual environments from the Unreal Game Engine Marketplace. The two environments contain a) 82 and b) 52 unique types of objects, derived from over a 100 graphical models each. The total number of object instances in each environment is (a) over 400 objects and (b) over 300 objects. . . . 130

5.5 The percent of matched object operational information when only using an action hierarchy, only using an object hierarchy, and using an action and object hierarchy. Error bars are shown as one standard deviation. 133
5.6 The percent accuracy of role detection vs. action data-set for ALET with two different data-sets vs. Pelkey and Allbeck. Error bars are shown as one standard deviation. 136
5.7 The percent recall of role detection vs. action data-set for ALET with two different data-sets vs. Pelkey and Allbeck. Error bars are shown as one standard deviation. . 137
5.8 The percent overlap vs. action set when using the matrix *VD*. No error bars are shown as there was no variance between object sets. 138
5.9 The average number of adverbs connected to action sets vs the cutoff value. The cutoff values used in our experimentation are shown as dotted lines. 141
5.10 The average number of adjectives connected to object sets vs the cutoff value. The cutoff values used in our experimentation are shown as dotted lines. 142
5.11 The average total accuracy of the system vs. the adjective cutoff value when using ALET. 143
5.12 The average object recall of the system vs. the adjective cutoff value when using ALET. 143
5.13 The average accuracy of detecting operational information $\mathbf{R} \in \mathbb{ACT}$ vs. the adjective cutoff value when using ALET. 144
5.14 The average total accuracy of the system vs. the adverb cutoff value when using ALET. 145
5.15 The average object recall of the system vs. the adverb cutoff value when using ALET. 146
5.16 The average accuracy of detecting operational information $\mathbf{R} \in \mathbb{ACT}$ vs. the adverb cutoff value when using ALET. 146
5.17 A virtual environment sold on the Unreal Engine Marketplace annotated with operational information. The action set shown is (a) ten actions from CMU and (b) nineteen actions from SmartBody. 147
5.18 A virtual environment sold on the Unreal Engine Marketplace automatically annotated with operational information for ten actions generated with CMU action set. . 148
6.1 An example scene (a) without processing to move the objects and (b) with processing to constrain the objects. 154
6.2 A possible pipeline for using scene-to-text in analytic software. Shown here is a data-set, action-set and visualization of a wireless sensor network that is currently being researched. We have designed the visualization for their system already. . . . 156

Abstract

AUTOMATED EXTRACTION OF ACTION SEMANTICS FOR EMBODIED VIRTUAL AGENT USING TEXTUAL KNOWLEDGE BASES

J. Timothy Balint, PhD

George Mason University, 2017

Dissertation Director: Dr. Jan M Allbeck

In games, training environments, and other virtual simulations, non-player characters (virtual agents) perform actions to interact with the 3-D graphical objects surrounding them to accomplish specified goals. These interactions and other components of a virtual agents behavior are generally created by hand by a simulation author. Artificial Intelligence (AI) and planning systems then utilize these created actions to allow virtual agents to reason over their abilities considering their environment. However, any behavior or interaction the simulation author wishes to have the virtual agent perform must be created beforehand. When using actions that have several components, the creation process is tedious, time consuming and possibly combinatoric. This is especially true when faced with many objects and actions, when multiple objects can be used for an action. While this burden is decreased to some extent with the use of hierarchies where information can be generalized, the total number of possible connections still explodes as the number of actions and objects in a simulation grow. It is also important for an AI system to have a consistent representation of the meaning of each action. Also, a simulation author hand-crafting actions can lead to inconsistencies between actions, such as having actions that can never be completed or should, but do not, connect actions to all usable objects. To address this issue, we show how to automate the processes for describing, organizing, and generating the meaning (semantics) of actions. Specifically, we formalized

and refined the Parameterized Action Representation (PAR) to better exploit its capabilities to enable the automated generation of actions in this language. PAR is unique in that it allows for intelligent organization of virtual agent actions through the use of action taxonomies. From this, we have formulated algorithms and measures to quantify the utility of action organization strategies based on their end application. Furthermore, we developed and evaluated novel methods to automate the population of action taxonomies from base action names through use of existing lexical databases. Our work culminates in a transformative pipeline to automate connections between virtual agent actions and 3-D graphical models as well as the population of action semantics. By defining and refining PAR, as well as automating populating the connections made between virtual agent actions and 3-D graphical models, action sets can be transferred between different virtual environments and ease the simulation authors burden.

Chapter 1: Introduction

Virtual humans have played an impressive role in creating more realistic virtual scenarios. They provide ambiance as Non-Player Characters in games [2], and are essential for testing new scenarios before physical humans work in them [3, 4]. Virtual humans require the ability to move and interact with objects and other agents in order to have meaningful impacts in the environment. This can be as simple as *walking in a straight line in a room* or as complex as *ordering a drink*, which would require several animations and connections to objects and other agents in the environment. One of the grand challenges of virtual reality in regards to increased levels of presence is taking into account the expectation of participants as they relate to real human behaviors [5]. Behaviors portrayed through animations performed by a virtual human can be executed either sequentially or in parallel as long as an understanding is in place for the proper order of the actions. Some common methods to control animations in game design and change the 3-D virtual environments is to script actions through state machines or decision networks [6].

While virtual agents perform actions in many environments, they do not tend to utilize many objects while in a rich virtual environment. Rich virtual environments are complex, containing dozens or even hundreds of objects. Creating agent behavior scripts for each action and object desired in a scenario can be a time-consuming and tedious process. For example, an *Argue* action script may use a *Microphone*, *Character*, or *Watering Can* object. While the first two objects would be objects that can argued at, a *Watering Can* would generally not be. However, it could be an object that is held while arguing. The *Microphone* would also be able to be held while arguing, filling in that action's component in multiple respects. Therefore, each action could have several categories of objects, with overlap between each category. Currently, a game designer works with a simulation author to build actions. The game designer supplies the requirements to the simulation author, who makes all the connections ad then iterates with the game designer to build up to the original idea of the agents. For rich virtual environments, it is easy to see how this becomes a tedious

task. Furthermore, a mistake in one script can have immersion breaking impact on the scenario. Modulization of domain knowledge on graphical models and animations can mitigate errors, as the scripts can then use known information parameters, (i.e. semantics), such as *density* and *grasp sites*. Note that these properties do not have to be directly related to the graphical objects, but instead can provide precise knowledge that can be used in other systems of the scenario, such as qualitative physics engines [7]. Animated actions can also have semantics attached to them in a similar way graphical objects do, such as the *goal* or *type* of animation. Action semantics are generally used to modularize scripts themselves, allowing high level agent decision making processes to determine which action should be performed.

Development of actions for virtual humans is generally an author-driven process, in that it still requires a user to provide grounded values for all components of the action. Even with templates, as the number of unique objects and actions grows, it becomes prohibitive to author large amounts of semantic data. Additionally, components which require several relationships in their representation, such as connecting virtual human actions to all participant objects (seen in the above *Argue* example), must have knowledge of the purpose and effect of each part of the relationship. As another example, a mystery game might have objects such as telephones, people, and candlesticks. The game may also have several animations like talking and fighting. A simulation author must describe and connect each object's role to each action if there is one. For rich environments, the necessity to connect actions to objects causes the development of actions (their components, also called Metadata) to be ad-hoc in nature, crippling the intended purpose of creating domain focused knowledge; namely transferability from one scenario to a similar one. This work creates the foundation for the automated inclusion of action and object semantics that is critical to increasing the utility of virtual environments. Consistent, automated methods of semantic generation would allow transferability of relationships between scenarios, as they should in theory generate similar semantics for similar scenarios.

In order to determine the usefulness of automated methods for actions in virtual scenarios, it is important to understand how an action is structured and what components are required for virtual agents and environments. The description and representation of the actions and objects in a

virtual simulation make up the **agent language**. Then, we are able to determine requirements for consistent automated techniques for semantic generation for virtual agents and environments. We accomplish this by first providing a new and novel formal definition for the Parameterized Action Representation(PAR). We then show the effects organizing actions into an **action taxonomy** using one component of PAR, namely the action parents, have on agent reasoning tools. We also discuss the implications on organizing actions. Finally, we describe novel and transformative methods to automate the population of action-object connections, action parents, and properties of the actions. As our automated methods consider the action set simultaneously, then the set of components (meta-data) of the actions should be understood, which requires a formalized definition of an action representation as a base.

1.1 The Need for a Consistent Action Set

Agent languages and representations, described in more detail in the related work of Section 1.3.4, have a common theme; the abilities of an agent need to meet the goals of both the agent and simulation author as well as connect (ground) the actions into the environment. For plan languages such as 3APL [8], control of the agent occurs only at the courser scales, that is, only the final desired result of the character is specified, and it is left up to the constructs of the language to determine a way to meet that goal. While planning techniques have been developed for full control methods such as behavior trees, these representations are focused and used in ways so that the author knows how the agent is going to behave. This creates a continuum between agent control and agent emergence, with many representations being on one extreme or another. This is not to say that representations that strike a balance between these two extremes do not exist, with PAR being one such example. PAR, however, has not been well defined, and so it is unclear how it strikes a balance. We have rectified this by describing in detail the formalization of PAR.

In addition to authorial balance, an issue arises when the total abilities of a virtual character grow. Recall that currently each individual action must be constructed by the simulation author. Therefore, as the number of desired actions increases, more time and care must be spent to ensure that each action is correct and that there is consistency between actions. For example, a *Walk* and

Run action should be similar in nature and utilize the same 3-D graphical objects in their execution. However, for large environments with rich behaviors, ensuring these properties becomes unobtainable. Simply creating the actions will devour the simulation authors' time, with no ability to further examine them. This leads to issues such as many actions only effecting one object and a few actions affecting their full set. This is inconsistent, as only a few actions have their full range of expressiveness. A more expressive action representation with more components will require more forethought of the entire set, but if the expressiveness of the representation allows for action sets to be condensed and generalized, the overall effort may decrease. However, this is only true if the set is generated such that it is consistent. Due to the increased forethought and work, errors in a more expressive action representation will more quickly manifest. An action set generated for consistency would allow for more expressive sets without the worry of errors appearing in the set.

1.2 Contributions of this Work

From the above and our previous work in [9], we have presented a number of issues with virtual agent actions. To address the issue of authoring virtual agent actions and connecting motion data to 3-D graphical objects, we make the following contributions:

- We formalized and refined an action representation (PAR) to better exploit its capabilities and enable automated generation methods.
- We demonstrated the utility of action taxonomies through an application based measure of efficiency gain.
- We developed and evaluated novel methods to automate the population of action taxonomies from base action names through use of existing lexical databases, published in previous work [10].
- We developed and evaluated a transformative pipeline, ALET (Agents Learning their Environment through Text), to automate connections between virtual agent actions and 3-D graphical models as well as the population of action semantics, published in previous work [11].

1.2.1 Organization of Thesis

In exploring these contributions, we provide a better understanding of action meta-data, and how to connect and populate high level understandings of actions to low level animations, taking into consideration the entire environment that the virtual characters exist in. This work is formatted in the following manner: Chapter 2 describes PAR, providing the notation and definitions used for the rest of the work. Chapter 3 describes which components must be considered when organizing actions into a hierarchy, taking a consistency and application based approach to the problem. Finally, Chapter 4 and Chapter 5 describe methods to populate action meta-data, specifically on how to generalize actions and connect them to the environment through ALET. These two chapters show how meaning can be populated provided some meta-data about the action is known. In Chapter 3, Chapter 4, and Chapter 5, we also provide measurements for judging different action's components, including the taxonomy and operational information. With test action sets, we show the utility of automated methods to generate action components from animation meta-data.

1.3 Related Work

1.3.1 Virtual Character Animation

Virtual characters are articulated 3-D graphical models capable of performing animations for the purpose of displaying changes to a virtual environment. In our work, all the characters are controlled through a bone hierarchy, with vertex weights mapping from each vertex to every bone, although other forms of agent control such as blend-shapes are also possible. These articulated characters perform animations through the use of one or more motion controllers, which deform the character's graphical model from one static pose to another. These controllers can either be based on Forward Kinematics, such as motion captured or key framed animations, or inverse kinematics. Inverse kinematics is a technique to determine the rotation of a chain of joints of an articulated figure given the desired end position of an end joint. Several inverse kinematic methods have been developed for both arbitrary joint chains [12] and human joint chains [13]. Our virtual characters use forward kinematics to perform key frame animations, resulting in either a single use gesture or to display

an animation on a loop while other controllers are also used, as seen in Figure 1.1. In our work, inverse kinematics are used for variable transition controllers, such as controlling the character's gaze or reach state, which can change depending on the start and desired end positions. Forward Kinematic techniques such as motion playback are used for any motion controller that changes the position of the root joint or requires a known, constant rotation of joints (such as a walking controller or waving gesture controller). Both techniques can be encapsulated, and connected to AI systems through meta-data, such as the name of the motion controller.

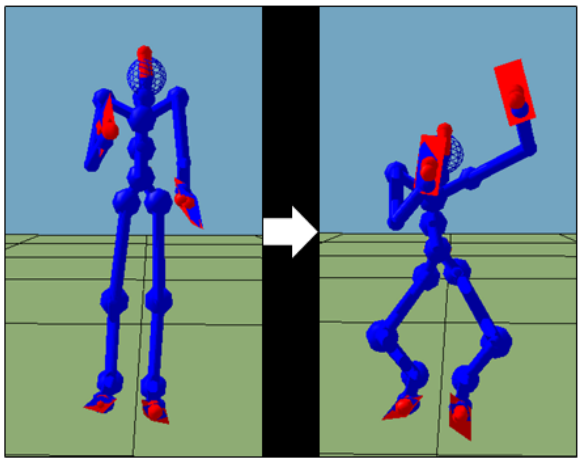


Figure 1.1: Two frames of a keyframe animation using a bone hierarchy.

Multiple motion controllers can be used by a character simultaneously to create complex actions. For example, a motion controller that translates the root bone along a path can be combined with a stepping animation to create a character that walks along a path. Different motion controllers can also be used for the same action depending on the state of the character. For example, a different motion controller for a *wave* action would be used if the agent had a sitting pose vs. a lying down pose. Understanding which motion controller or combination of controllers should be used for a given action is therefore scenario specific based on the context. However, meta-data of each controller, such as the name, will remain the same despite this. For this reason, we operate on general meta-data of actions and assume that combination of motion controllers are already provided

by the simulation author. Specifically, our work operates on the name of an action along with a sparse set of descriptive natural language keywords to describe the action.

1.3.2 Semantics for Simulations and Animations

In recent years, there has been much work in adding meaning to the objects of a virtual world [14]. There are various reasons for this, including creating more believable physics systems [15], procedural generation of scenes [16, 17], and character-object interaction [18]. As we are focused on actions, character object interaction (operational information) is being considered as the most important semantics that can be linked to objects in a game environment.

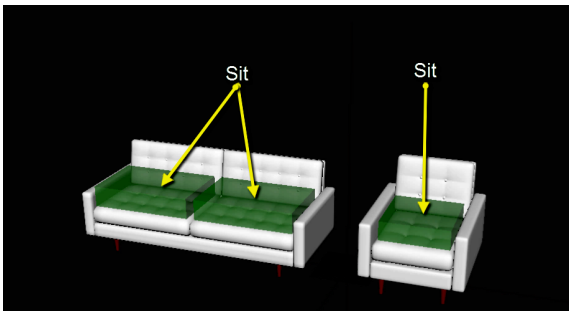
In the virtual agents' community there have been several proposals to connect actions to objects, mainly following the methodology of Smart Objects [18]. The terminology from these proposals is not consistent and can be compared in Table 1.1. Since there are slight differences in the definitions, this work will use *interactional* information when referring to the general case of action semantics attached to objects, such that the interaction is known to the virtual character through the object. Interaction information is a form of action semantics that connects 3-D graphical models with the animations a virtual human would perform, defined in Farenc et al. [19]. Peters et al. [20] introduced the idea of interactional information as slots, allowing multiple agents to interact using an object. This consisted of having a simulation author create an action-site pair and link connections with actions through a script. An action-site pair is the area on an object that a virtual agent can use to interact with that object, and can be seen in Figure 1.2a. Donikian and Paris [21] used the concept of *affordances* (general connections between actions and objects) to attach virtual objects to animations. Heckel and Youngblood [22] created a method to have virtual agents only consider affordances if the object was in a state that it could be operated on. Finally, Kraayenbrink et al. [3] used interactional information to control action selection for populations of virtual humans. The progress made in these works is meant to ease the cognitive decision process that virtual characters must make when selecting actions to perform, given available resources in the scenario.

Interactional information connects smart objects to virtual human animation through the use of

Table 1.1: A list of different names to connect action and objects in a virtual simulation. Different meanings of the same word have been broken up into separate entries.

Term	Used By	Meaning
Interactional	Kallmann and Thalmann [18], Farenc et al [19], Peter et al [20]	A site connected to an action with which a virtual human can activate that site to start a script for that action
Affordance (Any object action connection)	Donikian and Paris [21], Heckel and Youngblood [22], kapadia et al. [23]	A broadcasted action script that can be performed using an object
Affordance (Graphical Only)	Gibson [24], Sequeira et al. [25]	Determining contextual actions that can be performed on a graphical model based on the physical representation of the object.
Operational	Our previous work [10]	An object or set of objects connected to an action that will be used in either one step or during the entire action

action-site combinations. This relationship between actions and objects is in almost all cases connected to the object, providing the scripts to the character if the character wishes to use that object. This has the effect of forcing the character to decide on an action through the consideration of all objects in the environment. Many of the goal directed action representations available to Belief-Desire-Intention (BDI) agents(defined in Section 1.3.4) preform their planning over an action library, determining if an action can be performed by examining STRIPS-like pre-conditions [26, 27, 23]. In these cases, the objects in the environment become requirements to perform the action, as seen in Figure 1.2b. Examining interactional information as a requirement of an action instead of a container of an action (which from Table 1.1 is called *operational information*), has an advantage over interactional information, in that it allows virtual characters the ability to reason over actions. Operational information is another semantic of the action, and is in addition to other action semantics such as the effects of the action on the environment. Reasoning over interactional information forces agents to reason over objects, when in general the existence of an object in a given state is all that matters to the creation of an action. For automated methods of interactional information in rich environments, the sheer scale of choices can be cumbersome for virtual agents and impossible for



(a)

Sit (agent, object)
agent = Defined
object = {chair, couch}

(b)

Figure 1.2: A sample (a) interactional information and the equivalent (b) operational information for the action *sit* and two objects.

crowds of virtual agents to reason over. For example, Figure 1.3 shows all actions connected to objects (without site information). If a representation of semantics allows the connection that interactional information provides between agent actions and objects to be converted into operational information, such as converting Figure 1.2a to 1.2b, then operational information should be used for virtual human decision making.

1.3.3 Generating Meta-Data

Until this point, we have discussed the uses of semantics for characters in virtual scenarios, but have not described how these semantics are generated and added to a virtual scenario. This is because the literature on creating semantics for virtual humans and environments is largely silent, instead focusing on the representation of semantics. It therefore should be assumed that simulation authors add in semantics manually, which, as discussed previously, becomes prohibitive for large, rich virtual environments. Automatic generation of sites has been examined as an application of part recognition in computer graphics [28, 29, 30] and this information can be transferred from one object to another. Part recognition is valuable for interactional information, where parts can be automatically linked to actions. This would include being able to generate a handle for a grasp site or find large flat surfaces for a *Sit* action from a labeled set of parts. For complex actions involving several objects, or for objects that use actions not based off of affordances, other methods to generate semantics are needed.

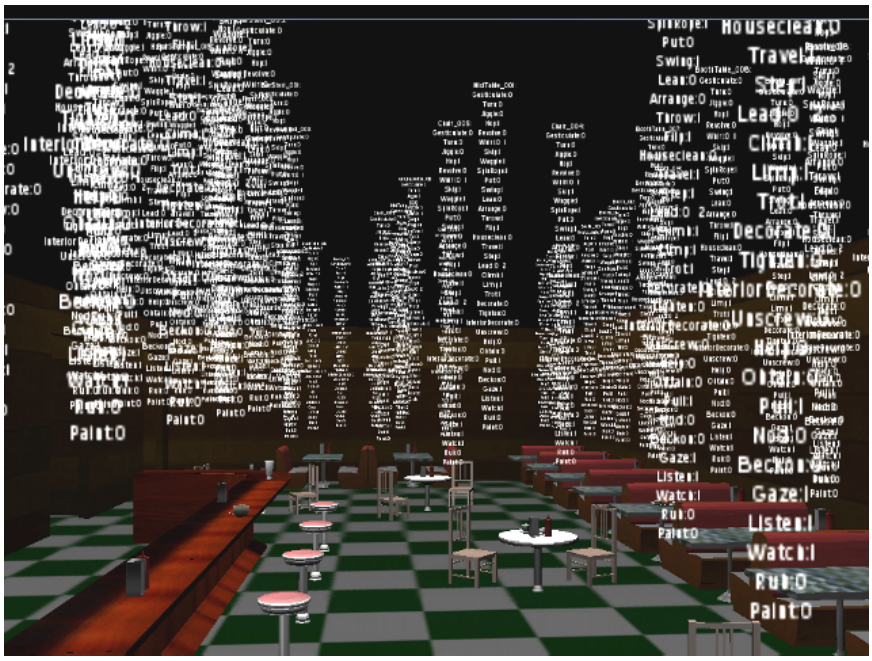


Figure 1.3: A virtual environment displaying all actions that can be used on each object. The number of actions used in this data-set is approximately sixty. There is a large amount of overlap for actions due to objects being of a similar template. A virtual human deciding on an action would examine all or a subset of the actions attached to all or a subset of the objects, which would require examining several redundant actions.

Semantic information for objects and actions to be stored is in an ontology. Ontologies contain relationships and taxonomies, such that information can be attached to an abstract concept, and links to that abstract concept can be assumed to have relationships between them. Taxonomies also allow semantic data to be placed at more general abstractions of an object or action, reducing the redundancy between objects. Ontologies have been used by several virtual simulations [27, 31, 15, 32] to connect and provide meaning to the objects and actions virtual humans must interact with. Ontology research such as Drumond and Giradrdi [33] makes a distinction between concepts with a taxonomic relationship (*IsA* or *IsPartOf* relationships) and non-taxonomic relationships (such as operational information). This break-down of relationships becomes a key concept in ontological learning methods.

If we assume that the 3-D models and animation information is connected together in an ontology, then techniques used in ontology generation become available for semantic generation. The

field of ontology generation and learning from large data-sets is a well studied topic, with several books [34, 35] and survey papers written on the topic [33]. Ontology learning from structured and semi-structured data sources generates consistent hierarchies over different parameters, as the learning methods are essentially pruning away un-needed data. Techniques to generate ontologies for virtual humans have made some progress using semi-structured data sources such as WordNet[36].

Semantic Generation for Virtual Humans

Throughout this thesis, it has been made clear that there are two important pieces of virtual humans interacting in virtual environments, the 3-D graphical models (objects) and animations (actions). Objects and actions are fundamentally distinct in their design, creation, and use, but can be connected through interactional or operational information. Therefore, we can divide an total virtual simulation ontology into two distinct hierarchies, which can possibly be evaluated separately and then connecting them together for the full ontology. Dividing the ontology into objects and actions focuses the information extraction process and removes any confusion attempting to process both at the same time entails. For example, graphical models and animations are related to nouns and verbs from natural language and there are several words that can be both, such as a *cook* (noun) who *cooks*(verb). If the names of the action or object are used to retrieve semantic information about it, then the system must determine its proper meaning from a larger set (the set of verbs and nouns). A smaller set would increase the accuracy of generation, and since objects and actions are disjoint, the author can propagate this distinction before semantic generation begins.

Information retrieval for virtual humans has seen progress in each of these two sub-problems. Pelkey and Allbeck [37] used the names of 3-D models to automate the population of a taxonomy of objects and attach binary semantics to each object. In their work, objects such as a *shrimp platter* are connected with a taxonomic relation to *food* and have a non-taxonomic relation to *edible*. Both of these semantics are attached to all shrimp-platters in the scene, and therefore, all *shrimp platter* are considered *edible*. For action semantics, in our previous work [10] we created a taxonomy of actions and populated operational information from a previously generated object hierarchy, considering only the objects in question. We motivated this with an example of *cooking*, which connect a *cook*

action to a *food* object in a pre-built object ontology. Using these two methods together would connect *cooking* to a *shrimp platter* object. This method was tested against a ground truth, with a maximum recall achievable at about 60%. It is discussed in the conclusion of this work that the other 40% of the operational information was contained through binary or set semantics that would be attached to objects.

Each of these techniques have been able to generate taxonomies and retrieve semantics for separate pieces of a virtual simulation. However, pieces of a virtual simulation should co-exist. That is, the actions a virtual human can perform should fit in with the objects available in an environment. For planning systems, having operational information is useful for determining what actions can be performed (at run-time or pre-planned), but operational connections can be determined during the specification phase, when the ontologies are generated. Provided a set of actions and objects are known, the important domain knowledge for virtual humans in that the system should exist before any planning system is necessary. Having an understanding of the actions and objects in a scenario should also allow a system to know what properties are not needed in the simulation. For example, if a simulation author does not have objects that have the property of *containing* a liquid, then it does not make sense to have an action require *containing* a liquid. If a simulation author then added an object with a *containing* property, the automated methods should propagate this change by rebuilding the taxonomies. From Pelkey and Allbeck, adding objects does not seriously impact the time to create a hierarchy, and so the generation systems could stay consistent, even with this change. Therefore, while progress has been made due to separating out the semantics, methods that will inevitability combine the relationships are still needed.

1.3.4 Agent Programming Languages

An autonomous (or even semi-autonomous) virtual agent is not only expected to perform an action, but be able to decide which action is appropriate to perform in a given scenario. This means they must be able to deliberate over their actions. Virtual agents have used motion controllers with operational information as the basis of the agent's action. However, this does not mean the agent is able to deliberate over these tasks. Furthermore, if a human wishes to interact with virtual characters

(whether as a simulation author controlling characters or as an avatar treated as a separate agent), there must be a rigid and understandable means of communication. Natural Language is known to be ambiguous, and this ambiguity can cause undesirable effects such as removing the control of characters from simulation authors. In order to program agents and connect them to their environment, several agent programming languages have been developed, and a survey of multi-agent programming languages can be found at [38]. Specifically, languages such as 3APL [8], FLUX [39], LORA [40], and AgentSpeak [41] have been used to create heterogeneous populations of agents, controlling their beliefs, desires, and intentions (BDI). Agent languages provide a more precise representation of a world then natural language, and as such, can be used to control virtual agents and provide communication between similar agents in the world.

Many agent languages, including the ones listed above, use a logical representation to describe their environment. Axioms describe relationships between known concepts in the world and can either be fluent based (in the case of FLUX) or state based (in the case of 3APL and LORA). Semantic properties for objects allow an agent to form a knowledge base, and from that an agent can deliberate on actions. This means that most agent languages are concerned with two main areas: choosing a strong representation of the agent's environment and providing a both grammar and logical rules that allow an agent to determine true statements about its environment.

The utility of having a basic understanding of agent languages is two-fold. First, by examining how agent languages operate, a better understanding of how the grammar for PAR can be established. As PAR is meant to be a mid-level action language, its syntax should be similar and executable by higher level agent planning languages. Secondly, examining the information used by axioms and states of these agent programming languages allows the necessary semantics to be determined. Automatically generating semantics for PAR should coincide with what is able to be understood by other agent languages. If this was not the case, then those semantics would not even be needed.

1.3.5 Hierarchical Task Networks

Semantic representations of actions and objects allow automated methods to create complex understandings in a game environment. However, on their own they do not provide an implementation of how an action should operate and need some method of decoding to do so. Therefore, a formalized representation of the task must be implemented that allows for both high level actions (such as *cooking*) and low level actions (such as *taking a step*) to be decomposed and performed. Hierarchical Task Networks(HTNs) [42, 43] are one way in which tasks for virtual humans are decomposed into *Primitive Actions*(low level actions) and connected to motion controllers. By decomposing a task into primitive animations, a virtual character can be controlled from high level commands.The granularity of a HTN should be based upon the motion controllers a simulation author has, as incorrect granularity would mean some primitive actions would be placeholders since they do not have motions attached to them. An example can be seen in Figure 1.4, which shows a base action being decomposed into several primitive actions. In this example, each primitive action would be connected to a single animation.

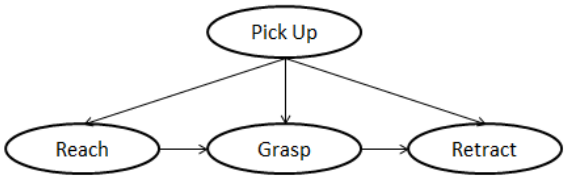


Figure 1.4: A sample task network decomposition for the action *PickUp*, which can be broken up into several smaller actions.

HTNs are difficult to create by hand and therefore several techniques have been introduced to learn correct task decomposition. Nejati et al. [44] used goals, as well as desired and required semantics, to compose tasks from a top down connection. In their earlier work [45] they attempted to use expert traces to learn tasks, but found that by doing so the agents could not perform as well as experts in a block world environment. In the robotic domain, Mohseni-kabir et al. [46] were able

to use experts to train HTNs for a car repair domain. There is an important nuance between using expert traces for explanation based learning of HTNs and demonstrative learning of HTNs. While they both allow HTNs to *shrink* the search space when learning, demonstrative learning generally requires a feedback loop between the character and user, providing more overall data to learn from. Therefore, care must be taken when choosing a data-source for learning HTNs, as the amount and type of data can determine the overall success of the learning method. While we are not specifically learning HTNs for PAR, examining HTN learning parallels populating semantics in a similar way. By understanding the purpose of the data-sources HTN learning is performed on, we can more accurately pin-point data-sources other semantics should be learned on.

1.4 The Parameterized Action Representation

To connect a human-like structured natural language to animations and motion controller of virtual characters, we use the Parameterized Action Representation(PAR) [27]. PAR relies on actions realized as HTNs that are controlled by finite state machines, using animation controlled through general callback functions. Each callback function connects to a primitive PAR action(name) and is representative of the motion controllers under that given name. PAR also allows for conjunctions of actions to occur both sequentially and in parallel, allowing the success of each action to be controlled with both *AND* and *OR* conjunctions. Actions are arranged in a *Forest of IsA Directed Acyclic Graphs* (FIDAGs), with more general understandings of a given action as parents of more specific ones. An example of a this seen in Figure 1.5. PAR also allows for representations of object hierarchies and semantic attachment and resolution of objects. Work on automatically populating object parents and semantics have seen promising results in Pelkey and Allbeck [37]. However, for the most part, objects are out of the scope of this thesis and therefore will not be covered in detail and will only be referenced in relation to the actions. One novel contribution of this work is a formal definition of the PAR system, including all of its components, seen in Chapter 2.

One benefit of PAR is that it operates as a controlled English language at runtime. A controlled English language is a construct of natural English that follows a regular pattern. An example would

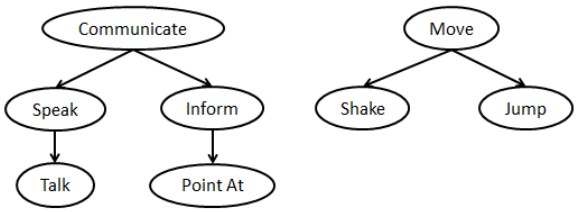


Figure 1.5: A sample action hierarchy in PAR.

be having all commands to a given agent provided as *Agent-Name Verb Arguments*. Using controlled English allows for easier communication between a physical and virtual human, provided the system takes the method of having a physical human use the controlled language into consideration [47]. Specifically, Schwitter [48] found that the usability of a controlled natural language is highly dependent on the method with which an author can input pieces of the language. Completely free-form text input requires the user to learn the structure of the language and can be challenging for novice users. However, if the input of the language matches the granularity at which it is understood, then the task becomes easier. The granularity of PAR during simulation consists of an action with associated operational information.

Chapter 2: Defining the Parameterized Action Representation

Before discussing how to automate the generation of components of the Parameterized Action Representation (PAR), it is important to define and differentiate the structure (syntax), meaning (semantics), and specific realization (pragmatics) in which our system will populate. An agent language is used to control virtual characters by describing how commands (and components of them) should be written and are understood by a character interpreting them. There are a variety of choices for agent languages [38], each with their own similarities and differences based on their application. We use the Parameterized Action Representation [49, 27], which is a language meant to bridge the gap between high level AI control (such as what is found in LTL [50, 51, 52] and low level graphical control). PAR has a number of unique features among agent languages, which we describe in more detail in Section 1.3.4. Unfortunately, there is no formal definition of PAR as a full action representation, making it difficult to describe the components and automate them. Therefore, we create, cement, and provide a formal understanding of PAR, describing the different components, how they should be written by a simulation author, and the effects of an agent's realization of an action. This should allow for a better understanding of what the automated processes should be responsible for, and define guidelines to create the different components of a PAR. Defining the syntax of PAR actions also allows for formal planning methods such as [40] to reason about the different components used for higher level autonomous agent behaviors. As PAR is an action representation, it should be noted that the syntax of PAR is designed to be used by agent authors, and that the syntax assumes un-instantiated PARs. An instantiated PAR is realized by an agent at run-time, and is the pragmatics of a PAR.

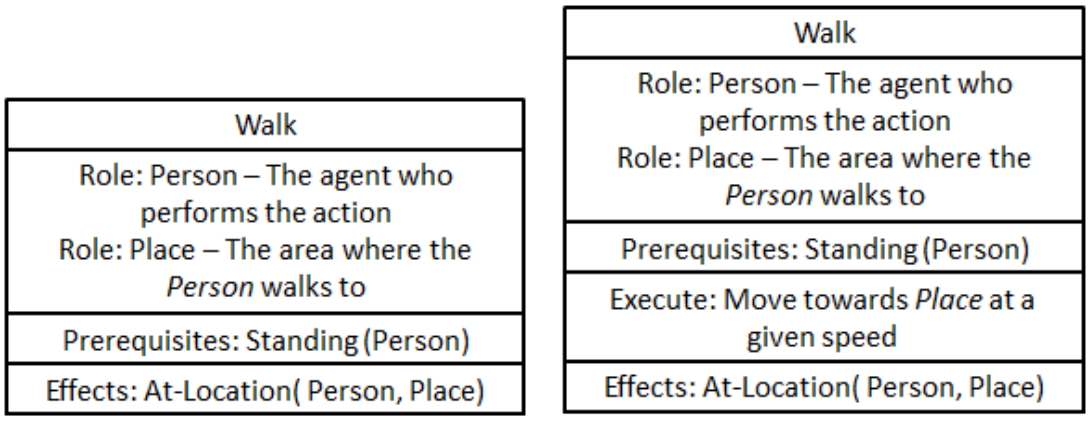
2.1 Agent Languages

The purpose of the syntax of PAR is similar to other agent languages [38, 8, 53, 39] in that it is used to describe the domain an agent exists in. The agent domain contains all knowledge that is necessary to the whole scenario, making it useful for planning and reasoning, as there are not unknown unknowns. As such, we describe other action representations, to show the similarities between PAR and these other representations.

2.1.1 Representations of Actions for Virtual Characters

To understand the need for a PAR grammar, it is important to understand different action definitions that have been formulated. To motivate PAR, we describe each representation in the context of text-to-simulation applications [10, 54, 55], specifically describing what information other representations provide and what information would still be needed for that particular application. While text-to-scene is an interesting application in that it requires an understanding of both human communicable language and graphical control, it is by no means the only use for any action representation described. We will consider a *Stanford Research Institute Problem Solver* (STRIPS)-like action [26], *hierarchical task network* (HTN) [43], and *events* [1]. A STRIPS-like action is comprised of pre-conditions and post conditions, creating a single action. An example can be seen in Figure 2.1a. For natural language command systems (text-to-simulation), executing a single command is impractical, as the text would have to explicitly state each action. This was solved by connecting together STRIPS like actions to into hierarchical structures, allowing the actions to be more complex. In essence, this is a hierarchical task network (HTN) [42]. To accommodate for the distinction of having an action be atomic (a *primitive action*) or made up of other actions (a *complex action*) more fields were added. These fields contained either a function definition (primitive actions) or the list of sub actions that make them up. The transformation from STRIPS to HTNs can be seen the execute field in Figure 2.1b. By allowing tasks to be chained together, HTNs solve the issue in text-to-simulation of having to describe each action in the text. Furthermore, the roles of objects in an HTN describe what objects are needed in a text-to-simulation system. However, HTNs only have pre-conditions and effects to describe an action, and do not have a way to describe

adverbs that may play an effect on the display and understanding of an action. So, instead of a text-to-simulation author needing to explicitly state every action, a simulation author must anticipate and create actions for every adverb that may be used, which is still insufficient for text-to-simulation applications. This does not diminish a HTN's usefulness, as they are still being developed today [56], but require other fields to be efficient in a text-to-simulation system.



(a) (b)

Figure 2.1: A high-level specification for *Walk* in (a) STRIPS format and (b) hierarchical task network (HTN) format.

One popular method for describing commands are behavior trees. Behavior trees are a design paradigm meant to ease the authorial burden of connecting actions together for use in games and simulations. A sample behavior tree can be seen in Figure 2.2. Behavior trees do not have a formal semantic representation by themselves like HTNs do. Work such as Shoulson et al. [1] and Kapadia et al [57] have attempted to add formal planning methods to behavior trees, either by wrapping the behavior tree around a grounded meta-representation (an *event*) or planning over the behavior tree explicitly. Planning over behavior trees fixes conflicts in parallel trees, but only if the trees are known to run together. Events [1] is closer to PAR's representation in that the event is similar to an action in PAR (shown in Figure 2.3), using behavior trees only to control the graphical animation. Combining events are done on the AI level by calling other events, suppling a similar behavior

to HTNs. However, events still do not contain the ability to be modified based on adverbial data. Therefore, while events are similar to PAR actions, they are not the same, and it will be shown that the definition of PAR provides a more expressive interface for simulation authors to command characters.

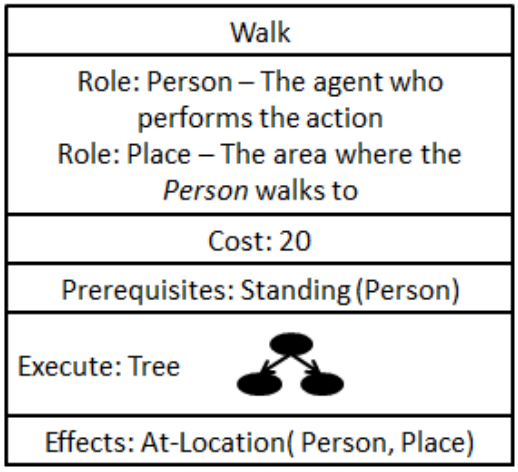


Figure 2.2: A high-level specification of *Walk* in Event format.

Finally, the Parameterized Action Representation [58, 59, 27] is a method for building actions specifically following natural language conventions. An example specification of *Walk* action for PAR may be seen in Figure 2.3. Like all other shown conventions, PAR has pre-conditions and effects of an action. Similar to HTNs and events, it also contains an executable set (which are referred to similarly as *tasks*) that is built from one or more actions. This allows knowledge to be built into an action, so that each individual command that is needed for an action to take place does not have to be specified by a text-to-scene author. Also similar to HTNs, the task is made up of other actions, making it easier to compare and parse tasks. Finally, PAR contains a field, *Semantics* that describe important characteristics of an action not specified by the roles. While this field is fully defined in Section 2.3, at a high level, it provides a way to control aspects of the action other than its beginning, end, and the objects that are used. This means that *Semantics* are

able to fill in the role of adverbs in linguistics, and are meant to pass information to the task that effects the underlying animation. Semantics may also be used to contain knowledge for planning systems that is not contained in the prerequisites, effects, or roles (such as what is seen in the work of [30, 14, 31, 15, 60, 47, 61]. The addition of semantics is a powerful field, one that has been available since the inception of PAR (used in EMOTE [62, 63]), but has not been fully defined. Furthermore, we show that the formal PAR task definition, novel to this work, can be used to compare and contrast tasks, providing even more information not normally available to planning systems. The rest of this chapter is as follows: We first define the language of PAR, describing all of the components that are necessary to specify a domain using PAR. Next, we present an explanation of *Semantics*, and explain how these are parsed through tasks. We then describe objects, laying out the properties that create an object hierarchy and attach semantics to it. After that, we define conditions and assertions, proceeding afterwards to actions. Finally, an understanding of tasks in PAR is presented, including properties that control how tasks may be created and compared.

Walk
Parents (**P**): Travel
Role (R): Person – The agent who performs the action Role (R): Place – The area where the Person walks to
Condition-Effect (**Ψ**): At-Location(Person, Place) -> Success
Prep-Specs (Θ): not Standing -> Stand(Person)
Execute: (θ) Move(Person, Place) or Turn(Person, Place)
Semantics (S): Duration, Manner, Cause

Figure 2.3: A high-level specification of *Walk* in PAR format

2.2 The Parameterized Action Representation Defined

We begin by defining the Parameterized Action Representation as a six-tuple system $\mathbb{L} = \{\mathbb{S}, \mathbb{OBJ}, \mathbb{\Phi}, \mathbb{A}, \mathbb{ACT}, \mathbb{F}\}$, where $\mathbb{S}$ is the set of semantics in the system, $\mathbb{OBJ}$ is an ordered set of objects, $\mathbb{\Phi}$ is the set of conditions in the system, $\mathbb{A}$ is the set of assertions, $\mathbb{ACT}$ is the ordered set of actions, and $\mathbb{F}$ is the set of task conjunctives. Unlike the HTNs of Erol et al., we do not differentiate between single actions (referred to as primitive actions in Erol et al.) and complex actions at the language level. The reason is due to how we treat the task network itself, and will become apparent in Section 2.6 and Section 2.8. Therefore, PAR is more closely related to the concept of *events* [1], and when deviating, will be explained accordingly.

It should be kept in mind that $\mathbb{L}$ is described as a design-time language. This means that what follows focuses on how a simulation author would create the PAR world, not in how they are realizing it. Thus, objects and actions are considered *un-instanced* (un-grounded) and are really object and action types. Therefore, while virtual agents will reason about the objects, actions, and realize behaviors created for them during a simulation, their creation and description is done prior to runtime. It should be noted that we do provide some algorithms for realization, but they are not the focus of this chapter.

2.2.1 A Note on Symbolic Notation

PAR has many layers, all of which are connected to one another. For symbolic consistency, we use special notation when a symbol refers to the language or a property thereof ($\mathbb{L}$ or $\mathbb{OBJ}$) vs. when a symbol is a sub-property of the language (λ, a sub-property of objects and actions). Furthermore, several sub-properties are themselves sets, either ordered or unordered. We denote a set sub-property using **bold face**, following the style of Events [1]. Furthermore, when discussing an item from a set, we will use a subscript notation (i.e $\mathbb{OBJ}_i$) when the property requires two or more items from that set or we mean to represent a specific item from that set. When any item from that set can be used, such as in Execution (Algorithm 1), lowercase notation will be used (i.e *obj*) unless that item is itself a set.

2.3 Definition of Semantics $\mathbb{S}$

We first define the properties of the world, contained in the semantic set $\mathbb{S}$. At the language level, $\mathbb{S}$ defines all possible properties necessary to reason about the world. Any semantic that is not defined in $\mathbb{S}$ cannot be used by the characters in their actions. This differentiates knowledge important to the characters from knowledge that a simulation uses but is not necessary for the characters. Formally, we define a semantic $s \in \mathbb{S}$ as:

Definition 2.3.1. *Let s be a semantic in the set of semantics* $\mathbb{S}$*. s is defined as* $s = \{\lambda, \mathbf{V}, \omega\}$*, where*

- λ *is the unique label of the semantic*
- $\mathbf{V}$ *is the set of values the semantic can take*
- ω *is a application of the semantics, and has the value: object, action, both*

Note that each semantic has a designation ω, which can be an *object*, *action*, or *both*. Object properties include the graphical object assigned to a given object (described in Section 2.4). It also allows for non-graphical properties that are necessary for reasoning about the objects. Action properties provides context sensitive fields related to the meaning of a given action. Fields such as the *duration* and *purpose* of a given action provide a grounded understanding of the action to an AI planning system and are similar to *adverbial clauses* in natural language. They also allow for fine grain controls of motion controllers between subsequent **realizations** of an action. Finally, there are semantics that have meaning to both actions and objects, designated by $\omega =$ both. In many cases, the semantics of an object (especially when the object is an agent) change the fine-grained control of an action. An example is the motion personality work of Durupinar et al [64]. In their work, the action was controlled by the desired personality of the agent, and had a direct effect on the animations performed by that agent. So, the personality is both a property of the agent(object) with values set on the agent and a property of the action with that value controlling parameters of the action. In this sense, it is important that a semantic *s* is understandable by both actions and objects, so that a common language may be used between them.

The set of semantic values $\mathbf{V}$ represents all possible grounded values that an object or action may be assigned at run-time for that given semantic. For example, a semantic *status* may have possible

values of *Idle*, *Busy*, and *Broken*. Note that the value at run-time would be selected from these possible values and is considered realized (Definition 2.3.2) at that time. We denote the difference between unrealized semantics ($\mathbf{V}$) and realized semantics as a lowercase (v), and describe it in Definition 2.3.2. Furthermore, $\mathbf{V}$ may be the empty set($\emptyset$), which the system understands to be a semantic that can take on any real value. So, if $\mathbf{V} = \emptyset$, then $-\infty < \mathbf{V} < \infty$. By utilizing $\mathbf{V} = \emptyset$, PAR can represent semantics such as *Start* and *End*, which do not have set values, and for the system to comprehend that they shouldn't.

Definition 2.3.2. *A **realized semantic** $\{\lambda, \mathbf{V}, \omega\}$ has a single property $|\mathbf{V}| = 1$, and can be denoted as $\{\lambda, v, \omega\}$*

Action semantics are not readily discussed in a generalized context (evident from them not appearing in Figure 2.1 and Figure 2.2) and for the most part, are widely used and tested without being considered part of the action. Parameters such as *Lexeme*, *Frequency*, and *Scale* for gestures in the Behavioral Markup Language [65] control how the action is going to unfold. This is also true for Laban Movement Analysis Parameters [64], and commands such as *Start* and *End* in SmartBody [66]. The last two parameters are useful from an AI perspective as well, as they can be used to reason about an action's beginning and end period. It should also be noted that semantics that are not normally considered part of a graphical control structure are important to allow for a generalized framework of actions. Work such as Liu et al. [67] experiment on changing parameters linked to an psychological perception of personality(extroversion and emotional stability). The base animations themselves are the same, but its movement is changed by adding a modifier onto it. This is a action semantic change that would not effect the definition of the action, just its execution. Having a method to generally represent these parameters allows for more general action sets to be created, and reduces the total number of actions needed to describe a behavior.

From the above discussion, there is an important property to define between semantics, one of how to compare semantics between actions, objects, and other semantics. Using the earlier personality example, the agent may have a single personality vlue and the action may be able to process a set of personality values, all of which derive from a personality semantic in $\mathbb{S}$. To know that the

object personality and action personality represent the same semantic even though they have different values ($\mathbf{V}$), PAR needs some way of equating the two. We therefore define **semantic equality** using Definition 2.3.3. Semantic equality is not determined by the set of values the semantics have, but rather by the label. In Section 2.4 and Section 2.6, when actions and objects can be assigned semantics, it is important for the character to know which sets the semantics are meant to represent. That is why Definition 2.3.3 is not constrained to be semantic sets in the language, but any semantic set (as long as it has a connection back to the language). The latter portion of Definition 2.3.3 is a check to ensure that equality is only considered if the semantic exists in the language. In reality, any semantic that exists in the system should also exist in the language.

Definition 2.3.3. *Two semantics $\boldsymbol{S}_i$ and $\mathbf{S}_j$ are considered equal if $\lambda \in \boldsymbol{S}_i = \lambda \in \mathbf{S}_j$ and there exists $\mathbb{S}_k \in \mathbb{S}$ where $\lambda \in \boldsymbol{S}_i = \lambda \in \mathbb{S}_k$*

PAR's system of semantics are extremely general, encompassing both set and continuous values. However, from Definition 2.3.1, there is no guidance on how they should be created or used. In reality, $\mathbb{S}$ simply contains all the possible atomic instances that the agent may have to deal with in the simulation. This can, of course, mean that a lot of information is needed in order to fully describe the state. Work such as Pelkey and Allbeck [37] and PASTE [68] use binary properties, so that all properties are realized. By expanding the definition to include real values and sets, a more nuanced reasoning can be applied to semantics to better match other work such as Laban Movement Analysis or EMOTE, in which semantic properties exist in sets or on a continuum.

2.4 Definition of Objects $\mathbb{OBJ}$

$\mathbb{OBJ}$ is the set of all objects, such as graphical objects, that make up a virtual environment. Clearly, for virtual agents to exist in a non-static world, they need to have some understanding of their environment. The total set of objects in $\mathbb{OBJ}$ consist of a grounded representation of objects (such that they are "tangible" in the virtual world) and their generalizations (closer in an idea to Aristotle's understanding of a perfect chair). This combination of the two in one format permits virtual agents to reason at several different levels about objects, only needing to ground objects in reality during

interaction (described in Section 2.6.2 and Section 2.8). We define a single object, $obj \in \mathbb{OBJ}$ in Definition 2.4.1.

Definition 2.4.1. *Let obj be an object in the set of objects* $\mathbb{OBJ}$. *obj is defined as* $\{\lambda, p, \mathbf{S}\}$, *where*

- λ *is a unique label*
- p *is the generalization or parent object, made up of zero or one* $p \in \mathbb{OBJ}$ *and* $p \neq obj$
- $\mathbf{S}$ *is the set of semantics of obj, defined in Definition 2.3.1.*

Note the generalization p can be another object or not exist. A graphical depiction of how generalizations can be visualized can be seen in Figure 2.4. We define a *path* between objects in the standard way that paths are defined in graphs, except that due to the nature of our generalizations, the paths are in reverse order.

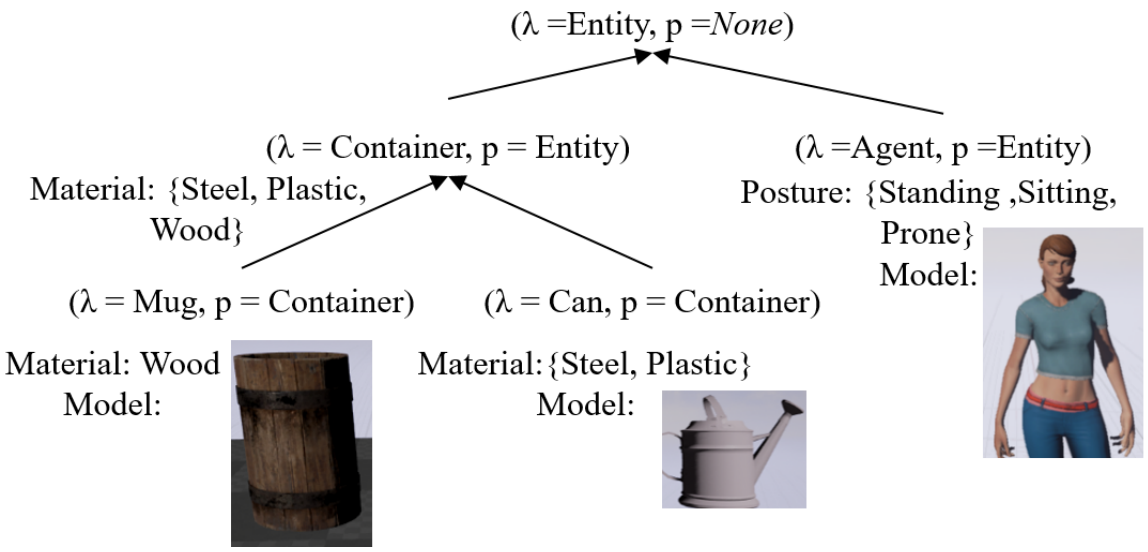


Figure 2.4: A set of sample objects connected to their generalizations. Note that in PAR, agents are also considered as objects

Definition 2.4.2. *A path*[1] *between two objects* $\mathbb{OBJ}_i$ *and* $\mathbb{OBJ}_j$*, denoted path*$(\mathbb{OBJ}_j, \mathbb{OBJ}_i)$*, exists if there is a sequence of objects* $\mathbf{OBJ} = \{\mathbb{OBJ}_i ... \mathbb{OBJ}_j\}$ *s.t* $\mathbf{OBJ}_k = p$ *and* $p \in \mathbf{OBJ}_{k-1}$ *for* $i \leq k \leq j$*.*

The definition of a path allows $\mathbb{OBJ}$ to be structured as a (possibly) connected graph. We further wish to restrict $\mathbb{OBJ}$ so that it contains no cycles. As p is the generalization of an object, then it follows that a generalization should not be a child of a more specific object. Therefore, we give Definition 2.4.3 describing that there are no cycles in $\mathbb{OBJ}$. This also means that paths only travel in on direction, and that there are no deviations in a path (there are no multiple paths, shown in Lemma 2.4.1.

Definition 2.4.3. $\forall obj_i \in \mathbb{OBJ}$*, There does not exists* $obj_j \in \mathbb{OBJ}$ *s.t.* $path(obj_i, obj_j)$ *exists and* $path(obj_j, obj_i)$ *exists.*

Lemma 2.4.1. *There exists at most one path between any two objects in* $\mathbb{OBJ}$*.*

proof: *From Definition 2.4.1,* $\forall obj \in \mathbb{OBJ}$*,* $p \in obj$ *is either another object or None. If* $obj'.p = obj, obj' \in \mathbb{OBJ}$*, then, from Definition 2.4.2, there is a single path connecting* obj' *to* obj *denoted* $path(obj', obj)$*, with the sequence being* obj, obj'*. Now, consider three objects,* $obj, obj', obj'' \in \mathbb{OBJ}$*, where* $\exists$ *path(*obj'*,*obj*) and path(*obj*,*obj''*). From Definition 2.4.1, an object has at most one generalization, which means that path(*obj'*,*obj''*) also exists, with one of the objects in the sequence being* obj*. As* $obj'.p = obj$*, then the only path from* obj' *to* obj'' *is through* obj*. Therefore, all paths are unique between any two objects in the hierarchy if a path exists.*

Definition 2.4.1 is similar to [15, 69] in that the semantics, and in particular graphical properties of the object, are attached as semantics (**S**) instead of being a named component of the object themselves. This separates the graphical from meta-data component, allowing agents to reason about types of objects instead of the graphical objects themselves. We also define semantics as non-graphical properties of the object that can be reasoned upon, such as the semantics in [25, 18]. However, we do not connect objects to actions in the semantics of objects (so we do not follow the work of Peters et al. [20]). This compartmentalizes the objects into a representation that is essentially a single parent hierarchy, with the only inter-relationship between objects being their

[1] Adapted from https://www.csee.umbc.edu/courses/undergraduate/341/fall98/frey/ClassNotes/Class14/trees.html

generalizations. However, this also allows for objects that may be used in a virtual environment to not have necessary graphical properties defined for them (such as an object not having a graphical model attached to it). In this respect, the generalizations can also be useful, as a more general object model should be defined higher up in the tree. This departs from the definition of semantics found in Pelkey and Allbeck [37], which stores the semantics on each object individually, providing the semantics to object instances. As the semantics of Pelkey and Allbeck are binary properties (i.e, every $\mathbb{S}_i \in \mathbb{S}$ is either TRUE or FALSE), generalizations of semantics have a limited effect on the simulation, meaning continuous values such as *velocity* cannot be represented, only that the object may have a velocity. Therefore, for any object, we define how semantics are stored and translated between objects in a hierarchy using Definition 2.4.4 to define *Object Semantic Assignment* and Definition 2.4.5 to define *Object Semantics Inheritance*.

Definition 2.4.4. ***Object Semantic Values****:For a given semantic* $\mathbf{S}_m \in \mathbf{S}$ *where* $\mathbf{S}$ *is the semantic set of object, and there is* $\mathbb{S}_n \in \mathbb{S}$ *where* $\mathbf{S}_m = \mathbb{S}_n$*, then the object's possible semantic property values* $\mathbf{V}$ *for*$\mathbf{S}_m$ *is* ***assigned*** *as* $\mathbf{V} \in \mathbf{S_m} \subset \mathbf{V} \in \mathbb{S}_n$

Definition 2.4.5. ***Object Semantic Inheritance****:* $\forall \mathbb{OBJ}_m, \mathbb{OBJ}_n \in \mathbb{OBJ}$*, if path*$(\mathbb{OBJ}_m, \mathbb{OBJ}_n)$ *exists, then let* ***V*** *be the values of a semantic in* $\mathbb{OBJ}_n$ *and* ***V'*** *be the values of the same semantic in* $\mathbb{OBJ}_m$*.* $\mathbf{V}'$ *is then a subset of* $\mathbf{V}$*.*

Lemma 2.4.2. *Object properties can be realized in un-instantiated objects*

Proof:*From Definition 2.4.5, for two* $\mathbb{OBJ}_m, \mathbb{OBJ}_n \in \mathbb{OBJ}$*, where* $p \in \mathbb{OBJ}_m = \mathbb{OBJ}_n$*, then* $\mathbf{S} \in \mathbb{OBJ}_m$*, denoted* $\mathbf{S}_m$ *is a subset of* $\mathbf{S} \in \mathbb{OBJ}_n$*, denoted* $\mathbf{S}_n$ *and all* $s \in \mathbf{S}_m$ *are subsets of* $s \in \mathbf{S}_n$*. Suppose* $\exists s \in \mathbf{S}_n$ *where* $|\boldsymbol{V}| \in s = 1$*. This object, and any child object, can then only take on one value during runtime, and therefore is realized. This means that the matching* $s \in \mathbf{S}_m$ *can only take on one property value, which is how un-instantiated and instantiated objects can have realized properties.*

The generalizations and semantics attached to all objects provides a powerful way for a simulation author to describe groups of objects. By forcing specification of semantics to exist on generalizations, an agent reasoning system can hierarchically reason about objects, and allow objects that fail the reasoning checks to be pruned from the system quickly.

2.5 Definition of Conditions and Assertions

The conditions field in the Parameterized Action Representation is equivalent to the prerequisites of other planning languages such as STRIPS [26] and defined in Definition 2.5.1. In effect, a condition is a statement that is resolved to either true or false. Conditions can be chained together using *AND* connectives.

Definition 2.5.1. *A condition $\phi \in \Phi$ is a conjunctive sentence that is evaluated to either TRUE or FALSE.*

Before we describe assertions, we define an important data-type to PAR that is the evaluated ending of an assertion in an action. Unlike conditions which have a boolean evaluation, the evaluation of an assertion is actually a subset of the status of an action. While this does not mean that assertions are only the subset of the status, it will be seen that the end evaluation is an important component of that. Therefore, we define the status of an action in PAR using Definition 2.5.2. Note that there are four different values for this status, signifying four base states that an action can take. Other action statuses can be added in addition to provide a more fine-grained understanding of the action (and in fact over the life-time of PAR, several have), but what we provide is the bare understanding needed for action operation.

Definition 2.5.2. *An Action status σ is a five-valued variable, which contains a value of either* NULL*,* INCOMPLETE*,* SUCCESS*,* FAILURE*, or* FINISHED*.*

Similar to the condition set Φ, the assertions on the world state are used to update the world model that occurs only upon completion of the action. An assertion on the world state performed by PAR is any non-graphical modification to the virtual environment, such as updating the contents of an agent's possessions or setting the known pose of an agent from *sitting* to *standing*. Graphical modifications brought about by motion controllers, such as setting the position of a character in the world or finishing an animation, should not be considered part of these assertions, as they may be used to determine when an action is finished. Multiple actions may run simultaneously, allowing multiple assertions to be run together. PAR has a specific assertion, the action status, that is always

in the assertion sentence of an action. The status of assertions is a subset of the total status, and may be set to either SUCCESS or FAILURE.

Definition 2.5.3. *An assertion $\delta \in \mathbb{A}$ is a conjunctive sentence that when evaluated changes the world state to match the conjunctives.*

As conditions and assertions may be connected to other conditions or assertions using *AND*, this means that both conditions and assertions may be made up of other conditions and assertions. As will be seen in Section 2.6, assertions are, in general, the end result of an action. It should also be noted that the sets Φ and $\mathbb{A}$ define all of the causes and effects that each action has on the world. Like most other things in PAR, both of these sets are parameterized, with the input values being either objects from $\mathbb{OBJ}$, actions from $\mathbb{ACT}$, or semantics from $\mathbb{S}$. The system also allows real-valued variables to be set. The end result is that conditions and assertions provide a tool for the agent to check and effect its game world.

2.6 Definition of Actions

A large component of PAR is the action set $\mathbb{ACT}$. Actions allow agents to understand and change the world state, and in essence, give the agents their volition. We define an action *act* $\in \mathbb{ACT}$ using Definition 2.6.1.

Definition 2.6.1. *Let act be an action in the set of actions $\mathbb{ACT}$. act is defined as $\{\lambda, \mathbf{P}, \mathbf{R}, \Psi, \Theta, \mathbf{S}\}$, where:*

- λ *is the name of the action,*
- **P** *are the parents of the actions, which consists of zero or more actions.*
- **R** *is a set of roles in the action, defined in Definition 2.6.9.*
- Ψ *is a set of condition-assertions that allow the action to reach a state of* SUCCESS *or* FAILURE*, as well as run any other assertions necessary. Ψ is defined in Definition 2.6.14*
- Θ *is the set of condition task pairs, defined as 2.6.5.*

Table 2.1: Set components of PAR

R	Operational Information
Ψ	Condition and Assertion statements
Θ	Task Information
S	Action Semantics

- **S** *is the set of action semantics, defined in Definition 2.3.1.*

λ and **P** are used to delineate action parents and create a hierarchy structure like the one seen in Figure 2.5. The parents **P** will either be a set of some $act \in \mathbb{ACT}$, or empty if the action does not have a parent. For the action hierarchy, we only allow one parent, and there cannot be any cycles in the hierarchy. Θ is the task set, defined as a set of hierarchical task networks, and described in Section 2.8. Θ provides conditions used to select a task. Much like the object definition, **S** is the set of action semantics. Many of the components of an action are themselves sets. A table with each set can be found in Table 2.1. The notation for these sets is that the set will be uppercase, with each individual as a lowercase element.

Similar to properties of objects from Section 2.4, we define several key properties of actions, namely, that a *path* can exist between two actions (Definition 2.6.2) and that there are no cycles between actions (Definition 2.6.3). However, there is a more relaxed definition of generalizations. An action's generalization is a set, meaning that there can be multiple paths from one action to another. This means that we do not have a unique path property, and the definitions for paths and no cycles are slightly different.

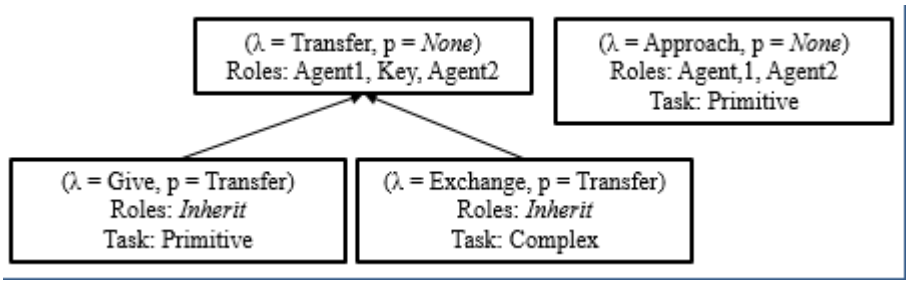
Definition 2.6.2. *A path between two actions* $\mathbb{ACT}_i$ *and* $\mathbb{ACT}_j$*, denoted as* $path(\mathbb{ACT}_j, \mathbb{ACT}_i)$ *exists if there is a sequence of nodes* $\mathbb{ACT}_i ... \mathbb{ACT}_j$ *s.t* $\mathbb{ACT}_k \in \boldsymbol{P}$ *where* $\boldsymbol{P} \in \mathbb{ACT}_{k-1}$ *for* $i \leq k \leq j$*.*

Definition 2.6.3. *No Action Cycles:*$\forall \mathbb{ACT}_i \in \mathbb{ACT}$*, there does not exist an* $\mathbb{ACT}_j \in \mathbb{ACT}$ *s.t.* $\mathbb{ACT}_j \in \boldsymbol{P}$ *where* $\boldsymbol{P} \in \mathbb{ACT}_i$ *and,* $\forall p \in \mathbf{P}$ *of* $\mathbb{ACT}_j$*,* $path(p, \mathbb{ACT}_i)$ *exists.*

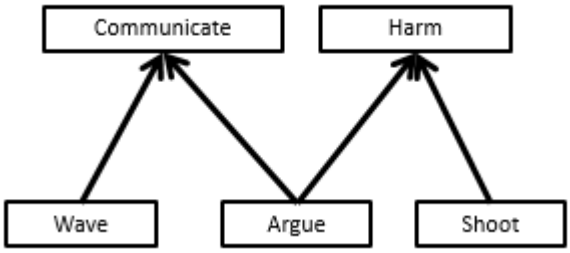
As actions have zero or more generalizations, we can compute similarity measures between actions. One similarity measure, using Wu-Palmer similarity [70], requires a *Least Common Subsumer (LCS)*. We define a Least Common Subsumer in Definition 2.6.4.

Definition 2.6.4. *A least common subsumer between two actions* $\mathbb{ACT}_i$ *and* $\mathbb{ACT}_j$*, denoted as* $LCS(\mathbb{ACT}_i, \mathbb{ACT}_j)$*, is an action* $\mathbb{ACT}_m \in \mathbb{ACT}$ *such that there exist a path between* $\mathbb{ACT}_i$ *and* $\mathbb{ACT}_m$ *and between* $\mathbb{ACT}_j$ *and* $\mathbb{ACT}_m$ *and there does not exists an action* $\mathbb{ACT}_k \in \mathbb{ACT}$ *s.t.* $|path(\mathbb{ACT}_i, \mathbb{ACT}_k) + path(\mathbb{ACT}_j, \mathbb{ACT}_k)| < |path(\mathbb{ACT}_i, \mathbb{ACT}_m) + path(\mathbb{ACT}_j, \mathbb{ACT}_m)|$

Definition 2.6.2 and Definition 2.6.3 allow the generalizations and specializations in $\mathbb{ACT}$ to form a directed acyclic graph structure. Because there is no guarantee of the connectiveness between generalizations, $\mathbb{ACT}$ can be thought of as a Forest of IsA Directed Acyclic Graphs (**FIDAG**). Two examples can be seen in Figure 2.5. Note how in Figure 2.5a, the action *Approach* does not contain any generalizations or specializations, whereas *Argue* contains two generalizations. This means that it is possible for the least common subsumer between two actions to not exist. When Wu and Palmer developed their similarity algorithm between words in WordNet [70] they allowed for a false root between all words, so that if there was no least common subsumer, their similarity algorithm would not fail. We take a similar approach, which is described in more detail in Section 2.8.3.



(a)



(b)

Figure 2.5: Two example action sets with generalizations. (a) Actions that are single entities. (b) Actions with multiple generalizations.

2.6.1 Execution of an Action

Actions inevitably need to be realized by an agent. We call the recipe for the execution of the action as the task, which is described in Section 2.8. While all of the components that make up a given action must be defined, some components are used in the execution of the action, while others are used in the execution of the task. Tasks of actions themselves fall into two categories: tasks that, depending on the state of the world *may* need to be executed, and the task that *always* needs to be executed. The ordered sequence of condition-task pairs Θ contains both of these, and is defined in Definition 2.6.5. We also describe a special condition-task pair that is required for every action, known as the *execution task* and defined in Definition 2.6.6. If the execution task is not defined for a given action, then the action may inherit from one of its parent actions, as described in Definition 2.6.8.

Definition 2.6.5. *A condition-task $\Theta_i \in \Theta$ is a 2-tuple $\{\phi, \mathbf{T}\}$, where:*

- *ϕ is unique condition sentence, defined as Definition 2.5.1*
- **T** *is the task, defined as Definition 2.8.1.*

Definition 2.6.6. *There exists a task, denoted the execution task, such that $\{\phi, \mathbf{T}\} \in \Theta$ and $\phi = \emptyset$*

Definition 2.6.7. *There may exists several tasks, denoted preparatory specs, such that $\{\phi, \mathbf{T}\} \in \Theta$ and $\phi \neq \emptyset$*

Definition 2.6.8. ***Task Inheritance**: For a given act $\in \mathbb{ACT}$, if $\nexists\{\phi, \tau\} \in \Theta$ where $\phi = \emptyset$, then the execution task is for act is the first parent $\mathbf{P}_i \in \mathbf{P}$ where $\exists\{\phi, \tau\} \in \mathbf{P}_i$ s.t $\phi = \emptyset$.*

Definition 2.6.6 shows an important design paradigm that is unique to PAR, in that there is an idea between what is necessary for an action and what is sufficient. The long-standing example of this is that, to open a door, the only animation that is *required* is an *opening* action, which of course may be broken up into a *grabbing* and *pulling* motion if the agent author has those actions instead. However, much more goes into opening a door than those actions. For instance, to open a door, one most likely has to be near a door, which may require *walking* to the door. This is a

canonical *Preparatory Specification* in that actions can be used to get into a state where the action is more likely to succeed. Other action representations, such as STRIPS, HTNs, and events, have pre-conditions that must be true for the action to begin. They then use the pre-conditions to plan other actions that are needed in order to get the world state in such a configuration that the action can be performed. Preparatory specifications in essence, cache this information at design time, so that the agent does not need to plan at run-time. Preparatory specifications consolidates design time plans. This is not to say that preparatory specifications cannot be created through planning at design time. In fact, doing so eases the authorial burden of PAR. Preparatory specifications simply remove that need at run-time. This is also why Definition 2.6.6 is important. Each action needs to have some functionality, which is represented as the *execution task*. The execution task does not need to prepare for all circumstances, as this will require processing several more components of a task than is needed. Using Θ should strike a balance between the two.

We briefly describe the algorithm used to execute an action in Algorithm 1 (Execute). Note that Algorithm 1 uses four other algorithms: Algorithm 3 in Section 2.6.3 (Evaluate Conditions), Algorithm 2 (Find), Algorithm 4(Enumerate) in Section 2.8, and Algorithm 5(Step) in Section 2.8.

Executing an action contains many parts. For easy understanding, we also provide a flowchart of the algorithm in Figure 2.6. In Figure 2.6, green lines signify the path taken when *Evaluate Conditions*(Algorithm 3) returns either *SUCCESS* or *FAILURE*. Red lines indicate a return of *INCOMPLETE*. Black lines are default execution steps that are always taken when there is no path.

In Algorithm 1, there are certain steps that are called *WAIT*. PAR execution is based off of a light-weight thread model, and execution of the action is halted at given frequencies. When multiple actions are run together in a task (see Section 2.7), there is a lock-step mechanism for allowing part of each task to run.

In order to know which tasks need to be executed, we define a *Find* procedure in Algorithm 2. The astute reader will notice that all conditions are examined in Find, and are matched to equality. Recall that all Ψ have unique condition strings ϕ. This does not preclude them from evaluating to the same value, so that multiple conditions may build up a task. The only exception to this is

Algorithm 1 The execution of an Action

```
1:  function EXECUTION ACTION(ACT)
2:      ACT ←The current action being executed, defined in Definition 2.6.1
3:      finished← the action status σ, defined in Definition 2.5.2
4:      finished←Evaluate(Ψ,0)
5:      if finished ≠ INCOMPLETE then
6:          return finished
7:      WAIT
8:      prep_task←Build(Θ,True)
9:      if prep_task ≠ ∅ then
10:         repeat
11:             Enumerate(prep_task)
12:             Step(prep_task)
13:             WAIT
14:         until prep_task.status = SUCCESS or prep_task.status = FAILURE
15:     exec_task ← Build(Θ,∅)
16:     repeat
17:         Enumerate(exec_task)
18:         Step(exec_task)
19:         finished←Evaluates(Ψ,1)
20:         WAIT
21:     until finished ≠ INCOMPLETE
22:     return finished
```

the execution task, whose condition is the empty set $\emptyset$. This means that execution steps are always created prior to simulation. Preparatory specifications must be defined prior to simulation, but the actual task is not known until run-time. Furthermore, Algorithm 2(Build) places a restriction on preparatory specifications that they are sequentially designed. This is a simplification based on the set of conditions, and a simulation author may override this by writing condition strings that are specific to each world state.

From Algorithm 1(Execute), any action that an agent performs at run-time must have an execution task, or the agent will not know how the action should be realized. We relax this requirement slightly in Definition 2.6.8. Definition 2.6.8 uses Definition 2.6.2 to inherent tasks. This will have further implications in Section 2.8.

2.6.2 Roles R

The set of operational items used in the action are the role set **R**. Recall from Table 1.1 that connection between objects and actions may be a field of either the objects or the actions. We refer to

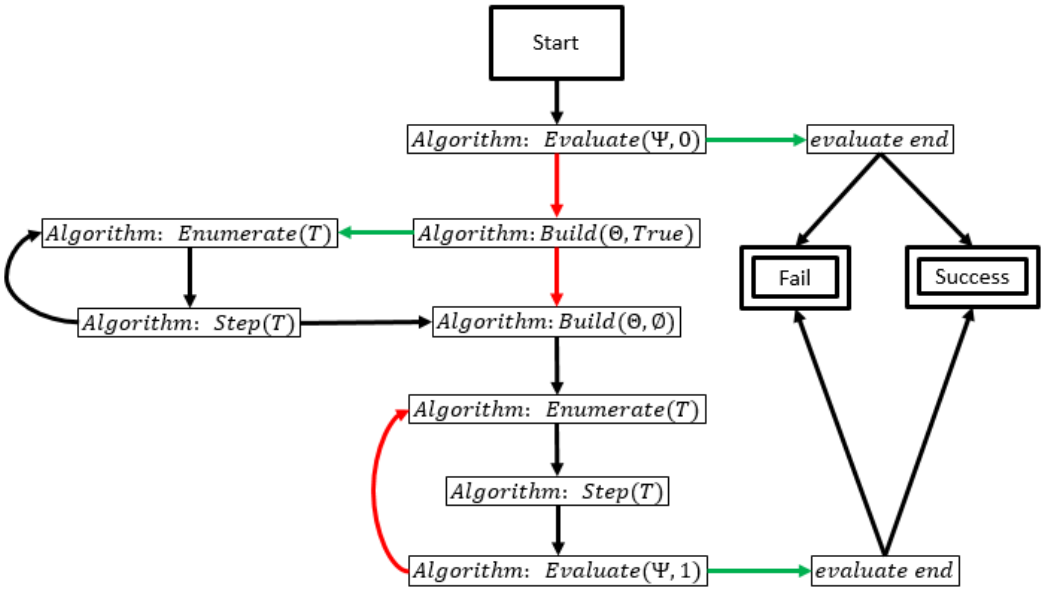


Figure 2.6: The processing steps for executing an action in PAR. This is a flowchart of Algorithm 1

.

this connection as the roles, and specifically when a field of an action as operational information (although it has also been described as affordances). From the original definition of PAR, each operational object in the role set can either be a necessary object participant, whose inclusion is required in order to perform the action, or augmentation objects, which are not required to perform the action but allow the action to be more expressive. Roles are defined in Definition 2.6.9 as a component of the action, which differs from interactional representations such as SmartObjects [18].

Definition 2.6.9. *A role $r \in R$ is a 2-tuple $\{\mathbf{OBJ}, \nu\}$, where:*

- **OBJ** *is the set of objects that can satisfy the requirements of the role. Each obj $\in$* **OBJ** *is representative of an object in $\mathbb{OBJ}$, defined from Definition 2.4.1.*
- ν *is a boolean representation signifying whether the role is necessary for the action to be used.*

Using an example of the action *cooking*, a necessary object, where $\nu = 1$, would be the *food* to cook, while a *spoon* would be an augmentation object, which is not necessary for cooking. This is different from the definition of operational objects seen in Erol et al. [42] or *events* [1], which only

Algorithm 2 Task Builder

```
1:  function BUILD(Θ,equality)
2:      Θ ←The set of condition-task passed in from an action
3:      equality ← The condition φ is being compared to
4:      finished ← empty set (∅) of actions
5:      for all (φτ) ∈ Θ do
6:          if evaluation of φ = equality then
7:              add τ to actions
8:      if |finished| > 1 then
9:          return AND conjoining all actions in finished
10:     return finished
```

consider necessary objects. Furthermore, note that **OBJ** is a set of objects, meaning that multiple objects may be used in the role, and that a single object is not assigned to that role until run-time.

We define three properties of roles necessary to reason over and realize actions: *equality*, *similarity*, and *specialization*. The first two properties, seen in Definition 2.6.10 and Definition 2.6.11 allow us to reason about the roles between actions. We show the use of role equality in organizing action parents in Chapter 3. The last property, seen in Definition 2.6.12, is used when an action is being realized (instantiated in the world). We also define the behavior of parent action roles in Definition 2.6.13.

Definition 2.6.10. ***Role Equality****: Let* **O1** *be* **OBJ** $\in \mathbf{R}_i$ *and* **O2** *be* **OBJ** $\in \mathbf{R}_j$ *for two roles* $\mathbf{R}_i$ *and* $\mathbf{R}_j$*.* $\mathbf{R}_i$ *is equal to* $\mathbf{R}_j$ *if* $\forall obj \in$ **O1**,$obj \in$ **O2** *and* $\forall obj \in$ **O2**,$obj \in$ **O1**.

Definition 2.6.11. ***Role Similarity****: Let* **O1** *be* **OBJ** $\in \mathbf{R}_i$ *and* **O2** *be* **OBJ** $\in \mathbf{R}_j$ *for two roles* $\mathbf{R}_i$ *and* $\mathbf{R}_j$*.* $\mathbf{R}_i$ *is similar to* $\mathbf{R}_j$ *if* $\exists obj \in$ **O1** *that also exists in* **O2** *or* $\exists obj \in$ **O2** *that also exists in* **O1**.

Definition 2.6.12. ***Role Instantiation****: For the set of possible objects that can fill a role,* ***OBJ****,* $\exists obj \in$ ***OBJ*** *and* $\mathbb{OBJ}_i \in \mathbb{OBJ}$ *with path(*$\mathbb{OBJ}_i$*,obj), then* $\mathbb{OBJ}_i$ *can be used to instantiate the role.*

Lemma 2.6.1. *Two roles are* ***similar*** *if an object in the set of objects in one role can be used to instantiate another role*

proof*:Consider two roles,* $\mathbf{R}_i$ *and* $\mathbf{R}_j$*, where* $\mathbf{R}_i \neq \mathbf{R}_j$ *and* $\{\mathbf{O1}, v\} = \mathbf{R}_i$ *and* $\{\mathbf{O2}, v\} = \mathbf{R}_j$*. Now, suppose their are two objects,* $o1 \in$ **O1** *and* $o2 \in$ **O2** *where path(o2,o1) exists. Then, by*

Definition 2.6.12, o2 can be used to instantiate $\mathbf{R}_i$. *Since o2 can be used in* $\mathbf{R}_i$, $\mathbf{R}_i$ *and* $\mathbf{R}_j$ *are similar.*

Definition 2.6.13. *If the set of roles* **R** *in a given action act are empty, but parent actions have roles, then the roles* $\boldsymbol{R} \in$ *act are the union of all parent sets in the action.*

Definition 2.6.10 and Definition 2.6.11 have a profound effect on computing the differences between roles, and converting one role to another. This is useful in agent reasoning systems, when an agent is comparing two actions and is considering the roles (and in some cases, the locations of the objects [71]). Role similarity is also important for tasks, as actions in a task may require a role be similar in order for the task to operate (described in Section 2.8). While roles and actions are created at design-time, an agent may be required to plan (or re-plan in the case of a failure) at run-time. Definition 2.6.11 coupled with Lemma 2.6.1 provide a loose comparison between roles, while Definition 2.6.10 provides an exact comparison between them. We cn also compare roles by defining a distance function based on the symmetric difference between two sets. This is defined in Equation 2.1. Note that the meaning of union and intersection is determined by whether roles need to be exact or similar. We show the effects of this on a test data-set in Section 2.9 and show the use of Equation 2.1 on comparing actions in the case study(Section 2.9).

$$A\Delta B = |(A \bigcup B)| - |A \bigcap B| \tag{2.1}$$

2.6.3 Condition and Assertions

The condition and assertion set Ψ defines the conditions that allow an action to complete, and the subsequent effects on the world Ψ is an ordered set. In many representations (STRIPS and events), the condition ϕ and assertion on the world δ are separate entities. This is because there is only one ϕ and δ for each action and so action failures [72] are not represented. Furthermore, planning systems such as Kontopoulous et al. [73] found that only having one ϕ and δ were not sufficient for their PDDL planning system, and that "conditional-effects" allow them to plan over changes in their resources (similar to objects). Therefore, a single ϕ and δ is insufficient to fully

describe the path an action may take. By using Ψ, the transition between the conditions and actions are better represented. Ψ allows for the conditions necessary for an action to succeed, fail, or even be describable by a simulation author, providing a more nuanced understanding of actions. Definition 2.6.14 defines a condition assertion in the ordered set.

Definition 2.6.14. *A condition-assertion of an action $\psi \in \Psi$ is defined as $\{\phi, \delta, e\}$, where:*

- *ϕ is a unique condition from the set of conditions Φ, defined in Definition 2.5.1.*
- *δ is an effect from the set of effects Δ, defined in Definition 2.5.3. Each delta ends with a σ(Definition 2.5.2) with either the value of* SUCCESS *or* FAILURE
- *e is a boolean flag signifying when ϕ is tested for its truth value*

Note that *e* is not found in any other representation, and describes when a condition is tested in the action. In reality, there are two times in which an action that is being realized may be considered complete, either before the action begins or while the task is being realized. When *e* is set to 0, then the condition is only tested before the action starts, and is known as *applicability conditions*. In the original definition of PAR [27], the authors described the need for applicability conditions, to determine if the action can even be run by the character. This is also described as pre-conditions in STRIPS [26]. For these condition assertion pairs, δ contains FAILURE as part of its connective sets. However, PAR treats tasks as pre-authored hierarchical task networks, which means that all of the desired action's effects may already be realized before the action begins. Therefore, applicability conditions in PAR are extended to have σ which contain SUCCESS as well. Because the *applicability conditions* describe if an action should, or even can, run, if no $\phi \in \Psi$ is found to be true, then the action's status is INCOMPLETE. A status of INCOMPLETE means that the action did not finish, and should continue in its evaluation. Therefore, the conditions should be written when the status of an action is resolved to either SUCCESS or FAILURE, and it is not assumed that the conditions always tend to one or another.

In addition to condition-assertions that occur at the beginning of an action, PAR has condition-assertions that occur during the execution of a task, when $e = 1$. This is analogous to culmination conditions and post assertions found in the original definition of PAR, but allows for actions to

evaluate to FAILURE in addition to SUCCESS. There has been a lot of work on failures in action execution [72, 74], and the inclusion of them in culmination conditions allows for failures to be considered in planning, as well as providing a nice symmetry between applicability conditions and culmination conditions. The agent evaluates Ψ using Algorithm 3, which determines which, if any condition is true, and executes the assertions on the world state if it is.

Algorithm 3 Evaluation of a condition set

```
1:  function EVALUATE(Ψ,type)
2:      Ψ ← The condition set being evaluated
3:      type← The conditional boolean the agent is processing
4:      End← The status σ of type SUCCESS,FAILURE,or INCOMPLETE
5:      End←INCOMPLETE
6:      for all {φ, δ, e} ∈ Ψ do
7:          if e = type then
8:              finished ← Evaluation of φ
9:              if finished = True then
10:                 End ← Evaluation of δ
11:     return END
```

From Evaluate(Algorithm 3), It is important to note that the ordering of $\{\phi, \delta\} \in \Psi$ matters. It is possible that the world state is in such a configuration that multiple conditions are true at the same time. As conditions are evaluated sequentially, race conditions will not exist, as described in Theorem 2.6.1. Therefore, the agent author must take into consideration which condition-effects are more important, and have those at the beginning of the ordered set $\Phi\Delta$.

Theorem 2.6.1. *An action will never perform more than one condition-assertion for an action*

***Proof**: Ψ is an ordered set. From Evaluate (Algorithm 3), the ordered set is executed sequentially for truth values. Therefore, $\forall\psi \in \Psi$, Ψ_i is executed before Ψ_j, as long as $i < j$. Also from Algorithm 3, a state-shift occurs when the action is set to either* SUCCESS *or* FAILURE*. From Definition 2.5.3, all assertions must contain either* SUCCESS *or* FAILURE*. Therefore, if $\phi \in \Psi_i$ is found to be true, $\delta \in \Psi_i$ will cause the action to end. Because the examination of Ψ is sequential, $\phi \in \Psi_j$ will not be evaluated, and so there will never be two conditions that are at the same time for*

the same action.

As Theorem 2.6.1 shows, condition-assertions will only allow one δ statement to execute for a given action. This affords flexibility in the action, as competing $\delta \in \Psi$ can be written into the action at design-time and due to Theorem 2.6.1, only one assertion will ever be evaluated. Thus, the total amount of actions that need to be written by a simulation author in the system can be compacted and reduced.

2.6.4 Semantics of Actions S

In addition to the semantics of objects, PAR's action may also have meaning attached to them. This is a departure from other action representations such as *events* or *Hierarchical Task Networks*, which do not contain action semantics. From a natural language perspective, actions should have semantics in the form of action descriptors (such as adverbs). Therefore, we define properties of action semantics, which are analogous to the object semantic definitions of Section 2.4.

Definition 2.6.15. ***Action Semantic Values****:For a given semantic* $\mathbf{S}_m \in \mathbf{S}$ *where* $\mathbf{S}$ *is a semantic set in an action, and there is* $\mathbb{S}_n \in \mathbb{S}$ *where* $\mathbf{S}_m = \mathbb{S}_n$*, then the possible values* $\mathbf{V} \in \mathbf{S_m}$*are a subset of*$\mathbf{V} \in \mathbb{S}_n$

Definition 2.6.16. ***Action Semantic Inheritance****:*$\forall \mathbb{ACT}_m, \mathbb{ACT}_n \in \mathbb{ACT}$*, if path*$(\mathbb{ACT}_m, \mathbb{ACT}_n)$ *exists, then let* $\boldsymbol{V}$ *be* $\boldsymbol{V} \in \mathbb{ACT}_m$ *and* $\boldsymbol{V}$*' be* $\boldsymbol{V} \in \mathbb{ACT}_n$*. Every* $\mathbf{V_i} \in \boldsymbol{V}$ *is also in* $\mathbf{V}'$ *and the values of* $\mathbf{V_i}$ *are a subset of its corresponding semantic in* $\mathbf{V}'$*.*

Lemma 2.6.2. *Action can have realized properties*

Proof:*From Definition 2.6.16, for two* $\mathbb{ACT}_m, \mathbb{ACT}_n \in \mathbb{ACT}$*, where path(*$\mathbb{ACT}_m$,$\mathbb{ACT}_n$*), then* $\mathbf{S} \in \mathbb{ACT}_m$*, denoted* $\mathbf{S}_m$ *contain a subset of the properties of* $\mathbf{S} \in \mathbb{ACT}_n$*, denoted* $\mathbf{S}_n$ *and all* $s \in \mathbf{S}_m$ *are subsets of* $s \in \mathbf{S}_n$*. Suppose* $\exists s \in \mathbf{S}_n$ *where* $|\boldsymbol{V}| \in s = 1$*. This object, and any child object, can then only take on one value during runtime, and due to Definition 2.3.2, the semantic is realized.*

Lemma 2.6.2 allows us to create actions that have singular semantics attached to them. This is a useful and important property of action semantics, as it allows a simulation author the ability

to implant concrete understandings of action sets before run-time. For example, if an author has three animations, *Run*, *Walk*, and *Jog*, the only difference in the understanding of these three actions are the animation itself and the movement speed. By creating three separate actions connected to one parent action *Movement*, action instantiation can contain the movement speed for each three actions, while allowing other knowledge of the action (the roles and condition-assertions) to remain the same. We can see an example of this in Figure 2.7. In Figure 2.7, the actions and animations are blended together based on a single property, *speed*. These two action semantics control which animation and how much of each animation are played when a *movement* action is called. The values each semantic has are then a set to be realized at runtime. Each animation is in that set is then a subset of that property (the standing action is not played above a certain speed threshold where walk begins). Lemma 2.6.2 describes this property, which is often used to create blend spaces but is not generally in action representations.

2.7 Conjunctives

While actions are considered atomic units, it should be noted from Figure 2.3 and the definition of an action's task that actions can be combined together in order to create more complex behaviors. While the exact method for doing so is described in Section 2.8, we first describe the kinds of connectives that are defined in PAR. While we realize that these are not the only way of connecting actions together, for now, it provides a strong base. We define six types of connectives in $\mathbb{F}$: $\{$AND, OR, PARINDY, PARJOIN, WHILE, GATHER $\}$, which are similar to the nodes described in Marzinotto et al. [61].

The three sequential conjunctives, AND and OR, and GATHER operate by evaluating each action that is a child of the conjunctive, and whose behavior can be seen in Figure 2.8. In Figure 2.8, at each action stage, the links show the possible outcome at each step of composite node. For an AND conjunctive, any action that fails will halt the execution of that entire branch. Likewise, an action that is successful in an OR conjunctive halts the execution of the entire branch. This is similar to composite in behavior trees. GATHER is a special connective, in that it operates on a collection of items. This connective enumerates through a collection of items, preforming the same

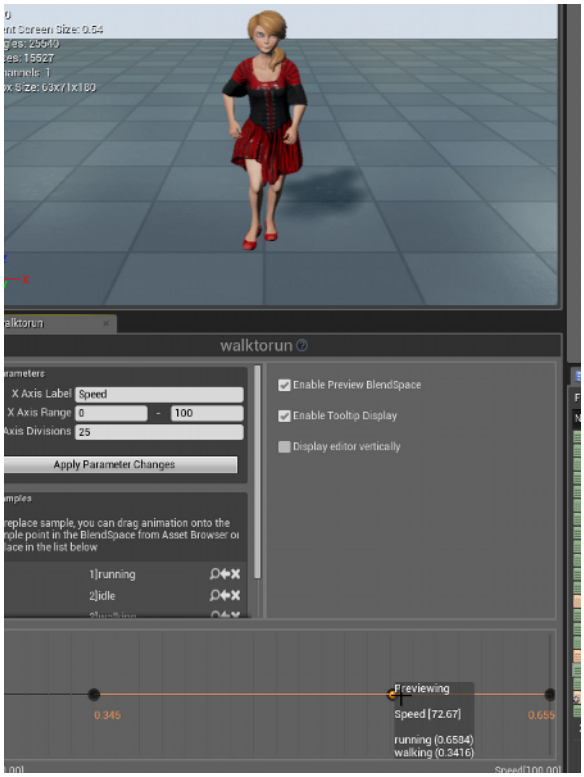
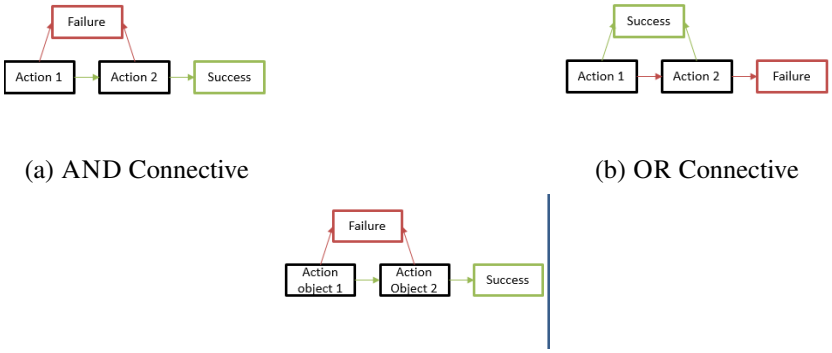


Figure 2.7: A blend space between three animations in a single action. This one dimensional blend is controlled by a speed parameter

.

action on each item individually. It can be thought of as an AND node that connects the same action using different objects. The failure of one action then is the failure of the entire branch. Because GATHER connects an unknown number of actions together at design time, it is considered a special conjunctive, in that GATHER may only operate on a single action. This is similar to a *foreach* node of [75].



(a) AND Connective

(b) OR Connective

(c) GATHER connective

Figure 2.8: A graphical depiction of the three sequential connectors, (a) AND,(b) OR, and (c) GATHER.

The three parallel conjunctives, (PARINDY, PARJOIN, and WHILE)[2] execute all actions on their branch at the same time, with a graphical representation of PARINDY and PARJOIN depicted in Figure 2.9. Their difference lies in how the individual actions and branches advance and halt execution. PARINDY evaluates all actions, and halts the execution of the branch when one action finishes. The result of that branch is then the result of the first action to finish. Likewise, PARJOIN halts the execution when the last action finishes, with the result of the branch being the result of the last action to finish. Finally, WHILE advances each step (from the execution algorithm in Algorithm 1) together. So, some actions that do not have preparatory specifications will halt and allow the actions that do to evaluate those first before continuing with the execution steps. If any action is to fail at any time, all actions will fail, so that while acts as a parallel AND.

[2]The names of these connections are due to historical reasons

(a) PARINDY Connective (b) PARJOIN Connective

Figure 2.9: A graphical depiction of two parallel connectors.

2.8 Explanation of Tasks T

Until this point, the descriptions of PAR and its components have been mainly focused on the descriptions of actions and objects and how those are used to define a knowledge base for agents. For a virtual character to realize and perform motion data using the virtual objects in their environment, there needs to be some explanation of how all the components of an action come together, that is, how the components should be realized by characters. Furthermore, actions defined so far are autonomous sets, but it has been well documented that actions can be a composite of other actions in the set [43, 56]. We designate *tasks*(**T**) to represent how actions are connected and realized for virtual characters.

We define a task **T** as an annotated, rooted tree, where each leaf node is an action $act \in \mathbb{ACT}$ and each interior node is a connector $f \in \mathbb{F}$. Formally, we define a task using Definition 2.8.1 and provide some terminology in Definition 2.8.4 and Definition 2.8.5. Also, as a task is a tree, we use terminology consistent with trees, such as *children*, *parents* and so forth. Definition 2.8.1 is a vast departure from the original definition of HTNs in Erol et al., and is due to the strides the community has made in representing tasks as a whole [56, 3]. This is also different from the definition of behavior trees found in work like [76, 61] as behavior trees allow for conditions to exist on the task network, whereas we only allow for actions and conjunctives. Task Networks have many different ways of connecting actions together, including parallel execution and selections of actions, and the conjunctives in PAR are described in depth in Section 2.7. We use this representation in opposition to *decorators* found in behavior trees. This is because most of the behavior tree decorators can be replicated using action semantics, described in Section 2.3.

Definition 2.8.1. *A task* $\mathbf{T}$ *is a directed, rooted tree constructed of nodes* $\mathbf{T}_i \in \mathbf{T}$ *where:*

- *Interior nodes are derived from* $\mathbb{F}$
- *Leaf nodes are act* $\in \mathbb{ACT}$*, defined in Definition 2.6.1*
- *Each non-leaf node, with the exception of a node specified as* GATHER *has two or more child nodes*

Definition 2.8.2. *A path between two nodes* $\mathbf{T}_i$ *and* $\mathbf{T}_j$*, denoted path(*$\mathbf{T}_i$*,*$\mathbf{T}_j$*) exists if there is a sequence of nodes* $\mathbf{T}_i ... \mathbf{T}_j$ *s.t* $\mathbf{T}_k$ *is a parent of* $\mathbf{T}_{k+1}$ *for* $i \leq k \leq j$ *or* $\mathbf{T}_k$ *is a parent of* $\mathbf{T}_{k-1}$ *for* $i \leq k \leq j$*.*

Definition 2.8.3. *The height of a task is the length of the longest path from the root node to any leaf node.*

Definition 2.8.4. *A* ***primitive task*** *is a task with a height of one. It is therefore made up of a single* $\tau \in \mathbf{T}$*.* τ *is then a realizable function that controls the pragmatics of the action.*

Definition 2.8.5. *A* ***complex task*** *is a task with height* > 1

Note that Definition 2.8.2 describes paths that travel from parent to child nodes. This is different than a path for an object (Definition 2.4.2) or action (Definition 2.6.2), which only observes paths as generalizations. The reasoning behind this is that the main purpose of a task is to be enumerated by the agent at run-time. Therefore, the agent always starts to examine a task from the root, whereas the agent will usually examine object and action types based on the grounded world (and therefore from a bottom-up perspective).

There is a circular, serendipitous relationship between actions and tasks. Tasks are composed of actions, and every action that can be realized by an agent has a task or can inherit a task (Definition 2.6.8). The breakup of actions and tasks are not found in hierarchical task networks. While events do separate out the animation component from the reasoning component (in terms of Parameterized Behavior Trees), parameterized behavior trees are not comprised of events, leading to the need for both PBTs and other events to be defined. The original definition of events does not

even contain this breakup, but instead assumes it, leading to the conclusion that PBTs can contain events, and that the animation is not removed from the reasoning. By defining actions as is done in Definition 2.6.1, we remove most of the reasoning components from the task itself.

2.8.1 Annotation

Each leaf node of the task will have one or more pieces of semantic data, henceforth called annotations of the tree. Specifically, as all roles of actions can be types of objects (objects in which *obj* is not connected to a graphical object), grounded objects (ones that have graphical objects) must be filled in during run-time. This is described on the task by annotating each leaf node with *obj* where *obj* can be used to instantiate the role, i.e Definition 2.6.11. We further consider non-role annotations to be action semantic specific, in that semantics from the task's parent action (*act*, where $\mathbf{T} \in \Theta$ and $\Theta \in act$) can be modified and passed to each leaf in the action. An example of tree annotations can be seen in Figure 2.10. From Figure 2.10, it should be seen that the same sub-action is run four times in parallel (Approach), with the difference being that the agent is a different guard each time. This forces restrictions on the parent action *act* based on the task. We describe how these restrictions manifest using Definition 2.8.6.

Definition 2.8.6. ***Roles used in tasks****: $\forall \mathbf{T} \in \Theta$, for some Θ in an action $\mathbb{ACT}_m$, if $\exists \mathbf{T}_i \in \mathbf{T}$ s.t $\mathbf{T}_i \in \mathbb{ACT}$, denoted $\mathbb{ACT}_i$, then the roles $\mathbf{R} \in \mathbb{ACT}_i$ must be contained in $\mathbf{R} \in \mathbb{ACT}_m$ for all necessary roles (where $\nu = 1$), based on role similarity (Definition 2.6.11). This is because if $\mathbf{T}_i \in \mathbb{ACT}$, then it contains at least one required role, that is, it contains the agent performing that action. Also, if a task cannot instantiate necessary roles for the sub-task, then the sub-task cannot be performed.*

There is one special annotation that has been seen in *Execute*(Algorithm 1), the status of the task. In reality, the status is a run-time value, and is used to understand the evaluation state of the task. In fact, each node in PAR has a status, as will be seen in *Enumerate*(Algorithm 4). All nodes start with a status of NULL, signifying that the node has not begun. While the node is being enumerated, it has a value of INCOMPLETE, and when its evaluation is completed, has the result of the executed assertion. Unlike role and semantic annotations, the status is not set by a simulation

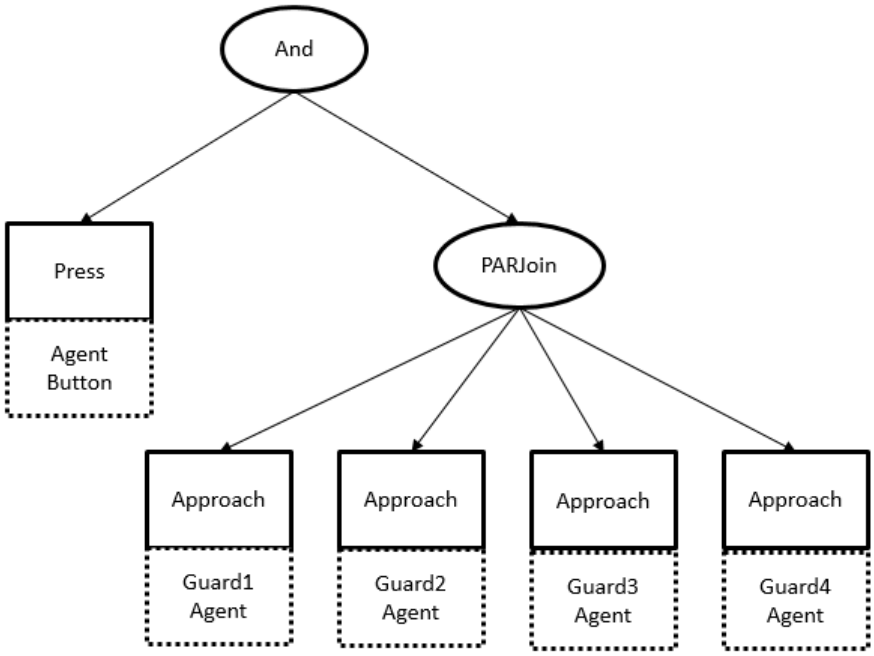


Figure 2.10: The task for the action *SoundAlarm* with its associated roles, shown as annotations of each leaf node in the task. This task was converted from [1] and can be found in Appendix A.

author, but is instead processed by the system. This does not mean that it cannot be accessed at run-time, simply that there is no need for an author to pre-set the status.

2.8.2 Task Grammar and Representations

By definition 2.8.1, a task is a tree, with certain properties for its interior and exterior nodes. Therefore, we can define a grammar for a task as the following:

1. $\tau \rightarrow act$
2. $\tau \rightarrow$GATHER(act)
3. $\tau \rightarrow F(\tau')$
4. $\tau' \rightarrow \tau\tau^{+}$
5. $F \rightarrow$ AND|OR|PARINDY|PARJOIN|WHILE

Rule 2 in the tree grammar shows the importance of GATHER as a special action form. All other conjunctives can appear as many times as necessary, and will have several other nodes attached to them. Note that the tree grammar provides a string representing a pre-order traversal of the tree. We can perform a post-order traversal by modifying rules two and three, seen below. We will subsequently define when the post-order modified grammar is used by designating a *PRE* or *POST* string.

2. $\tau \rightarrow (act)\text{GATHER}$

3. $\tau \rightarrow (\tau')F$

The task grammar allows us to now define two functions used in the execution of actions from *Execution*(Algorithm 1), *Enumerate*(Algorithm 4) and *Step*(Algorithm 5). These two algorithms deal directly with the task, with *Enumerate* describing how the system knows which branches of the task are the currently active ones. Because nodes on tasks can be annotated, the system uses this to keep track of which actions are currently running on the task. *Step* performs the execution of each node in the task.

Enumerate decides when to execute actions in the task, based on when the task is written. Notice that all nodes receive a task status, described in Definition 2.5.2. The status FINISHED is only used in parallel executions that require all actions to complete in order for that conjunctive to complete. The only action whose status matters in a PARJOIN connective is the last one. Once an action connected to a JOIN conjunctive has finished (and run its appropriate effects) that condition does not matter, and so the system only needs to remember that it has run and that it is no longer running.

The INCOMPLETE status is used in tandem with *Step*, and describes how the system executes actions. Line 5 of *Step* shows the interplay between tasks and actions, and describes how the system processes actions hierarchically. Each action has an associated task **T**, either built in the prep-specs or written by the simulation author before-hand in the execution steps. Each leaf node, as an action in $\mathbb{ACT}$, may therefore have another associated **T**, and so the process perpetuates. It is important to note that in Algorithm 5 (Step), each conjunctive (internal) nodes is ignored. The processing for conjunctive nodes is handled in *Enumerate*.

Algorithm 4 Enumerating a Task to Prepare it for execution based on PRE grammar

```
1: function ENUMERATE(T)
2:     T ← The task (or branch) to be enumerated, represented as a tree
3:     τ ← root of T
4:     if status of τ ≠ INCOMPLETE then
5:         return status of τ
6:     if τ ∈ 𝔸ℂ𝕋 then
7:         status of τ =INCOMPLETE
8:         return status of τ
9:     if τ is AND or τ is GATHER then
10:        children ← children of τ
11:        if any children status is FAILURE then
12:            set status of τ to FAILURE
13:            return status of τ
14:        cur ← the first node in children whose status is NULL or INCOMPLETE
15:        if status of cur =INCOMPLETE then
16:            return status of cur
17:        else
18:            return Enumerate(cur)
19:    if τ is OR then
20:        children ← children of τ
21:        if any children status is SUCCESS then
22:            set status of τ to SUCCESS
23:            return status of τ
24:        cur ← the first current running or incomplete node
25:        if status of cur =INCOMPLETE then
26:            return status of cur
27:        else
28:            return Enumerate(cur)
29:    if τ is PARINDY then
30:        children ← children of τ
31:        if any children status is SUCCESS then
32:            set status of τ to SUCCESS
33:            return status of τ
34:        if any children status is FAILURE then
35:            set status of τ to FAILURE
36:            return status of τ
37:        for all child ∈ children do
38:            Enumerate(child)
39:        set status of τ to INCOMPLETE
40:        return status of τ
41:    else
42:        children ← children of τ
43:        fin ← number of finished children, initially zero
44:        stat ← The status of all children that completed in the last iteration
45:        for all child ∈ children do
46:            Enumerate(child)
47:            if status ofchild = SUCCESS or status of child = FAILURE then
48:                stat ←status of child
49:                set status of child to FINISHED
50:            if status of child = FINISHED then
51:                fin ← fin + 1
52:        if fin = |children| then
53:            set status of τ to stat
54:        return status of τ
```

Algorithm 5 Executing a Task Step

```
1: function STEP(τ)
2:     τ ← The task step to be executed
3:     for all τ_i ∈ τ do
4:         if status of τ_i =INCOMPLETE and τ_i ∈ 𝔸ℂ𝕋 then
5:             finished ← execute(τ_i)
6:             if finished = SUCCESS or finished = FAILURE then
7:                 set status τ_i to finished
```

2.8.3 Measure for Behavior Similarity

For comparison of behaviors, a consistent representation is needed, one that allows all components of the tree to be compared. To do so, we convert each task from a tree into a post-order string. An example of this can be seen in Figure 2.11a being converted to Figure 2.11b. In order to compare components of sub-trees, we then convert the regular post-order string into a non-regular grammar. To do so, we count the immediate children of each conjunctive term, converting to our post-order representation to the representation seen in Figure 2.11c. The representation seen in Figure 2.11c allows a quick understanding of the total number of children attached to any given conjunctive. It removes the ability to quickly see which children are attached to which conjunctives.

To compare two tasks **T1** and **T2**, we convert both behaviors into the representation seen in Figure 2.11c. From Figure 2.11a, we perform a post order traversal of the tree, giving us the representation seen in Figure 2.11b. So that we can compare the number of children attached to each node, we then do a non-linear transformation to Figure 2.11c. This describes how many children are under each connective explicitly. We then compute the edit distance between each tasks using Equation 2.2. In Equation 2.2, each character in linear format between the two strings is compared, solving smaller sub-problems to determine the total similarity. The cost of each step is calculated as c, and d is the distance metric, seen in Equation 2.3 and Equation 2.4. The total cost is then calculated as $c(\mathbf{T1}_{|\mathbf{T1}|}, \mathbf{T2}_{|\mathbf{T1}|})$ or the cost after comparing all parts of **T1** and **T2**.

$$\begin{aligned} c(\mathbf{T1}_i, \mathbf{T2}_j) = d(\mathbf{T1}_i, \mathbf{T2}_j) + min(&c(\mathbf{T1}_{i-1}, \mathbf{T2}_j), \\ &c(\mathbf{T1}_i, \mathbf{T2}_{j-1}), \\ &c(\mathbf{T1}_{i-1}, \mathbf{T2}_{j-1})) \end{aligned} \tag{2.2}$$

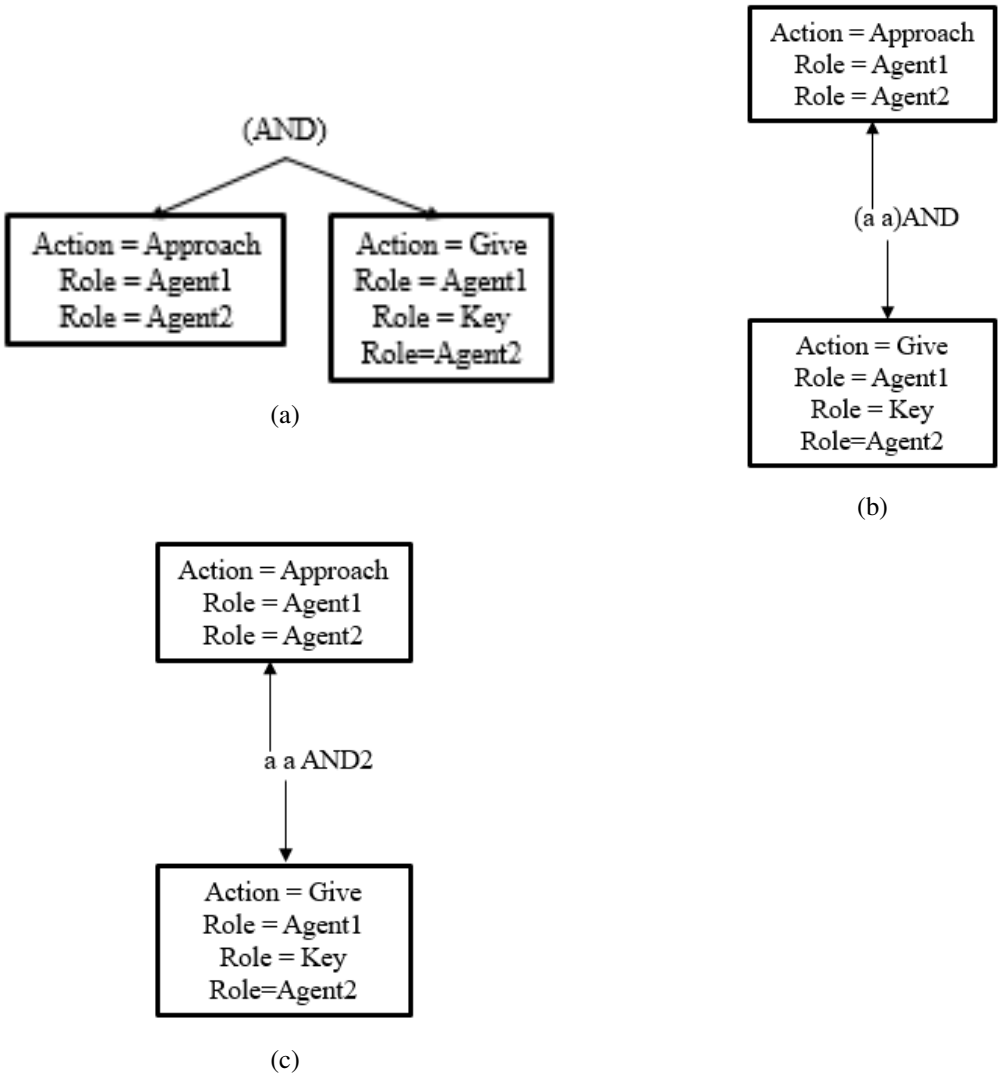


Figure 2.11: Three representations for the task Exchange, adapted from [1] using their name. (a) The task as a tree. (b) A post-order representation of the task for *Exchange*. (c) Our processed representation of the task for *Exchange*

The edit cost between two components of each behavior is an important part of the equation. Before determining the edit cost, we determine the similarity between the two component of the tasks using the generalizations of actions. We use the equation seen in Equation 2.3 if $\mathbf{T1}_i$ and $\mathbf{T2}_j$ are primitive actions. In Equation 2.3, the similarity is controlled by a least common subsumer (*lcs*), using Definition 2.6.4. Recall that the *lcs* of two primitive actions is the generalization that most closely defines the actions. Examining Figure 2.5a, the *lcs* of *Exchange* and *Give* is *Transfer*. The *lcs* of *Approach* and *Give* does not exist. To compensate, we consider a false root to connect all disjoint action sets. This makes the depth of an *lcs* of *Approach* and *Give* to be 1. If both components are connective pieces, $d(\mathbf{T1}_i, \mathbf{T2}_j)$ is solved using Equation 2.4.

$$d(\mathbf{T1}_i, \mathbf{T2}_j) = \frac{2 * depth(lcs)}{depth(\mathbf{T1}_i) + depth(\mathbf{T2}_j)} * Annotations(\mathbf{T1}_i, \mathbf{T2}_j) \tag{2.3}$$

$$d(\mathbf{T1}_i, \mathbf{T2}_j) = \begin{cases} 1 & if \mathbf{T1}_i = \mathbf{T2}_j \\ \frac{1}{5} & else \end{cases} \tag{2.4}$$

To calculate the total costs of the annotations between two primitive behaviors (that is, calculating $Annotations(\mathbf{T1}_i, \mathbf{T2}_j)$), we calculate the symmetric difference, seen in Equation 2.5. For this step, we liberally connect the roles, meaning the symmetric difference is calculated using Definition 2.6.11. This allows our measure to consider possible connections, instead of strict connections.

$$Annotations(\mathbf{T1}_i, \mathbf{T2}_j) = symmetricdifference(\mathbf{R} \in \mathbf{T1}_i, \mathbf{R} \in \mathbf{T2}_j) \tag{2.5}$$

The end result of Equation 2.2 is the minimum effort to transform one behavior into another. At this stage, our measure provides the cost of changing behaviors, and may be useful for re-planning systems. To compute similarity between behaviors, we perform **L2** normalization between each overall set of behavior. The normalized cost is then subtracted from 1 to provide an overall similarity comparison between behaviors.

2.9 Case Study

To show PAR's representation and what can be understood from the syntax of actions using PAR's measures, we re-create, to the best of our ability, the events from Shoulson et al. [1]. From their paper, several events were extracted and converted over to PAR actions, specifically from their Figure 2 and Table 1. Objects and object semantics that are mentioned by name in their work are converted into PAR objects. We also automatically generate generalizations for both actions and objects using WordNet as the base hierarchy. This creates a total of 62 actions and 31 objects. For this test, there are no preparatory specifications. Semantic sets were only parsed for objects. The reason for this is that PAR is unique in its preparatory specifications and action semantics, so it would not make sense that when translating them over to have those fields filled out. We found during the case study that not all events used were described in [1], and so we also added several primitive actions from the descriptions of the behavior trees in their Table 1 as well as the primitive action *Guard*, which is not described anywhere in the paper, but fits in with primitive events mentioned in passing in the paper, such as *Close* and *Daze*. An in-depth description of each action and object can be found in Appendix A.

The first demonstration in our case study shows the effects of using role equality(Definition 2.6.10) vs. role similarity(Definition 2.6.11). We use Equation 2.1 to compute the total edit distance between two sets of roles in an action. We provide a qualitative analysis, in that we show, for all actions, the most and least similar actions based on the roles, using both equality and similarity. Ties are broken by assuming the first action is closer. We also describe the total number of ties in Table 2.3. We also create a word-cloud to show the entailment between different actions based on their roles in Figure 2.12.

From Table 2.2, it should be seen that, for many actions in our study, the most and least similar actions are the same. The exception has to do with the action whose most similar action is *Approach*. This is because *Approach* does not have a restriction on what can be approached, meaning any physical object can be used in that setting, and so many actions have one role that is similar to *Approach*. This is not reciprocal. While actions such as *Press* and *Trap* may have a role generalize to *Approach*, *Approach* does not have a role that will generalize to them. This displays the important

Table 2.2: The minimum and maximum distance actions based on roles for both exact and similar measurements. Actions that differ between similar and exact are shown in bold.

Action	Minimum Similar Action	Maximum Similar Action	Minimum Exact Action	Maximum Exact Action
Hide	Approach	TrapGuardsAlarm	Lock	TrapGuardsAlarm
Lock	Guard	TrapGuardsAlarm	Guard	TrapGuardsAlarm
EscapeCell	Lock	TrapGuardsAlarm	Lock	TrapGuardsAlarm
Press	Approach	TrapGuardsAlarm	Hide	TrapGuardsAlarm
Guard	Lock	TrapGuardsAlarm	Lock	TrapGuardsAlarm
Trap	Approach	TrapGuardsAlarm	Hide	TrapGuardsAlarm
TrapGuardsAlarm	TrapGuards	DistractGuards	TrapGuards	Unlock
TrapGuards	TrapGuardsAlarm	Lock	TrapGuardsAlarm	Unlock
Draw	Daze	TrapGuardsAlarm	Daze	TrapGuardsAlarm
Daze	Draw	TrapGuardsAlarm	Draw	TrapGuardsAlarm
Call	Draw	TrapGuardsAlarm	Draw	TrapGuardsAlarm
Approach	Hide	TrapGuardsAlarm	Hide	TrapGuardsAlarm
Give	Exchange	TrapGuardsAlarm	Exchange	TrapGuardsAlarm
Exchange	Give	TrapGuardsAlarm	Give	TrapGuardsAlarm
Open	Lock	TrapGuardsAlarm	Lock	TrapGuardsAlarm
Unlock	Lock	TrapGuardsAlarm	Lock	TrapGuardsAlarm
Take	StealKey	TrapGuardsAlarm	StealKey	TrapGuardsAlarm
StealKey	Take	TrapGuardsAlarm	Take	TrapGuardsAlarm
SoundAlarm	TrapGuardsAlarm	DistractGuards	TrapGuardsAlarm	DistractGuards
Close	Lock	TrapGuardsAlarm	Lock	TrapGuardsAlarm
DistractGuard	EscapeCell	TrapGuardsAlarm	EscapeCell	TrapGuardsAlarm

point that role similarity does not happen in both directions, and is dependent upon the path between objects.

It should also be seen that actions with more overall roles are less similar to actions that only use one or two roles. This should be expected, as the number of possible matches is going to be lower (and therefore cost more to transform one into the other) when there are more roles in the action. An interesting exception to this is when an action is part of the task of another action. For example, *TrapGuardsAlarm* has its task definition as *And(SoundAlarm,TrapGuards)*. So even though *TrapGuardsAlarm* contains several roles and is generally considered the least similar action, because *TrapGuardsAlarm* must contain the roles of its subsequent actions in the task (for annotation purposes), it is actually considered very similar to those actions.

(a) Similarity word cloud

(b) Exact word cloud

Figure 2.12: Entailment groupings for all actions in the case study when only comparing roles, using (a) similarity metric or (b) exact comparison. This is a visual explanation of Table 2.2

One issue that occurs when examining the least and best related actions is that there may be more than one action that fits that category. Table 2.3 describes, for both measures, the number of actions that have the same cost and are minimum. It can be seen from Table 2.2 and Table 2.3 that for many actions, a change in the actions between role similarity and role exactness does not correspond to the lowest cost action changing. In several cases, such as for *Daze* and *Close*, the number of actions that are the same decrease when using a stricter comparison, which is to be expected. However, there are a few actions (*Press* for example) whose number of ties actually increase dramatically. Specifically for *Press*, the most similar action changes when using different measures, which means that the exact measure in Table 2.3 shows that there are many actions that are the least cost, but that when using the similarity measure, they are beaten out by *Approach*. A larger, more varied action set that utilizes the object hierarchy and Definition 2.6.12 would raise the number of similar cost ties, causing the similar and exact cost ties to become more closely related.

The second example in our case study describes how Θ can be used as a storage mechanism for plans. We first describe an *event* from Shoulson et al [1] and shown in Table 2.4. When translating the actions of [1], we purposefully did not create preparatory specifications, as events does not

Table 2.3: The number of ties for each action, using both similarity measure and exact measure.

Action	Number of Minimum Similar Cost Ties	Number of Minimum Exact Cost Ties
Draw	3	2
EscapeCell	8	7
Daze	3	2
Press	1	11
TrapGuardsAlarm	1	1
TrapGuards	1	1
Trap	1	11
StealKey	1	1
Take	1	1
Open	4	3
Call	3	2
Exchange	1	1
Hide	1	11
Close	4	3
DistractGuard	2	2
Approach	11	11
Lock	4	3
Give	1	1
SoundAlarm	1	1
Unlock	5	4
Guard	4	3

recognize them. However, PAR stores this preparatory specifications as tasks, that are built up at runtime. Therefore, we show the exponential buildup of actions based on the starting conditions $\phi \in \Psi$ when $e = 0$ and ending effects $\delta \in \Psi$ when $e = 1$. In Table 2.5, we provide Θ, as well as the modified Ψ for action 62 (*Distract Guard*). The changes to all other actions can be found in Appendix A.

Table 2.4: The action *DistractGuard* from Shoulson et al. [1]

λ	DistractGuard
R	Guard
R	Door
R	Prisoner
R	Prisoner
R	WayPoint
R	WayPoint
ϕ	getProperty(door,"status") =="guarded" and canReach(agent,guard) and canReach(prisioner1,waypoint1) and canReach(prisoner2,waypoint2)
δ	setProperty(door,"status","Idle")
τ	AND(Hide,Hide,Draw)

From Table 2.5, it can be seen that many of the conditions in *Distract Guard*(ϕ in Table 2.4) can actually be met by other actions, such as *canReach* being an effect of the action *Approach*. This simplifies the overall conditions required to start to simply be that the door needs to be guarded. If the door is not guarded, then the action has already succeeded, and there is no need to run through any of the task. It should be noted that setting the action to SUCCESS is a design choice in this study, one that would be made by a simulation author, and is not a forced parameter of PAR. A simulation author could also say that an unguarded door should return FAILURE without running *DistractGuard*, because the action itself would be pointless to do. What is important to realize is that this is a design choice, and that using applicability allows a greater control over the action, as the combination of preparatory tasks and condition-assertions allows the conditions that are needed to be moved to ones that cause the task to be possible or impossible to perform. This flexibility in

Table 2.5: The action from Table 2.4, written out as a PAR.

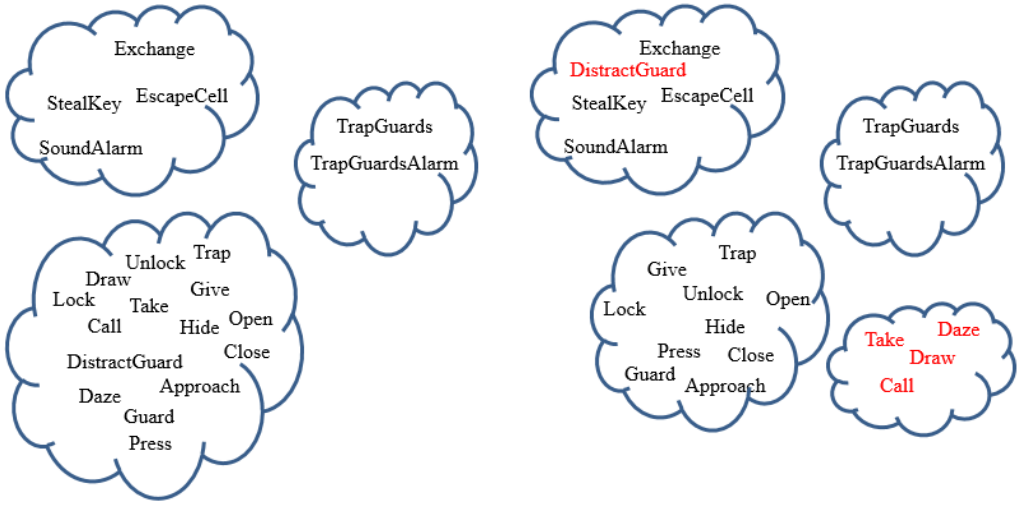
λ	DistractGuard
P	Distract
R	Guard
R	Door
R	Prisoner
R	Prisoner
R	WayPoint
R	WayPoint
$\Psi(e=0)$(PAR)	getProperty(door,"status") != "guarded$\leftarrow$ *SUCCESS*
$\Psi(e=1)$(PAR)	Finished$\rightarrow$ setProperty(door,"status","Idle") and *SUCCESS*
Θ	not canReach(agent,guard)$\rightarrow$Approach(agent,guard)
Θ	not canReach(prisoner1,waypoint1)$\rightarrow$Approach(prisoner1,waypoint1)
Θ	not canReach(prisoner2,waypoint2)$\rightarrow$Approach(prisoner2,waypoint2)
$\Theta(\phi=\emptyset)$	And(Hide,Hide,Draw)

action design means that simulation authors must be more direct in which conditions they wish to plan around and therefore differentiate between conditions that can be met by other actions in Θ and which conditions they do not and should remain in Ψ. This is not seen in the*event* representation of Table 2.4, requiring that the entire action set be planned over or re-planned over during run-time. By having the plan made before execution, we remove that costly bottleneck to agent simulation. By using preparatory specifications as a previous step measure and having it be part of the language, it is more obvious that it is necessary to consider during design time.

Another choice for the guarded condition in Table 2.5 is that the condition can be moved to Θ if there is an action that causes the door to be guarded. If the action must take place, then it is useful for there to be a way for the action to start. However, in the recreation of events, there is no action that sets the door property to be guarded, which is why Table 2.5 has that property stay a condition-assertion pair when being converted to PAR. By adding an action, *Guard* to the set, this complex action can then always happen, again with what conditions are members of Ψ being a design choice. By forcing some of the preparatory planning into the action definition, a simulation author can also determine if their action set covers what they want to by looking at which actions can be used in Θ. What should be understood from this is that using the condition-assertions and

preparatory specifications allow a simulation author options on how the system should behave that are not available using work such as events, which do not have this designation

Finally, we show how task similarity can be used to generalize actions. The original definition of events does not contain generalizations ($act.P$), but action generalizations have been used in planning methods to help organize the actions. One method of generalizing actions is to group actions that are most similar together. As a starting point, we take the actions in event-centric planning, and use task edit distance to (Equation 2.2) to calculate the cost between each action. The lowest and highest cost actions are shown in Table 2.6 and in Table 2.7. Table 2.6 assumes that $act.P = \emptyset$, so that no action starts with a parent. Table 2.7 uses WordNet [36] as a starting point, assigning each action a parent. We include the WordNet parent assignment as additional actions in Appendix A. Furthermore, we perform entailment on the set of tasks that are the lowest cost in order to determine groupings between actions. The sets without using Wu-Palmer similarity can be seen in Figure 2.13. The entailment sets with Wu-Palmer similarity can be seen in Figure 2.14.



(a) Without Roles (b) With Roles

Figure 2.13: Entailment groupings for all actions in the case study when no information about generalization is used. Red action names signifies changes between (a) and (b).

Table 2.6: The minimum and maximum cost actions when comparing tasks using an exact measurement. The task measurement is shown using both role cost and non-role cost.

Action	Lowest Cost Task	Highest Cost Task	Lowest Cost Task with Roles	Highest Cost Task With Roles
Hide	Lock	TrapGuardsAlarm	Approach	TrapGuardsAlarm
Lock	Hide	TrapGuardsAlarm	Guard	TrapGuardsAlarm
EscapeCell	StealKey	TrapGuardsAlarm	StealKey	Give
Press	Hide	TrapGuardsAlarm	Approach	TrapGuardsAlarm
Guard	Hide	TrapGuardsAlarm	Lock	TrapGuardsAlarm
Trap	Hide	TrapGuardsAlarm	Approach	TrapGuardsAlarm
TrapGuardsAlarm	TrapGuards	Hide	TrapGuards	Give
TrapGuards	TrapGuardsAlarm	Hide	TrapGuardsAlarm	Give
Draw	Hide	TrapGuardsAlarm	Daze	TrapGuardsAlarm
Daze	Hide	TrapGuardsAlarm	Draw	TrapGuardsAlarm
Call	Hide	TrapGuardsAlarm	Draw	TrapGuardsAlarm
Approach	Hide	TrapGuardsAlarm	Hide	TrapGuardsAlarm
Give	Hide	TrapGuardsAlarm	Approach	TrapGuardsAlarm
Exchange	StealKey	TrapGuardsAlarm	StealKey	TrapGuardsAlarm
Open	Hide	TrapGuardsAlarm	Lock	TrapGuardsAlarm
Unlock	Hide	TrapGuardsAlarm	Lock	TrapGuardsAlarm
Take	Hide	TrapGuardsAlarm	Draw	TrapGuardsAlarm
StealKey	Exchange	TrapGuardsAlarm	Exchange	TrapGuardsAlarm
SoundAlarm	Exchange	TrapGuardsAlarm	Exchange	Give
Close	Hide	TrapGuardsAlarm	Lock	TrapGuardsAlarm
DistractGuard	Hide	TrapGuardsAlarm	StealKey	TrapGuardsAlarm

Examining Table 2.6 and Table 2.7, the task with the largest tree is the most distinct from all other actions. What is a bit surprising is how the lowest cost actions manifest themselves. If each action grouping for Figure 2.13a and Figure 2.14a are considered their own forest, then some obvious and not so obvious meaning between tasks appear. The first is that, using only an exact measurement (not using $act.P$ in the comparison), *TrapGuardsAlarm* is only connected to the first part of that action *TrapGuards* and not the second part *SoundAlarm*. This is because *TrapGuards* is a longer task than *SoundAlarm*, and so the cost estimate is much greater. Looking at the actual cost, removing 19 components to a tree is a much greater edit than adding six. When using Wu-Palmer similarity on the tree, the edit cost for changing an action actually decreases, so tasks with more actions (opposed to connectives) are favored, which is why *TrapGuards* and *SoundAlarm*

swap clusters in Figure 2.14a. This means that clustering actions using only task similarity (and not using roles in their cost) may not semantically capture similarity in tasks. There is a lot of semantic meaning in the roles of an action, and therefore, they are an important consideration when comparing actions.

One issue when examining the actions based on entailment is that there are times when an action's highest cost counterpart is in the same set as the lowest cost actions. This occurs specifically when using the Wu-Palmer similarity measure, and can be seen in Table 2.7. The reason that this occurs is that, with Wu-Palmer, the cost of action editing becomes much less than the cost of adding or removing an action, so the edit distance creates much higher costs associated with longer actions, which it should do. This isn't seen in Table 2.6, because we aren't assuming anything about the actions themselves. At a first glance, this may seem as though using generalizations actually hinders the comparison between actions. For primitive actions like *Daze* and *Draw*, generalizations provide a quick relationship between the two, and as will be shown later on, can assist in end goals of the agent. What should really be noted here is that it is not really appropriate to disambiguate complex actions using WordNet, because the name of the action does not necessarily reflect everything that is going on in the task. This means that other measures are needed, such as the edit distance, which in Figure 2.13, especially Figure 2.13b connects complex tasks into two groups. Therefore, when examining complex tasks, it is not enough to look at only the generalizations or the roles, but the entire definition of that action.

The entailment groupings in Figure 2.14 show the effect of using both semantic generalization knowledge and role information in creating groupings of *act*. Especially interesting is Figure 2.14b, which has a grouping containing *Open*, *Lock*, *Unlock*, *Guard* and *Close*. This should be an obvious grouping due to the similarity of the meaning of these five actions (based on WordNet generality) as well as the fact that they were all generated from primitives in Shoulson et al, dealing with the same events. Using similarity with roles has also differentiated all of the complex actions from the primitive ones, with complex actions appearing in the left-most cloud of Figure 2.14b. While separating complex from primitive actions is true for both Figure 2.13b and Figure 2.14b, using an already pre-processed tree has entailed all complex actions together. This means that simply using

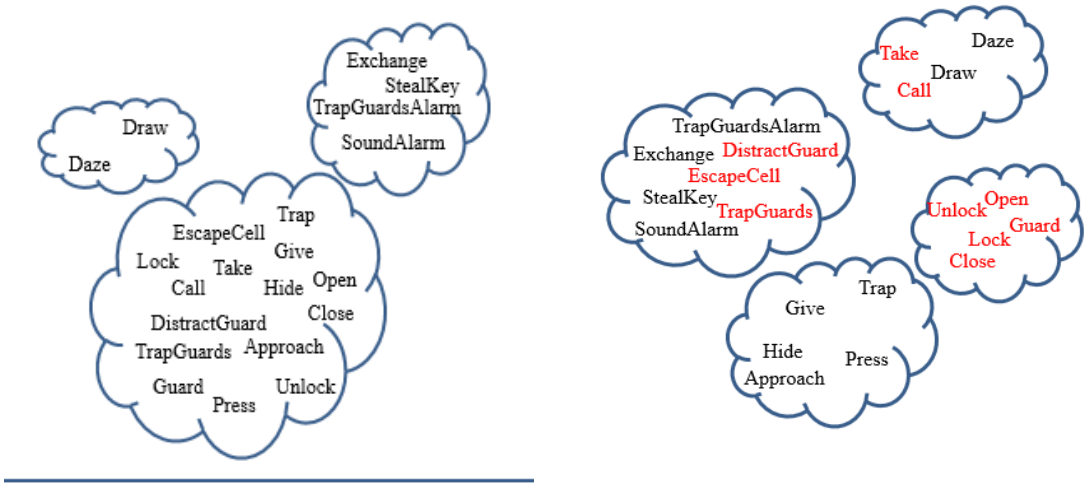
Table 2.7: The minimum and maximum cost actions when comparing tasks using Wu-Palmer similarity measurement. The task measurement is shown using both role cost and non-role cost.

Action	Lowest Cost Task	Highest Cost Task	Lowest Cost Task with Roles	Highest Cost Task With Roles
Hide	Open	TrapGuardsAlarm	Approach	TrapGuardsAlarm
Lock	Hide	TrapGuardsAlarm	Open	TrapGuardsAlarm
EscapeCell	DistractGuard	TrapGuardsAlarm	StealKey	Give
Press	Hide	TrapGuardsAlarm	Approach	TrapGuardsAlarm
Guard	Hide	TrapGuardsAlarm	Open	TrapGuardsAlarm
Trap	Hide	TrapGuardsAlarm	Approach	TrapGuardsAlarm
TrapGuardsAlarm	SoundAlarm	Call	TrapGuards	Give
TrapGuards	EscapeCell	Call	SoundAlarm	Give
Draw	Daze	TrapGuardsAlarm	Daze	TrapGuardsAlarm
Daze	Draw	TrapGuardsAlarm	Draw	TrapGuardsAlarm
Call	Hide	TrapGuardsAlarm	Draw	TrapGuardsAlarm
Approach	Hide	TrapGuardsAlarm	Hide	TrapGuardsAlarm
Give	Hide	TrapGuardsAlarm	Approach	TrapGuardsAlarm
Exchange	StealKey	TrapGuardsAlarm	StealKey	TrapGuardsAlarm
Open	Unlock	TrapGuardsAlarm	Close	TrapGuardsAlarm
Unlock	Open	TrapGuardsAlarm	Open	TrapGuardsAlarm
Take	Hide	TrapGuardsAlarm	Draw	TrapGuardsAlarm
StealKey	Exchange	TrapGuardsAlarm	Exchange	TrapGuardsAlarm
SoundAlarm	StealKey	TrapGuardsAlarm	Exchange	TrapGuardsAlarm
Close	Hide	TrapGuardsAlarm	Open	TrapGuardsAlarm
DistractGuard	Hide	TrapGuardsAlarm	StealKey	TrapGuardsAlarm

WordNet to generalize complex actions may not be appropriate, as the actions are so far removed from the primitive actions that they form one group. It also shows the importance of roles when examining actions, as adding role similarity was able to break up the large cloud of Figure 2.14a into more semantically meaningful sets. For these reasons, role similarity is a useful measure for calculating distances between actions, and should not be discounted.

2.10 Conclusions

We presented a systematic description of the Parameterized Action Representation. The syntax and semantics of PAR have, in the past, been very abstract, and what is provided is a novel, concrete



(a) Without Roles

(b) With Roles

Figure 2.14: Entailment groupings for all actions in the case study when generalizations are used. Red action names signifies changes between (a) and (b).

understanding of the Parameterized Action Representation. The work we have contributed thusly allows us to better describe how actions can be organized and generated using PAR as a baseline. PAR has several advantages over other action descriptions, including generalizations of objects and actions, as well as the separation and co-dependence of tasks and actions. The creation of a PAR requires several steps, as do all action languages. The additions of condition-assertions, condition-tasks (preparatory tasks), and action semantics to the agent language repatriate provide more options for action control, at the expense of the simulation author. Having exact definitions for each of these components, and describing their use should standardize the creation of these, and hopefully lead to wider adoption. The use of roles and action parents were also highlighted in a comparison measure, tying together knowledge of the virtual agent's environment. As we later show that the population of these two fields, object semantics and action semantics can be automated, their use becomes critical in allowing agents to have a larger understanding of their abilities and environment, freeing up a simulation author to provide more control in condition-assertions and condition-tasks.

2.11 Summary of Key Components

2.11.1 Symbols for the Language

The PAR language is denoted as $\mathbb{L}$, and contains all necessary knowledge for an agent to reason about and interact with its virtual environment. Components of the language are listed below. Many components have sub-components, which are listed as sub-lists. We also provide a short description of each component when necessary with their numbered definition.

- $\mathbb{S}$: Semantics-Definition 2.3.1
  - λ: Semantic Descriptor (name)
  - **V**: Set of possible values at runtime
  - ω: Flag signifying what the semantic describes
- $\mathbb{OBJ}$: Objects-Definition 2.4.1
  - λ: Object Descriptor (name)
  - p: Object generalization (parent)
  - **S**: Semantics of object
- Φ: Conditions-Definition 2.5.1
- $\mathbb{A}$: Assertions-Definition 2.5.3
- $\mathbb{ACT}$: Actions-Definition 2.6.1
  - λ: Action Descriptor (name)
  - **P**: Action generalizations (parents)
  - σ: The action status
  - **R**: Roles-Definition 2.6.9
    - **OBJ**: Set of objects that can fulfill that role
    - ν: Flag signifying if the object is necessary in the action

  - Ψ: Action condition assertion statements-Definition 2.6.14
    - ϕ: Executable condition
    - δ: Assertions upon the world
    - e: Flag signifying when the condition is examined
  - Θ: Action condition task statements - Definition 2.6.5
    - ϕ: Executable condition
    - **T**: Task description - Definition 2.8.1
  - **S**: Semantics of action
- $\mathbb{F}$: Task Connectives

2.11.2 Key Functions

There are also several key functions that allow PAR to be understood and realized by virtual agents. We list those below, along with their argument types and references.

- $path(\mathbb{OBJ}_i, \mathbb{OBJ}_j)$-Definition 2.4.2
- $path(\mathbb{ACT}_i, \mathbb{ACT}_j)$-Definition 2.6.2
- $lcs(\mathbb{ACT}_i, \mathbb{ACT}_j)$-Definition 2.6.4
- $symdiff(\mathbb{ACT}_i, \mathbb{ACT}_j)$-Equation 2.1
- $ExecutionAction(act)$-Algorithm 1
- $Build(\Theta)$-Algorithm 2
- $Evaluate(\Psi, type)$-Algorithm 3
- $Enumerate(\mathbf{T})$-Algorithm 4
- $Step(\mathbf{T})$-Algorithm 5

Chapter 3: Considerations on Action Organization

3.1 Introduction

As an action representation, PAR allows for multiple parent generalization, as well as inheritance from those generalizations. When we describe generalizations and inheritance, what we mean is that PARs have an inherent structure based on the path, with an example seen in Figure 3.1. A *generalization* is the action's parent(s), from which the child action can inherit components of if those components are not defined specifically for that action. In Figure 3.1, the task for *Walk* and *Run* are inherited from *Travel*, as are both actions' roles. The property of *speed* are specific subsets of the property in the parent.

There are several interesting consequences to action parents in PAR. One is that action data may come from several inherited sources, and apply action data to several, more specific actions. Given the interplay between semantics (**S**), roles (**R**), and tasks($\mathbf{T} \in \Theta$), it should be seen that components can be templates, with specifications constraining those templates in different manners. Figure 3.1 provides a good example of how organization of the actions reduces the overall amount of data in the system. Ergo, a design paradigm should enable a simulation author to maintain less overall knowledge in the system while still achieving the same fidelity. Ideally, a well created hierarchy would also reduce bugs in the action base.

Another advantage to using a well designed hierarchy comes from the planning community. Hierarchical task planners allow actions to be connected together into complex plans. This thought process can be reversed, in that reasoning systems may only need to consider a subset of the total actions in the entire set of $\mathbb{ACT}$. Generally, in order to prune the action space, the agent must still examine the entire action space. A well-formed hierarchy would allow planning and reasoning systems to understand large swatches of the space on a general level, assisting in pruning out undesirable actions without necessarily examining them. For rich environments consisting of many

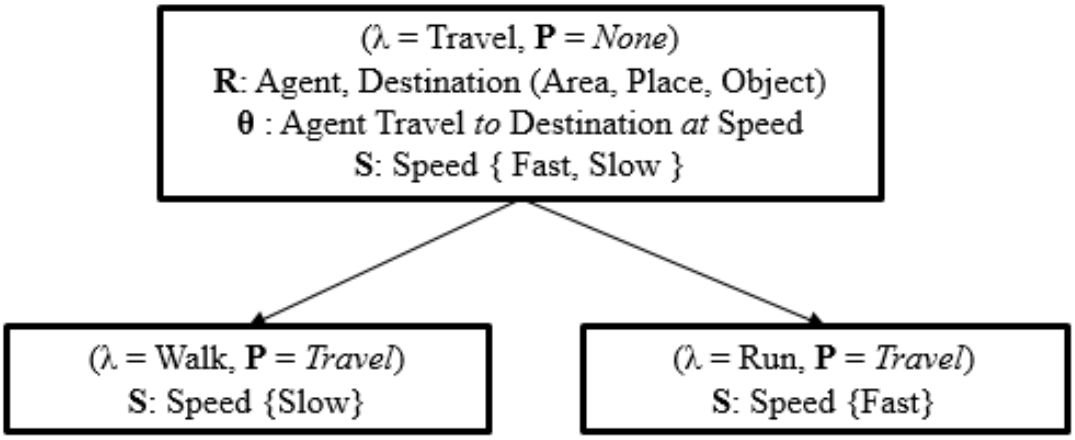


Figure 3.1: An action set of three actions, where one (*Travel*) is the parent of the others.

actions, using a well formed hierarchy of actions means that using planning algorithms should still be a practical approach to virtual agent behavior.

What has not yet been defined is the term "well-formed" hierarchy as it pertains to an action hierarchy. We show that this should be thought of as an application dependent problem, in that the hierarchy is only well-formed if it improves the reasoning abilities of virtual characters. What is described in this chapter is not a rigid definition of a well-formed hierarchy, but on techniques to guide agent authors to better form their hierarchies. Specifically, we focus on two particular questions:

1. In what instances does using multiple generalizations (or specifications) require a simulation author to add additional information, specifying a certain action and removing inherited properties from the action space?

2. How does the information about actions and the organization of those actions effect the end abilities of the virtual characters?

3.2 Parent-Child Action Relationships

An action's parent-child relationship provides generality between actions, and creates a convenient way to group actions into taxonomies, which can be seen in Figure 3.1. Actions such as *Water* and *Sprinkle* are obviously related to *Wet*, and one way in which to organize these actions is to have *Wet* be a parent of both *Water* and *Sprinkle*. This is certainly not the only way in which these actions can be broken up, especially depending on the meaning of each action. If a simulation author meant *Water* to mean **fill with tears**, then it would not be related as well to *sprinkle*. Creating parent-child relationships are a precise and important task when needing to to quickly locate similar actions, and as we will show are therefore useful to agent planning systems.

Traditionally, PAR has represented parent-child relationships as a single parent hierarchy for both actions and objects. From Ontological literature, this relationship does not cover instances where an action can be described by one or more parents. For example, an action for a motion controller *CrouchAndShoot* which would comprise of a single motion, is best described by two dissimilar parent actions *Crouch* and *Shoot*. To account for this, we have implemented PAR actions as a *Forest of IsA Directed Acyclic Graphs* (FIDAGs). FIDAGs are a more appropriate data structure to capture the rich information necessary to have virtual agents reason over their environment. The basic operation of searching for an action by generalization of the hierarchy will need to be examined, as this operation will become more complex with multiple parents. Therefore, we discuss the guidelines of FIDAG creation, including when it is appropriate for actions to have multiple parents.

FIDAGs create many issues that need to be considered. However, we first describe the utility of FIDAGs as a data-structure. For action information that is parsed as a sequence (such as Ψ), FIDAGs allow a simulation author to inherit from multiple sources, allowing for input into failures from all parents. Recall from the definition of *Task Inheritance*(Definition 2.6.8) and *Semantic Inheritance*(Definition 2.6.16) in Chapter 2 the ways in which actions inherit information from their parents if they are absent in the action's definition. One challenge of having an action set that contains several actions is that the definitions must be filled in. Inheriting from action parents eases a simulation author's burden, as information can instead be parameterized. Multiple data-sources provide a more nuanced inheritance, as long as the parent actions do not provide conflicting

inherited information.

Primarily where conflicts in FIDAGs arise is in the execution of the task of a given action. For example, by the definition of *Semantic Inheritance*, the semantic set **S** of an action must be a subset of the action's generalizations, provided multiple generalizations have the semantic set. Consider as a motivating example a semantic $\mathbf{S}_i$ that is used as an annotation in a task. If the parents in the FIDAG of an action that contain $\mathbf{S}_i$ are disjoint sets, then $\mathbf{S}_i$ is empty, and does not contain a usable value suitable for annotation. If a task in $\mathbb{ACT}$ expects a grounded semantic from $\mathbf{S}_i$ which cannot be filled in, the action's parents are then in conflict.

It should be noted from the definition of an action (Chapter 2, Definition 2.6.1) that the syntax of an action has several components. When discussing how these components are inherited from generalization (**P**), it is important to discuss which components may come into conflict with each other. Of particular interest is the definition of *Condition Tasks* (Definition 2.6.5), *Semantic Inheritance*(Definition 2.6.16), *Condition-Assertions* (Definition 2.6.14) and of inheriting roles (Definition 2.6.13). Actions with multiple parents may have different components of each of these, or may be written with components that need to be handled together. In this section, we describe how $act \in \mathbb{ACT}$ inherit from the parent set **P** when there are multiple parents to inherit from. This will led to a discussion of techniques to handle conflicts between parents, which are shown to only appear between annotations on tasks themselves. What should be remembered is that inheritance on the tasks, roles, and condition-assertions do not occur on a given action if the component is defined on that action. This is a tool the simulation author can use to resolve conflicts.

3.2.1 Task Inheritance

From Figure 3.1, it should be seen that the task is essentially the same for all three actions. The only real difference between these tasks are the speed (and usually animation itself). All other functional code, such as actually moving the character, is the same. Therefore, it should be seen why Θ where $\phi \in \Theta = \emptyset$ can be generalized for *Walk* and *Run* into the parent action *Travel*. As each child action is used by the agent, it examines the generalizations in order determine the encoding of the action. An example of this using a blendspace can be seen in Figure 3.2. Blendspaces are a common practice

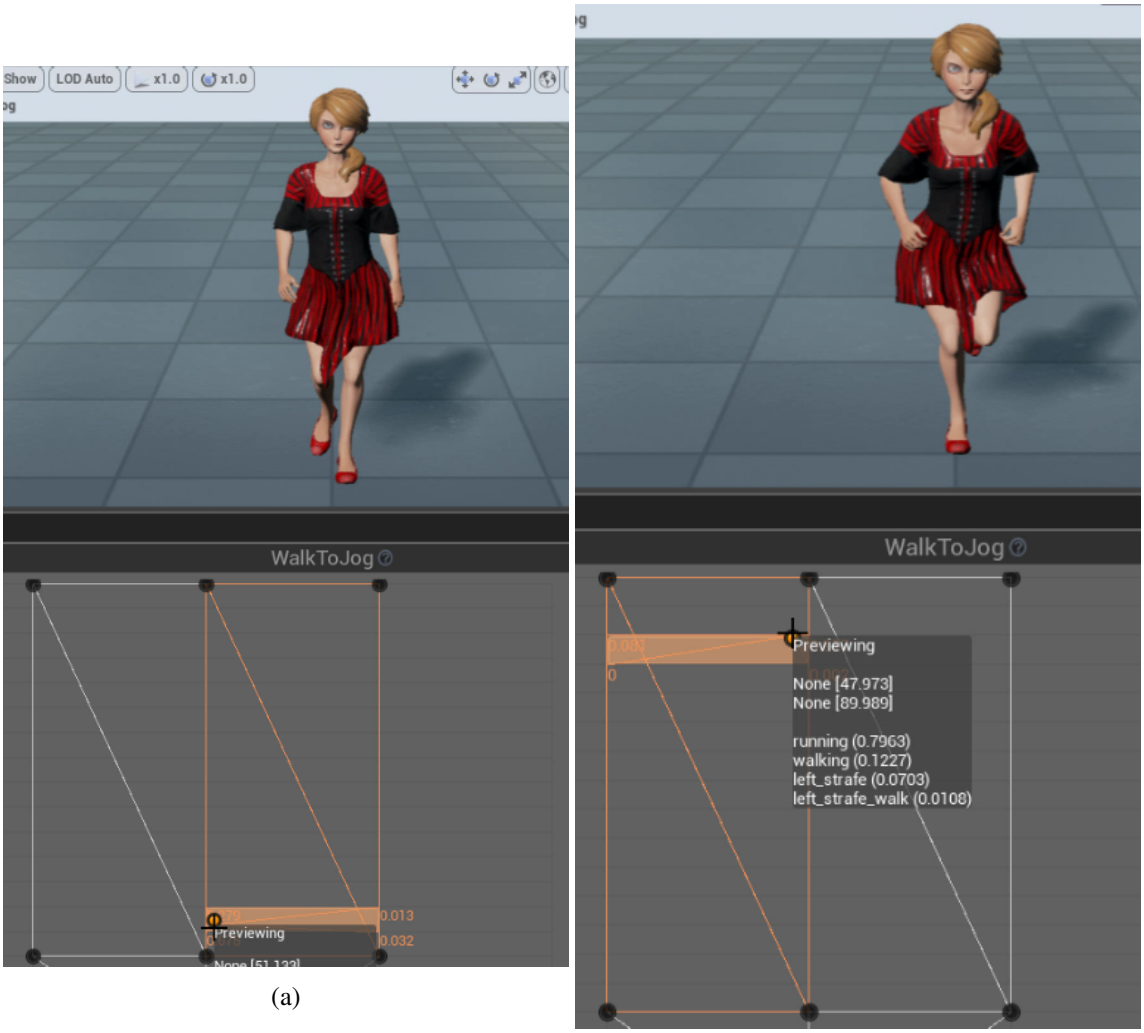
for virtual characters[1], and as can be seen in Figure 3.2, is controlled by a simple parameter (*Speed*). As the speed parameter changes from Figure 3.2a to Figure 3.2b, the percentages of each animation change. This is the thought behind task inheritance. Child actions that can be controlled by a single parameter (which could be a semantic of each action) should have a single task that is varied by that parameter. Organizing actions such that this is the case can reduce the burden of a simulation author, in that most of the data he would write is stored in a single PAR, with only changes being stored in the child action.

There are two instances when task sets from multiple parents have an effect on the action: when multiple execution tasks are defined for an action and when multiple preparatory tasks are defined for an action. Recall from the definition of execution step inheritance (Definition 2.6.8) that execution steps are inherited by examining $act \in \mathbf{P}$ in order. This means that an inherited task will always come from higher in the FIDAG. This provides an important consideration about action ordering. The order in which actions are in **P** play an effect on an action's inheritance, and so parent actions should be ordered by how much alike they are to the child action.

3.2.2 Semantic Inheritance

S is a set of semantics for a given action, with each semantic set $\mathbf{V} \in \mathbf{S}$ a subset of the parent actions'. This means that generalizations play a huge role in action semantic sets. We refer to *Semantic Inheritance* (Definition 2.6.16) to describe how, for one parent, the action's semantics operate. When there are multiple action generalizations, the semantic set must be a subset of all parent actions, and therefore is the intersection of its parents' semantics. This gives rise to two issues. The most common issue is that one parent will not have a given action semantic at all, meaning that the subset is empty. If this is not the intended case, care must be taken that subsets are described on the parent for all actions on the set, so that they may be properly propagated to the children. The other issue is that a simulation author may only tag an object with a semantic set, and not fill in the parents, as seen in Figure 3.3a. This is a common occurrence, as it save the simulation author time to only enter semantics on the actions where they are relevant. Note how *manner* is

[1] A tutorial can be found in https://docs.unrealengine.com/latest/INT/Engine/Animation/AnimHowTo/BlendSpace/



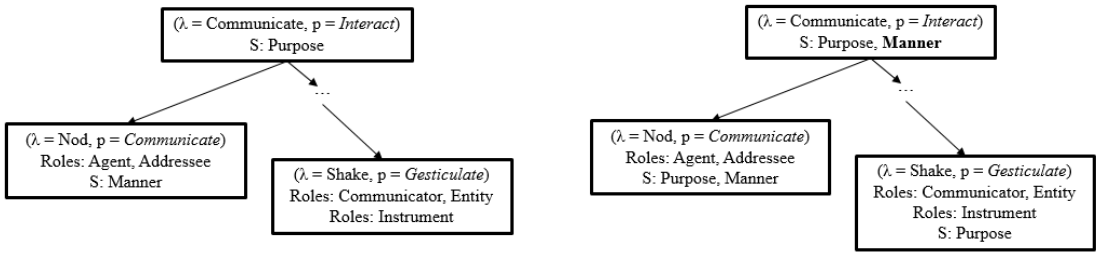
(a)

(b)

Figure 3.2: A motion blend of several animation, including (a)*Walk* and (b)*Run*

only on *Nod* in Figure 3.3a. For the action set to be correct, communicate should also contain *Manner* as a semantic for the parent action. Of course, if a certain set of semantics is required on a child action, the set can be back-propagated to all parents that do not have the semantic set during the design phase. This will increase the total amount of data but provide a consistent meaning on generalizations. Then, it should simply be recognized that other children of a parent would have a possibility of those sets and would need to be defined as an empty set if that is the understanding of the simulation author. So, in Figure 3.3a, *Manner* should not propagate down to *Shake*. This further

increases authorial burden, as the author must mark those semantics as empty. To compensate, PAR contains a special back-propagation algorithm to create valid generalizations. In summary, we assume that the semantics inherit the parent's full set unless a simulation author specifies otherwise. If a simulation author sets a semantic in a set that does not appear in the parent, the semantic is not added to the parent until all other semantics are added. Thus, a simulation author only needs to add semantics to the highest level generalization of the action for that set to be passed down. The system should take care of correcting any set discrepancies, thereby reducing the amount of data an author needs to generate while providing ample data to the system. This can be accomplished using *Correct Semantics* (Algorithm 6), which will, in general, combine all equal components of type *type* in a given action set. *Correct Semantics* is only used when properties are first added to the system in order to keep consistency with Definition 2.6.16. Algorithm 6 is only used after all semantic sets are determined, and is only meant to propagate missing semantics that are hand placed on the actions (i.e, only as a tool to ensure correctness of **S** for a given action). The result of *Correct Semantics* on Figure 3.3a can be seen in Figure 3.3b. Note the bold *Manner*, and how it is propagated up to *Communicate*, but not down to *Shake*.



(a) uncorrected

(b) corrected

Figure 3.3: An action hierarchy with (a) semantics on the action and (b) semantics after being combined

Algorithm 6 Recombining semantic properties for correctness.

```
1: function CORRECT SEMANTICS(act)
2:     for all sem ∈ S do
3:         for all p∈ P do
4:             PSem← S ∈ p
5:             if sem ∉ Psem then
6:                 add sem to Psem
7:                 Correct Semantics(p)
```

3.2.3 Role Inheritance

Unlike tasks or semantics, roles by definition are an additive component of an action. Recall from the definition of Roles (Definition 2.6.9) that a role has two components, **OBJ** and v. **OBJ** itself is a set of objects. Because of role similarity (Definition 2.6.11) and role instantiation(Definition 2.6.12), roles may be similar, with one object superseding another in the role set. Using the actions of event-centric planning as an example [75], *Approach* (Table A.12) and *Hide* (Table A.1) have a single role each. Also, the *Entity* object in role supersedes the *Waypoint* object, an *Entity* is a parent of *Waypoint* in $\mathbb{OBJ}$ (Appendix A Section A.1). For virtual agents, what is important for roles is that an object conforms to the operational information. That is, that an object can be instantiated into the role. Since there can be multiple objects for a role, adding more objects to that role simply widens what can be used for that role. Therefore, having multiple parents with different objects in the role simply adds to what can fit in the role. Following the definition for combining roles (Definition 2.6.13), an action inheriting from both *Approach* and *Hide* would have a role defined as $\mathbf{R} : ((\mathbf{OBJ}, 1))$ where $\mathbf{OBJ} = (\textit{Entity}, \textit{Waypoint})$, which due to role similarity, reduces to $\mathbf{OBJ} = (\textit{Entity})$. Of course, to circumvent this behavior, a role on the action itself will not check the action's parents.

3.2.4 Condition-Assertion Inheritance

There are many similarities between the condition-assertion component in an action and the task component in an action. Both of them are ordered sets, and both of them contain a condition (ϕ) string as one of the components. Therefore, inheritance of condition-assertions from parents are similar. While tasks are broken into execution and preparatory tasks, condition-assertions can be

broken into once and every sets. In each set, we discuss them as a whole set (so, Ψ). This is a limitation of the PAR system, in that the system does not examine each individual condition-assertion when inheriting condition-assertions. We do this to ease the authorial burden, which can become more complicated with multiple parents. From Theorem 2.6.1, two conflicting conditions and assertions cannot occur at the same time because condition-assertions are checked sequentially. While this could allow for multiple sets to be concatenated together, there is no mechanism in PAR at this time to do so. Furthermore, as tasks share many components with condition-assertions, keeping this component symmetric should assist in building hierarchies.

The understanding of task inheritance and condition-assertion inheritance in this section show the importance of organizing action parents intelligently. Not only do the FIDAG structures have an effect on understanding the generalizations, but the ordering of actions in $\mathbf{P} \in \mathbb{ACT}$ and $\mathbb{ACT}$ play an important role in the execution of virtual agent actions. Therefore, it is up to the simulation author to determine not only what generalizations are important for the simulation, but how they are organized to perform the behaviors the author desires. It is important to keep in mind that a simulation author can supersede the parent structure by keeping tasks and condition-assertions on each executing action. This would be unfortunate, as we will describe in Section 3.3, because exploiting the parent actions and intelligently organizing these actions can assist not only in keeping the actions easy to read for any humans, but can make it easier for agents to reason over them. Furthermore, in the future tools to organize action sets based on the inherited components could be developed, further easing he burden on a simulation author.

3.3 An Application-based Measurement for FIDAG Organization

Throughout this work, we have described PAR, and how to encode meaning those actions. This has always been under the assumed end goal of having an action representation for virtual human reasoning and action selection. To this end, the knowledge encoding and organization of $\mathbb{ACT}$ should be performed in such a way as to increase the ability of the reasoning system. One popular and important metric to evaluate a reasoning or planning decision is time. In several cases [4, 77], as the number of actions grow, the time to select actions grows. Ideally, an action's parents could

subsume and encode the necessary information of its children on the parent, meaning fewer actions examined by reasoning tools. It is this insight that we explore in the following sections.

3.3.1 Narrative Planning and Behavior Selection Algorithms

Virtual characters' ultimate purpose is to appear more plausible [78] in conveying the end goals of a simulation author. Therefore, virtual characters must be able to use the knowledge that is provided to them. While being able to understand and realize an action is important, rote scripting of characters is a long, arduous, and tedious process. The need for planning and reasoning tools to control agents was realized early on in AI. Of course, if the agent decides every step in a simulation, control is wrestled away from the author, and the chain and choice of behaviors, while interesting, may not be appropriate. There is a belief that a balance is needed between authorial control and agent autonomy [79]. Therefore, we briefly describe characteristics of reasoning and behavior selection tools for virtual characters, and then explain how these tools are used to show the effects of different parent-child arrangements.

Role-based Methods

Some user control methods that have components which are dependent on the objects in the environment and roles are CAROSA [80] and Normoyle et al. [4]. In CAROSA, aleatoric actions consist of sub-actions that are chosen according to an authored distribution and are meant to give variety to a given space. The ability of these actions to be used depends on the resources (objects) available to the virtual agent. Normoyle et al. is a stochastic method which determines transitions between actions to maintain distributions of crowds, similar to the aleatoric methods in CAROSA. Given a set of crowd activities, locations, and desired steady state distributions, this method determines the probability of a given entity in the crowd choosing a new action based on the previous action. While it does not guarantee that the system has exactly the distribution at a given time, it does maintain roughly the distribution, taking into account distance between activities and preferences of agents in the crowd. Because Normoyle et al. determines transitions using convex optimization, the computation time increases dramatically as the number of actions and locations grow. Therefore, for

rich virtual environments with several actions, the pre-computational time is immense. Reducing the total number of actions (through the use of generalizations) should reduce this problem while still assisting the simulation author in determining distributions of crowds.

For measuring the effectiveness of action hierarchy organization, we reproduce to the best of our abilities the role-based method of Normoyle et al [4]. From the results of that paper, the more actions and locations that are available to a group of agent, the longer the convex optimization must run to converge upon a solution. An action organization that reduces the number of actions should allow the convex optimization to more quickly converge, which we postulate makes the method an effective measure for role-based agent behavior selection.

One technique to use with Normoyle et al. is to assume that every action can be followed by every other action, meaning that for *N* actions, there are *NxN* possible connections, which is prohibitive for a large number of actions and locations. We then combine actions into groups of actions **ACTS** $\in \mathbb{ACT}$, based on two criteria. An *acts* $\in$ **ACTS** may exist if, for all actions in **ACTS** the roles are the same, and there is a least common subsumer between all actions in *acts*. We do this because Normoyle et al. is location based, and therefore an action at a location (that has the same object requirements) can have all the actions with the same object requirements performed at that location.

Goal-based Methods

While Role-Based Methods rely on the location of objects in the environment for their decision making system, goal-based methods [81, 82, 72, 44, 77, 83] use $\Psi \in \mathbb{ACT}$ in order to determine the next action that an agent should perform. Many of these methods are planning methods, in that the virtual character's end goal is provided, and the system determines how to reach that end goal. A special kind of goal-based reasoning system is the Belief-Desire-Intention (BDI) system [8, 40]. In this system, the agent's understanding of their environment is stored in their beliefs. The agent's goals are their desires, and the action chosen to reach those goals are the intentions.

Goal-based reasoning has seen alot of attention over the years, particularly in the form of narrative variation. Narrative variation attempts to construct different stories from a set of actions and

objects based on one or more goal states and the causal believability of the actions. In effect, it reasons out the actions and interactions between entire sets of characters. Work such as Riedl and Young [84] and Kartal et al. [77] plan stories using these two measures as a heuristic, attempting to increase both the narrative variation and length of the story using different planning methods. As narrative generation relies on design time understandings of actions, more actions will provide more varied stories. The *ANTON* system [85] builds larger sets of actions by automating the building of negations of actions. Narrative planning methods, like other goal-based methods, suffer from branching overload. As the number of action choices grows, the number of possible combinations also grow, requiring more resources in order to build a viable story. An intelligent organization of the actions can be used to essentially prune actions, as two actions that are similar can be grouped together.

To show the effect action parents have on goal-based methods, we reproduce, to the best of our abilities, the monte carlo search tree narrative generation algorithm of Kartal et al. [77]. This algorithm uses a monte carlo search tree to generate narrative variations based on the action and object sets. Stories are rated using a multiplicative heuristics on how many goals the current set of tasks accomplish and how believable the current story is, based on user defined preferences. To seed the search tree, Kartal et al. implements a random rollout of actions so that a branch can be evaluated when no goals have yet to be reached. The method also contains a selection bias and rollout bias based on the global times an action has been selected.

We implement the believability of a given set of actions using Equation 3.1. Equation 3.1 takes two factors on the action set **ACT** into account for each branch of the monte carlo tree search algorithm, the total number of actions and the total prerequisites of the actions that are met up to that point. Recall that goal-based methods may require multiple actions in order to reach an action that solves the goal, with the other actions being used to reach that action. Therefore, if prerequisites for a given action are met by past actions, the entire action set should be more believable. A set that begins with an action that requires the world state to be in a certain configuration is therefore not as believable and those branches are pruned quickly because of the monte carlo tree search. Furthermore, we attempt to control for multiple actions having similar δ by favoring in our believability

smaller action sets that meet the same criteria.

$$believe = \frac{1}{|\mathbf{ACT}|} \prod_{i=0}^{|\mathbf{ACT}|} bscore(\mathbf{ACT}_i, \mathbf{ACT}_{0\ldots i-1}) \tag{3.1}$$

$$bscore(act, \mathbf{ACTs}) = \left\{ \begin{array}{l} \frac{matched(\phi \in act, \delta \in \mathbf{ACTs})}{|\phi \in act|}, |\phi \in act| > 0 \\ 1, else \end{array} \right\} \tag{3.2}$$

Hybrid Methods

The role-based and goal-based methods described use either $\mathbf{R} \in \mathbb{ACT}$ or $\Psi \in \mathbb{ACT}$ in order to determine an agent's best next step. While it can be argued that goal-based methods encompass role-based methods (as many goals require objects and roles), when roles and goals are broken up, this is simply not the case, as $\mathbf{R}$ is only examined in terms of Ψ (and therefore, is not actually examined). There do exist hybrid methods that consider agent goals and the objects in the virtual environment as separate components. One such example are the opportunistic actions of CAROSA [80]. Opportunistic actions are a subset of the entire action reasoning system in CAROSA, in which actions attempt to be selected and scheduled based on needs (i.e goals). However, while possible actions may be selected based on goals, they are not used if the roles and timing locations do not work out. Therefore, opportunistic actions encompass both subsets of action reasoning systems described. In many cases, hybrid methods require examining the action sets from multiple facets in order to get a useful hierarchy.

3.3.2 Data Compression

We have described several different agent reasoning tools, and discussed the implementation of two of them using the PAR system. During this discussion, one important issue appeared multiple times, that is that the size of the considered action set has an effect on the ability of the reasoning tools. Easily, the considered action set may be a subset of the total action set, in that actions appear the same to the reasoning mechanism. If we know that a specific component of the action (i.e $\mathbf{R}$ or

Ψ) is used by the reasoning system, then we can determine if one hierarchy is more appropriately structured than another. To do so, we use Algorithm 7 (Combine Properties) to determine how well a component of our action set generalizes. In Combine Properties, *type* refers to the component of the action that is being examined, and **Roots** are all the root actions in $\mathbb{ACT}$.

Algorithm 7 Combining data on action properties.

```
1:  function COMBINE PROPERTIES(Roots,type)
2:      for all root ∈ Roots do
3:          stack ← empty stack
4:          Perform Breadth First Search on root, inserting each node into stack
5:          while stack is not empty do
6:              top← pop(stack)
7:              parents← P ∈top
8:              for all parent∈parents do
9:                  Children← all children ofparent
10:                 type∈parent←type∈parent∪ ⋒_c^Children type ∈ c
11:         Par← queue initialized with parents of all sink nodes in root
12:         while Par is not empty do
13:             first← dequeue(Par)
14:             Children← all children offirst
15:             for all child∈Children do
16:                 Remove from type∈child type∈first
```

Algorithm 7 describes how components of an action may be moved around in a FIDAG. To understand the effects it has on the hierarchy as a whole, we provide Equation 3.3, which describes how many fewer actions a system needs to examine **if** it is only using one component.

$$reduction = 1 - \frac{|type \in \mathbb{ACT}| - |type \in CombineProperties(\mathbb{ACT})|}{|type \in \mathbb{ACT}|} \tag{3.3}$$

While Equation 3.3 is useful in determining the amount of redundancy in an action hierarchy, there are several occasions (shown in Section 3.4) in which it is too optimistic a measure. As Algorithm 7 moves partial information around, it may not be applicable to components of actions that require the entire component to be moved. In those instances, it is preferable to use a binary

reduction, seen in Equation 3.4. As binary reduction determines the change in the number of actions with a component, it provide a more realistic approximation of how the hierarchy is structured and how much information is un-necessary for actual processing.

$$reduction = 1 - \frac{|\mathbb{ACT} \text{ with } type| - |CombineProperties(\mathbb{ACT}) \text{with} type|}{|\mathbb{ACT} \text{ with } type|} \tag{3.4}$$

3.4 Experimentation

Up until this point, we have discussed different considerations when organizing actions into an action hierarchy. To show the effects of organization, we examine an application based approach. We have implemented, to the best of our abilities, two different agent reasoning algorithms: Normoyle et al. [4] as an aleortic action selection mechanism and Kartal et al. [77] as a narrative generation mechanism. Each of the methods, as well as any changes made to them, are described earlier in this chapter. The purpose of these two methods are to display the changes in run-time that intelligently organizing action sets afford. We use the actions from event-centric planning [1](Appendix A).

We wish to show the effect structuring $\mathbb{ACT}$ has for our two application based components, roles and condition-assertions. Therefore, we run our compression algorithm on different tree structures for both components, with the results being seen in Figure 3.4. Note that a couple of tree structures, the *WordNet Hypernym Single Parent* trees (shown in Appendix A) and *Word vector clusters* were generated using the methods described in Chapter 4. The two hand-done trees use the *WordNet Hypernym Single Parent* tree as a base generation. For roles, actions were placed under the same parent as long as their exact semantic difference (using Equation 2.1) is zero. A similar procedure was performed for condition-assertions. In Figure 3.4, we show two measures for compression, partial (Equation 3.3) and binary(Equation 3.4). This is to show the contrast and how similar each are when measuring movement of information in the hierarchy.

As can be seen from Figure 3.4, using **P** allows for components of an action set to be combined together, causing compression in the actions. This occurs, for the most part, whether the method of building an action is focused on those components or not. What is important to keep in mind is

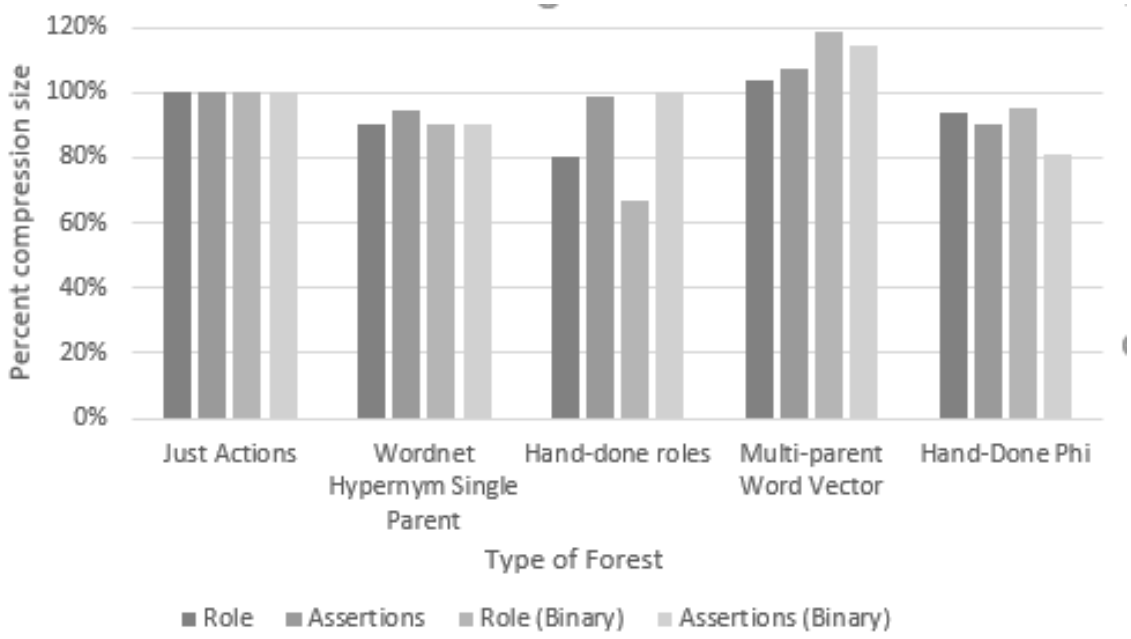


Figure 3.4: The compression size for both Roles and Assertions for our action sets

non-binary compression is being measured on the components, and not the actions themselves. So, a role focused method may have actions that are exact in all of their roles as siblings, but partial matches will also be compressed down. While we do not explore moving $r \in \mathbf{R}$ in our tests, it is an interesting and possibly useful effect of Algorithm 7.

In should also be noted from Figure 3.4 the FIDAG word-vector method showed negative compression. This is because the actions' parents branch just after each defined action. This causes the roles and condition-assertions to go to multiple parents, which means any compression would be undermined by the component existing on multiple parents. Compression may occur on different word-vector models, or if property movement was cleaned up by removing the multiple parent redundancies

We first show the effects of role compression on run-time using a role-based action selection mechanism, Normolye et al. [4]. We compute the change probabilities for each tree based on the roles in the system. To do so, our algorithm finds all actions with roles attached to them, and considers this the action set the virtual agents have access to. End probabilities and connectivity

between actions are randomized for each run. It should be noted that a single action set of twenty is a small set for this algorithm, and so we exaggerate the computational effects by allowing each action with roles to have ten locations (so that the system is calculated on ten times as many actions). This test was run on an Intel Xeon 2.3GHz eight core processor on a single thread. We run 1000 trials, reporting the average in Figure 3.5.

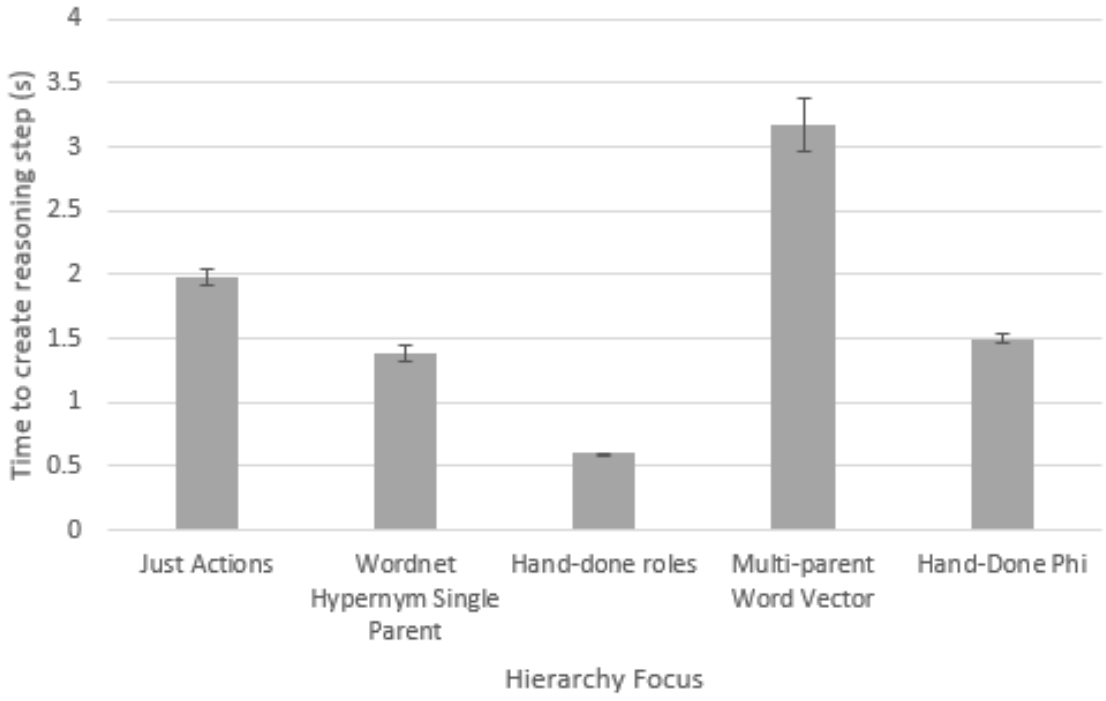


Figure 3.5: The time to compute a reasoning step of Normoyle et al. vs. the method of organizing a hierarchy.

What is not surprising from Figure 3.5 is that, as the compression of the roles in the action tree increases (so overall number of actions decrease), the average time to run decreases. As the number of actions decrease, the number of equations the convex optimization algorithm must solve decreases, and so the result is expected. What is a bit more interesting is that the standard deviation

decreases dramatically as the compression increases. This should also be expected. As the connectivity of the system is controlled by the overall number of considered actions, then reducing the total number of actions will reduce the total possible connectivity. Less variance in the connectivity will mean less variance overall. Figure 3.5 follows the binary effect shown in Figure 3.4, showing that for this method, the measure presented in Equation 3.3 is useful for quickly calculating if an organization is useful.

We also test out the effect different organizations of the same hierarchy have on conditions-assertions Note from Figure 3.4 that the compression ability of condition-assertions were much less than roles. Condition-assertions are much more varied than roles, and are therefore harder to compress. We use Kartal et al. [77] as our goal based testing algorithm, with the changes described previously. Each tree was tested with five random assertions to complete nineteen times on an Intel i5 four core processor on a single thread. We report the average runtime in Figure 3.6.

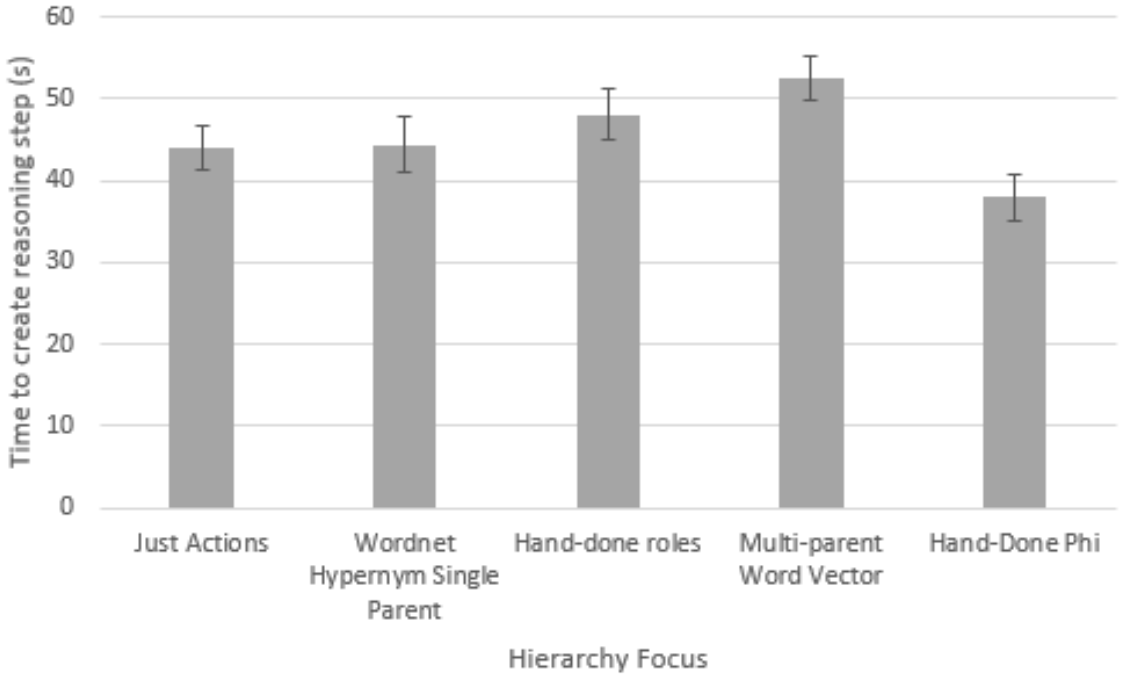


Figure 3.6: The time to compute a narrative plan of Kartal et al. vs. the method of organizing a hierarchy.

As can be seen from Figure 3.4, the binary hand-done condition-assertion focused is the best for condition-assertion reduction, whereas there is not as much difference when using Equation 3.3. Figure 3.6 shows that the binary reduction better reflects how the change effects the runtime for Kartal et al. This is because reduction on the search tree is affected by the **total** number of actions. If all actions are included, then the tree is in a terminal state. With less actions, the chance of all actions increases, as the the probability that are the assertions being searched for are found. The action sets consisting of only the actions, the role-focused set, and the WordNet hypernym set all performed similarly, but not as well as the hand-done condition-assertions. From Figure 3.4, the binary roles are different for the hand-done condition-assertion set, but are similar to the others. Therefore, when choosing which compression measure to use on the taxonomies, how the reasoning method works should be taken into consideration.

3.5 A Quick Note on Pragmatics

Until this point, we have discussed virtual agent actions during their design phase. At runtime, the realization of virtual agent actions is the pragmatics of the action, that is, the context specific instantiation of it. In PAR, the pragmatics of an action are assigned during simulation, creating Instantiated Parameterized Action Representations (IPARS). IPARS are stored per-agent in an action table, and are useful for examining how agents act in the environment. Correct IPAR generation is assisted by defining semantics for a given action, but is outside the scope of this work.

Pragmatics of actions are useful when defining plans for agent architectures, such as BDI-agents. Because pragmatics are defined based on the exact state of the environments, situations [86] can be defined. Using the semantics of an action allows the placeholders to be set-up for having experts define pragmatics, and agent behavior can be created from techniques such as Explanation-Based Learning [87]. Since the amount and flow of knowledge is pyramid-like, rule-sets for defining pragmatics are left to future work.

3.6 Conclusions

In this chapter, we have shown the considerations required for structuring $\mathbb{ACT}$ in FIDAGs. Due to inheritance on the FIDAG, properties of an action such as the roles and condition-assertions can be organized so as to reduce the total information that a simulation author or virtual agent must parse through. While this understanding is useful for reasoning tools, it is also important to note that there are many cases where this is simply not possible. Conflicts between multiple parents may not allow for generalizations to happen in order to assure that the components of an action are what the simulation author intended.

To assist in understanding this assumption, we have shown several different organizations of the same action set [2]. Each of these organizations have been used in two different agent reasoning and action selection algorithms, and the difference in time is shown. This should further build the case for the intelligent organization of actions. In many cases, the actions themselves are not as important to the system as the agent's ability to reason about them, and so properties of actions that assist in an agent's reasoning ability are extremely important components of an action set.

[2] the action set is found in Appendix A

Chapter 4: Automated Generation of Action Hierarchies

The addition of a definition and description of PARs does not by itself ease the authorial burden of creating actions. It is difficult, even with data structures such as FIDAGs that allow inheritance, to fill out all the parameters of PAR actions (such as **P**, **OBJ** and **S** in $\mathbb{ACT}$). Methods to automate the generation of parameters of actions would reduce the overall effort creating actions, affording virtual characters more abilities by connecting more motion controllers and having more nuanced actions for AI reasoning systems. Having automated methods operating over a pool of data will also create more consistent actions, as semantic parameters that appear in several actions in a given data-set should be picked up by an automated system.

Methods to generate ontologies have been well studied in the case of general information retrieval [34, 33, 88]. When generating ontologies, it is normal to compare the generation system to a gold standard ontology. This provides an accuracy, precision and recall factor that can be compared to other generation systems. Some variations on this include posterior precision (in addition to prior precision) and comparing generation to human ontology building. Comparing to human labelers is one method that is used in this chapter to determine the ability of the system and is useful for understanding how similar the automated systems function when comparing it to humans. However, specifically examining actions and generating an action FIDAG ($\mathbf{P} \in \mathbb{ACT}$) for motion controllers has not been examined before. The restrictions that are inherent to a virtual character operating in a rich virtual environment must be taken into account during the generation process. Mainly, it is important that the information can be retrieved quickly by an agent and not contain a lot of redundant information, as redundant information must still be compared in an AI system to ensure it is the same during reasoning phases. Also, as described in Chapter 3, action information that can be generated on a more general parent (and higher up the action hierarchy) means that there is less information for the simulation author to examine, in that the information percolates downward to child actions. Therefore, the amount of information should be compact. Another metric to show the

efficacy of automating the generation of **P** is to measure the change of the system when it undergoes Algorithm 7(Data Compression) from Chapter 3, and use the metrics associated with that.

In order to automate semantic generation of actions, the exact meaning of an action must be understood by the system, such that ambiguities inherent in action understanding are removed. To do so, we use the name of the action as the identifier and then the proper *sense* of that action's name is determined, when possible. From natural language, this is a process known as *Word Sense Disambiguation* and a survey can be found at [89]. To dismabiguate a word's sense, the word must have context, which generally comes from a sentence. There has been some effort to build features of descriptions based on motion parameters [90]. There has also been some work on determining how humans perceive differences in motion [71] and in semantically segmenting actions [91] which could be used to examine properties of motion data. The work on building and segmenting features from motion data has been focused on describing character movements such as *walk* and *limp*. More complex actions that could use several motion controllers, such as actions like *water plant* and *cook* are not described by a single motion parameter, and are more naturally described through language. Motion data with lower resolution, such as a character that does not have finger joints for an animation, would also not work though motion controller parsing, but would work by looking at meta-data like the name. Higher level actions such as *Live Life* have not been examined, and are outside the scope of this work. From this, we formulate the overall hypothesis for this chapter as:

- H 4: The creation of action generalizations ($\mathbf{P} \in \mathbb{ACT}$) can be automated from text if some prior knowledge of the actions are provided.

To explore H 4, we create three different methods of automated action generalization. Two use a natural language knowledge base in order to cement the exact meaning of the action and build up generalizations. The other uses machine learning to determine relationships between the names of actions. We explore these methods in order to determine how a change in the process affects a change in the generation and to provide a baseline to compare each method to. The general form of this process is seen in Figure 4.1. Note that this process uses user supplied names and keywords and a data-source such as a lexical database, and creates an action taxonomy that can be examined by the simulation author, who can then change the keywords to produce different results.

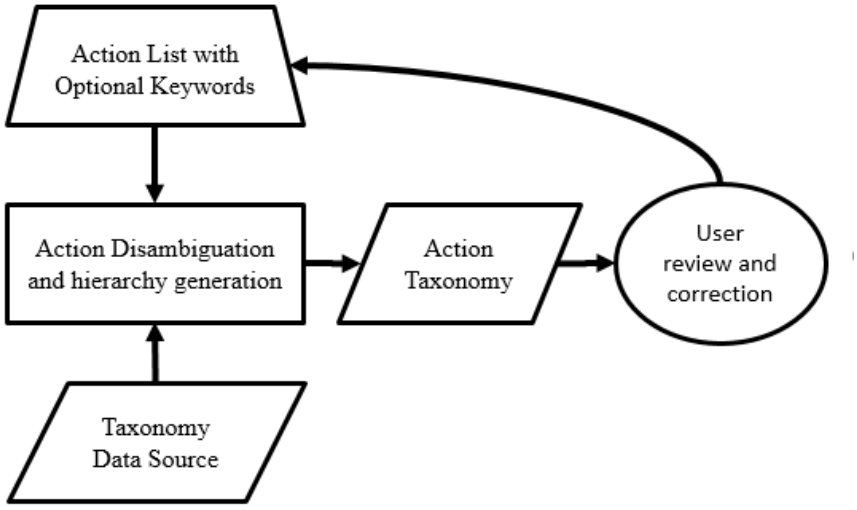


Figure 4.1: System Overview

4.1 Data Sources for Automated Generation

Converting action names into an action hierarchy requires understanding the *sense* of a given action, that is, the proper context a given action would exist in. This can manifest itself through either its related words or meaning, as seen in Table 4.1. As another example, the phrase *pick up* has at least two senses, *to lift an object* or to *understand an idea*. Other properties of an action that can participate are vastly different depending on which sense the user desires, and different animations for a virtual human would accompany each sense. To disambiguate polysemous words, physical humans will examine a word's context, and determine which definition best fits the context. This is known as word sense disambiguation. A good survey of the topic can be found in [89]. Most word sense disambiguation techniques focus on a given context, such as the sentence in which the word is found. Unfortunately, even well named motion clips and processes do not readily appear in full sentences with enough context to determine their sense.

For both word sense disambiguation and meaning generation, the choice of knowledge bases impacts our system's abilities. For our system, we use a knowledge base that captures a linguistic

Table 4.1: An example of the polysemous word *Cook* taken from WordNet. Only the verb senses are shown.

Term	Synonyms	Definition
Cook	n/a	Prepare a hot meal
Cook	Fix, Ready, Make, Prepare	Prepare for eating by applying heat
Cook	n/a	Transform and make suitable for consumption by heating
Cook	Fudge, Manipulate, Fake, Falsify, Wangle, Misrepresent	Tamper, with the purpose of deception

understanding of actions, WordNet [36][1]. We use WordNet to determine our initial sense due to its larger collection of verbs and the more prevalent parent-child relationships. The tree structure of WordNet allows verbs to be understood more generally, which we exploit in our process. In generating the hierarchy, we are more concerned with WordNet as our knowledge-base.For some of our tests, we also utilize FrameNet [92][2]. FrameNet contains operational information in the form of **Frame Elements (FE)**, which are the participants of a frame of a verb. For example, a *sleep* verb contains the FEs *Sleeper* and *Place*. Shi and Mihalcea have connected FrameNet frames to WordNet senses [93]. To show the efficacy of our approach (Section 4.5), we use FrameNet Frames to show that connections made in later chapters are possible.

4.2 Using WordNet to Generate Ontologies

4.2.1 A Multi-sense Method to determine the Sense of an Action

To determine parent actions and build an action hierarchy, our method first reasons about the actions that a simulation author has created. Figure 4.1 illustrates how these lexical databases fit into our overall system. The list of actions available to be performed in the scenario are disambiguated, mapped to WordNet verb synsets, and formed into an action taxonomy. These actions can come from a variety of sources such as low level atomic animations or procedural controllers (e.g. picking up an object). The sense candidates chosen from WordNet are directly related to the name a

[1] Also found online at http://wordnet.princeton.edu

[2] Also found online at https://framenet.icsi.berkeley.edu

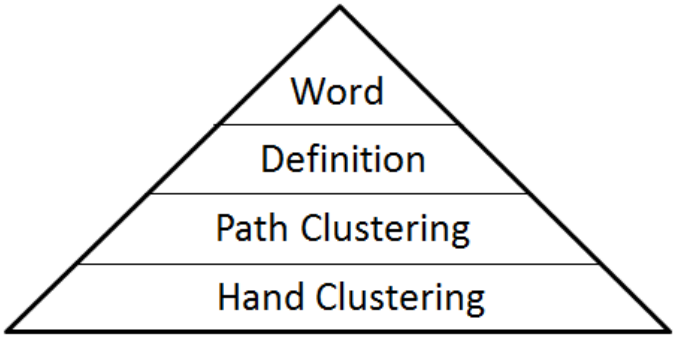


Figure 4.2: The Multi-pass technique's testing conditions. Each level determines word senses with the most precise methods higher up in the pyramid. Techniques lower in the pyramid are less precise but have greater coverage.

simulation author provides for the action. Therefore, the level of detail and ability of an author to describe an action has a large impact on the system's ability to derive its sense, with more general methods being used to disambiguate fuzzy situations as is done cognitively by physical humans [94]. To enable this ability in our system and provide more information to an agent reasoning about actions, we maintain much of the synset hierarchy as a taxonomy of actions. The hierarchical nature of verbs also lends itself naturally to a hierarchical, or multi-sieve, approach, as seen in Figure 4.2. Our hierarchical method examines word senses using information on the word itself, its children, definition, and the relationship to other found senses.

Unlike previous word sense disambiguation methods, our method examines an action name λ given a small constrained set of user provided keywords $k = \{k_1, \ldots, k_n\}$ and determines the proper sense of λ from the set of candidate senses $\mathbf{c} = \{c_1, \ldots, c_m\}$. λ can be any word or word phrase, such as *pick up* or *duck and shoot*. For the verb *cook*, a user might provide the keywords *food* and *heat*. As our domain specifically focuses on actions for virtual agents, we assume that λ contains at least one verb that can be performed by a virtual actor. This is an important assumption as it greatly prunes the search space of candidate senses for a given verb. It is also a reasonable assumption for our target applications. The keywords should contain some context to the given sense of λ, however, due to the descriptions found in animations, this may not always be the case. The name of the motion itself may only give a partial clue to its nature. We process λ to determine $\mathbf{c}$ by first

searching for **c** using λ. If no results are found, we determine if λ is a phrase, and search for **c** from each verb in the phrase. If no results are found at this stage, **c** is considered unresolvable.

Once our system has determined the set of candidate senses **c**, we search for the most likely candidate sense, c_{found}, which will be the label for the parent action. We find c_{found} by testing **c** against each method seen in Figure 4.2 and described in more detail in the following paragraphs, using Equation 4.1. α is used to reject c_{found} for low matching senses, and allows other methods to be tested in an attempt to find a better match, while not undermining high probability matches. Through testing we have found that $\alpha = 0.2$[3] provides a strong threshold while not being too discriminating.

$$s_{found} = argmax(method(s)) > \alpha \tag{4.1}$$

Word Disambiguation: The first methods used in our system attempts to disambiguate using only the sense set **c** and a set of k, similar to the disambiguation of agent commands done in [96]. As was done in [96], we use the lemmas of a sense, which are closely related to synonyms, as well as the parent verbs in WordNet's hierarchy, and determine the number of k that matches each candidate in **c** on these metrics. We have also added a third technique, which compares the lemmas of the sense's child verbs in WordNet's hierarchy to k by performing a Breadth First Search on the tree of the sense. In order to get a percentage score to compare to α, we divide the total matches by $|k|$. For our example of *cook* with keywords *food* and *heat*, this method results in a score of zero. Neither keyword is found in the set of synonyms for *cook*.

Definition Disambiguation: If the above technique cannot disambiguate the sense of the word from its given components, we then examine the definition of each sense. The definition of a sense is a more relaxed search, and provides context to each sense in **c**. Testing the definition of a sense against k is a relaxed string matching approach. If a keyword matches any of the words in the definition, then it is considered a match, and is scored similarly to the word disambiguation technique. We also determine a percentage score for these techniques by dividing the total matching found by $|k|$. Here our *cook* example yields a score of 0.5, because the keyword *heat* is found in

[3] After a more extensive analysis, we found experimentally that 0.2 is a good cutoff metric for sense matching using Equation 4.1. This differs from our work that was published in [95], in which we used 0.3. In reality, the choice of α is dependent on the dataset used, which is why it is left as a free parameter.

one of the sense definitions, but *food* is not. As we have found a sense whose score is higher than our α, this sense is chosen.

Frame Clustering (Confidence Based Symbiotic Actions Only[4]): This disambiguation metric is only used for disambiguating actions and object information about the virtual environment when the objects are already known. It uses k combined with λ, for each $\lambda \in \mathbb{OBJ}$ (added to k) to determine how many connections there are. Frame Clustering is based upon the Functional Elements of FrameNet [92]. Our heuristic computes the number of keywords that appear in the FrameNet frame compared to all other keywords. Any candidate that does not have an corresponding frame in FrameNet (using its hypernyms) is removed from consideration. These heuristics, including our novel functional heuristic, are scored using the Jaccard index seen in Equation 4.2. The Jaccard index does not consider order, and so the end result of Equation 4.2 is how similar the frame is to whichever set (keywords or object names) it is being compared to.

$$score_h(\mathbf{c}_i) = \frac{|param(\mathbf{c}_i) \cap k|}{|param(\mathbf{c}_i) \cup k|} \tag{4.2}$$

Path Disambiguation: The final automated technique used to determine the sense of λ is to compare $\mathbf{c}$ to the already disambiguated verb senses. This technique follows [97], which has been previously used to connect WordNet senses and FrameNet frames. Provided the system has determined one correct sense, this technique uses the Wu-Palmer Similarity [70] of $\mathbf{c}$ to determine the percent similarity to the already found senses of actions. This method is an iterative approach, and so will attempt to connect senses until no new senses are found.

For our confidence based method, we perform a slightly different version of path disambiguation. We still compute a global context score using the Wu-Palmer similarity metric [70] between all candidates of all other actions/objects. This metric keeps the highest similarity score, All similarity scores for a given action and object are normalized between all candidates. Note that this is a slight departure from **path similarity** mentioned earlier, as it compares the paths between all candidates,

[4]Through testing, we found it had no effect on the multi-sieve method. We postulate that this is due to the inclusion of an object hierarchy in the confidence based method

and not just between the ones that were found. As our confidence based method does not determine confidence until after all measures are tested, we do not have known actions to compare to. Therefore, we simply compare to the candidates.

Hand Clustering: When there is not enough information to disambiguate **c** or if none of the techniques are able to resolve the sense of a verb, then manual disambiguation is necessary. Our system provides tools for a user to choose the correct sense given a definition and list of synonyms of each sense in **c**. If the user cannot find a given sense at this stage, then the action name is considered unresolvable.

4.2.2 Hypernym Tree Generation

After the initial examination and disambiguation of verbs, we build a FIDAG, which exploits WordNet's hypernym hierarchy for its creation. This allows a virtual agent to reason about an action using a broader definition (e.g. being able to understand that a *pirouette* is a *turn* or that *baking* is a form of *cooking* which is in turn a *creation action*). To construct $\mathbb{ACT}$, we first generate the full parent list of each disambiguated verb. For example, in Figure 4.3(b), the black circle is an action, and all other circles are the hypernym parents of that action. We then perform a node comparison, using Wu-Palmer path similarity [70] to the leaves of all previously generated trees to find candidate trees that the sense may belong to (See Figure 4.3(a)). We then examine all nodes in the list and candidate trees, searching for a direct string match for a given verb sense. When one is found, that node and all child nodes are directly connected to the tree. This allows a large amount of information to be reasoned over, increasing the total coverage of the system.

Once the actions' word senses have been determined and the action taxonomy created, the actions can be further processed, such as linking the actions to object participants, thus providing agents with information about how the objects can be used in virtual worlds.

4.2.3 Confidence Based Symbiotic Generation

Our method of symbiotic generation for actions and objects relies on the idea that a virtual character has a known set animations and graphical objects and that there is some underlying meaning in the

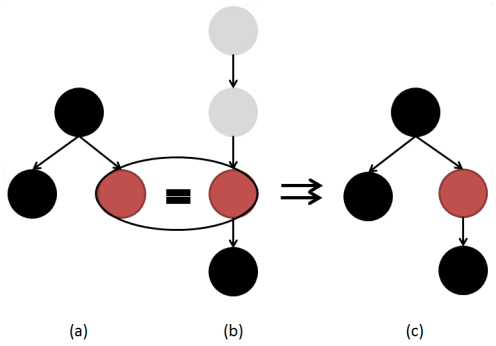


Figure 4.3: A node in an action hierarchy being added (b) matches a node in an existing tree in (a). The hierarchies are merged at the common node and the ancestors of (b)'s matching node are removed (c).

connections between actions and objects. We start with the same assumptions as our multi-sieve method. Our method first builds a taxonomy of objects and actions, starting from an input list of actions and objects in the scenario. Each action represents a meaningful name of a motion controller and each object name represents the meaningful name of a graphical object. A high level explanation of our population algorithm can be seen in Algorithm 8. This algorithm disambiguates objects using keywords provided by the simulation author as well as the base names of the actions, generates potential properties for each object, and then uses that information to augment the keywords a simulation author uses for actions. A disambiguated object set ($\mathbb{OBJ}$) may also be used, removing the need for steps 5-12. It is important to note that the population phase is now based on keywords (similar to [37, 10]) and information inherent to the simulation (i.e. the other actions and objects). The information inherent to the simulation helps the population algorithm determine global context, as the virtual agent can only perform the actions in the simulation and only interact with objects in it.

The disambiguate function on lines 10 and 17 of Algorithm 8(Confidence Based Generation) provides the most likely candidate sense for a given object or action, provided one exists. To determine both the most likely sense of an action/object and the confidence that the given sense represents the animation/graphical model, we compute a set of features that are processed using a Regression Decision Tree [98] model. The decision tree uses mean squared error as its splitting function, and

Algorithm 8 Our Hierarchy Population Algorithm

```
1: function CONFIDENCE BASED GENERATION PROCEDURE(OBJ,ACT)
2:     OBJ←The set of object names and associated keywords
3:     ACT←The set of action names and associated keywords
4:     C ← new Hash Table
5:     for all obj ∈ OBJ do
6:         c ← candidates of obj
7:         k ← keywords of obj ∪ λ ∈ ACT
8:         insert (c,k) into C
9:     H ← new Hierarchy of 𝕆𝔹𝕁
10:    insert Disambiguate(C) into H
11:    Empty(C)
12:    for all act ∈ ACT do
13:        c ← candidates of act
14:        for i from 1 to —c— do
15:            if c_i has an attached frame then
16:                add frame(c_i).FE to keywords only for c_i
17:            else
18:                remove c_i from consideration
19:        k ← keywords of act ∪ λ ∈ 𝕆𝔹𝕁
20:        insert (c,k) into C
21:    insert Disambiguate(C) into H
22:    return H
```

we only require two samples for nodes in the decision tree to split. As we have a heuristic that is specific to actions, we train a separate model for object disambiguation and action disambiguation. Finally, when we insert the disambiguated terms into lines 10 and 17, we insert the WordNet [36] sense with its entire hypernym tree, creating a forest of hierarchies for actions and a single hierarchy for objects.

$$c_i = set(\frac{score(c_i)}{avg(score(c - c_i))}) \tag{4.3}$$

In order to compute features to predict the confidence of a candidate, we provide several heuristics. Many of these heuristics use the context provided by the scenario and simulation author (such as the names of the objects and other descriptors for actions) and are contained in the set k, discriminated to each set of candidates. We use all the heuristics found in Section 4.2.1, but each candidate receives a score for each heuristic, shown in Figure 4.4.

A note on the global context heuristics, using the frames and action paths. Obviously, both of

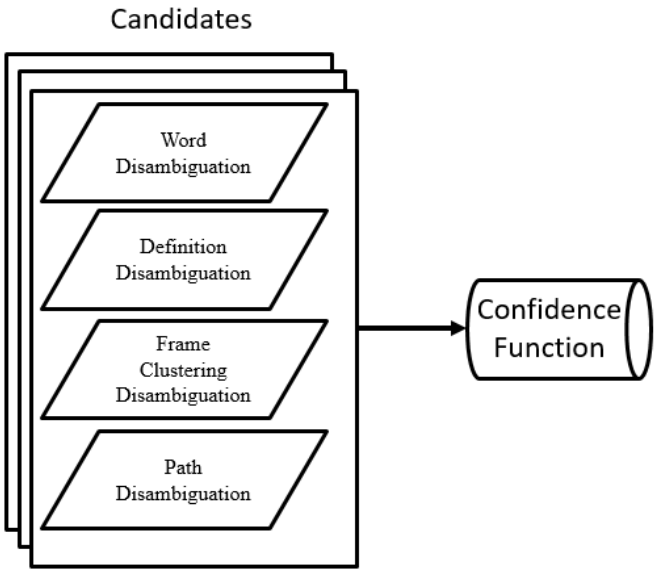


Figure 4.4: An overview of how our confidence-based method disambiguates action candidates.

these heuristics are sensitive to the global context and data with which they derive their scores. This has the possibly unintended effect of biasing the actions and objects towards one action, making different animations and graphical objects which are not meant to be considered synonymous synonyms. For example, if a virtual character can *run* a program and *run* on a treadmill, the global context heuristics do not pick up the difference, but instead score based on the same candidate. However, as all other heuristics are keyword based, they may not be able to find a connection if the set of keywords are not descriptive enough.

4.3 Using Continuous Bag of Words to Generate Hierarchies

In the past few years, there have been recent developments in word representations. Work such as WordNet [36] represents words as synsets, which contain links to generalizations, specializations, synonyms, antonyms, and other information about the word in a human readable format. Another way to represent a word is by a *Bag of Words* representation [99]. A bag of words assumes a finite, unchanging set of words in a dictionary. The entire set can then be represented as a vector, where

each position of the vector corresponds to a word in the set. An example five item bag of words is shown in Figure 4.5. In the example, a word vector is created for the sentence "jane eats", where each word corresponds to a given position in the bag. Multiple sentences can then be compared to each other as vector differences. Note that the position is lost between words in a bag of words model, so that some semantic information is lost on the sentence level.

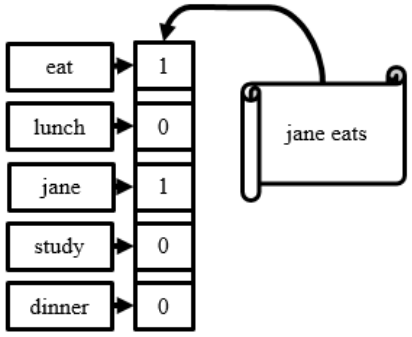


Figure 4.5: An example bag of words, and the word vector from the sentence "jane eats".

Similar to traditional bag-of-words models are Continuous Bag of Words and Skip-gram models [100]. However, instead of each word having a specific position in the vector representation, the vector representation is a (generally unknown) set of features. Words are then categorized by continuous values of these features. Word Vector models have grown in popularity since their introduction in [101], including such work as Word2Vec [100], Sense2Vec [102], and Shape2Vec [103] Shape2Vec combines word vectors with vectors describing 3-D graphical models in an attempt to have a common language between graphical objects and words. Word vectors have also been described as being able to contain levels in their knowledge representation [104]. If they can contain graphical objects and generalizations, then word vectors may be useful as an abstract method of representation. Because word vectors are not human-readable, then it wouldn't be appropriate to present them to a simulation author as is. Furthermore, one of the benefits of the PAR language is that actions can be added and removed by a simulation author without affecting other actions. A

pure skip-gram generalization (such as what is claimed in [104]) would need to be retrained for each new concept, making that hierarchy more static. Word Vectors could, however, be used in place of WordNet sysnsets for the population of action generalizations, just not for the actual representation. Therefore, we test out the effects of creating hierarchies from word vectors.

4.3.1 Creating a Hierarchy using Word Vectors

Once a model has been created, a vector representation of each action can be determined. However, the vector by themselves does not indicate relationships between the actions. We wish to find an indeterminate known grouping of words that may exist on multiple levels between actions. Furthermore, we wish for this hierarchical grouping to have a human understandable meaning. Therefore, the word vectors developed are used to determine generalizations between groups of words. We cluster word vectors together, using DBScan [105] in order to determine similar words. DBScan is a neighborhood searching method that builds clusters by examining distances between different data-points (word vector), clustering points that fall under a given threshold. If enough points are within the threshold (chained together, see Figure 4.6), then the cluster will be created. Any points that cannot be clustered together are labeled as such. DBScan is generally used on large data-sets because its clustering method is fast. For small data-sets, we reduce the number of minimum samples needed to create a cluster (such as allowing two close enough data-points to create a cluster). Because word vectors are trained on large corpora, verbs and nouns generally have several similar components. Therefore, we run PCA analysis on each word vector before using DBScan, transforming the data-points into their two most representative components.

Once a preliminary grouping is determined, a generalization of the grouped words is needed. Unlike WordNet, which has a rote generalization for many actions, word vectors do not have generalizations trained. Furthermore, the words that should be generalized together will change based on the clustering parameters. Therefore, generalizations between words are based on the word most closely related to all other words in the cluster. We do this through cosine similarity on the vector space. The generated word becomes a new action in our hierarchy, and the process is run until no more actions can be clustered together or a desired height of the hierarchy is reached.

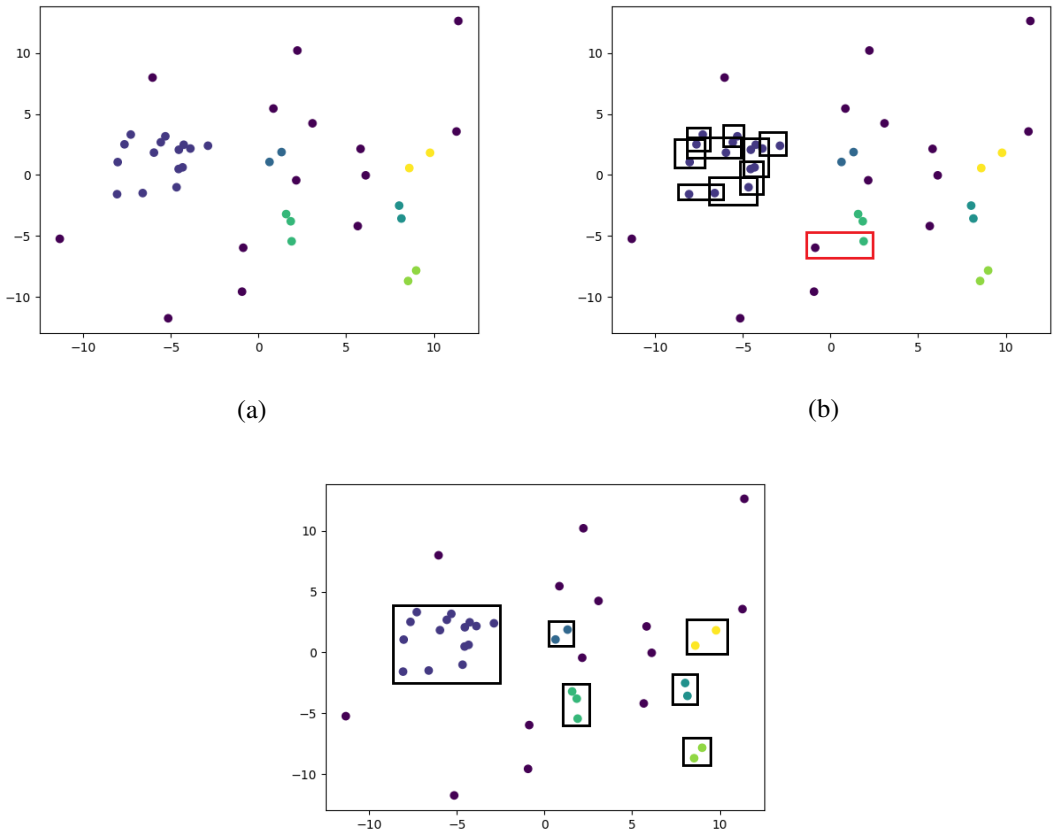


Figure 4.6: A two component PCA-based word vector representation of the CMU data-set. (a) The actions are shown with their two components. (b) DBScan measures the distance between actions, connecting any neighbors based on their distance. (c) The final clusters are combined based on the connected neighborhoods.

One issue with using a continuous bag of words model is the semantic meanings of action parents. Work such as Mikolov et al. [101] describe the ability of these models to determine features such as changing one feature can transform a singular to plural word or that makes a word masculine or feminine. This is done by examining the transformation from one word to another for several pairs of words, and essentially learning the transformation from this data-set. This is important because combining several word vectors together finds the most similar word to all of them, which is in reality another word from the total set of words and not what would be considered the generalization

of that word. So long as the action hierarchy can be nonsensical (i.e so long as the parent of *Talk* and *Walk* can be *Laughter*), word vectors are a great representation for actions and their generalizations. What may also be desired is determining the topic from a set of word vectors [106]. Topic modeling using word vectors has been performed in Liu et al. [106], and combines Latent Direlecht Allocation [107] with word vectors to determine topics based on the clustered words. They provide three different models, two in which topics are considered as pseduo words and one in which the topic word embeddings are simply concatenated to the word embedding. The real difference between the three is simply in how the topic is used as context, and show that the three produce different results.

While each of these methods produce different relationships between words, they all rely on the same idea, that words can be represented as a vector, and that relationships between vectors correspond to meaningful relationships between words. If the relationship between words is known, then relationships between vectors can be determined. Without that knowledge, methods to determine relationships between vectors are needed, in order to build some semantic understanding of the set.

4.4 Automated Generation of FIDAGs

Single generalizations are not the only generalizations that can be populated using our methods. As described in Chapter 2 Definition 2.6.1, actions may have multiple generalizations, creating a Forest of IsA Directed Acyclic Graphs (FIDAGs) structure instead of an IsA hierarchy. In Chapter 3, we have described the use and consideration of FIDAGs in regards to traditional IsA hierarchies. Therefore, here we also describe how to populate FIDAG generalizations for actions.

One way to populate the FIDAG is to associate more direct generalizations (candidates) with a given action or object. Note that both the multi-sieve and symbiotic generation algorithms use a cutoff threshold α when determining which candidate best suits the action or object. By keeping more candidates, a FIDAG can be generated (seen in Figure 4.7). The two methods we examine for FIDAG creation slightly change Equation 4.1, such that we are able to consider multiple candidates as actual parents. The first, which is an all parents method, uses Equation 4.4. This assumes any candidate that is greater than α during the phase α is compared is a parent of the action. The second, a highest parent method, still assumes that the correct parent is the highest parent greater than α

during the phase α is compared. This means that only the highest ties are considered disambiguated parents of the action, instead of having ties being broken arbitrarily as is done in the single parent method of our multi-sieve method [95].

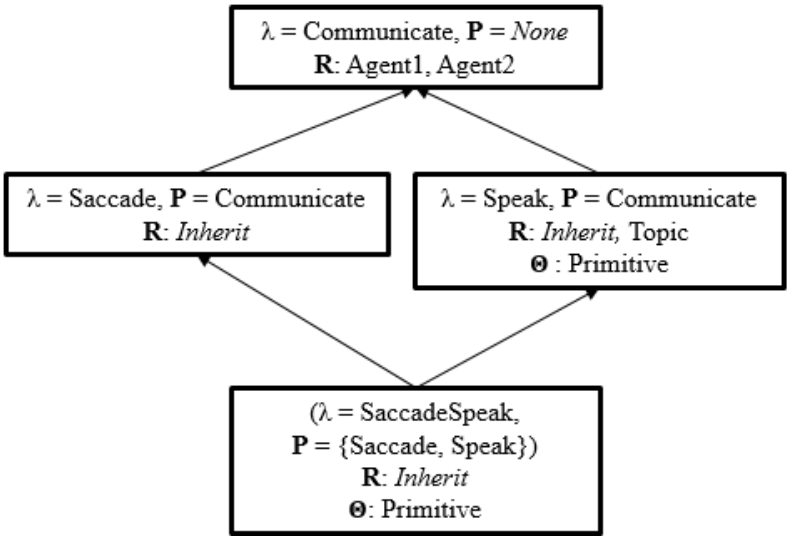


Figure 4.7: A Sample FIDAG generated for SaccadeSpeak using a hypernym multi-sense method.

$$\mathbf{s_{found}} = s_i, \forall s, method(s) > \alpha \tag{4.4}$$

In addition to multiple generalizations being determined during disambiguation, creating a word vector FIDAG can be accomplished by assuming many actions or objects are actually compound words. Splitting up these words increases the chances of finding a word vector for that word, and associates more word vectors with the given action or object name. An example of this can be seen in Figure 4.8. Due to the nature of word vector tree creation, the parents have a tendency to converge, although its ability to is dependent on the number of cluster models used and their size.

Both of these methods produce similarly structured FIDAGs, in that much of the multiplicity occurs lower in the structure. This allows us to assume something about the tree itself, in that much

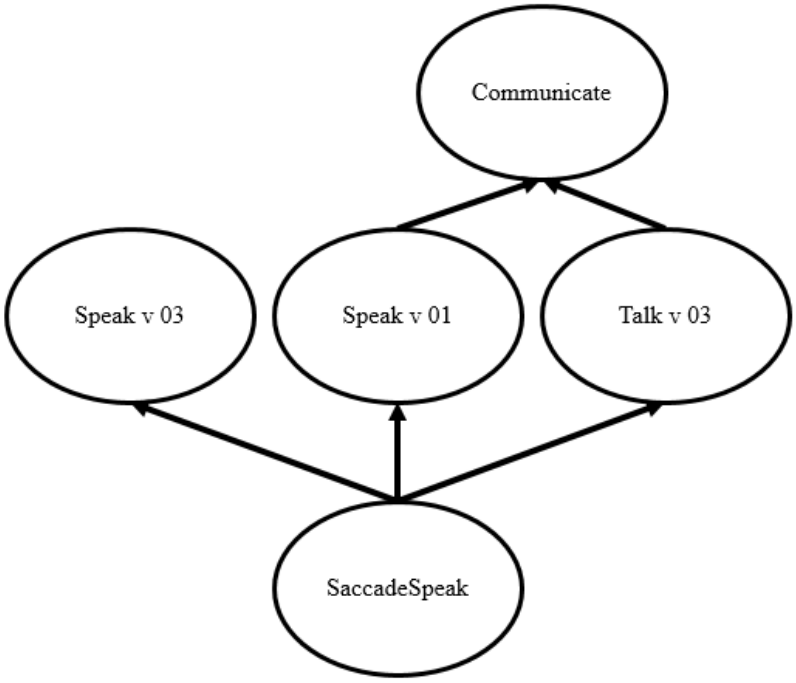


Figure 4.8: A Sample FIDAG generated for SaccadeSpeak using the word-vector tree building model.

of the considerations discussed in Chapter 3 are going to occur close to the action and object itself, meaning conflict resolution can be performed on each action, without having to examine most of the FIDAG structure. This was found when combining properties in Figure 3.4 Lowering the total number of nodes in the entire tree that need to be examined by a simulation author eases the authorial burden while still providing expressibility in generalizing our action sets.

4.5 Experimentation

In order to test out **H 4** and the ability of our method to determine the sense of an action from its name and keywords, we have formulated and tested several hypotheses:

- **H 4.1**: A consistent action hierarchy can be created using well named motion data and on-line lexical databases.
- **H 4.2**: Using a mutli-pass method is a more accurate way of determining the sense of a verb

obtained from an animation's name than any one method.

- **H 4.3**: The keywords attached to each action should describe the sense of the action, and therefore count occurrence should not be used to disambiguate the actions.

- **H 4.4**: Using a confidence based decision method produces a higher precision-recall curve than using a multi-sieve method

- **H 4.5**: Both a word vector tree-building method and hypernym tree-building method are able to connect to FrameNet frames

- **H 4.6**: Using FIDAGs connection provides more first parent generalizations, and those first parent generalizations are compatible with one another

- **H 4.7**: An all-above threshold for FIDAG generates less fully compatible parents than a highest threshold method, and does not more accurately match ground truth

- **H 4.8**: Using a hypernym tree-building method produces more functional action and object FIDAGs than using a word vector model for both single parent hierarchies and FIDAGs

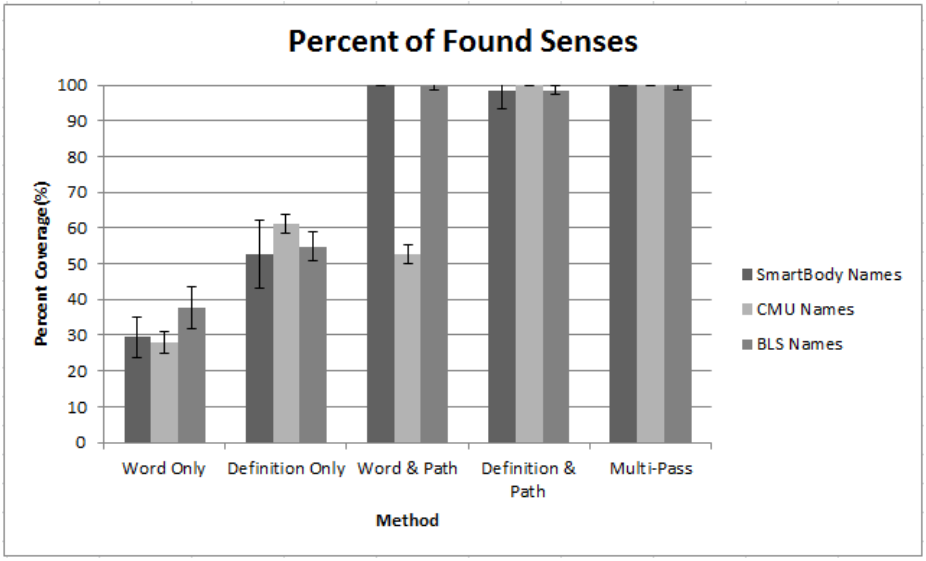
The first three hypotheses are concerned with the ability to connect actions to a knowledge base in order to build up generalizations. This is the core of **H 4** in that **H 4.1** and **H 4.2** show that actions can be better understood if they are connected to other knowledge sources. **H 4.3** adds that the ability of the system is only as good as the data going in, and that the keyword descriptors play a vital role in the ability of the system. **H 4.4** and **H 4.5** examine more methods to determine action disambiguation, which shows more generally that these action semantics can be realized through text. To consider the multiple parents of PAR, **H 4.6**, **H 4.7**, and **H 4.8** are tested, which describes $\mathbf{P} \in \mathbb{ACT}$ instead of a single parent. These hypotheses support **H 4** for multi-parent actions instead of just single parent actions, which **H 4.1-5** are concerned with. While **H 4.7** and **H 4.8** are not required to prove **H 4**, it does provide some understanding into the effects of having more information on an action set.

To test **H 4.1** and **H 4.2** we used action names from CMU's motion capture library [108], a scan of the SmartBody documentation [66], and a higher level data set that represents common behaviors

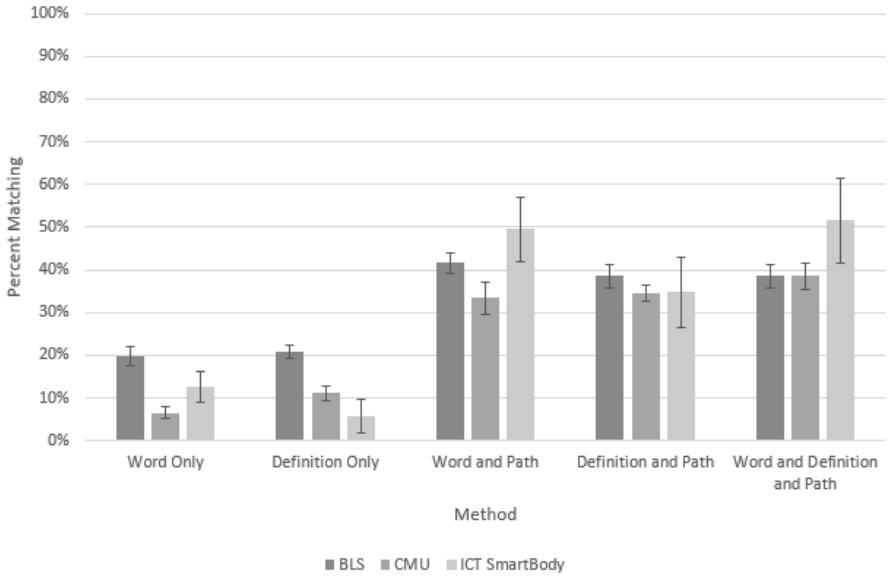
listed in tables provided by the Bureau of Labor Statistics (BLS) [109]. There is some overlap between senses from the SmartBody and CMU action sets. This creates a list of sixty actions, fifteen actions, and forty two actions respectively that a virtual human would be capable of. Naturally, behaviors in the BLS set do not inherently correspond to virtual human capabilities, but the set provides us with another source of potential behaviors. We create a ground truth for each data-set that contains the correct sense for each action by hand examining the senses in WordNet. This was performed by two encoders, one of which only had knowledge that the action was performed by a virtual human. The other coder had access to the animations, but did not reference them when determining ground truth. There agreement between the two coders was around 50%. For each dataset, we also create two lists of keywords[5]: a definition list from the Merriam Webster's dictionary definition[6] (generally the longest words in the definition) and a list containing synonyms from both Merriam Webster's dictionary and thesaurus.com[7]. We then create several tests from the set of keywords, choosing either the definition or synonym set for each action. These are used to compare our overall method to each of its components. The results can be seen in Figure 4.9. As the path similarity metric requires one action sense to examine WordNet paths, we combine that method with each of our other metrics.

As can be seen from Figure 4.9a almost all of the multi-pass methods that included path disambiguation [97] were able to overwhelmingly find a sense to attach to an action's name. This means that, more often then not, using path disambiguation [97] will allow for some sense to be found and an action taxonomy to be generated for most of the actions without the need for an author to manually add them. Using our word disambiguation with path disambiguation for CMU's data set was an exception. This is because this method did not find enough senses using word disambiguation to connect senses using [97]. A lower α value would allow the process to find more senses, but may reduce the overall accuracy of the system. As a result, the high percentage of found senses confirms **H 4.1**. When examining Figure 4.9b, there is an overall decrease in the system's ability to choose

[5]The definition and synonym lists can be seen in Appendix B.
[6]www.dictionary.com
[7]www.thesaurus.com



(a)



(b)

Figure 4.9: The percent of found (a) and correct (b) senses for our ten sample test. Error bars represent one standard deviation. A single factor ANOVA analysis provided a negligible p-value for both figures, with a Tukey-Kramer test showing significant difference between our method and the word only, definition only, and word with path methods for CMU's data set, and word only and definition only for Smartbody's data set. A Tukey-Kramer test shows significant differences between all methods and the multi-path method for the BLS data set.

the correct sense. Therefore, while [97] allows for a hierarchy to be created, it may not be the correct one. However, compared to methods that did not use a path method, Figure 4.9b shows that a multiple pass method is preferred when creating an action hierarchy from action names, confirming **H 4.2**.

To examine **H4.3**, we compare keywords generated from a variety of sources. We use the keywords generated for **H 4.1** and **H 4.2** as intelligently chosen hand-done keywords. We also auto-generate keywords from several text sources, including the entirety of Wikipedia, the Billion Word News Corpus, and a selection of two genres (fantasy and mystery) from Project Gutenberg [110]. Each corpus found the top keywords generated from ALET (Chapter 5), and were broken up into descriptors and verb-noun co-occurances. The action descriptors were unable to disambiguate **c** for any word, and were dropped from the analysis for that reason. We believe that because the action descriptors mainly provided adverbs to describe the actions, which did not fit into any of our disambigaution metrics. The top three and five keywords were chosen for each generated corpus, and ten samples were created by mixing combinations of the three to five keywords. These were then compared to two human disambiguation participants, and accuracy was determined if one matched either of them. The results can be seen in Figure 4.10. Test were performed both in and between data-sets for significance, showing that there is a pairwise significance between data-sets and between both the human chosen keywords and the automated keywords for each data-set.

From Figure 4.10, it should be seen that having a human choose the keywords over an automated method is generally preferred. This is because automatically generating the keywords creates a chicken and egg scenario. The disambiguated words are needed to know what keywords better describe the action, and the keywords are used to know how the action is disambiguated. The one inconsistency in this is in the BLS dataset, where the automated connections actually connect better. This is because the actions themselves are higher level, and both the system and humans had trouble determining a WordNet sense that accurately described the action. By having the default be that the action cannot be disambiguated, when human coders have more trouble, poorer data (through the keywords) actually performs better. This confirms **H 4.3**, in that keywords chosen to represent the action are more useful to the disambiguation of the action, meaning that the human needs to remain

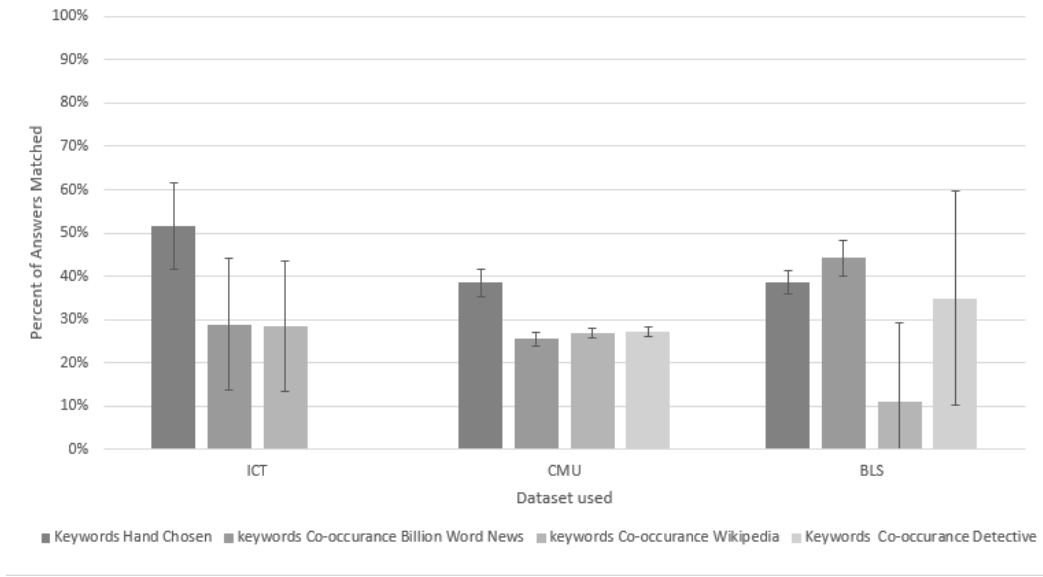


Figure 4.10: The percent matched vs. dataset when using keywords picked by a system compared to those generated from text corpora. Error bars are given as one standard deviation.

in the loop when disambiguating actions.

To test **H 4.4**, we have included 4 object datasets. Two data-sets are the names of graphical objects from two environments on the Unreal Engine Marketplace. Both of these object sets have close to 100 objects, although many are different models of the same object. We also have two academic datasets, the 101 Object Set [111] and ModelNet [112]. ModelNet has over 600 3D graphical models. Each of these sets were disambiguated by hand to create an object hierarchy. We then train a decision tree regressor by choosing one action and object data-set, training the regressor on all candidates, and using that regressor to determine confidences for all other combinations of each other data-set. This is done for each combination of actions and objects. To get different values of precision and recall, we change the value of α. We test each action dataset against each object dataset. The results can be found in Figure 4.11.

As can be seen from Figure 4.11, the confidence-based method produces a similar precision to that of the multi-sieve method for low recall levels. At each of the bounds (precision = 1.0 or recall = 1.0) the multi-sieve method outperforms the confidence based method. However, in Figure 4.11 where there is more area where the precision is closer to the recall for the confidence-based method,

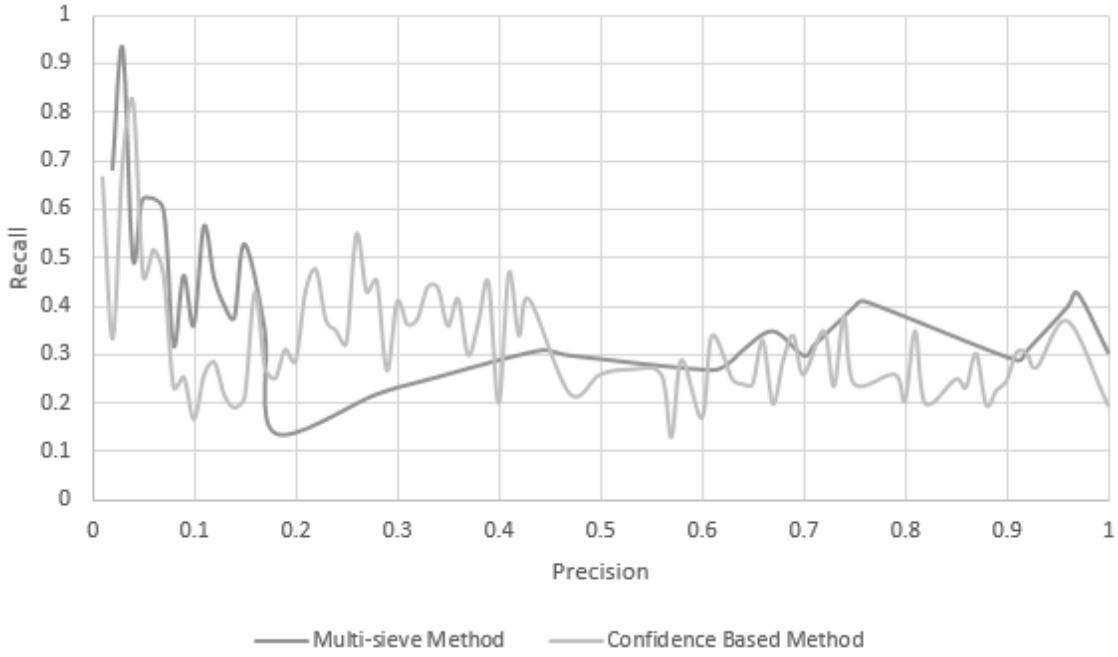


Figure 4.11: The precision-recall graph obtained in testing the confidence-based method and the multi-sieve method on our action data-sets over all three data-sets.

such as when recall = 0.4. This means that the confidence based method produces more balanced results between precision and recall than the multi-sieve method, thereby confirming **H 4.4**.

In addition to our multi-sieve method and confidence-based method, we show the effect of our word-vector model on the data-set. We generate our word2Vec models (using gensim [113]) from two separate large text corpora, a plain text parsing of the entirety of Wikipedia [8] and the The 1 Billion Word Language Model Benchmark[9]. We generate Word2Vec models that have window sizes of 3 and 5 words, and a word was not trained in the model unless it had a count frequency of greater than 50 occurrences. For clustering actions, we used scikits-learn [114] DBSCAN, with epsilon sizes of 2 and 3.5 and a minimum leaf size of 2. All word vectors for a given set are first processed using Principle Component Analysis, keeping the two most descriptive terms. We examined how many actions were identified, as well as the total size of the tree and how many actions were able to connect to FrameNet in Table 4.2.

[8] en.wikipedia.org

[9] http://www.statmt.org/lm-benchmark/

Table 4.2: Statistics for single parent word vectors on each action set used in our tests. The number of actions are the base number of actions and the total items in the hierarchy include generalizations. The % for actions with frames is calculated from the base number of actions.

Name	**# of Actions**	**% of Found Actions**	**Total Actions and General-izations**	**% of Actions with Frames**
Smartbody	19	65.8±3%	22.5±0.5	68.4±0%
CMU	58	69.8±1%	70.5±2.5	40.5±1%
The American Time Use Survey	43	25.6±0%	46.75±2.25	19.2±7%

From Table 4.2, it should be seen that the word vectors were able to identify more actions for the two lower level action sets (Smartbody and CMU) than for BLS actions. This should be apparent, as lower level actions are more likely to have a single unique word identifier, whereas the higher level actions in BLS would not. This also accounts for the lower number of FrameNet frames connected to each action. It should also be noted that the total number of parent actions is lower for this method than for any method that uses a WordNet hypernym structure. Because our word-vector model clusters several actions together, the branching factor for the tree should be greater than the long chains that are seen in the hypernym models. The ability of the system to recognize actions and cluster them into a hierarchy proves **H 4.5**.

H 4.6 and **H 4.7** focus on how automated FIDAG generation affects the action generation as a whole. Using Multi-sieve WordNet generation, we provided two methods to prune candidate senses for an action. Before being able to describe how these generation methods effect the hierarchy, we examine the two methods and their ability to match to human participants. This, like in **H 4.1**, gives a baseline to the extent of using FIDAG generation methods. To do so, we determine the number of matches between our each method and human generation, showing the percent that one sense using automated generation methods is matched to one of the human selected senses. The results can be seen in Figure 4.12. We also show, on average, how many parents are generated for each action data-set. A higher average means that more candidates are selected for a given action. The results are shown in Table 4.3. For this analysis, we do not consider the action sets that do not receive a sense.

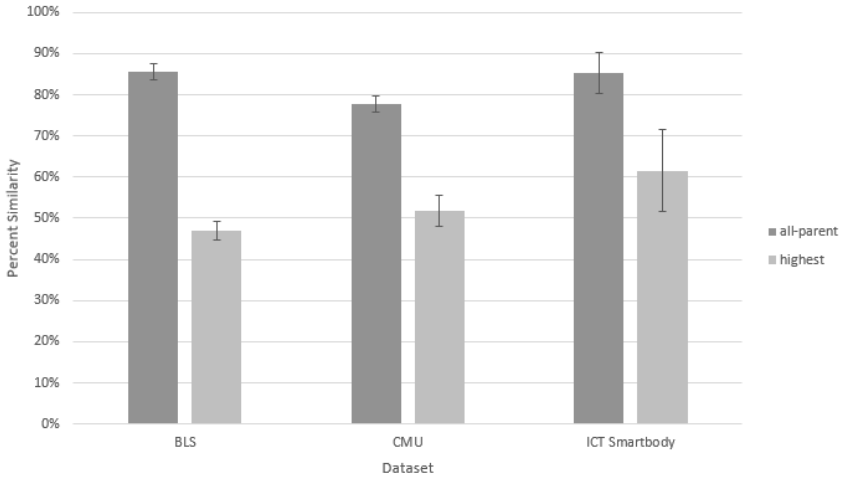


Figure 4.12: The percent matched vs. dataset when all keywords matched over the threshold vs. using the highest matched over the threshold.

From Figure 4.12, using a threshold to determine good candidates is able to more often match to a sense generated by human coders. This is due to the fact that using the all-parents method provides more parents than if we look for only the highest action, as shown by Table 4.3. The lower total number of actions shows that, in general, one synset is found to be more favorable than others by the system. In many respects, using a highest parent model produces similar results to those found in the multi-sense method of Figure 4.9b to the hand selected keywords of Figure 4.10, showing that using a highest parent cutoff value is similar to using a single parent hierarchy method. However, **H 4.6** only discusses the ability of either method to produce FIDAGs, which both methods are able to do. Therefore, either method proves the first part of **H 4.6**, in that either method can produce FIDAGs. The ability of the all-parent method to identify possible connections that are similar to human taggers also disproves the first portion of **H 4.7**.

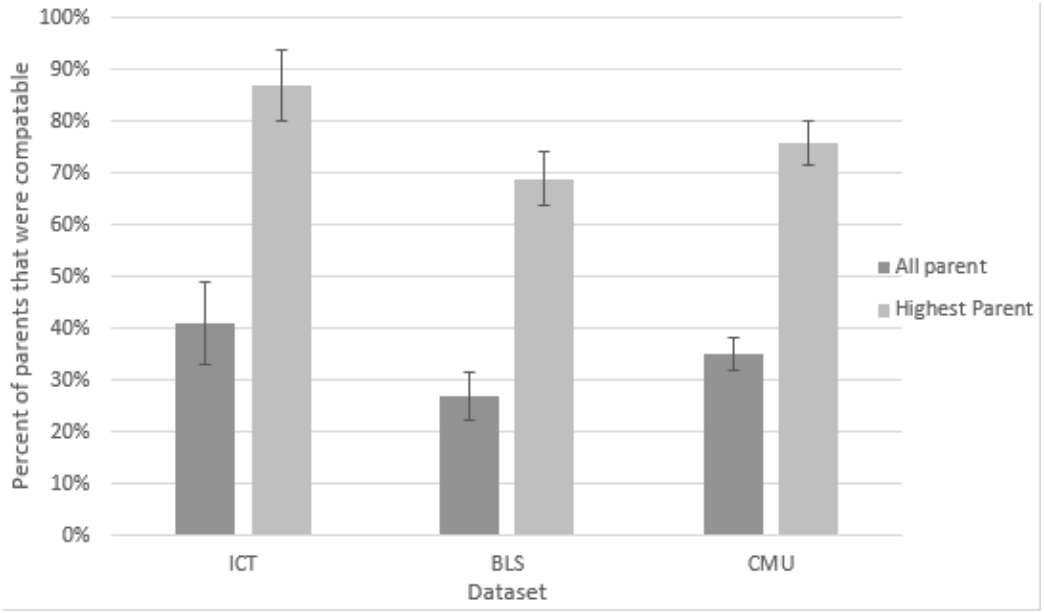
To completely examine **H 4.6**, we must also look at the compatibility of each parent. To do so, we examine the number of roles and semantics that connect to each action, and the amount of

Table 4.3: The average number of parents found when generating FIDAGS, using either a single threshold or using the highest action and a threshold.

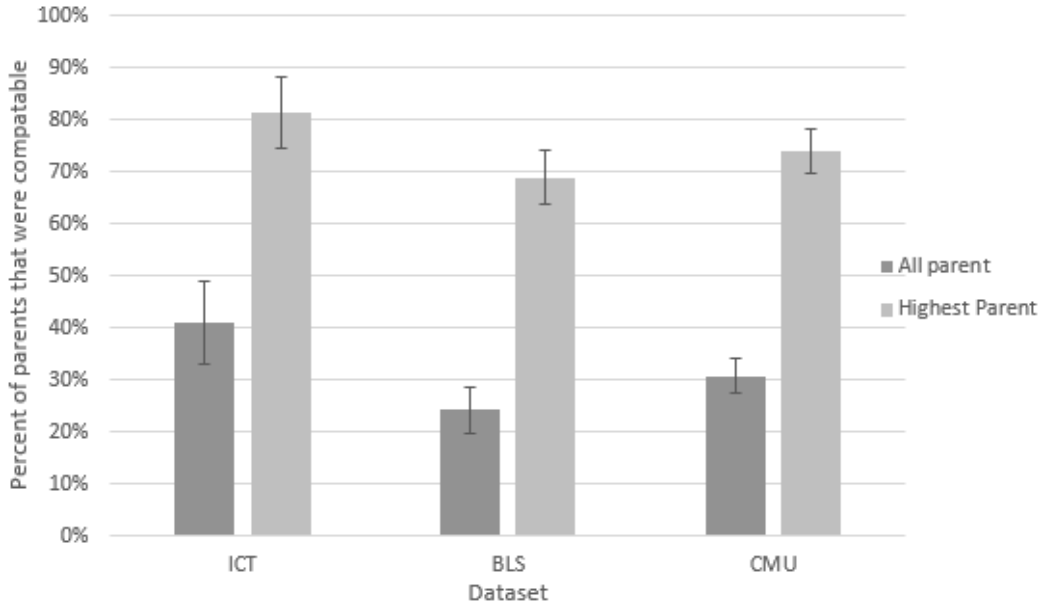
Dataset	Average Number of parents for all-parent method	Average Number of parents for highest-parent method
ICT Smartbody	5.3	3.4
CMU	6.1	3.8
BLS	7.5	4.8
Average over all three data-sets	6.3	4

overlap between each set. For both an above-all threshold and highest parent threshold, the disambiguated senses become the parents of the action. For automated generation, this means that the properties of actions are determined through the parents. We therefore determine how many properties of these parents match for each method. Parent properties are attached to the disambiguated senses, and so for this experiment we do not build a complete tree, but only examine parents one level deep. We compare complete properties for which all semantics and roles match for all parents, with the results in Figure 4.13 This uses the binary components equation (Equation 3.3). We also determine what percentage of overlap between parents exists for an individual action, with the results in Figure 4.14. Semantics and roles are automatically generated using ALET, which is described in Chapter 5. It is also because of ALET that these two measures provide an understanding of how varied the parents generated for actions for the two methods are.

From Figure 4.13, it should be seen that using an above-all threshold provides significantly less fully compatible actions then using a highest-sense threshold, for both semantic generation and role generation. Part of this is simply due to there being fewer parents per action (as seen in Table 4.3). It also means that, when multiple parents are determined by automated methods, having a large number of parents is not necessarily better, because the parents might not be compatible with one another and a simulation author would have to go in and rectify this information. Therefore, using an above-threshold method would increase the work a simulation author has to perform, which proves the second part of **H 4.6** for the highest-parent threshold method, but shows that there are many incompatibilities for the above-all threshold method, and shows that **H 4.6** does not hold for



(a)

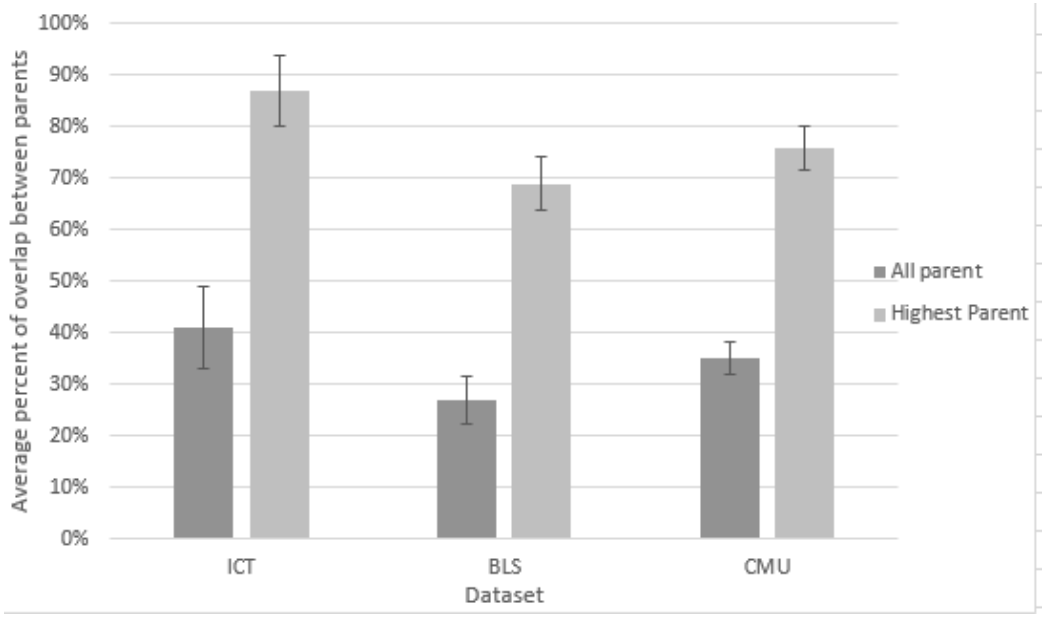


(b)

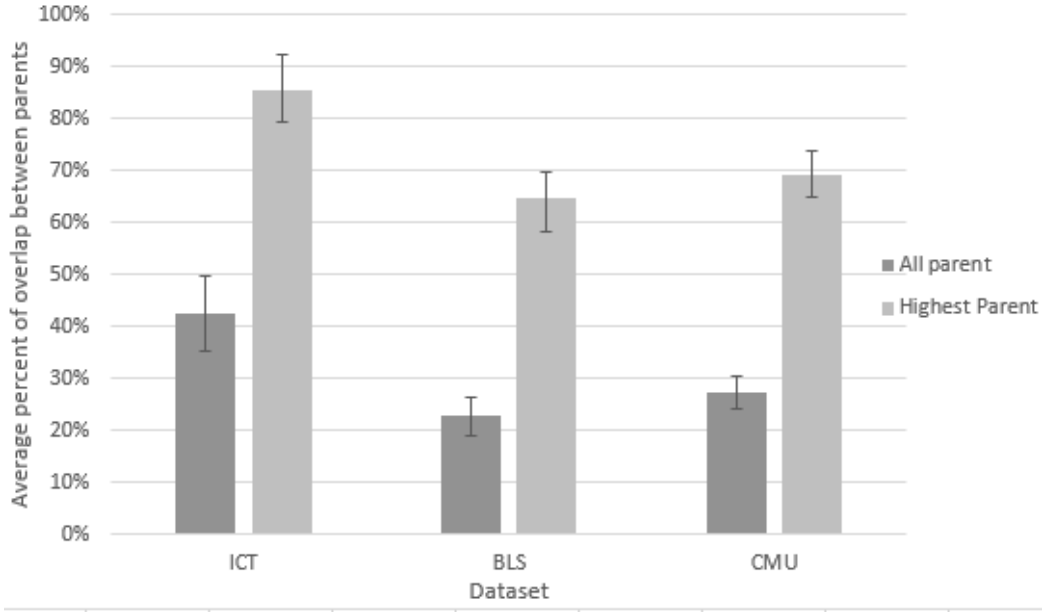
Figure 4.13: The average percentage of parents with completely compatible (a) semantics and (b) roles for both an above-all threshold and a highest threshold.

that method.

One interpretation of the child action is that it would be a composite of all the parent actions (i.e



(a)



(b)

Figure 4.14: The average percentage of (a) semantics and (b) roles that are the same across all parents.

for $\mathbf{R} \in \mathbb{ACT}$). In that case, the amount of overlap between actions parent actions shows the agreement between parent actions. Figure 4.14 shows the average percent overlap between actions, using

automated semantic tagging tools. This gives us an idea about how much redundant information is available. As can be seen from Figure 4.14, using a highest parent method produces more redundant information than using the all-parents method, and this occurs for both semantic properties in Figure 4.14a and roles in Figure 4.14b. When examining Figure 4.14 with Figure 4.13, it should be noted that there are many cases in which the total compatibility between properties on the parent is much lower when the threshold for what actions are allowed to be a parent is much lower. As the highest-parent method provides that extra constraint on choosing a sense, that technique can be useful to prune out knowledge such that there is more confidence in the ability of the system. This proves the latter half of **H 4.7**, but as the first portion of **H 4.7** was dis-proven, means that it is difficult to say if one method for FIDAG creation is more useful than another. In reality, it depends on the needs of the simulation author. If a simulation author wants more information about the action to sift through, then using an above-all threshold will provide the author with those tools. However, if the simulation author wishes for the system to only find parents that are the most obvious parents, then the highest-parent method will not provide as many parents, creating a smaller and more similar tree.

For our final experiment, we examine FIDAG creation using word-vectors. We use the same set-up as we did for experiment **H 4.5**, but now attempt to split any action name into multiple parents. We test for the same parameters and using the same word-vector models as was previously discussed. The results can be seen in Table 4.4.

Table 4.4: Statistics on each action set used in our tests. The number of actions are the base number of actions and the total items in the hierarchy include generalizations. The % for actions with frames is calculated from the base number of actions.

Name	**# of Actions**	**% of Found Actions**	**Total Actions and Generalizations**	**% of Actions with Frames**
Smartbody	19	97.7±2.7%	43.25±2.5	100%
CMU	58	97.7±0.7%	169.25±2.5	82.9±1.4%
The American Time Use Survey	43	100%	169.25±2.5	96±6%

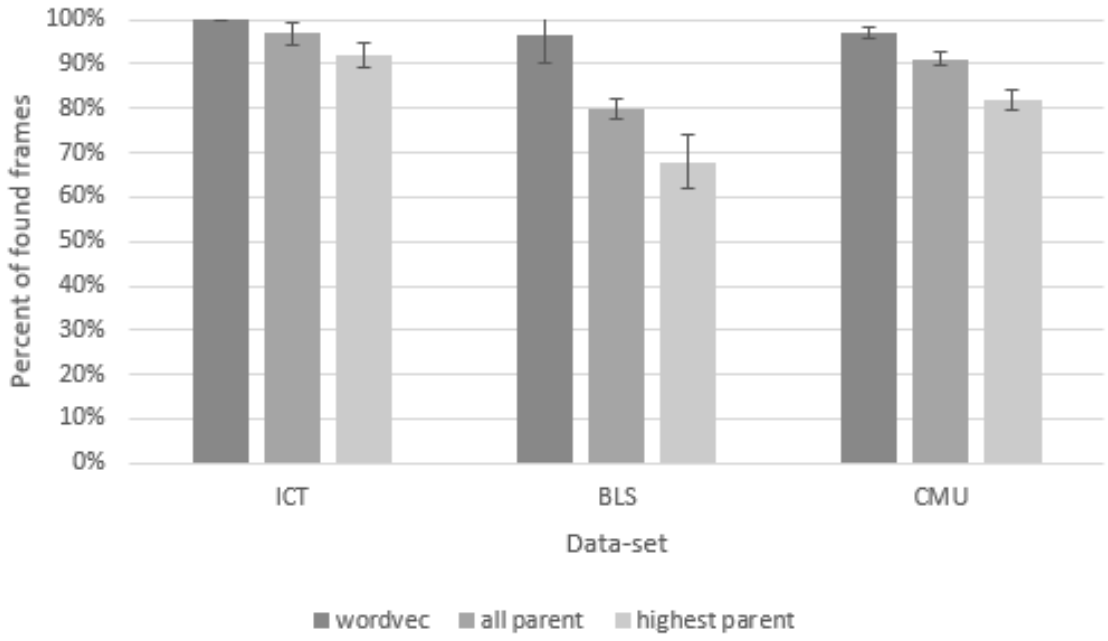


Figure 4.15: The percent of Frames found for each multiple parent tree building method. A single factor ANOVA shows no difference between the three methods, with $p = 0.1$

It should be immediately apparent from Table 4.4 that assuming multiple parents in a word-vector model has a significant impact in the ability of the system to identify the action, as well as connect it to FrameNet frames. This is most noticeable for the BLS dataset, which was able to determine a parent for all actions using multiple parents. Because our word-vector method breaks up the word when there are multiple parents, it shows that this particular action set are not singular, atomic actions, but are more complex, higher level actions. The increase in total frames found when compared to Table 4.2 for all action sets also show that having more possible actions and more data from the parent really assists this method's ability to connect to FrameNet frames.

While the results from Table 4.4 show that multiple parents for a word vector are beneficial, **H 4.8** is concerned with the number of Frame connections that can be made. Therefore, we average the number of FrameNet frame connections and compare to the average number for a WordNet hypernym generated tree using the highest-parent method. The results can be seen in Figure 4.15.

Surprisingly, Figure 4.15 shows that a word-vector method is able to identify frames for the action as often or more often than using either hypernym method. This is most likely due to the

word-vector tree building method searching for frames using the lemma, which equates to the names of the action and its parents using. Since the word vector method, when action names are examined on their parts is more likely to find parents and cluster actions together, it is more likely that those parents will have a frame. Our Multi-sieve WordNet methods connect frames using [93], which is a more precise method of connecting frames. While the hypernym tree method is more precise in its connection of FrameNet frames, the ability of a "data-dumb" method to perform so well is quite fascinating. It also disproves **H 4.8** in that, for simply finding frame connections, our word-vector tree building model is as capable as our hypernym model.

4.6 Conclusions

We presented methods to automate the population of parents given the name of an action. The parents of an action can be generated from either a well-understood data-set or well defined keywords that accurately represent the action in question. We also showed how different methods and different data-sets effect the ability of the system, and how newer techniques such as word-vectors can be used to populate parents. By generating action parents, we have laid the foundation work for Chapter 5 to populate both the roles and semantics of actions, providing a more complete and concrete connect for agents interacting in rich environments.

The ability of word-vectors and clustering to create action parents is important because of the work connecting graphical models to names (i.e shape2Vec [103]). While this work is concerned with graphical models, it is not unreasonable to hypothesis that animations could also be connected to word-vectors. In this case, simply connecting the animations to names is not enough to be able to reason about it (as shown by the inability of the single parent word-vectors models to find FrameNet Frames). However, by using multiple parents and building a hierarchy (using the word-vector method) it is conceivable that there is overlap between what is presented here and future endeavors with word-vectors. What we provide here is proof that word-vectors are a viable option to build an agent's understanding of the generalizations to their actions. As we will show in Chapter 5, having parent actions are crucial to populating semantic information.

Chapter 5: Automated Generation of Semantics

In addition to populating action parent sets in Chapter 4, a simulation author would still need to fill in all other components of an action set. Even if the simulation author has a completed set, new properties or requirements of the characters may appear, and components would need to be added to or changed. This is especially true if new objects were added (as $\mathbf{R} \in \mathbb{ACT}$ would need to be updated) or if $\mathbb{S}$ were modified. In these instances, a simulation author would be required to recreate or modify all actions to include this change, which creates a burden on the simulation author. In commercial cases [115] where this recreation is not feasible, actions become inconsistent, and can lead to undesired results and incorrect simulations. Most previous work in semantic information for virtual scenarios (or actions) focused on how ontologies for virtual agent environments should be structured. While we have described how the object and action sets fit together, the contribution of this chapter is in generating those semantics from text in a system we call ALET[1] (Agents Learning their Environment through Text). ALET is similar to both Fast et al.'s Augur system [116] for activity recognition and our work in [10] (with a component seen in Chapter 4) for operational population. However, our system provides several marked improvements over both systems for semantic information generation. ALET uses a combination of our action sets and text to infer attributes of the action and attributes of objects, which neither previous work is concerned with. This allows us to populate more operational information than we were previously able to in [10] and described in Section 5.1, as this works was only concerned with the object's name and semantic type. Furthermore, as our domain has a specified finite set of actions and objects based on the animation and model sets, the text parsing is only concerned with that set, removing any actions and objects that do not fall within that set from consideration.

As another strategy employed to ease the burden of creating operational information connection, some researchers have tried to use human computing, also known as crowd sourcing. Human

[1] Published in [11]

computing [117] relies on simplifying a problem into smaller pieces and having humans perform the simpler computation. Human computing has been used to create complex behavior [118] and operational information [119]. While the technique is an interesting method to ease the authorial burden, in reality it only works well for tasks that the "average" person can perform. For highly scientific action sets, the human created actions may not work well. This is because the "average" person may not have the required domain knowledge, which means that not as many people will perform the computation. Furthermore, humans can be inconsistent in their labeling, making it difficult to get more reliable, replicatable results. Therefore, human computing is limited in its ability to create action sets.

At a high level, ALET relies on the idea that a virtual character has a known set of animations and graphical objects and that there is some underlying meaning in the connections between actions and objects. We assume that each action and object are have parental information. This can be accomplished by having a simulation author choose the correct sense given a graphical model or animation clip, or through automated processes such as Pelkey and Allbeck [37] for graphical models or one of the methods seen in Chapter 4. We further assume no other information in our ontology, that is, that there is no interactional information or object semantics connected. We also assume our ontology contains all the models a virtual human would encounter and all motion data that the character can use. This means that we would simply re-perform our method if new actions or objects became available to the simulation. We show a graphical representation in Figure 5.1.

In addition to resources that contain generalizations of actions such as WordNet, there are several resources that attempt to connect actions and objects. Two well known resources are FrameNet [92] and PropBank [120]. Both of these are large, semantic databases that describe roles attached to verbs. In some instances, these roles are similar to $\mathbf{R} \in \mathbb{ACT}$, but may also include Non-Interactional Functional Information that describes the actions (and is more similar to $\mathbf{S} \in \mathbb{ACT}$). Both of these resources are used as templates in Semantic Role Labeling [121], which diagrams a sentence based on the components of the sentence and the necessary components that should be attached to the verb. Semantic role labeling has started to be used for virtual agent animation [104], specifically for text command and simulation. Ludwig et al [104] uses prop-bank and a bayesian

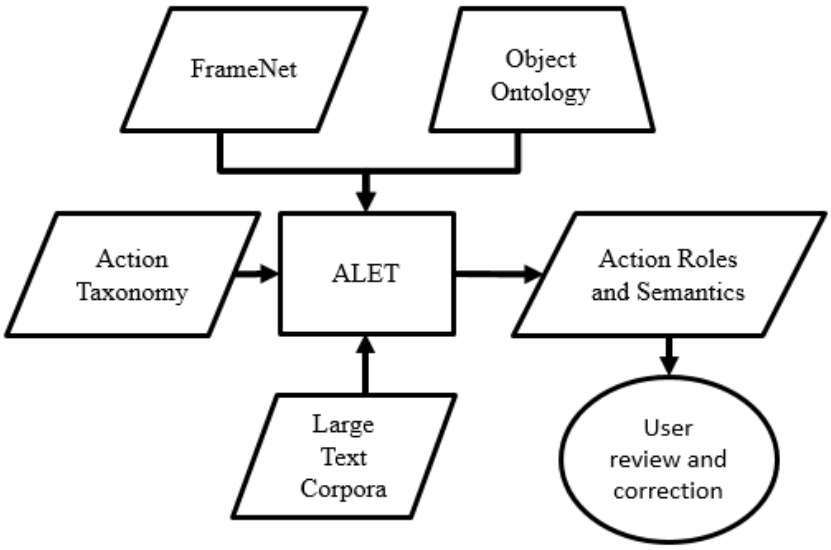


Figure 5.1: An overview of the basic techniques for populating action roles and semantics

system to identify the objects used with actions in a text. However, it makes no mention of if these are the correct objects to use, just that they are identified as the role. Without ontological support, an agent designer has no knowledge or control over what objects actually fit in that role, which is partially do to the generality of prop-bank. On the other hand, FrameNet has been shown to provide excellent coverage in identifying roles, especially over other ontologies such as dbpedia [122]. We therefore demonstrate how to populate semantics of virtual environments ($\mathbf{S} \in \mathbb{OBJ}$,$\mathbf{S} \in \mathbb{ACT}$, and $\mathbf{R} \in \mathbb{ACT}$) using FrameNet as a base for these connections. These are methods to add meaning to virtual agent actions, continuing the trend from Chapter 4 in populating an understanding of virtual agent actions. The core of this chapter is therefore to examine **H 5**, shown below. The utility of **H 5** arises in that resources to help populate actions ease the simulation authors burden and allows virtual characters to have more abilities.

H 5: Meaning can be generated for actions using the knowledge of objects, knowledge bases, and large textual data-sets.

5.1 Connecting Objects to Actions using FrameNet

Object operational information requires a connection between a given action and the objects that can be used in it. For several basic actions, this simply forms a triplet *(agent, action, object)*, where *agent* is the agent initiating the *action* on an *object*. However, actions can become very complex, requiring not only knowledge of *what* objects can participate, but *how* they can as well. For example, an agent mopping a floor requires not only a space in which to mop, but an instrument with which to perform the action. This creates a two-fold process, in which the correct sense of a frame must be disambiguated. Disambiguating the correct ordering of such an action combination is essential to streamlining an agent's decision making process [21].

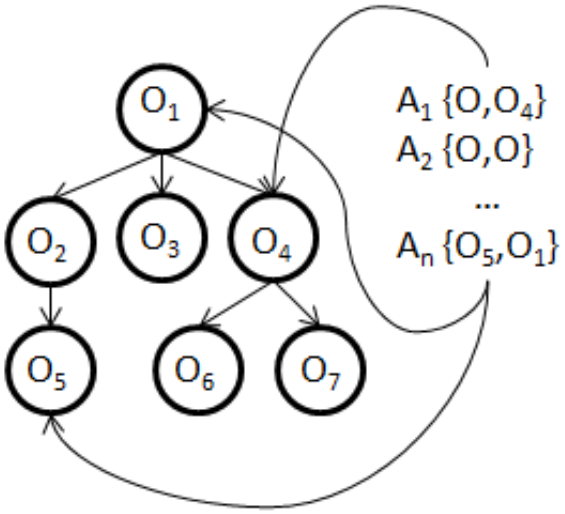


Figure 5.2: A pictorial overview of the connection step. The object parameters of actions are linked to object types in the object ontology.

One component of the FrameNet database are *Functional Elements* (*FE*s), which are functional (operational and semantic) information related to a verb. An example for the verb *cook* can be seen in Table 5.1. It should be noted from Table 5.1 that not every FE is related to graphical objects.

In fact, only *Cook*, *Heating_instrument* and *Produced_food* are graphical in nature and the rest are NIFI. NIFI such as *Manner* in Table 5.1 can be used to control how an action is performed. Therefore, in addition to connecting roles, at this stage we also extract NIFI ($\mathbf{S} \in \mathbb{ACT}$). This does not mean ALET plans motions based on NIFI, just that it finds NIFI that can be used in motion planning. Motion planning based on NIFI is left to future work. There are also FEs that, while useful for the action, do not match any object that exists in the simulation. We cannot only use FE as $\mathbf{R} \in \mathbb{ACT}$, but need to match them with object types in the system so the agent knows what objects are matched with a given role.

Table 5.1: Functional Elements for the Frame *Cook*, specifically *Cooking Creation*. The first three are operational information $\mathbf{R} \in \mathbb{ACT}$, while the last two are NIFI $\mathbf{S} \in \mathbb{ACT}$.

Functional Element	**Definition**
Cook	The Cook prepares the Produced_food.
Produced_food	The Produced_food is the result of a Cook's efforts.
Heating_instrument	This FE identifies the entity that directly supplies heat to the Food.
Manner	This FE identifies the Manner in which the cooking creation is performed.
Purpose	This FE identifies the Purpose for which the cooking creation is performed.

5.2 Agents Learning their Environment through Text (ALET)

Creating rich semantic environments requires connecting all possible actions to every object that could be used in that action, in what is known as operational information [9] or roles [1]. Particularly for virtual agents, operational information must be available prior to simulation so the information is retrievable in real-time during simulation. Operational information can be programmed in by hand, but becomes infeasable as the number of actions and objects grow. Methods that generate affordance[2] information from 3-D models [28] are unable to connect virtual human actions that rely

[2]Affordance information here is from Gibson's original definition: that the actions that can be performed on objects are perceivable by the shape of the object

on non-graphical properties of objects. For example, an *eat* action would not have its operational information resolved based on graphical properties of 3-D models. Other methods use knowledge bases for natural language processing and ontologies to resolve operational information [10], but these methods only create connections based on the name and generalization of the objects. These methods are not able to resolve connections described using adjectives. For example, a knowledge base using *ingestible* (an adjective) as the item to *eat* and not *food* (a noun) would be unable to connect edible items to eating. To connect operational information based on properties, the objects must contain those properties. For non-graphical properties, this would be done ad-hoc by a simulation author, leading to inconsistent or omitted connections. Therefore, textual knowledge bases offer candidates for operational information, though they still may require expert knowledge to know which connections are appropriate for a given simulation. Our method uses large textual corpora to determine candidate semantics for use in virtual environments. We delineate two matrices that contain our semantics, a noun-description matrix for *semantic information*, and a verb-description information to describe actions. Two of the matrices are generative, in that only the rows are known before examining the text.

5.2.1 Dependency Grammar Parsing

The construction of co-occurrence data-tables begins with ALET parsing large textual corpora to find verbs and nouns matching the labels of actions and objects ($\lambda \in \mathbb{ACT}$ and $\lambda \in \mathbb{OBJ}$) in the system. We note that this problem is a highly functional one, in that the co-occurrences represent relationships between words in the text. Spacing co-occurrence, in which words co-occur based on their spacing, suffers from ambiguity when phrases contain descriptors that could describe either actions or objects. Therefore, we use a Dependency Parser [123], to read through the text, and parse out *Universal Functional Dependencies* [124] creating 3-tuples that give meaningful relationships between the text. The output of parsing is a list of 3-tuples denoted t. Each $t_i \in t$ contains a designation d and two words (denoted $w_j^{t_i}$ where $j = (0,1)$). At this stage, we can prune t to only include d that have meaningful matches to the co-occurrences we are searching for. Specifically, we search for $d = \{amod, nmod\}$ dependencies when construction our noun-descriptor data-table

and d = $\{advmod\}$ when constructing our generative verb-description table. Pruning the tuples at this stage allows for only important tuples to be compared, allowing the system to have a better understanding of the data for later processing. A graphical depiction of this process can be seen in Figure 5.3.

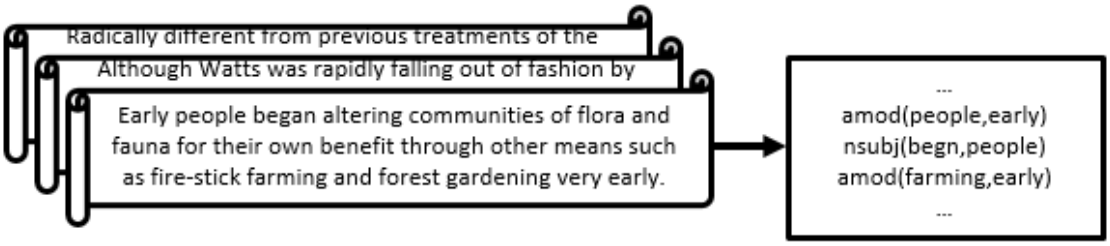


Figure 5.3: A graphical representation of dependency parsing on text.

5.2.2 Co-occurance Connections

While ALET is determining desired dependency tuples, it also disambiguates a more meaningful representation of each w^{t_i} themselves. While base words may appear as action and object labels in our representation, the actual meaning of the word can be quite different. This problem is known as *Word Sense Disambiguation*. Recall our action and object representation has a generalization field $\mathbf{P} \in \mathbb{ACT}$ and $p \in \mathbb{OBJ}$. When using automated generation tools such as [37, 10], these generalizations can represent WordNet hierarchies and preserve more detailed information on the action and object. To generate descriptors that more closely match the meaning, ALET disambiguates each w^{t_i} into an associated synset.

To better understand matches in our system, we use Lesk's Algorithm [125] to determine the WordNet synsets for each word in a tuple. Lesk's algorithm is a simple and well-studied algorithm for word sense disambiguation. The algorithm takes the context of all the surrounding words in the sentence that the object came from, similar to the definition method presented in Chapter 4. Specifically, for each word w^{t_i} in each tuple t_i, we calculate the probability that each sense *w.candidate*

contains the correct meaning given the sentence s that w^{t_i} originates from using the WordNet definition, $w.candidate.def$ attached to each sense and Equation 5.1. As we know the context of $w_j^{t_i}$ from parsing the tuples, we only need to consider each possible meaning of that type (such as only considering *verbs* if we know *cook* is an action). Equation 5.1 is run for each possible meaning of w^{t_i}, and the sense with the highest probability is considered the correct sense. We then only use the tuples who match the row term in our ontology.

$$w = argmax\{c.def \in w.sense : \frac{|s \cap c.def|}{|s| * |c.def|}\} \tag{5.1}$$

It should be noticed that a large textual corpora should contain a given disambiguated tuple multiple times if the connection is truly salient. Furthermore, many disambiguated words will not contain between each other as well. The end result of ALET are two data-tables created from COO sparse matrix[3], where each row is a term in our ontology and each column is a term found connected to an object in our text. We generate two data-tables that correspond to different semantic information, with an example of our noun-descriptor table in Table 5.2. A sample of our verb descriptors can be seen in Table 5.3. Our two matrices describe semantics attached to each hierarchy. The object-adjective matrix *ND* contains nouns *N* and generated descriptions *D* and the action-adverb *VD* has verbs *V* and the descriptions *D* that were found in the corpora.

Table 5.2: A sample of the matrix *ND* for the object *Food.n.01*, along with its counts.

Noun	**Descriptor**	**Count**
Food.n.01	fresh.a.01	587
Food.n.01	natural.a.01	250
Food.n.01	fried.s.01	106
Food.n.01	american.a.01	124

[3]Also known as an *ijv* or *triplet* matrix

Table 5.3: A sample of the matrix *VD* for the action *Cook.v.01*, along with its counts.

Verb	Descriptor	Count
Cook.v.01	frequently.r.01	64
Cook.v.01	immediately.r.01	16

5.2.3 Operational Information Population

The results of Section 5.2.2 are two data-tables, with an example of object descriptors seen in Table 5.2. Each **object** and **descriptor** represents a link found in the text, with **count** being the number of times that link was found in the text. However, while these connections are meant to assist generating roles ($\mathbf{R} \in \mathbb{ACT}$) and action semantics ($\mathbf{S} \in \mathbb{ACT}$), the connections only provides descriptions of actions and objects. To specify the roles of each object in an action, we combine the results of our co-occurrence algorithm with FrameNet [92] in order to embed a deeper understanding of each virtual object's role

We first leverage the matrix *VD* to determine the set of action descriptions $\mathbf{S} \in \mathbb{ACT}$ for a given action set. As the set of operational and NIFI information is disjoint, we build the action descriptions first so that each $\mathbf{R} \in \mathbb{ACT}$ is only generated from FEs that we know are not descriptors. To generate $\mathbf{S} \in \mathbb{ACT}$, we extract a unique set of all descriptors from *VD* as a set of keywords using Equation 5.2. The keywords are compared to the FEs connected to an action, with matches for a given action creating $\mathbf{S} \in \mathbb{ACT}$. Note that we use FE as our action descriptors, and not *VD*. The descriptions in *VD* are numerous for several of the actions, whereas FE tends to be a smaller set across several actions. Creating a proper subset of descriptions for each $\mathbf{S} \in \mathbb{ACT}$ is outside of the scope of this work and left to future work.

$$\mathbf{S} = \bigcup_{i=act \in \mathbb{ACT}} VD_{V=\lambda \in i} \tag{5.2}$$

Keyword generation for operational information $\mathbf{R} \in \mathbb{ACT}$ comes from our object representation and noun-description data-table *ND*. We first consider the set of keywords $K = \{\lambda, obj \in \mathbb{OBJ}\}$. In other words, our set of keywords is the set of the names and generalizations of objects available to the agents in our scenario.

There are also FEs that do not represent an object but rather a property that can be applied to several objects. In order to capture this functional information, we use the matrix *ND* created in Section 5.2.2. Recall that this data-table consists of nouns that possibly objects from our virtual environment, as well as candidate semantics described through adjectives in the text, with counts representing the number of co-occurances in the corpora. Therefore, we attach the descriptions found in data-table *ND* as keywords in our search, using only the descriptions that match objects in the hierarchy, using Equation 5.3. This is done with the set of object label keywords. However, the matches on FE from this require an extra step to link the descriptors that were matches back to the objects they are matched with.

$$K_d = \bigcup_{i=obj \in \mathbb{OBJ}} ND_{N=\lambda \in i} \tag{5.3}$$

We also connect the object descriptors on the objects as the set $\mathbf{S} \in \mathbb{OBJ}$, which become important object properties in our virtual environment. FEs that were added to $\mathbf{R} \in \mathbb{ACT}$ using K_d link to a descriptor in *ND*. Therefore, for each linked descriptor FE_i, we determine the set of objects that contain that descriptor using Equation 5.4. We then add FE_i to $\mathbf{S} \in \mathbb{OBJ}$ for each object. This is done in addition to adding the object to $\mathbf{R} \in \mathbb{ACT}$ because the descriptions in *ND* are treated as possible properties of an object, which may or may not exist on different objects of the same type in a virtual environment. Creating a two stage connection provides the simulation author with more control, in that the system provides the agent with the objects to look for, as well as the properties to check on, the objects after they are found. So, an agent may know that an *apple* is *ingestable* for the action *Eat*, and will also know to check the property on an *apple* to make sure it is *ingestable*.

$$D_o = ND_{D=FE_i} \tag{5.4}$$

It is important to understand the difference between ALET and Pelkey and Allbeck [37]. Pelkey and Allbeck determined $\mathbf{S} \in \mathbb{OBJ}$ by examining part of speech tags on WordNet definitions that matched $\lambda \in \mathbb{OBJ}$ for all objects in the hierarchy. The insight in Pelkey and Allbeck was that certain parts of speech tags (such as *NOUN* and *ADJECTIVE*) are representative of descriptors,

and if they are part of the definition, must be descriptors used to describe the object. ALET, by comparison, uses textual relations that are known to describe objects and actions, and connects those textual descriptions back to the actions in the data-set. This provides two marked improvements: the descriptions found through relations are known to describe the data-set, and there is much more data in the system to smooth out the noise inherent in natural language.

The use of more data was also seen in Augur [116], which ALET is another inspiration of ALET. Augur uses verb-noun co-occurrences on a large data-source (modern fiction) in order to learn roles. These roles are different from $\mathbf{R} \in \mathbb{ACT}$ in that they are unordered, which for their application (action recognition in video) is useful. However, for agents to know how the roles operate, especially when objects may fill multiple roles in an action, $\mathbf{R} \in \mathbb{ACT}$ is assumed to be ordered (or at least have some semantic meaning for the different roles). We do not consider co-occurrences between descriptors for the same reason we used dependency parsing over part of speech tagging in Pelkey and Allbeck.

5.3 Experimentation

To show the efficacy of using FrameNet and text corpora to populate action semantics, we postulate several hypotheses, namely:

- **H 5.1**:The resolution provided by FrameNet and a virtual object ontology covers the usable objects in a scenario.
- **H 5.2**:The use of generalizations provide better coverage then when generalizations are absent
- **H 5.3**:$\mathbf{S} \in \mathbb{OBJ}$ will provide a higher recall of identifying $\mathbf{R} \in \mathbb{ACT}$, but lowers the overall accuracy
- **H 5.4**:Verb descriptors can populate $\mathbf{S} \in \mathbb{ACT}$ when used in conjunction with FrameNet
- **H 5.5**:The use of verb descriptors and noun descriptors produces a higher overall accuracy on the system over only noun descriptors
- **H 5.6**:The choice of text data has an impact on ALET's behavior and abilities.

H 5.1 and **H 5.2** show the ability of FrameNet to act as a starting point for determining $\mathbf{R} \in \mathbb{ACT}$, but requires more, unstructured data(shown in **H 5.3**, **H 5.4**, and **H 5.5**) to fully compute a partial semantic representation of the actions. Using unstructured data requires other considerations, proven in **H 5.6**. The combination of these hypotheses build an understanding of the requirements in populating actions, and their combined realization show that action roles and semantics can be populated using large textual corpora and knowledge bases, proving **H 5**.

5.3.1 Datasets used in Experiment

In order to test our method, we populate several sources of objects and actions. Objects are generated from ModelNet [112], the Caltech 101 object image data-set [111], and two virtual environments that can be downloaded from the Unreal Game Engine Marketplace, seen in Figure 5.4. The Unreal Game Engine scenes represent different time periods and different authors. Each object data-set is disambiguated by hand and connected to WordNet sysnets. The generalizations are also created in our representation, allowing us to fill in $p \in \mathbb{OBJ}$. Information on each set can be seen in Table 5.4. The most important statistic in this table is the percent of objects that were matched with a WordNet synset (% of found objects). As ALET determines WordNet synsets during its disambiguation phase, objects that do not have a synset are not included in $\mathbb{OBJ}$.

Table 5.4: Statistics on each object set used in our tests. The total number of items in the hierarchy includes objects that are generalizations of other objects.

Name	**# of Objects**	**% of Found Objects**	**Total Objects and Generalizations**
ModelNet	656	93%	1063
CalTech 101	101	95%	331
Office (Figure 5.4a)	82	84%	192
Pub (Figure 5.4b)	52	88%	181

Our action data-sets are comprised of a total of five data sources. Motion data makes up three

(a)

(b)

Figure 5.4: Two sample object rich virtual environments from the Unreal Game Engine Marketplace. The two environments contain a) 82 and b) 52 unique types of objects, derived from over a 100 graphical models each. The total number of object instances in each environment is (a) over 400 objects and (b) over 300 objects.

data-sets, and there is one action recognition source and one plain text action source. Both of the non-motion data sources could be turned into animations. The motion data sources that we use are Behavioral Markup Language actions from SmartBody [66], a subset of the CMU motion capture database [108], and *The ICS Action Database* [126]. Our action recognition source is *The Human Motion Database* [127], and our plain text actions are from *The American Time Use Survey* [109]. We also provide statistics for this data-set, as seen in Table 5.5. In addition to connecting action names to synsets, we also connect each action to a FrameNet frame if one exists.

Table 5.5: Statistics on each action set used in our tests. The number of actions are the base number of actions and the total items in the hierarchy include generalizations. The % for actions with frames is calculated from the base number of actions.

Name	**# of Actions**	**% of Found Actions**	**Total Actions and General-izations**	**% of Actions with Frames**
Smartbody	19	100%	49	89%
CMU	60	98%	150	75%
The ICS Action Database	25	96%	51	76%
The Human Motion Database	51	90%	123	75%
The American Time Use Survey	43	83%	96	47%

To determine matches for ALET, we examine the FEs of all found frames connected the test action set, and separate known FEs for each actions into roles $\mathbf{R} \in FE$ and NIFI $\mathbf{S} \in FE$[4], if possible. This is a multi-step process, where the first step examines the generalization designated by FrameNet of each FE, if one exists. Any FE that is generalized to a *Physical Entity*, *Goal*, or *Source* are placed into $\mathbf{R} \in FE$, and any FE that can be generalized into *Attribute*, *Time*, or *State of Affairs* is considered part of $\mathbf{S} \in FE$. However, this only sorts a fraction of the FEs that are inherent to the frames attached to our ontology. Therefore, we have identified a few special cases if a given FE does not have a generalization. These exceptions search for keywords in the definition of an FE (searching for *person* or *entity* for $\mathbf{R} \in FE$). We also use keywords in the name (*path* or *entity* for

[4]The designation of FEs can be found in Appendix C.3

$\mathbf{R} \in FE$ and *Event* or *Duration* for $\mathbf{S} \in FE$). Finally, we compare all remaining FEs to those that are already designated in $\mathbf{R} \in FE$ and $\mathbf{S} \in FE$. Some FEs have a set of *excluded FE*s that cannot be used with them, and if an excluded FE is in $\mathbf{R} \in FE$ or $\mathbf{S} \in FE$ we add it to the same set. So, if a *place* is excluded from an *area*, and an area is known to be in $\mathbf{R} \in FE$, we assume that *place* is also in $\mathbf{R} \in FE$. Any FE that is not identified as $\mathbf{R} \in FE$ or $\mathbf{S} \in FE$ using this process is left as unknown and not used for these experiments. Placing FEs into sets provides some feedback into the ability of each system to generate $\mathbf{R} \in \mathbb{ACT}$ and $\mathbf{S} \in \mathbb{ACT}$ without confusing those sets.

5.3.2 Connecting Objects to FrameNet

In order to test **H 5.1** and **H 5.2**, we examine the full connections and determine the percent coverage for using the object and action data-sets described in the previous section. Our method then computes coverage for each of the found object trees, using our automated method. We also compute a maximum ground truth using the ground truth action hierarchy for each data set. For **H 5.1**, frames are connected to each action or a generalization of that action (i.e, using all $act \in Act$). For comparison, we show the result of not using generalizations when populating $\mathbf{R} \in \mathbb{ACT}$ in two ways. In one, we only use leaf objects (any objects that are not a parent or generalization of another object). We also do the same for actions. The results can be found in Figure 5.5.

When using an action hierarchy to get a fuller definition of an action, it can be seen from Figure 5.5 that the closer to ground truth the starting senses are, the closer the expected connections are to the maximum coverage using our method. As there is a statistically significant difference for each data-set using each method, we cannot confirm **H 5.1** and the true ability of the coverage. We can infer from the operational connections in the ground truth seen in Figure 5.5 that our method can resolve over half of the operational connections between our object and action ontologies. This already greatly reduces the effort needed by a simulation author and encourages further work in this area.

When no action hierarchy is used, the ability to determine expected operational information decreases drastically, as seen by the low percentages in Figure 5.5. This should be expected as the likelihood for a WordNet sense to have a corresponding FrameNet frame using [93] will increase

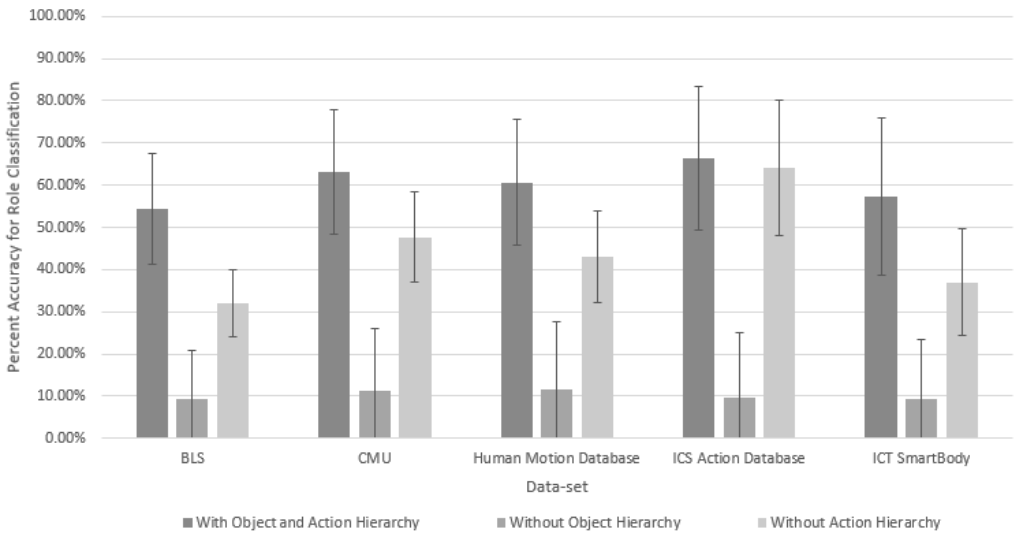


Figure 5.5: The percent of matched object operational information when only using an action hierarchy, only using an object hierarchy, and using an action and object hierarchy. Error bars are shown as one standard deviation.

when a more general sense understanding can be exploited, because many of these connections are stored as generalizations in the system. This is more apparent when object generalizations are not used, and only the names of graphical objects (and their associated synsets) are connected as $\mathbf{R} \in \mathbb{ACT}$ without an associated hierarchy. From Figure 5.5, the standard deviation is larger than the total accuracy, because the roles in $\mathbf{R} \in FE$ are general terms. Therefore, the use of object generalizations is crucial to a complete understanding and connection, with the use of action generalizations still being important. This confirms **H 5.2**.

5.3.3 ALET Datasets

Our system populates semantic information for autonomous virtual agents' use in virtual environments by examining large textual sources and combining the results with FrameNet functional elements. The system uses several data-structures in order to generate and discriminate between operational information $\mathbf{R} \in \mathbb{ACT}$ and (NIFI) $\mathbf{S} \in \mathbb{ACT}$ for actions in our set. FrameNet provides a base for both $\mathbf{R} \in \mathbb{ACT}$ and $\mathbf{S} \in \mathbb{ACT}$, which are matched to the data-tables for actions and objects that ALET generates, and the actual graphical objects that a simulation author has procured for

their virtual environment. We compare ALET to two other generation methods, our work in [10] and Pelkey and Allbeck [37], showing the abilities of each method. Furthermore, to show the effect that the data-source has on the ability of our system, we use two text corpora, *Wikipedia*[5] and *The 1 Billion Word Language Model Benchmark*[6] .

5.3.4 Analysis of Role Connection and ALET

We also test the effect matrix *ND* has on finding operational information in connection with our object ontology. Recall that we use *ND* in conjunction with the object ontology to build a larger set of keywords with which to match to FE. Using *ND* to find noun descriptors is analogous to the work of Pelkey and Allbeck. Therefore, we compare against their method of finding object descriptors, and its overall effect on populating $\mathbf{R} \in \mathbb{ACT}$. At this stage, we are only concerned with finding $\mathbf{R} \in \mathbb{ACT}$, and so are not examining the total comparison with NIFI. Cross tables for our test action and object sets can be found in Table 5.6(ALET with Wikipedia), Table 5.7 (ALET with the Billion Word News Corpus), and Table 5.8 (Pelkey and Allbeck's method). The average accuracy over all object data-sets are shown in Figure 5.6. A two factor ANOVA on Figure 5.6 with replication shows no difference over the action sets ($\rho = 0.1$), but does not show no difference between data-sets and methods ($\rho < 0.001$).

Table 5.6: The average accuracy for detection $\mathbf{R} \in \mathbb{ACT}$ in ALET using Wikipedia as a data-source

	BLS	CMU	Human Motion Database	ICS Action Database	ICT Smart-body
101 Object	44.69%	49.24%	47.26%	44.71%	50.80%
ModelNet	43.44%	52.28%	50.40%	50.75%	59.46%
Office	49.29%	55.56%	53.65%	52.89%	59.45%
Pub	49.29%	55.56%	53.65%	52.89%	59.45%

From Figure 5.6, we can see the effect of using properties to connect actions and objects. It was

[5]https://dumps.wikimedia.org/backup-index.html
[6]http://www.statmt.org/lm-benchmark/

Table 5.7: The average accuracy for detection $\mathbf{R} \in \mathbb{ACT}$ in ALET using the Billion Word News Corpus as a data-source

	BLS	CMU	Human Motion Database	ICS Action Database	ICT Smart-body
101	44.69%	49.24%	47.26%	44.71%	50.80%
ModelNet	48.65%	57.58%	54.95%	55.01%	65.73%
Office	49.75%	55.56%	53.65%	52.89%	59.45%
Pub	71.53%	76.33%	75.53%	78.08%	76.36%

Table 5.8: The average accuracy for detection $\mathbf{R} \in \mathbb{ACT}$ in using Pelkey and Allbeck's Method

	BLS	CMU	Human Motion Database	ICS Action Database	ICT Smart-body
101	37.76%	42.37%	41.28%	39.17%	46.26%
ModelNet	40.70%	46.21%	44.98%	46.87%	52.70%
Office	35.71%	37.79%	37.10%	35.62%	42.61%
Pub	49.68%	58.83%	58.55%	61.60%	65.94%

suggested in our previous work [95] that using properties could assist in finding operational information. We find using adjective descriptors does increase the amount of operational information connected, whether or not that set should belong to $\mathbf{R} \in \mathbb{ACT}$ or $\mathbf{S} \in \mathbb{ACT}$. The fact that the action sets are statistically similar is also useful, in that the action sets span a range of specificity, with BLS being the most high level actions. However, this does not mean that each individual set is the same, as can be well seen in Table 5.7. In the table, the accuracy of model sets has a strong impact on the overall accuracy. This is to be expected. Different sets have varying objects, and those objects have varying descriptors, which may or may not connect well with FrameNet. Not connecting with FrameNet should not be seen as an error, as the objects may just not exist, and so the role should not exist.

The accuracy difference between data-sets (Wikipedia and The Billion Word News Corpus) and

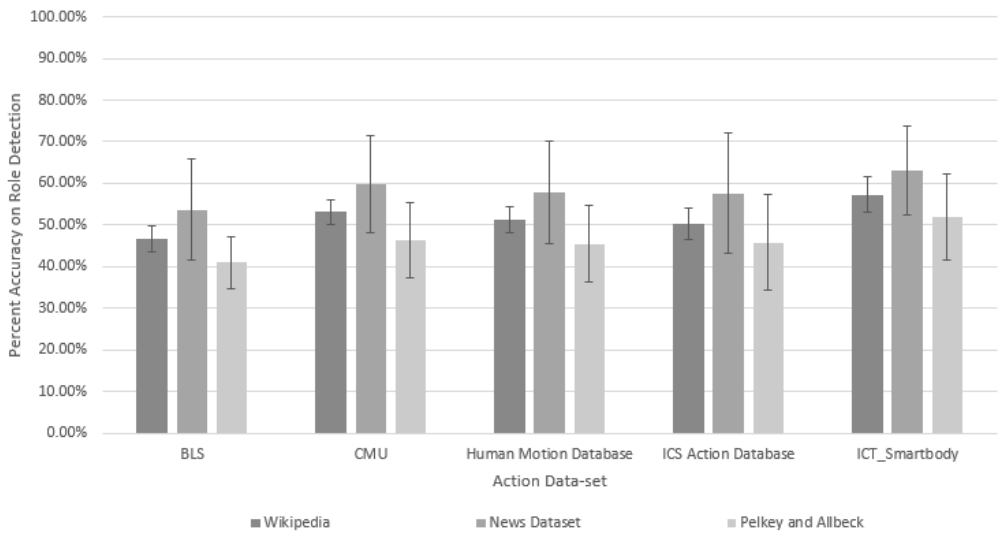


Figure 5.6: The percent accuracy of role detection vs. action data-set for ALET with two different data-sets vs. Pelkey and Allbeck. Error bars are shown as one standard deviation.

methods also shows that object descriptors by themselves cannot generate all $\mathbf{R} \in \mathbb{ACT}$. The descriptors found may connect to other information (such as that found in $\mathbf{S} \in FE$), thereby decreasing the total accuracy of connecting $\mathbf{R} \in \mathbb{ACT}$. This also means that Figure 5.6 cannot prove or disprove **H 5.3** by itself. Therefore, we also look at the systems ability to find any objects, described as the recall of the system. We compare the the data-sets and methods in Figure 5.7. A two factor ANOVA on the methods and action sets shows no significant difference between the methods and between the action sets, with $\rho = 0.05$ for both.

From Figure 5.7, it can be seen that the ability of each method to connect roles is much higher than the overall accuracy. The recall does not take into account mis-classifications of $\mathbf{S} \in FE$ as $\mathbf{R} \in \mathbb{ACT}$, which means that each data-set is able to find most of the operational information in FrameNet. This is much higher than only using the object hierarchy, but due to the mis-classifications, does not produce more accurate results because their are more mis-classifications, thereby proving **H 5.3**.

Figure 5.6 shows an increase in classification of NIFI information as operational information. As the system also discovers action descriptors, we test to see the effect matrix *VD* has in determining NIFI information from FEs. In our system, *VD* is used to classify potential FE as being part of the

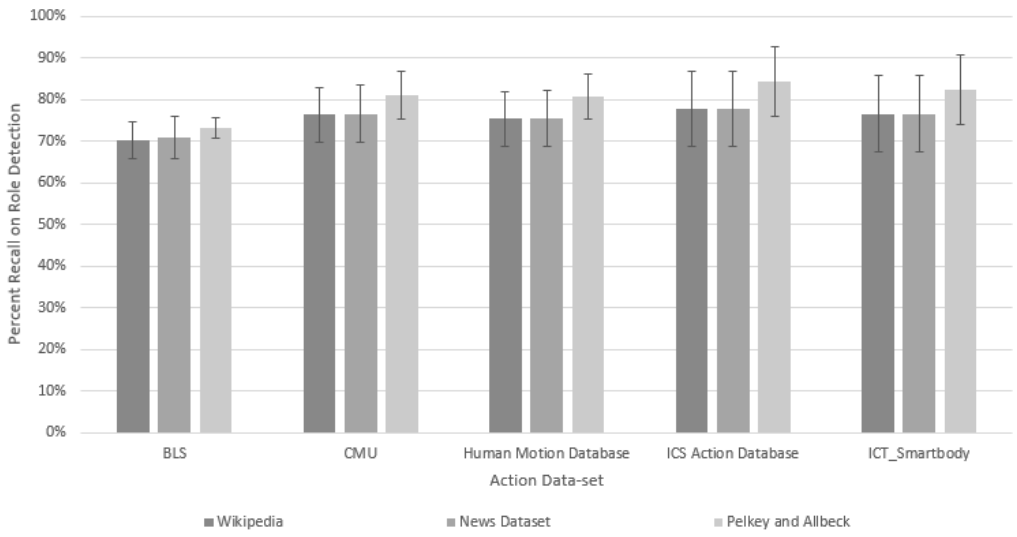


Figure 5.7: The percent recall of role detection vs. action data-set for ALET with two different data-sets vs. Pelkey and Allbeck. Error bars are shown as one standard deviation.

set $\mathbf{S} \in \mathbb{ACT}$. We compare our sets against using all the found descriptors of *VD* that appear for a particular ontology. Therefore, if one action has *quickly* as a descriptor, we assume that it is possible for any action in that particular test data-set to have that as a descriptor, using the FEs to determine if another particular actions in the set should have that descriptor. We show the results in Figure 5.8. For three of the five action sets, no operational information was found using the matrix *VD*.

Table 5.9: The average accuracy for detection $\mathbf{S} \in \mathbb{ACT}$ using ALET with Wikipedia

	BLS	CMU	Human Motion Database	ICS Action Database	ICT Smart-body
101	38%	30%	32%	36%	26%
ModelNet	38%	30%	32%	36%	26%
Office	38%	30%	32%	36%	26%
Pub	38%	30%	32%	36%	26%

Figure 5.8 shows how using the matrix *VD* can classify several NIFI elements in a given FE,

Table 5.10: The average accuracy for detection $\mathbf{S} \in \mathbb{ACT}$ using ALET with the Billion Word News Corpora

	BLS	CMU	Human Motion Database	ICS Action Database	ICT Smart-body
101	37.58%	29.72%	32.16%	35.76%	25.63%
ModelNet	37.58%	29.72%	32.16%	35.76%	25.63%
Office	37.58%	29.72%	32.16%	35.76%	25.63%
Pub	37.58%	29.72%	32.16%	35.76%	25.63%

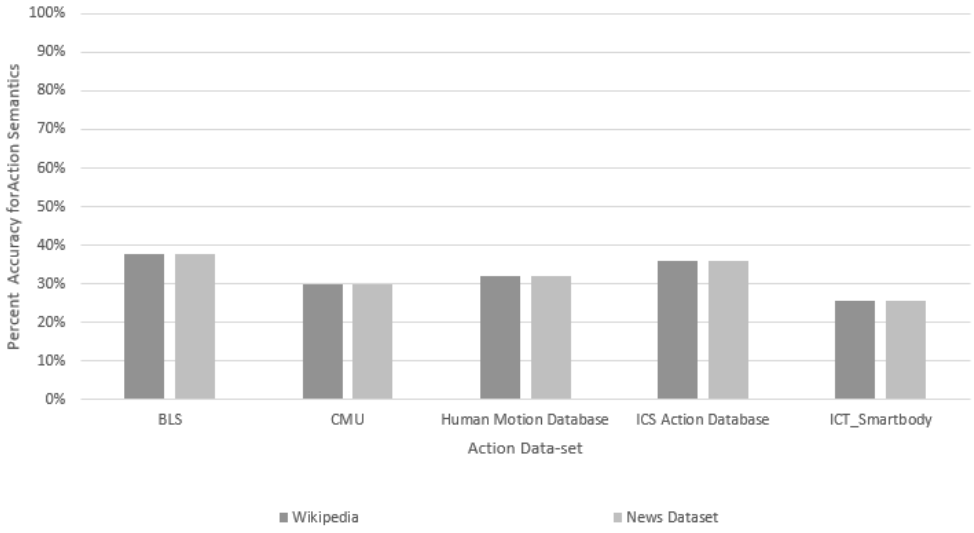


Figure 5.8: The percent overlap vs. action set when using the matrix *VD*. No error bars are shown as there was no variance between object sets.

proving **H 5.4**. *The American Time Use Survey* data-set was able to classify fewer of the FEs for actions in that ontology. There are two possible causes for this, either that the data-set itself did not have many descriptors in the text, or that the descriptors in the text did not match. For both circumstances, it shows the effect the text corpora has on the system's ability as a whole. The *American Time Use Survey* has much higher level actions (like "GoToStore" and "Research") than the other action sets, which may be described less or differently than lower level data-sets based off

of more primitive actions.

Figure 5.8 also shows that the matrix *VD* mainly classifies NIFI. For two of the data-sets, the descriptors match to FEs that are expected to be operational information. Therefore, using the matrix *VD* may have an impact on the overall system's ability to classify operational information. For a majority of the data-sets, we see no overlap in operational information, since most of the FEs found using *VD* are descriptions. However, because some of the data-sets had action descriptors we expect to be operational information, the system may suffer in its classification of the two sets. Therefore, care must be taken when finding NIFI information for actions.

To determine the efficacy of our approach, we compare ALET to the method described in our previous work [10]. We also compare ALET to a combination of our previous work [10] and Pelkey and Allbeck, which shows the effect of using semantics to find operational information. We test for both the accuracy of each method in finding operational information and NIFI, which previous methods are completely unable to do. The averaged results over each action and object set are seen in Table 5.11.

Table 5.11: The average accuracy for identifying functional elements as either operational information or NIFI. Errors are shown as one standard deviation. A single factor ANOVA found significance between tests, with $\rho < 0.001$.

Method	**% Average Accuracy**
Object Ontology Only	30.7%±12.9%
Pelkey and Allbeck	30.0%±7.0%
ALET (News)	53.9%±20.8%
ALET (Wikipedia)	51.6%±12.4%

Table 5.11 shows that ALET had a higher average accuracy than previous methods. This is mostly due to ALET's ability to detect and describe action descriptors. Previous methods can only determine operational information. As ALET is able to determine action descriptions, we can more fully describe an action with terms such as *duration* and *frequency*, which are widely used when

controlling virtual characters, and appear in specifications such as the Behavioral Markup Language [66]. We also examine each system's ability to identify roles compared to our FE breakdown. This second test takes into account mismatches between $\mathbf{S} \in FE$ and $\mathbf{R} \in \mathbb{ACT}$. This shows the ability of ALET to correctly detect only roles. The results can be seen in Table 5.12.

Table 5.12: The average accuracy for identifying functional elements only as operational information. Errors are shown as one standard deviation. A single factor ANOVA found significance between tests, with $\rho < 0.001$.

Method	**% Average Accuracy**
Object Ontology Only	60.9%±25.9%
Pelkey and Allbeck	45.9%±14.0%
ALET (News)	62.2%±25.7%
ALET (Wikipedia)	71.4%±20.7%

Table 5.12 shows that there is an upper bound in matching objects to roles, replicating the results in [10]. However, simply adding descriptors to objects, using Pelkey and Allbeck, actually decreases the overall accuracy of the system. This is because the semantics described there are too greedy, and match object to $\mathbf{S} \in FE$. As ALET first determines $\mathbf{S} \in \mathbb{ACT}$ before matching roles, it reduces the number of false positives that it finds, while increases the overall accuracy of the roles found. The data-source used affects ALET's ability, with the larger data-source (Wikipedia) having more accurate results. The ability of ALET using both the Billion Word News Corpus and Wikipedia against methods that do not take this data into account prove **H 5.5**.

5.3.5 Analysis of Adjectives and Adverbs in Dataset

ALET generates co-occurrence values based on the relational tuples searched for. Co-occurrences of the same type are summed up, and a cutoff value can be established, removing relations that are not found often in the corpus. This presents the interesting question of "What is an optimal cut-off value?". Furthermore, is there a given number of descriptors that are adequate to match $\mathbf{R} \in \mathbb{ACT}$ and $\mathbf{S} \in \mathbb{ACT}$ correctly while minimizing mis-matches. To answer this question, we examine the

descriptors, searching for an optimal threshold based on the object and action data-sets. Figure 5.9 and Figure 5.10 show the effect of the cutoff value on the average number of descriptors provided to the system, based on the objects and actions in the test data-set. We also show the values that we used for the experiments in Section 5.3.4.

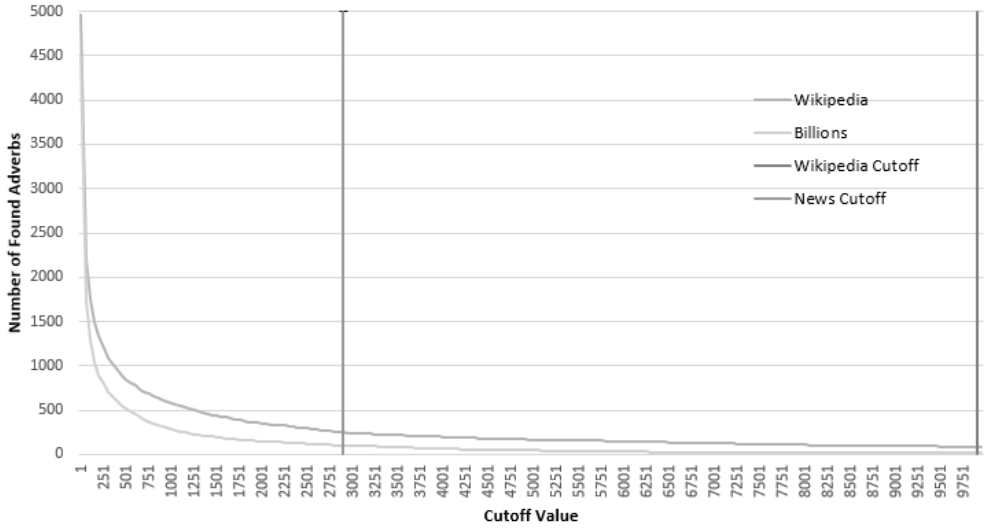


Figure 5.9: The average number of adverbs connected to action sets vs the cutoff value. The cutoff values used in our experimentation are shown as dotted lines.

Both Figure 5.9 and Figure 5.10 display a steep drop-off curve as the cutoff value is increased. As expected, the descriptors of the News data-set are initially much less, and drop off quicker than the number of descriptors in the Wikipedia data-set. This is simply due to the size of the data-set in general, with the Wikipedia data-set being much larger than the News data-set. This means that the optimal cutoff value should be less for the news data-set than the Wikipedia data-set. Based on the cut-off values used and the score similarity in Section 5.3.4, this is what we found. This fact is similar for both Figure 5.9 and Figure 5.10, showing that the size of the data-set has an effect on the cutoff value, without having a large effect on the ability of the system. Therefore, care should be taken for choosing cutoff values for tuples based on the size of the data-set.

We further examine the effect the cutoff value has on the metrics we used in Table 5.11. As

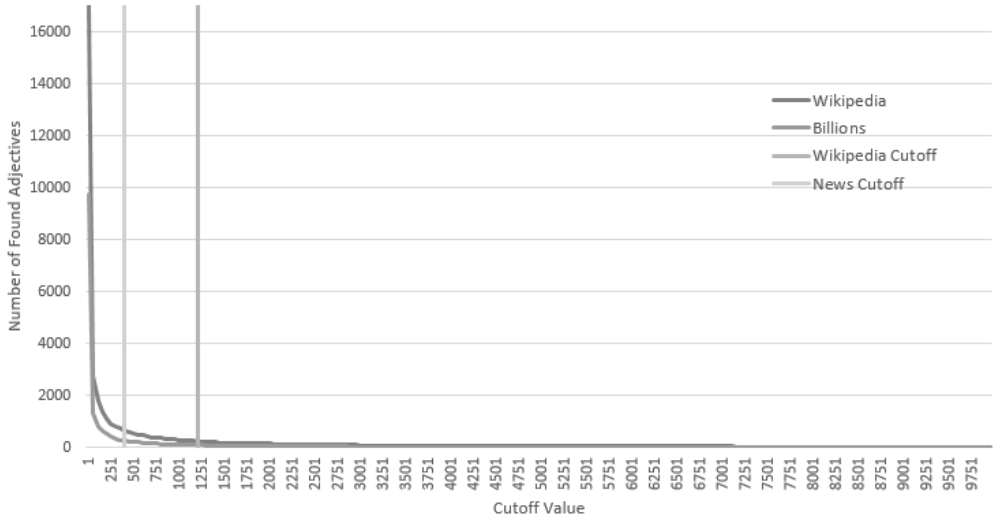


Figure 5.10: The average number of adjectives connected to object sets vs the cutoff value. The cutoff values used in our experimentation are shown as dotted lines.

the cutoff value has an effect on the amount of information being fed into the system, it follows that a good cutoff value would show better results for the system. We show the effect the adjective cutoff value has on the total accuracy in Figure 5.11, object recall in Figure 5.12, and accuracy in detecting roles in Figure 5.13. For all three experiments, the adverb cutoff value was held constant, with Wikipedia being 10000 and the news Corpus being 1500, which are beyond the control cut-off lines.

Figure 5.11 shows the effect different data-sets has on the total accuracy of the system. What should be noted from the figure is that the two data-sets converge onto a given accuracy. This is due to having cutoff values high enough that adjectives are not matched in the system, and therefore only objects are considered as matches. This is an important consideration for a simulation author. Attempting to use only the most used adjectives does not necessarily mean that the adjectives will be connected to funtional elements. It should also be noted that the two data-sets display similar shape, but at different adjective cutoff values. This should also be expected, as the size of the two data-sets are different, and so the data-set with less overall data (Billion Words News) should have smaller cutoff values than Wikipedia. Therefore, the cutoff value for adjectives are dependent on the size of the data. Examining Figure 5.10 with Figure 5.11, it can be seen that there is an optimal

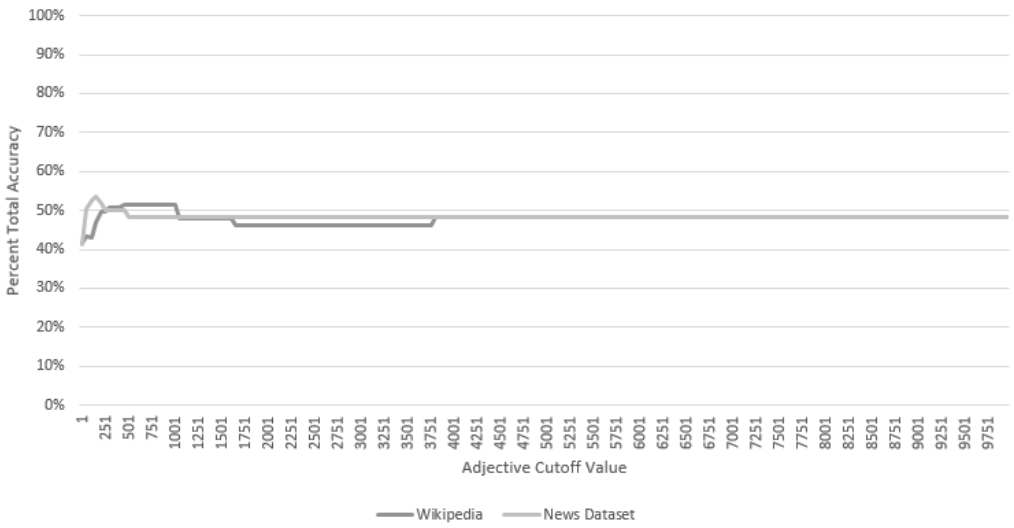


Figure 5.11: The average total accuracy of the system vs. the adjective cutoff value when using ALET.

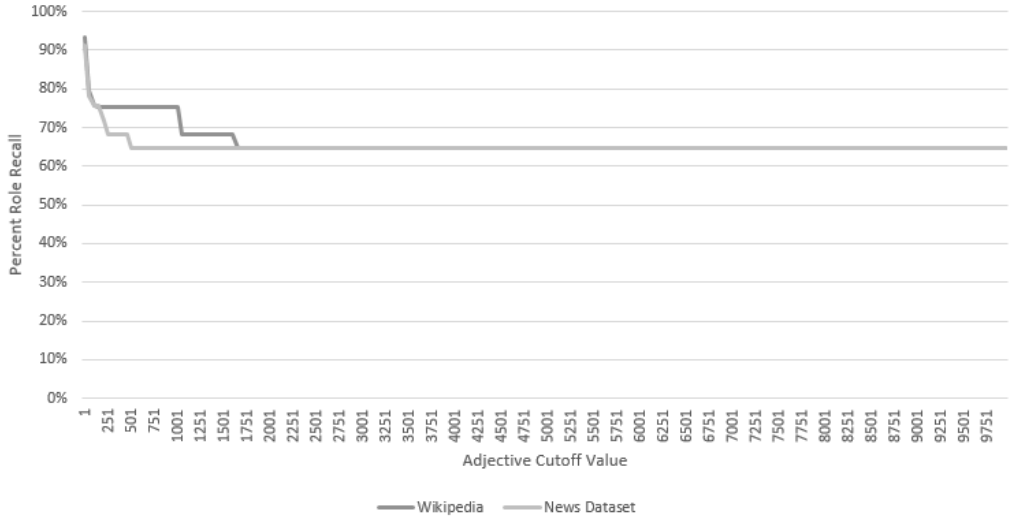


Figure 5.12: The average object recall of the system vs. the adjective cutoff value when using ALET.

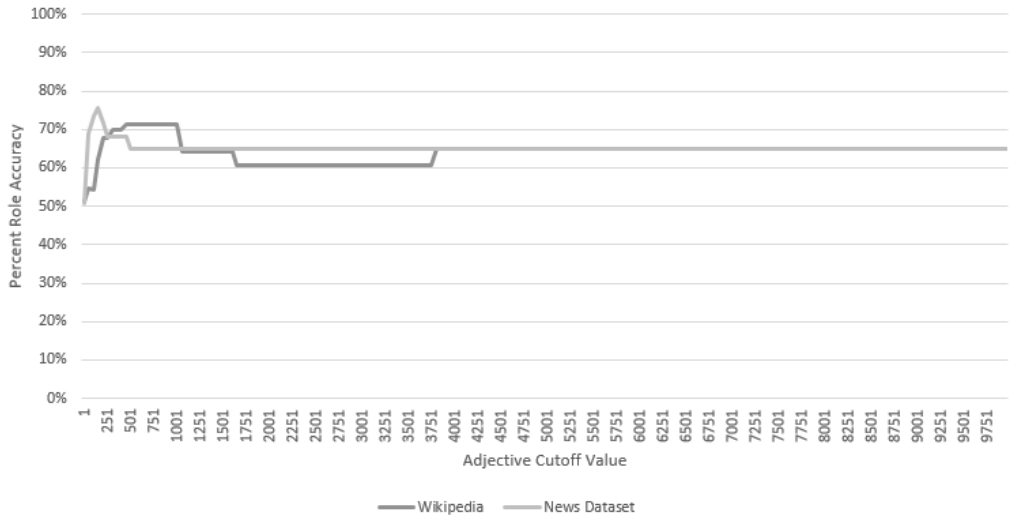


Figure 5.13: The average accuracy of detecting operational information $\mathbf{R} \in \mathbb{ACT}$ vs. the adjective cutoff value when using ALET.

number of adjectives for both data-sets, and the optimal value is around 200. While not all adjectives in the optimal value are used to connect actions to objects, 200 is a large enough mix to make the connections.

In addition to Figure 5.11, we examine the effect adjectives have on the recall and accuracy of populating $\mathbf{R} \in \mathbb{ACT}$. Similar to Figure 5.11, the two data-sets converge on a single value, which is mostly due to them being related measurements (Figure 5.13 is a component in Figure 5.11). Figure 5.12 shows that the adjectives have an effect on finding operational information to a given point, and so if a simulation author is only concerned with populating $\mathbf{R} \in \mathbb{ACT}$, then lower values are more useful in this task. However, the total accuracy of populating $\mathbf{R} \in \mathbb{ACT}$ increases with the cutoff value (to a point), because information that should be considered NIFI are also being found. Examining the Wikipedia data-set in Figure 5.13, it can actually be seen that the accuracy decreases for certain cut-off values, because it is finding information from $\mathbf{S} \in FE$ and categorizing it as $\mathbf{R} \in \mathbb{ACT}$.

In addition to the effect the cutoff value for adjectives have on ALET, we examine the effect the adverbs have on ALET. Recall that action descriptors are used to find $\mathbf{S} \in \mathbb{ACT}$ **before** roles are determined, causing adverbs to affect both NIFI population and role population. As such, we use

the same three metrics that we used when describing adjectives. We present the result of changing the adverb cutoff value on ALET's whole ability in Figure 5.14. The residual effects of adverbs on role information can be seen in Figure 5.15 and Figure 5.16.

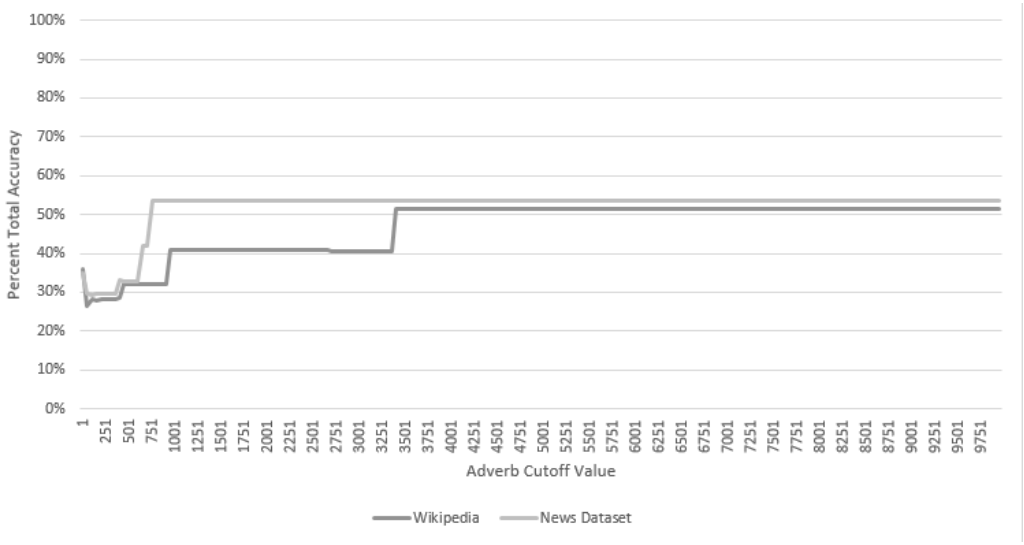


Figure 5.14: The average total accuracy of the system vs. the adverb cutoff value when using ALET.

Unlike Figure 5.11, Figure 5.14 does not converge between the two data-sets. This has several possible reasons. As we see this effect for Figure 5.14, Figure 5.15 and Figure 5.16 (where Figure 5.15 and Figure 5.16 are concerned with objects), we can infer that the false positive role matches hve an inverse relation with the adverb cutoff value. This should be expected, as the verb descriptors matched to Functional Elements are removed from role consideration. It should also be inferred from Figure 5.14 that there is not much difference after the cutoff plateau. This can also be seen in Figure 5.9, which shows that the total number of descriptors does not decrease much after a certain point (appearing as exponential decay). Therefore, the descriptors that are removed in later sets do not have, for these test cases, have much of an impact on finding roles or action semantics.

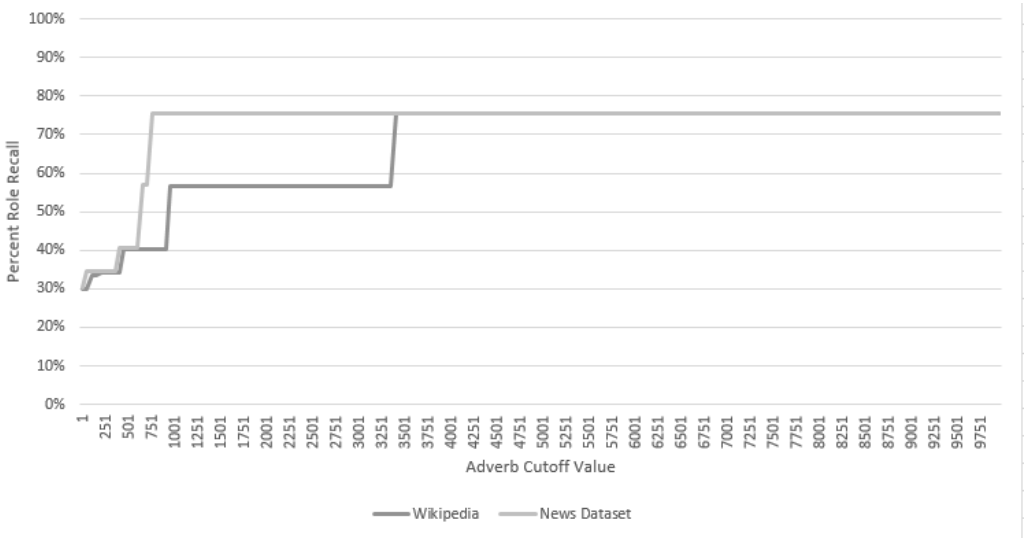


Figure 5.15: The average object recall of the system vs. the adverb cutoff value when using ALET.

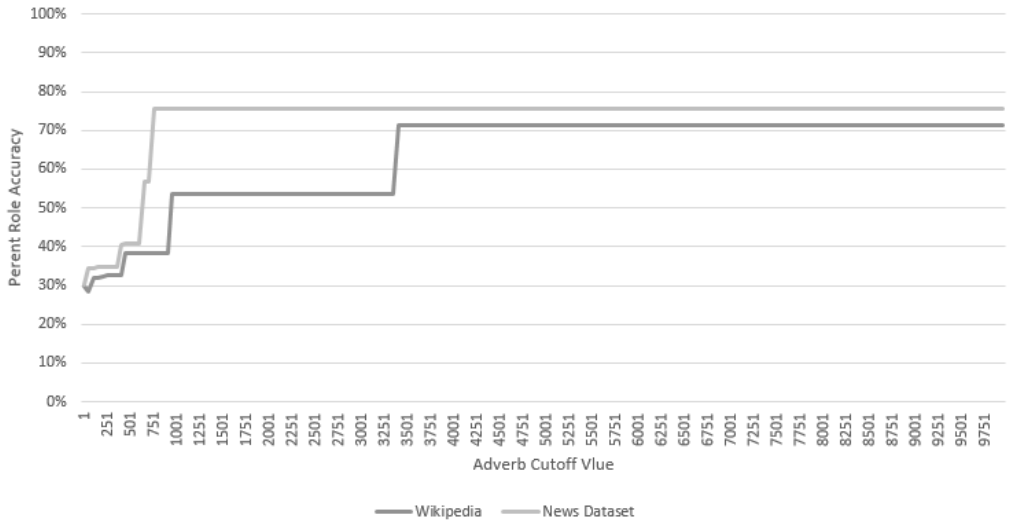
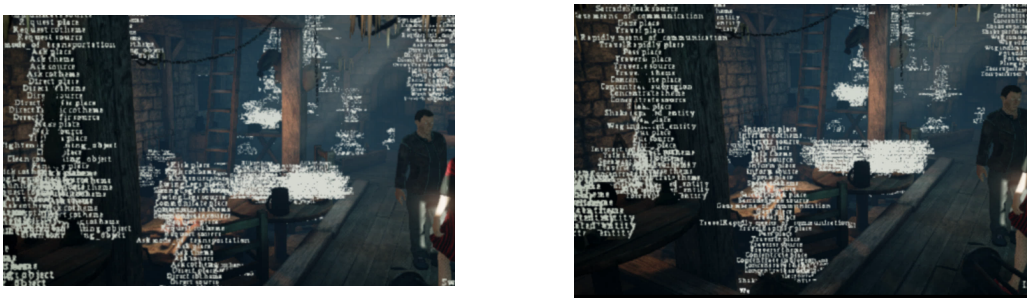


Figure 5.16: The average accuracy of detecting operational information $\mathbf{R} \in \mathbb{ACT}$ vs. the adverb cutoff value when using ALET.

5.3.6 Demonstration

ALET is meant to ease the burden of connecting objects in a virtual environment to performable actions. To that end, we show ALET connecting action sets to roles for two environments, using the Wikipedia text corpora. The results of different action sets on the same environment can be seen in Figure 5.17. In this example, we can see that many actions have similar roles, such as objects being a *Theme* of actions *Kick*(Figure 5.17a) or *Nod*(Figure 5.17b). This is not true for all roles, as most objects are not connected to *Place*(Figure 5.17a) and that it is specialized to the building itself. Thus, ALET can take several action sets and generate connections for the same object set, taking into account the differences in the virtual agent actions and their objects.



(a) (b)

Figure 5.17: A virtual environment sold on the Unreal Engine Marketplace annotated with operational information. The action set shown is (a) ten actions from CMU and (b) nineteen actions from SmartBody.

We also show how changing the environment on the same action set affects the operational information. Figure 5.18 uses the CMU action set on a different constructed virtual environment from a different time period. While the object sets are different, and from different time periods, the fact that actions can be used on a variety of objects shows the the overall roles are similar. Thus, ALET can find descriptors to connect different sets of objects to the same action. This is the work a simulation author would have to perform, meaning ALET can be used as a tool to assist agent authors in populating the knowledge of virtual humans.

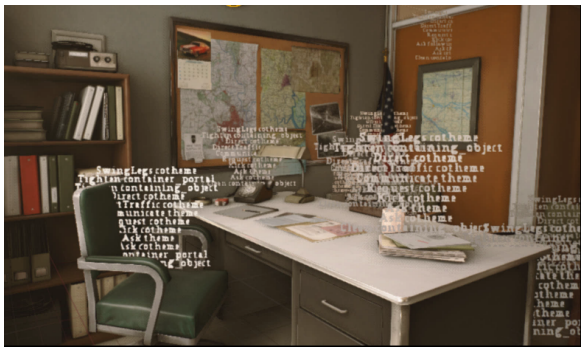


Figure 5.18: A virtual environment sold on the Unreal Engine Marketplace automatically annotated with operational information for ten actions generated with CMU action set.

5.4 Conclusions

We present methods to automate the creation of semantics for actions, specifically focusing on $\mathbf{R} \in \mathbb{ACT}$ and $\mathbf{S} \in \mathbb{ACT}$. One way to create $\mathbf{R} \in \mathbb{ACT}$ is to use knowledge bases such as FrameNet as an intermediary between $\mathbb{OBJ}$ and $\mathbb{ACT}$. Using FrameNet does not determine $\mathbf{S} \in \mathbb{ACT}$ and leaves out several possibly important connection. Therefore, we developed ALET to connect intelligent virtual human actions and objects and create an understanding of how the objects can participate in actions. The system can also provide candidate description slots, providing a virtual human with more information about the actions it can perform. ALET leverages text to find candidates for both ontological connections as well as object and action descriptions. FrameNet is used to provide a base and describe the roles objects play in a given action.

Our system of learning candidate descriptions of our virtual environment is based on the available objects, actions, and text. While the first two are obvious and should effect determining descriptions (as virtual characters should not worry about actions they are never able to perform and have no knowledge of), the choice of text is a bit more nuanced, and has an effect on the system, as seen in Table 5.11 and Table 5.12. The difference between data-sets in Table 5.11 clearly shows that the choice of knowledge base for ALET has an effect on the ability of the system. Figure 5.11 and Figure 5.14 strengthen that understanding, and show the dual effect that verb descriptors play on the system as a whole. The combined efforts of the system are such that $\mathbf{R} \in \mathbb{ACT}$ and $\mathbf{S} \in \mathbb{ACT}$

are disjointed even when, from a natural language perspective, they may not need to be.

AELT is also able to determine NIFI for actions. NIFI are used to control graphical aspects of actions, such as the *manner* in which an action is performed. Currently, ALET only detects when this data is available to actions, but does not classify how to fill in these fields and connect them to the animations. Future work will also examine how to cluster action descriptors for NIFI generalizations. This will expand ALET to assist in not only virtual human planning, but also in motion planning, which is currently outside the scope of this work.

Chapter 6: Conclusions and Future Work

6.1 Summary and Conclusions

The overall research goal of this work was to advance the ability of simulation authors in creating actions based on their animations. We did this by first defining actions, and then though the use of automated semantic generation that encapsulates the animations along with important meta-data that instruct both AI and graphical systems in the action's use. We summarize the key contributions of this work as follows:

6.1.1 Key Contributions

- In Chapter 2, we formalized and refined an action representation (PAR) to better exploit its capabilities and enable automated generation methods.
- In Chapter 3, we demonstrated the utility of action taxonomies through an application based measure of efficiency gain.
- In Chapter 4, we developed and evaluated novel methods to automate the population of action taxonomies from base action names through use of existing lexical databases, published in previous work [10].
- In Chapter 5, we developed and evaluated a transformative pipeline, ALET, to automate connections between virtual agent actions and 3-D graphical models as well as the population of action semantics, published in previous work [11].

To show the efficacy of our approach, we have provided a task based examination of action hierarchies, as well as proven and dis-proven several hypotheses about our automated generation

approaches using more traditional measures when possible. We have shown that automated methods can be used to create generalizations if we understand some textual information about the action, such as its name and some intelligently examined keywords. With known action parents, text corpora and an object hierarchy, we have shown that connections between 3D graphical models and animations can be made. Using large text corpora, it is also possible to add in non-interactional meaning to actions, increasing the overall understanding of action sets.

6.2 Lessons Learned

Throughout this work, we were careful to use the term *automated* instead of *automatic*. This is due to the fact that generalization methods are not perfect, but instead provide candidate solutions to action parents, roles, and semantics. The experimentation that we performed in Chapter 4 and Chapter 5 also supports this. In reality, action generation will never be 100% accurate. The work presented in these chapters should be thought of as a tool to aid a simulation author instead of replace a simulation author. In the field of information retrieval, there is the concept of precision-recall, a metric that we have used for some of our experimentation as well. Ideally, a system should maximize the true positives while minimizing the false positive results, and there is a trade-off between the two. By providing the author with a lot of candidates (relying more on recall than precision), the simulation author can remove connections that are unnecessary. One lesson learned is that our generation methods will never be perfect, but that the focus should be more on maximizing recall (finding action-object connections) than minimizing precision(pruning connections to only have correct ones), with the best solution being a happy medium between the two.

Chapter 4 and Chapter 5 use action and object hierarchies, whose names come from motion data and 3-D graphical models. The names of this data and models play an important role in the overall ability of our system. If the name has no relation to the graphical component that it represents, then our automated methods will not be able to represent that graphical component. Therefore, naming conventions are important. We stated in Chapter 4 that each action name must have at least one verb in order for WordNet to generate candidates. Similarly, a noun must appear for objects in Pelkey and Allbeck [37]. One lesson learned is that naming conventions for graphical objects and motion

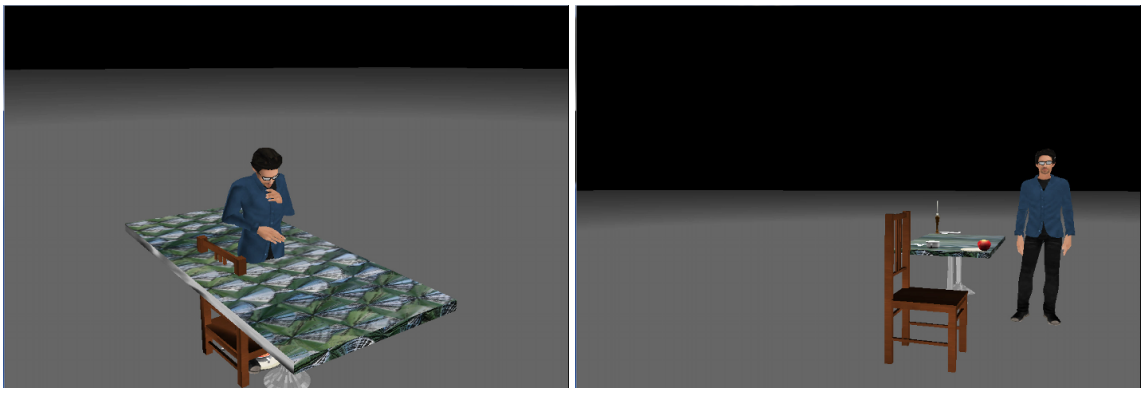
data does not always follow this convention. In reality, the names of graphical data vary widely between authors, with different capitalization patterns and mis-spellings being abundant. Our rules for naming actions in Chapter 4 are general, but understand that a simulation author may have to rename downloaded data in order to meet these requirements.

Also in Chapter 4, we discussed a user defined α for our multi-sieve and confidence based method. The choice of α has an effect on the system's ability to disambiguate the actions. Recall that the comparison method determines a percent, which means α can range from zero to one. When α is low, the higher heuristics of the multi-sieve method dominate, such that the data-set appears closer to only using the word heuristic. At higher α, the word and definition methods do not find synsets, which means there is no set to seed the path heuristic. We found through this work and previous work [10] that an α value of $0.2 - 0.4$ strikes a balance between too low and too high, based on our test sets. In reality, multiple values for both the Multi-sieve and Confidence based methods should be attempted by the simulation author in order to generate the taxonomy that is desired.

Another important consideration we discovered while examining automated methods is the need for a large amount of textual data in order to reduce the noise in our dataset. With a small amount of textual data (on the order of a few million words), the ability of ALET and our word vector models is greatly diminished While the deep learning literature describes the need for a large amount of data, ALET requires a large amount of text in order to find patterns of descriptors. When there is not much data, overall counts of descriptors are low, meaning ALET cannot prune what in reality are strange connections. With a large amount of textual data, what are thought of as "normal" connections appear more frequently. One of the important lessons learned is that the data chosen is extremely important. Having the right kind and enough data is going to have an effect on the system.

6.3 Future work

This work shows the possibility of populating actions from descriptive names of animations. From this work, we are able to populate the action's parents (**P**), the roles of the action (**R**) and semantics types of actions (**S** from $\lambda \in \mathbb{S}$). These are not the only components of actions, and future work should focus on populating the other components of actions. Several planning methods exist to create complex action chains that describe entire plans and phenomena [42, 45], which in PAR, would be automating the generation of Θ. Recently, interactive action creation has examined task creation from a human centered approach in which the human has complete control of the connections [46, 57]. With these approaches, the actions that are being constructed are also domain specific, and require an expert in action creation to build the semantic base (primitive set) of the actions, which require the cause and effects of the primitive set of actions. Work in causal chains of actions has also seen some progress towards constructing complex actions [128]. However, this work has seen little progress because it attempts to build chains from overly-specified domains such as a cooking domain (using only recipes) and construction domain (using instruction manuals). The data that they are using for these domain already have the causal chains laid out in them, such that the process is really just memorizing the instructions, usually under the assumption that any verb is an action. Since we have also used the assumption that a verb can be matched to an action, we believe that one promising future direction is to generate both task and condition-assertion generation. The action recognition domain is also starting to examine learning the prerequisites of actions from a variety of sources [129, 128]. This is because if the cause of an action is known, the action can be better predicted. Specifically, Yordanova et al. [128] examines causal steps for actions in a cooking domain. However, the cooking domain is a step by step domain, in which the cause of one action can be clearly inferred from the end of another action. This means that the examined action generation techniques would not work on domains where the cause cannot be clearly inferred, such as a scientific journal domain or human behavior domain. To examine these domains, the effects, as well as the prerequisites, of the action must be able to be learned from a data-set. While it is possible to learn primitive (base) actions from motion meta-data, generating the correct complex actions is a more open-ended task, as shown with STRIPS actions in Branavan et al. [130].



(a) (b)

Figure 6.1: An example scene (a) without processing to move the objects and (b) with processing to constrain the objects.

While action generation is an important goal to provide a larger, more nuanced repertoire, it does not by itself increase the abilities of virtual agents. However, by increasing action understanding, new abilities that require a more nuanced understanding can be developed. Two such applications are text-to-scenario generation and agent inference methods. Text-to-scene generation has been explored over the past decade [54, 131, 55]. Similarly, work on automated scene generation [132, 133] has been able to create virtual environments based on either learned or programmed rules. However, this work mainly focuses on static scenes, with the exception being Ma's work [55], which focused on text to a single animation. Because of the cost associated with creating actions, it becomes prohibitive to connect motion data to text, especially when the motion data is expected to change the surrounding environment over time. While current text to scene systems can spatially place objects based on the verbs in a given text, any temporal conflicts manifest themselves immediately, and require an artist to simulate the system. Using actions can account for some temporal conflicts, and can also be used to show where parts of the scene are under or over specified, shown in Figure 6.1. Larger action sets would have more actions accounted for in the scene, allowing for greater fidelity in a text-to-simulation system.

In a similar manner, having consistent and well defined actions would advance the ability of machines to understand the world around them. Besides having high fidelity agents, generating

consistent action sets would allow domain expert agents to be created. These agents could assist researchers in analyzing their simulations. This is the stance of the action recognition community, who have started to identify actions by object roles from domain corpora such as amateur modern fiction [116]. This work identifies objects in a video, and guesses the most likely action from that. Having a machine understand more complex scientific visualizations is going to require a more complete understanding of that domain. But with this understanding, the scientific community can have machines start to mine visualizations and describe the patterns that it finds. This is known as scene-to-text, and has seen some work on simplistic visualizations [134], using mainly object and action connections. Being able to mine a visualization and describe what is occurring in the visualization would allow for new analytics tools, assisting researchers in discovering the underlying phenomenons. A possible example for wireless network sensor data can be seen in Figure 6.2. In Figure 6.2, the action set is used by the observer agent to match and identify actions in the visualization. The visualization is a 3-D environment simulating real data from a wireless sensor network. The observations of the agent should be considered a data transformation, allowing the transformation to be analyzed in terms of the actions themselves. One application of this transformation is that researchers would have a system to check their simulations, to determine if the simulation matches the researcher's expectation. If the analysis from the agent came back incorrect for one simulation and not for others, then the researcher could check that one simulation and debug it.

In Chapter 4, we presented three methods to generate action parents and organize actions based on their meta-data. However, we did not attempt to combine different creation and organization methods. An interesting extension of Chapter 4 would be to use word-vectors to disambiguated the actions into WordNet synsets, and then use create hypernym trees. Disambiguated synsets could also be converted into a word-vector representation, and clustered based on their similarity. Simple changes to our provided techniques could be used to combine the techniques, and future work will examine these combinations.

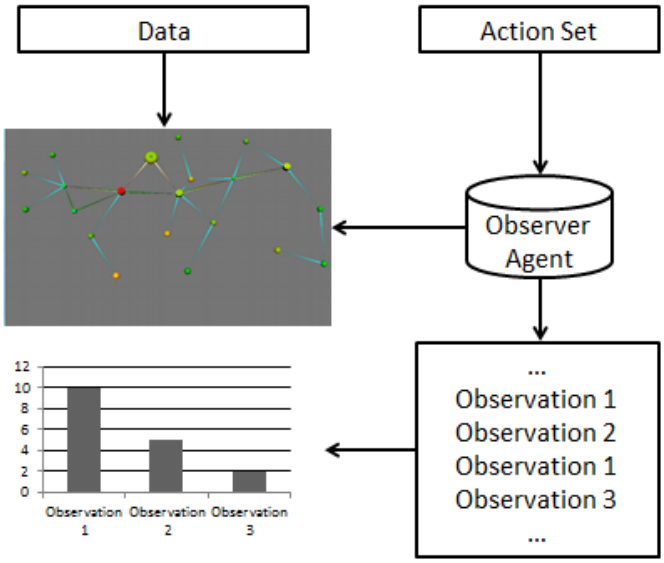


Figure 6.2: A possible pipeline for using scene-to-text in analytic software. Shown here is a data-set, action-set and visualization of a wireless sensor network that is currently being researched. We have designed the visualization for their system already.

Appendix A: PARS used in Case Study

The case study from *The Syntax of PAR* was converted over from the work of Shoulson et al. **??**. Below is all the information used in that case study.

A.1 Object Designation

Each object $obj \in \mathbb{OBJ}$ has a designation of $\{\lambda, p, S\}$, which we display below.

1. $\lambda =$ Entity $p =$ None $S =$
2. $\lambda =$ Physical_Entity $p =$ Entity $S =$
3. $\lambda =$ CausalAgent $p =$ Physical_Entity $S =$
4. $\lambda =$ Object $p =$ Physical_Entity $S =$
5. $\lambda =$ Person $p =$ CausalAgent $S =$
6. $\lambda =$ Agent $p =$ CausalAgent $S =$
7. $\lambda =$ Whole $p =$ Object $S =$
8. $\lambda =$ Location $p =$ Object $S =$
9. $\lambda =$ Preserver $p =$ Person $S =$
10. $\lambda =$ Unfortunate $p =$ Person $S =$
11. $\lambda =$ Artifact $p =$ Whole $S =$
12. $\lambda =$ Point $p =$ Location $S =$
13. $\lambda =$ Defender $p =$ Preserver $S =$
14. $\lambda =$ Prisoner $p =$ Unfortunate $S =$ { status:['Free', 'Idle', 'Active', 'Trapped']}

15. λ = Instrumentality p = Artifact S =

16. λ = Structure p = Artifact S =

17. λ = WayPoint p = Point S =

18. λ = Guard p = Defender S = { status:['Free', 'Dazed', 'Idle', 'Active', 'Trapped']}

19. λ = Device p = Instrumentality S =

20. λ = Obstruction p = Structure S =

21. λ = Key p = Device S =

22. λ = Mechanism p = Device S =

23. λ = Trap p = Device S = { status:['Idle', 'Active']}

24. λ = Alarm p = Device S = { status:['Idle', 'Active']}

25. λ = Barrier p = Obstruction S =

26. λ = Control p = Mechanism S =

27. λ = MovableBarrier p = Barrier S =

28. λ = Switch p = Control S =

29. λ = Door p = MovableBarrier S = { status:['Guarded','Idle'] locked:['Lock', 'Unlock'] open:['open', 'close']}

30. λ = PushButton p = Switch S =

31. λ = Button p = PushButton S =

A.2 Designation of Actions

We include in this set of tables the actions that are defined in event-centric planning as well as ones that are assumed to be part of the action set (i.e *Guard* and *Open*). We also provide a WordNet hierarchy representation of the actions, generated through the hand-selection of a single synset, such that each action can be generalized.

Table A.1: Action Number 1:Hide

λ	Hide
P	None
R	WayPoint
$\Psi(e=1)(PAR)$	finishedAction(self.id)→SUCCESS
Θ	Hide

Table A.2: Action Number 5:Lock

λ	Lock
P	Fasten
R	Door
$\Psi(e=0)(PAR)$ $\Psi(e=1)(PAR)$	canReach(agent,Door)→FAILURE finishedAction(self.id)→setProperty(Door,"lock") = "locked" AND SUCCESS
Θ Θ	not getProperty(Door,"open") = "closed"→Close Lock

Table A.3: Action Number 10:EscapeCell

λ	EscapeCell
P	Escape
R	Guard
R	Door
$\Psi(e=0)(PAR)$	not contains(Guard,Key)→FAILURE
$\Psi(e=1)(PAR)$	finishedAction(self.id)→changeContents(agent,Key) AND setProperty(Guard,"status") = "trapped" AND setProperty(Door,"look") = "locked" AND setProperty(Door,"open") = "closed" AND SUCCESS
Θ	not canReach(Guard,Door)→Approach
Θ	not getProperty(Door,"open") = "closed"→Close
Θ	not getProperty(Door,"lock") = "locked→Lock
Θ	And(Call,Flee,Daze,And(Approach,Take) ,And(Lock,Lock))

Table A.4: Action Number 12:Press

λ	Press
P	Touch
R	Button
$\Psi(e=1)(PAR)$	finishedAction(self.id)→SUCCESS
Θ	Press

Table A.5: Action Number 16:Guard

λ	Guard
P	Watch
R	Door
$\Psi(e=0)(PAR)$	getProperty(Door,"status") = "guarded"→SUCCESS
$\Psi(e=0)(PAR)$	getProperty(agent,"status") = "dazed"→FAILURE
$\Psi(e=1)(PAR)$	finishedAction(self.id)→setProperty(Door,"status") = "guarded" AND SUCCESS
Θ	not canReach(agent,Guard)→Approach
Θ	Guard

Table A.6: Action Number 19:Trap

λ	Trap
P	Capture
R	Trap
$\Psi(e=1)(PAR)$	finishedAction(self.id)→setProperty(Agent,"status") = "Trapped" AND SUCCESS
Θ	Trap

Table A.7: Action Number 20:TrapGuardsAlarm

λ	TrapGuardsAlarm
P	Trap
R	Guard
R	Guard
R	Guard
R	Guard
R	Prisoner
R	Button
R	Trap
R	Button
$\Psi(e=0)(PAR)$	hasProperty(alarm,"status") = "active"→FAILURE
$\Psi(e=0)(PAR)$	not controls(Button2,alarm)→FAILURE
$\Psi(e=0)(PAR)$	not hasPath(Guard3,waypoint) = trap→FAILURE
$\Psi(e=0)(PAR)$	not controls(Button1,trap)→FAILURE
$\Psi(e=0)(PAR)$	hasProperty(trap,"status") = "active"→FAILURE
$\Psi(e=0)(PAR)$	not hasPath(Guard1,waypoint) = trap→FAILURE
$\Psi(e=0)(PAR)$	not hasPath(Guard2,waypoint) = trap→FAILURE
$\Psi(e=0)(PAR)$	not hasPath(Guard4,waypoint) = trap→FAILURE
$\Psi(e=1)(PAR)$	finishedAction(self.id)→setProperty(Guard1,"status") = "trapped" AND setProperty(Guard2,"status") = "trapped" AND setProperty(Guard3,"status") = "trapped" AND setProperty(Guard4,"status") = "trapped" AND setProperty(trap,"status") = "active" AND setProperty(Button,"status") = "active" AND SUCCESS
Θ	not canReach(Prisoner,Button2)→Approach
Θ	not canReach(Agent,Button1)→Approach
Θ	And(And(Press,Join(Approach,Approach,Approach,Approach)) ,And(Approach,Join(Call,Call,Call,Call) ,Join(Approach,Approach,Approach,Approach) ,Press,Join(Trap,Trap,Trap,Trap)))

Table A.8: Action Number 21:TrapGuards

λ	TrapGuards
P	Trap
R	Guard
R	Guard
R	Guard
R	Guard
R	Prisoner
R	Button
R	Trap
R	WayPoint
$\Psi(e=0)(PAR)$	not hasPath(Guard3,waypoint) = trap→FAILURE
$\Psi(e=0)(PAR)$	not controls(Button,trap)→FAILURE
$\Psi(e=0)(PAR)$	hasProperty(trap,"status") = "active"→FAILURE
$\Psi(e=0)(PAR)$	not hasPath(Guard1,waypoint) = trap→FAILURE
$\Psi(e=0)(PAR)$	not hasPath(Guard2,waypoint) = trap→FAILURE
$\Psi(e=0)(PAR)$	not hasPath(Guard4,waypoint) = trap→FAILURE
$\Psi(e=1)(PAR)$	finishedAction(self.id)→setProperty(Guard1,"status") = "trapped" AND setProperty(Guard2,"status") = "trapped" AND setProperty(Guard3,"status") = "trapped" AND setProperty(Guard4,"status") = "trapped" AND setProperty(trap,"status") = "active" AND SUCCESS
Θ	not canReach(Agent,Waypoint)→Approach
Θ	not canReach(Prisoner,Button)→Approach
Θ	And(Approach,Join(Call,Call,Call,Call) ,Join(Approach,Approach,Approach,Approach) ,Press,Join(Trap,Trap,Trap,Trap))

Table A.9: Action Number 24:Draw

λ	Draw
P	Change
R	Guard
$\Psi(e=1)(PAR)$	finishedAction(self.id)→SUCCESS
Θ	Draw

Table A.10: Action Number 26:Daze

λ	Daze
P	Stun
R	Guard
$\Psi(e=1)(PAR)$	finishedAction(self.id)→setProperty(Guard,status) = dazed AND SUCCESS
Θ	not canReach(Agent,Guard)→Approach
Θ	Daze

Table A.11: Action Number 35:Call

λ	Call
P	Order
R	Guard
$\Psi(e=1)(PAR)$	finishedAction(self.id)→SUCCESS
Θ	Call

Table A.12: Action Number 38:Approach

λ	Approach
P	Come
R	Entity
$\Psi(e=1)(PAR)$	finishedAction(self.id)→SUCCESS
Θ	Approach

Table A.13: Action Number 40:Give

λ	Give
P	Transfer
R **R**	Prisoner Key
$\Psi(e=1)(PAR)$	finishedAction(self.id)→SUCCESS
Θ	Give

Table A.14: Action Number 41:Exchange

λ	Exchange
P	Give
R **R**	Prisoner Key
$\Psi(e=0)(PAR)$ $\Psi(e=0)(PAR)$ $\Psi(e=1)(PAR)$	not contains(Agent,Key)→FAILURE contains(Prisoner,Key)→SUCCESS finishedAction(self.id)→SUCCESS
Θ Θ	not canReach(Agent,Prisoner)→Approach And(Approach,Give)

Table A.15: Action Number 43:Open

λ	Open
P	None
R	Door
$\Psi(e=0)(PAR)$	not getProperty(Door,"open") = "closed"→SUCCESS
$\Psi(e=0)(PAR)$	getProperty(Door,"lock") = "locked"→FAILURE
$\Psi(e=1)(PAR)$	finishedAction(self.id)→setProperty(Door,"open") = "open" AND SUCCESS
Θ	not canReach(Agent,Door)→Approach
Θ	Open

Table A.16: Action Number 44:Unlock

λ	Unlock
P	Open
R	Door
R	Key
$\Psi(e=0)(PAR)$	getProperty(Door,"open") = "closed"→FAILURE
$\Psi(e=0)(PAR)$	not getProperty(Door,"lock") = "locked"→SUCCESS
$\Psi(e=0)(PAR)$	not getProperty(Door,"status") = "guarded"→FAILURE
$\Psi(e=1)(PAR)$	finishedAction(self.id)→setProperty(Door,"lock") = "unlocked" AND SUCCESS
Θ	not canReach(Agent,Door)→Approach
Θ	Unlock

Table A.17: Action Number 45:Take

λ	Take
P	None
R	Guard
R	Key
$\Psi(e=1)(PAR)$	finishedAction(self.id)→SUCCESS
Θ	Take

Table A.18: Action Number 47:StealKey

λ	StealKey
P	Steal
R	Guard
R	Key
$\Psi(e=0)(PAR)$	not contains(Guard,Key)→FAILURE
$\Psi(e=0)(PAR)$	contains(Agent,Key)→SUCCESS
$\Psi(e=1)(PAR)$	finishedAction(self.id)→changeContents(Agent,Key) AND SUCCESS
Θ	not canReach(Agent,Guard)→Approach
Θ	not getProperty(Guard,"status") = "dazed"→Daze
Θ	And(Approach,Take)

Table A.19: Action Number 54:SoundAlarm

λ	SoundAlarm
P	Alarm
R	Guard
R	Guard
R	Guard
R	Guard
R	Alarm
R	Button
$\Psi(e=0)(PAR)$	not control(Button,alarm)→FAILURE
$\Psi(e=0)(PAR)$	getProperty(alarm,"status") = "active"→SUCCESS
$\Psi(e=1)(PAR)$	finishedAction(self.id)→setProperty(alarm,"status") = "active" AND SUCCESS
Θ	not canReach(Agent,Button)→Approach
Θ	And(Press,Join(Approach,Approach,Approach,Approach))

Table A.20: Action Number 55:Close

λ	Close
P	None
R	Door
$\Psi(e=0)(PAR)$	finishedAction(self.id)→setProperty(Door,"open") = "closed" AND SUCCESS
$\Psi(e=0)(PAR)$	not getProperty(Door,"open") = "open"→FAILURE
Θ	not canReach(Agent,Door)→Approach
Θ	Close

Table A.21: Action Number 62:DistractGuard

λ	DistractGuard
p	Distract
R	Guard
R	Door
R	Prisoner
R	Prisoner
R	WayPoint
R	WayPoint
$\Psi(e=0)$(PAR)	getProperty(door,"status") != "guarded← *SUCCESS*
$\Psi(e=1)$(PAR)	Finished→ setProperty(door,"status","Idle") and *SUCCESS*
Θ	not canReach(Agent,guard)→Approach(Agent,guard)
Θ	not canReach(prisoner1,waypoint1)→Approach(prisoner1,waypoint1)
Θ	not canReach(prisoner2,waypoint2)→Approach(prisoner2,waypoint2)
$\Theta(\phi=\emptyset)$	And(Hide,Hide,Draw)

Table A.22: The actions used to create a generalization set for the actions in the case study.

Action ID	Action Name	Parent Name
2	Connect	None
3	Attach	Connect
4	Fasten	Attach
6	Leave	None
7	Scat	Leave
8	Flee	Scat
9	Escape	Flee
11	Touch	None
13	Analyze	None
14	Check	Analyze
15	Watch	Check
17	Get	None
18	Capture	Get
22	Change	None
23	Desensitize	Change
25	Stun	Desensitize
27	Move	None
28	Transfer	Move
29	Convey	Transfer
30	Communicate	Convey
31	Request	Communicate
32	Ask	Request
33	Request	Ask
34	Order	Request
36	Travel	None
37	Come	Travel
39	Transfer	None
46	Steal	Take
48	Act	None
49	Interact	Act
50	Communicate	Interact
51	Inform	Communicate
52	Warn	Inform
53	Alarm	Warn
56	Make	None
57	Arouse	Make
58	Upset	Arouse
59	Embarrass	Upset
60	Confuse	Embarrass
61	Distract	Confuse

Appendix B: Keywords used in disambiguation test

B.1 Keyword List

Samples for our experimentation section were generated by sampling the list from either the definition or synonym section for each word in the list below.

B.1.1 Smartbody Definition Keywords

- SaccadeTalk : rapid,movement,eye,fixation
- SaccadeSpeak : rapid,movement,eye,fixation
- SaccadeListen : rapid,movement,eye,fixation
- Waggle : move,short,quick,side
- Speak : forming, nouns,manner,characteristic
- Shake : tremble,vibrate
- Nod : lower,raise,head,slightly,briefly
- PointAt : direct,attention,position,direction,finger
- Gaze : look,steadily,intently
- Step : lift,set,foot
- Walk : move,regular,slow,pace
- Jog : run,steady,gentle,pace
- PickUp : collect,something,left,elsewhere
- PutDown : move,place,particular,position
- Toss : throw,lightly,easily

- Touch : close,contact,come
- Jump : push,surface,air,muscles,legs,feet
- Run : move,speed,faster,walk,feet
- Wiggle : move,down,side,rapid,movements

B.1.2 Smartbody Synonym Keywords

- SaccadeTalk : jerk,jerking,jolt
- SaccadeSpeak : jerk,jerking,jolt
- Speak : address,communicate, converse, declaim, deliver
- Shake :agitate, appal, appall, brandish, churn
- Nod : agree, concur, dip
- PointAt : beckon, indicate
- Gaze : gix, google,peer,stare
- Waggle : faw, wobble,wag
- Step : march,pace,stride
- Walk : accompany,amble,hike
- Jog : canter, clip,trot
- PickUp : collect,fetch
- PutDown : lay,set
- Toss : chuck,fling,heave
- SaccadeListen : jerk,jerking,jolt

- Touch : meet,join,contact
- Jump : leap,spring,bound,hop
- Run : spring,race,dart,rush,dash,scurry
- Wiggle : jiggle,wriggle,twitch,shimmy

B.1.3 CMU Definition Keywords

- FowardJumps : move,body,upward,pushing
- Climb :move,up,feet,hands
- Hang:Attach place,held,support
- Swing:move,backward,side,hanging
- Sit:Position,bottom,resting,upright
- Lean:incline,bend,support
- Walk:move,legs,slower
- Jog :run,steady,gentle,pace
- Jump: move,body,upward,pushing
- Balance : weight,spread,equally,fall
- Punch : hit,hard,fist
- Fencing:activity,skill,fencing
- WashSelf:clean,water
- Dance:move,guide,music
- Pirouette:full turn,front

- ArabesqueDance: ballet,foot,arm
- FoldArms: lay,another
- CartWheel : athletic,movement,hand,ground
- JeteDance:springing,jump,ballet
- Dribble : fall,flow
- Shoot: eject,impel,release
- Run:move,legs,speed
- Kick:strike,foot,feet
- Screw:attach,fasten,twist
- UnScrew:loosen,remove
- Drink:take,liquid,mouth,swallow
- Laugh:show,happy,funny,smiling
- Box:hit,hand
- WashWindows:clean,water
- DirectTraffic :give,order,instruction
- Sweep:clean,brush,dirt,litter
- Wave:move,hand,greeting
- Point:direct,attention,position,direction
- JumpingJack : conditioning,exercise,legs,spread
- SideTwist:cause,rotate,turn
- flick:propel,sudden,movement,fingers

- BendOver:incline,body,downward
- Squat:crouch,sit,knees,thighs
- Stretch:straighten,extend,body
- Mop:clean,soak,wiping
- SwingLegs:move,backward,side,hanging
- LayUp:shot,near,basket,backboard
- Pass:move,direction
- Dive:plunge,head,water
- EgyptianWalk:dance
- Hobble:walk,awkward,pain
- Shake:move,up,side,rapid,forceful
- Pull:exert,force,
- Resist:withstand,action,effect
- Lead:cause,go,holding,moving,forward
- Catch:intercept,hold
- Wait: stay,delay,action,time
- SpinRope:turn,whirl
- GenieDance:move,guide,music
- Yawn:involuntarily,open,mouth,inhale,deeply
- Skip:move,lightly,stepping,hop,bounce
- Hammer:hit,beat
- Eat:Food,mouth,chew,swallow

B.1.4 CMU Synonym Keywords

- FowardJumps : bound, hop, leap, spring, vault
- Climb: clamber, scrabble, scramble, swarm
- Hang:clamber, scrabble, scramble, swarm
- Swing:sway, oscillate, vibrate
- Sit:set
- Lean:incline
- Walk:ambulate,step
- Jog:run,steady,gentle,pace
- Jump : bound, hop, leap, spring, vault
- Balance :counterpoise, equilibration, equilibrium, equipoise, poise
- Punch:bang,bash,clobber,whack
- SwordPlay:activity,skill,fencing
- WashSelf:bathe, lap, lave, lip, splash
- Dance:step,shake
- Pirouette:spin, reel, revolution, roll, rotation, twirl, wheel, whirl
- Arabesque: ballet,foot,arm
- FoldArms:crease,bend
- CartWheel: athletic,movement,hand,ground
- Jete:springing,jump,ballet
- Dribble:drip, drop, trickle

- Shoot:blast,loose
- Run:sprint, race, dart, rush, dash, hasten, hurry, scurry
- Kick:boot, punt, drop-kick
- Screw:fasten, secure, fix, attach
- UnScrew:loosen,remove
- Drink:swallow,gulp,quaff,guzzle
- Laugh:chuckle,chortle,guffaw,cackle
- Box:fight,spar
- WashWindows:bathe, lap, lave, lip, splash
- DirectTraffic:: instruct, tell, command, order
- Sweep:brush, clean, scrub, wipe, mop
- Wave:gesture, gesticulate, signal
- Point:direct,attention,poisition,direction
- JumpingJack:conditioning,exercise,legs,spread
- SideTwist:turn,twirl,spin,rotation
- flick:click, snap, flip, jerk
- BendOver:stoop, bow, crouch, hunch,
- Squat:crouch,hunker
- Stretch:extend
- Mop:wash, clean, wipe, swab
- SwingLegs:sway, oscillate, vibrate

- LayUp:shot,near,basket,backboard
- Pass:go, proceed, move, progress
- Dive:plunge, nosedive
- EgyptianWalk:dance
- Hobble:limp
- Shake:jiggle,joggle,agitate
- Pull:tug, haul, drag, draw, tow,
- Resist:withstand,combat,endure
- Lead:cause,go,holding,moving,foward
- Catch:seize, grab, snatch
- Wait:hold,stand,sit
- SpinRope:revolve, rotate, turn
- GenieDance:step,shake
- Yawn:involuntarily,open,mouth,inhale,deeply
- Skip:caper, prance, trip, dance, bound, bounce
- Hammer:beat, forge, shape, form, mold
- Eat:consume, devour, ingest

B.1.5 BLS Definition List

- Sleeping:rest,asleep
- Grooming:brushing,cleaning

- HealthRelatedSelfCare:provide,need
- PersonalActivities:do,thing
- Eating:chew,swallow
- Drinking:liquid,swallow
- HouseWork:cleaning,shopping,cooking
- FoodPreparation:food,selection,preparing
- FoodCleanUp:wiping,brushing,santizing
- LawnCare:provide,yard,look
- GardenCare:provide,plant,look
- HouseholdManagement:dealing,controlling,home
- InteriorMaintenance:preserving,caring,internal
- InteriorRepair:fix,mend,internal
- InteriorDecoration:adorning,internal
- ExteriorMaintenance:preserving,caring,external
- ExteriorRepair:fix,mend,external
- ExteriorDecoration:adorning,external
- Purchase:acquire,buy,pay
- GroceryShopping:purchasing,store,foodstuffs
- Banking:finance,services,business
- EducateChild:instruct,guide,youngster
- Work:activity,effort,achieve

- InterviewForJob:meet,consult,examine
- TravelToJob:journey,move,work
- AttendClass:present,lecture
- DoHomeWork:schoolwork,home
- DoResearch:investigation,study
- AttendMeeting:present,gathering
- AttendConference:present,lectures
- AttendTraining:present,trained
- Socialize:mix,others
- Relax:lessen,tense,loosen
- Communicate:share,information,exchange
- WatchTV:observe,television
- PlaySports:engage,activity,compete
- Exercise:activity,physical,effort,skill
- PaintArt:cover,decorate,depict,produce
- WatchSports:observe,exercise
- CallPhone:contact,telephone
- CallOut:yell,word
- WriteEmail:compose,document
- SendEmail:deliver,document

B.1.6 BLS Synonym List

- Sleeping:doze,nap,catnap
- Grooming:brush, comb,clean
- HealthRelatedSelfCare:tend,nurse,attend
- PersonalActivities:pursuit
- Eating:feed,snack,ingest,consume
- Drinking:guzzle,imbibe,sip,consume
- HouseWork:housecleaning,housekeeping,homemaking
- FoodPreparation:cleaning,cooking,sanitizing
- FoodCleanUp:wash,cleanse,wipe,sponge,scrub,mob
- LawnCare:tend,attend,minister
- GardenCare:tend,attend,minister
- HouseholdManagement:administration,managing,organization,running
- InteriorMaintenance:conserving,perpetuation,keeping
- InteriorRepair:restore,overhaul,service
- InteriorDecoration:ornament,bauble,trinket,spangle
- ExteriorMaintenance:conserving,perpetuation,keeping
- ExteriorRepair:restore,overhaul,service
- ExteriorDecoration:ornament,bauble,trinket,spangle
- Purchase:obtain,take,procure
- GroceryShopping:get,obtain,purchase

- Banking:investing
- EducateChild:tutor,coach,drill,train
- Work:labor,toil,slog,exertion
- InterviewForJob:meeting,discussion,conference,examine
- TravelToJob:voyage,explore,wander
- AttendClass:participate
- DoHomeWork:work
- DoResearch:analysis,fact-finding,fieldwork
- AttendMeeting:participate
- AttendConference:participate
- AttendTraining:participate
- Socialize:converse,interact,mingle
- Relax:slacken,ease
- Communicate:commune,interface,interact
- WatchTV:view,eye,gaze
- PlaySports:amuse,entertain,enjoy,leisure
- Exercise:workout,task
- PaintArt:coloring,portray,picture
- WatchSports:view,eye,gaze
- CallPhone:ring,buzz
- CallOut:cry,shout,hail

- WriteEmail:note,register
- SendEmail:dispatch,mail,consign,forward

B.2 Object Leafs for Experimentation Section

This is a list of items that we used to create the object hierarchy. The names provided below are the names of the graphical models that could be used in our environments.

- Human : person actor human virtual
- Lectern : desk
- StudentDesk : desk furniture
- Sofa : furniture sit
- Printer : machine paper ink
- Light : source heat
- ComputerServer : computer machine
- Microwave : machine food heat
- Sink : kitchen wash water
- EyeWash : wash water
- Toilet : water bathroom
- Pen : handheld ink write
- Mirror : reflect
- WhiteBoard : write erase notes
- Storage : store hold

- GarbageCan : trash
- WaterJug : water jug
- Locker : storage
- Plant : plant living organism leaves
- OfficeChair : chair furniture
- OfficeDesk : furniture
- TrashCan : garbage
- LabDesk : furniture
- RoundTable : table furniture surface
- Bookshelf : furniture
- CopyMachine : copy
- LectureDesk : furniture
- Server : person
- LoungeChair : furniture relax
- Laboratory : room place
- Office : room place
- LoungeArea : room place
- Restroom : room place
- Hallway : place
- ClassRoom : room place learn
- Weapon : fight attack

- Sawhorse : horse toy
- Shovel : equipment work dig tool
- BoltCutter : equiment work tool
- Hammer : tool equipment hand
- NeedlenosePliers: tool pliers
- Wrench : tool
- Syringe : medical
- Drum : instrument music
- Key : door lock
- Coffee : drink bean
- WaterFountain : drinking
- Bookcase : books furniture
- Fork : silverware food
- Spoon : silverware dinner food serve
- Knife : silverware dinner food serve
- Plate : dinnerware dinner food serve
- Sponge : cleaning device
- ClothNapkin : cleaning wipe
- PaperNapkin : cleaning wipe
- Mop : cleaning tool
- Broom : cleaning tool

- Register : machine electronic money
- Receipt : paper
- Bill : money currency
- Coin : money currency
- CreditCard : money bank
- Rag : cleaning device cloth
- Bowl : food server container
- Pot : food cook prepare
- Pan : cook food
- BarSegment : table furniture
- BarCorner : table furniture
- BoothTable : table seat furniture diner
- Booth : table seat furniture diner
- MidTable : table furniture
- Shelf : hold furniture
- CoffeeMug : cup mug drink
- Counter : table surface
- Grill : food cook heat
- BarStool : stool seat chair
- NapkinHolder : napkin container holding device
- KetchupBottle : condiment

- Clock : time
- CoffeeToGo : coffee drink
- DessertPlate : food
- DipBowl : food chips
- ExitSign : sign exit
- FireExtinguisher : fire
- Flute : instrument music
- FullPlate : food
- MeatTray : food
- OliveBowl : food
- Phone : talk machine device
- Pilsner : beer alcohol
- ShrimpPlatter : seafood food tray
- SnackTower : food
- SoupBowl : food
- SweetsTower : food
- KitchenSink : sink kitchen dishes water wash
- Cup : cup drink abstraction object
- FilingCabinet : file paper furniture
- Chair : furniture sit seat
- Fridge : furniture machine food cold

- CorkBoard : cork board notes
- Axe : tool
- Room : location general abstraction
- TurkeyLeg :
- Soup : liquid,food,meat

Appendix C: The data and Associated Synsets used for Automated Population

C.1 Objects used in our Analysis and their Associated Synsets

We describe the object data-sets that we used for each analysis, as well as their associated synsets. This does not include the generalizations, but only the graphical models and their synsets. We also only show unique objects, so that if there is a *object1* and *object2*, it will be displayed here as *object*.

C.1.1 ModelNet

- wine_glass:Synset('glass.n.02')
- windsheild:No Synset
- skeleton:Synset('skeletal_system.n.01')
- hooks:Synset('hook.n.05')
- ice:Synset('ice.n.01')
- railing:Synset('railing.n.01')
- chair:Synset('chair.n.01')
- track_light:Synset('light.n.02')
- trolley:Synset('streetcar.n.01')
- desk_drawer:Synset('drawer.n.01')
- photo_album:Synset('album.n.02')
- bicyclee:No Synset
- trousers:Synset('trouser.n.01')

- cable_box:Synset('cable_television.n.01')
- chess_set:No Synset
- wall_divider:Synset('wall.n.01')
- display_case:Synset('case.n.20')
- blanket:Synset('blanket.n.01')
- screwdriver:Synset('screwdriver.n.01')
- vase:Synset('vase.n.01')
- spoon:Synset('spoon.n.01')
- fan:Synset('fan.n.01')
- person_walking:Synset('person.n.01')
- fireplace:Synset('fireplace.n.01')
- coffee_table:Synset('table.n.02')
- tricycle:Synset('tricycle.n.01')
- button:Synset('button.n.01')
- metal_shutter:Synset('shutter.n.02')
- magnet:Synset('magnet.n.01')
- air_conditioner:No Synset
- candelabra:Synset('candelabrum.n.01')
- rack:Synset('rack.n.01')
- bulletin_board:Synset('display_panel.n.01')
- clothes:Synset('apparel.n.01')

- toy_car:Synset('toy.n.02')
- fence:Synset('fence.n.01')
- wine_rack:Synset('rack.n.01')
- sign:Synset('signboard.n.01')
- dirt_track:Synset('racetrack.n.01')
- clock:Synset('clock.n.01')
- sun:Synset('sun.n.01')
- stool:Synset('stool.n.01')
- suv:Synset('sport_utility.n.01')
- net:Synset('net.n.06')
- microchip:Synset('chip.n.07')
- gazebo:Synset('gazebo.n.01')
- bird:Synset('bird.n.01')
- tie_fighter:Synset('fighter.n.02')
- commercial:No Synset
- paper_roll:Synset('paper.n.01')
- bookcase:Synset('bookcase.n.01')
- baseball:Synset('baseball.n.02')
- sink:Synset('sink.n.01')
- box:Synset('box.n.01')
- luggage:Synset('baggage.n.01')

- leaves:Synset('leaf.n.01')
- soap_bottle:Synset('bottle.n.01')
- bunk_bed:Synset('bunk_bed.n.01')
- pillow:Synset('pillow.n.01')
- shovel:Synset('shovel.n.01')
- handrail:Synset('bannister.n.02')
- apple:Synset('apple.n.01')
- scales:Synset('scale.n.10')
- lumberr:No Synset
- cane:Synset('cane.n.01')
- lamppost:Synset('lamppost.n.01')
- duck:Synset('duck.n.01')
- faucet:Synset('faucet.n.01')
- glass_box:Synset('box.n.01')
- cloud:Synset('cloud.n.02')
- cd_disk:Synset('compact_disk.n.01')
- projector:Synset('projector.n.02')
- model_boat:Synset('model.n.04')
- game_table:Synset('table.n.02')
- coins:No Synset
- bush:Synset('shrub.n.01')

- camera:Synset('camera.n.01')
- iphone:No Synset
- balustrade:Synset('bannister.n.02')
- calculator:Synset('calculator.n.02')
- pencil:Synset('pencil.n.01')
- entrance:Synset('entrance.n.01')
- bidet:Synset('bidet.n.01')
- door:Synset('doorway.n.01')
- bottle:Synset('bottle.n.01')
- grand_piano:Synset('grand_piano.n.01')
- person_sitting:Synset('person.n.01')
- dishwasher:Synset('dishwasher.n.01')
- chocolate:Synset('chocolate.n.02')
- glass:Synset('glass.n.02')
- flag:Synset('flag.n.01')
- train:Synset('train.n.01')
- chess_piece:No Synset
- stick:Synset('stick.n.02')
- rabbit:Synset('rabbit.n.01')
- blinds:Synset('blind.n.03')
- grill:No Synset

- berth:Synset('berth.n.03')
- plastic_chair:Synset('chair.n.01')
- scaffolding:Synset('scaffolding.n.01')
- palm_tree:Synset('palm.n.03')
- tea_pot:Synset('teapot.n.01')
- tunnel:Synset('tunnel.n.01')
- car:Synset('car.n.01')
- cap:Synset('cap.n.04')
- roof:Synset('roof.n.01')
- cat:Synset('cat.n.01')
- soap_dish:Synset('dish.n.01')
- hot_air_balloon:Synset('balloon.n.01')
- can:Synset('can.n.01')
- glass_set:Synset('glass.n.07')
- shark:Synset('shark.n.01')
- fish_tank:Synset('tank.n.02')
- lock:Synset('lock.n.01')
- dolphin:Synset('dolphin.n.02')
- light_bulb:Synset('light_bulb.n.01')
- drawer:Synset('drawer.n.01')
- single_leg:Synset('peg.n.04')

- airplane:Synset('airplane.n.01')
- dress:Synset('dress.n.01')
- extractor_hood:Synset('hood.n.09')
- court:Synset('court.n.04')
- sailboat:Synset('sailboat.n.01')
- door_way:Synset('doorway.n.01')
- newtonian_toy:Synset('toy.n.02')
- machine:Synset('machine.n.01')
- lamp:Synset('lamp.n.02')
- refridgerator:No Synset
- sword:Synset('sword.n.01')
- coffee_machine:Synset('machine.n.01')
- gate:Synset('gate.n.01')
- furnace:Synset('furnace.n.01')
- flying_bird:Synset('bird.n.01')
- pencil_holder:Synset('holder.n.01')
- plant:Synset('plant.n.02')
- cupboard:Synset('cupboard.n.01')
- briefcase:Synset('briefcase.n.01')
- staircase:Synset('stairway.n.01')
- ladder:Synset('ladder.n.01')

- train_car:Synset('car.n.02')
- towel_rack:Synset('rack.n.01')
- notebook:Synset('notebook.n.01')
- arch:Synset('arch.n.04')
- plank:Synset('board.n.02')
- coat:Synset('coat.n.01')
- mezzanine:Synset('mezzanine.n.01')
- counter:Synset('counter.n.01')
- swimming_pool:Synset('pool.n.01')
- switch:Synset('switch.n.01')
- truck:Synset('truck.n.01')
- wrench:Synset('wrench.n.03')
- biplane:Synset('biplane.n.01')
- sprinker:No Synset
- wine:Synset('wine.n.01')
- eye_glasses:Synset('spectacles.n.01')
- deck:Synset('deck.n.04')
- conveyor_belt:No Synset
- plant_pot:Synset('pot.n.04')
- jacuzzi:No Synset
- motorcycle:Synset('motorcycle.n.01')

- oven:Synset('oven.n.01')
- keyboard:Synset('keyboard.n.01')
- butcher_knife:Synset('knife.n.01')
- monitor:Synset('monitor.n.04')
- conference_table:Synset('table.n.02')
- chips:Synset('chip.n.04')
- backpack:Synset('backpack.n.01')
- frying_pan:Synset('pan.n.01')
- mattress:Synset('mattress.n.01')
- forecourt:Synset('forecourt.n.01')
- dolly:Synset('dolly.n.02')
- orange:Synset('orange.n.01')
- conical:No Synset
- carton:Synset('carton.n.02')
- grille_door:Synset('grille.n.02')
- dam:Synset('dam.n.01')
- dock:Synset('pier.n.01')
- snake:Synset('snake.n.01')
- hairdryer:No Synset
- decorative_platter:Synset('platter.n.01')
- headboard:Synset('headboard.n.01')

- easel:Synset('easel.n.01')
- door_knob:Synset('doorknob.n.01')
- side_table:Synset('table.n.02')
- ray_roll:No Synset
- slide:Synset('slide.n.04')
- insrument_panel:Synset('control_panel.n.01')
- embankment:Synset('embankment.n.01')
- rock:Synset('rock.n.02')
- tray:Synset('tray.n.01')
- trailer:Synset('trailer.n.04')
- field_grass:Synset('grass.n.01')
- hanger:Synset('hanger.n.02')
- map:Synset('map.n.01')
- fish:Synset('fish.n.01')
- torch:Synset('torch.n.01')
- spider:Synset('spider.n.01')
- water_fountain:Synset('fountain.n.01')
- fruit_bowl:Synset('bowl.n.01')
- hay_roll:Synset('hay.n.01')
- satellite_dish:Synset('dish.n.05')
- harp:Synset('harp.n.01')

- album:Synset('album.n.01')
- swingset:No Synset
- flower:Synset('flower.n.01')
- jeep:Synset('jeep.n.01')
- school_desk:Synset('desk.n.01')
- military_tank:Synset('tank.n.01')
- hangers:Synset('hanger.n.02')
- umbrella:Synset('umbrella.n.01')
- cables:Synset('cable.n.02')
- cart:Synset('handcart.n.01')
- traffic_light:Synset('light.n.02')
- geographic_map:Synset('map.n.01')
- toy:Synset('plaything.n.01')
- flower_box:Synset('box.n.01')
- one_peak_tent:Synset('tent.n.01')
- filing_cabinet:Synset('cabinet.n.01')
- coffee_pot:Synset('coffeepot.n.01')
- shelves:Synset('shelf.n.01')
- headphones:Synset('earphone.n.01')
- safety_belt:Synset('belt.n.02')
- mailbox:Synset('mailbox.n.01')

- round:Synset('round.n.01')
- router:Synset('router.n.02')
- castle:Synset('palace.n.01')
- wooden_plank:Synset('board.n.02')
- blind:Synset('blind.n.03')
- cheese:Synset('cheese.n.01')
- bouquet:Synset('bouquet.n.01')
- crib:Synset('crib.n.01')
- ring:Synset('ring.n.08')
- razor:Synset('razor.n.01')
- horse:Synset('horse.n.01')
- basketball:Synset('basketball.n.02')
- lego:Synset('lego.n.01')
- elevator:Synset('elevator.n.01')
- shower_head:Synset('showerhead.n.01')
- potted_plant:Synset('plant.n.02')
- streetlight:Synset('streetlight.n.01')
- banana:Synset('banana.n.01')
- cabin:Synset('cabin.n.02')
- gear:Synset('gear.n.01')
- cuddly_toy:Synset('plaything.n.01')

- shorts:Synset('short_pants.n.01')
- shelf:Synset('shelf.n.01')
- ceiling_fan:Synset('fan.n.01')
- green_screen:Synset('screen.n.01')
- hut:Synset('hovel.n.01')
- rifle:Synset('rifle.n.01')
- wardrobe:Synset('wardrobe.n.01')
- flowers:Synset('flower.n.01')
- stairs:Synset('stairs.n.01')
- ipad:No Synset
- television:Synset('television_receiver.n.01')
- room_divider:Synset('partition.n.01')
- iceberg:Synset('iceberg.n.01')
- race_car:Synset('car.n.01')
- tree:Synset('tree.n.01')
- bed:Synset('bed.n.01')
- bee:Synset('bee.n.01')
- shower:Synset('shower.n.01')
- trex:No Synset
- large_sail_boat:Synset('boat.n.01')
- light:Synset('light.n.02')

- sculpture:Synset('sculpture.n.01')
- bridge:Synset('bridge.n.01')
- display_stand:Synset('rack.n.05')
- french_bread:Synset('bread.n.01')
- multiple_peak_tent:Synset('tent.n.01')
- one_story_home:Synset('dwelling.n.01')
- sconce:Synset('sconce.n.04')
- space_shuttle:No Synset
- soap_dispenser:Synset('dispenser.n.01')
- cistern:Synset('cistern.n.03')
- lid:Synset('lid.n.02')
- mountain:Synset('mountain.n.01')
- microphone:Synset('microphone.n.01')
- escalator:Synset('escalator.n.02')
- dishes:Synset('dish.n.01')
- washbasin:Synset('washbasin.n.01')
- chessboard:Synset('chessboard.n.01')
- projector_screen:Synset('screen.n.01')
- couch:Synset('sofa.n.01')
- bagel:Synset('bagel.n.01')
- equipment:Synset('equipment.n.01')

- chest:Synset('chest.n.02')
- folding_door:Synset('doorway.n.01')
- exercise_machine:No Synset
- soap:Synset('soap.n.01')
- paintbrush:Synset('paintbrush.n.01')
- butterfly:Synset('butterfly.n.01')
- window_frame:Synset('window.n.01')
- printer:Synset('printer.n.03')
- vending_machine:Synset('machine.n.01')
- pail:Synset('bucket.n.01')
- track:Synset('path.n.04')
- blackboard:Synset('blackboard.n.01')
- eggs:Synset('egg.n.02')
- basket:Synset('basket.n.01')
- place_mat:Synset('mat.n.01')
- sheets:Synset('sheet.n.03')
- sedan:Synset('sedan.n.01')
- medal:No Synset
- webcam:Synset('webcam.n.01')
- stereo:Synset('stereo.n.01')
- mineral:Synset('mineral.n.01')

- wooden_planks:Synset('board.n.02')
- urn:Synset('urn.n.01')
- napkin:Synset('napkin.n.01')
- dog:Synset('dog.n.01')
- face:Synset('face.n.01')
- pipe:Synset('pipe.n.01')
- ingots:Synset('ingot.n.01')
- barren:Synset('barren.n.01')
- barrel:Synset('barrel.n.02')
- stones:Synset('rock.n.02')
- candleholder:No Synset
- walking:No Synset
- stove:Synset('stove.n.01')
- crate:Synset('crate.n.01')
- chandelier:Synset('chandelier.n.01')
- radio:Synset('radio_receiver.n.01')
- shoe:Synset('shoe.n.01')
- ottoman:Synset('ottoman.n.03')
- blender:Synset('blender.n.01')
- tire:Synset('tire.n.01')
- jar:Synset('jar.n.01')

- basketball_hoop:Synset('hoop.n.02')
- tree_sculpture:Synset('sculpture.n.01')
- tape:Synset('tape.n.01')
- piano:Synset('piano.n.01')
- finger:Synset('finger.n.01')
- plate:Synset('plate.n.04')
- handle:Synset('handle.n.01')
- watch:Synset('watch.n.01')
- billiard_table:Synset('table.n.02')
- cushion:Synset('cushion.n.03')
- hedge:Synset('hedge.n.01')
- handgun:Synset('pistol.n.01')
- chimney:Synset('chimney.n.01')
- ping_pong_table:Synset('table.n.02')
- skull:Synset('skull.n.01')
- ashtray:Synset('ashtray.n.01')
- floor_lamp:Synset('lamp.n.01')
- two_story_home:Synset('dwelling.n.01')
- bag:Synset('bag.n.01')
- microscope:Synset('microscope.n.01')
- paper:Synset('paper.n.01')

- cell_phone:Synset('cellular_telephone.n.01')
- workbench:Synset('workbench.n.01')
- pizza_box:Synset('box.n.01')
- frame:No Synset
- pedestal:Synset('pedestal.n.03')
- cologne:Synset('cologne.n.02')
- computer:Synset('computer.n.01')
- hammer:Synset('hammer.n.02')
- exit_sign:Synset('signboard.n.01')
- hourglass:Synset('hourglass.n.01')
- loudspeaker:Synset('loudspeaker.n.01')
- wire:Synset('wire.n.02')
- closet:Synset('wardrobe.n.01')
- aqueduct:Synset('aqueduct.n.01')
- feline:Synset('feline.n.01')
- wooden_pillar:Synset('pillar.n.03')
- mug:Synset('mug.n.04')
- skyscraper:Synset('skyscraper.n.01')
- music_keyboard:Synset('keyboard.n.01')
- tent:Synset('tent.n.01')
- soccer_ball:Synset('ball.n.01')

- dome:Synset('dome.n.04')
- projection_screen:Synset('screen.n.01')
- guitar:Synset('guitar.n.01')
- aircraft:Synset('aircraft.n.01')
- wall_stand:Synset('stand.n.04')
- storage_rack:Synset('rack.n.01')
- drum:Synset('drum.n.04')
- cow:Synset('cow.n.01')
- trash_can:Synset('rubbish.n.01')
- drawer_knob:Synset('knob.n.02')
- vault:Synset('vault.n.01')
- canopy:Synset('canopy.n.03')
- lemon:Synset('lemon.n.01')
- litter_bin:Synset('litterbin.n.01')
- jersey:Synset('jersey.n.03')
- vent:Synset('vent.n.01')
- loaf:Synset('loaf_of_bread.n.01')
- light_switch:Synset('switch.n.01')
- hot_dogs:Synset('hotdog.n.02')
- cleaner:Synset('cleansing_agent.n.01')
- fighter_jet:Synset('jet.n.01')

- armchair:Synset('armchair.n.01')
- pot:Synset('pot.n.01')
- jeans:Synset('jean.n.01')
- plastic_box:Synset('box.n.01')
- cd:Synset('compact_disk.n.01')
- filing_shelves:Synset('shelf.n.01')
- pole:Synset('pole.n.01')
- stalacite:No Synset
- table:Synset('table.n.02')
- bench:Synset('bench.n.01')
- boat:Synset('boat.n.01')
- belt:Synset('belt.n.02')
- lighthouse:Synset('beacon.n.03')
- comb:Synset('comb.n.01')
- motorbike:Synset('minibike.n.01')
- lamp_shade:Synset('lampshade.n.01')
- stamp:Synset('revenue_stamp.n.01')
- window:Synset('window.n.01')
- american_flag:Synset('flag.n.01')
- submarine:Synset('submarine.n.01')
- eraser:Synset('eraser.n.01')

- curtain:Synset('curtain.n.01')
- brick:Synset('brick.n.01')
- fire_alarm:Synset('alarm.n.02')
- flashligh:No Synset
- control_tower:Synset('tower.n.01')
- air_vent:Synset('vent.n.01')
- multi_fuselage:Synset('fuselage.n.01')
- arcade_machine:Synset('machine.n.01')
- x_wing:No Synset
- flower_with_stem:Synset('flower.n.01')
- bulb:Synset('light_bulb.n.01')
- glider:Synset('glider.n.01')
- case:Synset('case.n.05')
- phone_handle:Synset('telephone.n.01')
- tank:Synset('tank.n.01')
- door_lock:Synset('lock.n.01')
- kitchen_items:Synset('item.n.03')
- ceiling_lamp:Synset('lamp.n.01')
- monster_truck:Synset('truck.n.01')
- rope:Synset('rope.n.01')
- flower_pot:Synset('pot.n.01')

- bathtub:Synset('bathtub.n.01')
- spotlight:Synset('spotlight.n.02')
- bin:Synset('bin.n.01')
- pyramid:Synset('pyramid.n.03')
- pickup:Synset('pickup.n.01')
- sticker:No Synset
- coffee_cup:Synset('cup.n.01')
- purse:Synset('bag.n.04')
- telephone:Synset('telephone.n.01')
- cone:No Synset
- violin:Synset('violin.n.01')
- mouse:Synset('mouse.n.04')
- fire_escape:No Synset
- rectangular_table:Synset('table.n.02')
- helmet:Synset('helmet.n.02')
- shopping_cart:Synset('handcart.n.01')
- balloon:Synset('balloon.n.01')
- toaster:Synset('toaster.n.02')
- bowl:Synset('bowl.n.03')
- lava:Synset('lava.n.01')
- flying_saucer:No Synset

- speaker:Synset('loudspeaker.n.01')
- dome_church:Synset('church.n.02')
- balcony:Synset('balcony.n.01')
- pan:Synset('pan.n.01')
- wheel:Synset('wheel.n.01')
- satellite:Synset('satellite.n.01')
- ball:Synset('ball.n.01')
- rail:Synset('rail.n.04')
- ice_rink:Synset('ice_rink.n.01')
- fruit:Synset('fruit.n.01')
- grandstand:No Synset
- person_skiing:Synset('person.n.01')
- wooden_toy:Synset('plaything.n.01')
- slot_machine:Synset('slot.n.07')
- ice_cream:Synset('ice_cream.n.01')
- file_box:Synset('box.n.01')
- catus:No Synset
- river_water:Synset('river.n.01')
- drain:Synset('drain.n.03')
- charger:Synset('charger.n.02')
- hatchery:Synset('hatchery.n.01')

- person:Synset('person.n.02')
- tombstone:Synset('gravestone.n.01')
- usb_drive:Synset('drive.n.10')
- awning:Synset('awning.n.01')
- cash_register:Synset('cash_register.n.01')
- antique_car:Synset('car.n.01')
- snowman:Synset('snowman.n.01')
- pool_ball:Synset('ball.n.01')
- coaster:Synset('coaster.n.03')
- kettle:Synset('kettle.n.01')
- money:No Synset
- laptop:Synset('laptop.n.01')
- desk:Synset('desk.n.01')
- cutting_board:Synset('board.n.03')
- footbridge:Synset('footbridge.n.01')
- stretcher:Synset('stretcher.n.03')
- human:Synset('homo.n.02')
- sailboat_with_oars:Synset('sailboat.n.01')
- dish_rack:Synset('rack.n.01')
- ham:Synset('ham.n.01')
- cup:Synset('cup.n.01')

- remote_control:Synset('remote_control.n.01')
- bell:Synset('bell.n.01')
- desktop:No Synset
- pallet:Synset('pallet.n.02')
- board:Synset('board.n.02')
- tissue_box:Synset('box.n.01')
- bread:Synset('bread.n.01')
- hat:Synset('hat.n.01')
- knob:Synset('knob.n.02')
- deck_chair:Synset('chair.n.01')
- radiator:Synset('radiator.n.02')
- ruler:Synset('rule.n.12')
- stall_shower:Synset('shower.n.01')
- baggage_cart:Synset('handcart.n.01')
- lockheed_airplane:Synset('airplane.n.01')
- security_camera:Synset('camera.n.01')
- furniture:Synset('furniture.n.01')
- dirigible:Synset('airship.n.01')
- tree_trunk:Synset('trunk.n.01')
- desk_chiar:Synset('desk.n.01')
- towel:Synset('towel.n.01')

- onion:Synset('onion.n.02')
- semi:Synset('trailer_truck.n.01')
- sofa:Synset('sofa.n.01')
- pillar:Synset('column.n.06')
- xbox:No Synset
- window_shelf:Synset('shelf.n.01')
- voting_booth:Synset('booth.n.02')
- tower:Synset('tower.n.01')
- eletric_box:Synset('box.n.01')
- sneaker:No Synset
- stacked_chairs:Synset('chair.n.01')
- doll:Synset('doll.n.01')
- foot_rest:Synset('footstool.n.01')
- church:Synset('church.n.02')
- night_stand:Synset('stand.n.04')
- garage_door:Synset('doorway.n.01')
- sponge:Synset('sponge.n.04')
- palm:Synset('palm.n.03')
- mirror:Synset('mirror.n.01')
- billboard:Synset('billboard.n.01')
- dining_chair:Synset('chair.n.01')

- dvd_player:No Synset
- fire_extinguisher:Synset('fire_extinguisher.n.01')
- scale:Synset('scale.n.07')
- shrub:Synset('shrub.n.01')
- teddy_bear:Synset('teddy.n.01')
- tube:Synset('tube.n.01')
- steering_wheel:Synset('steering_wheel.n.01')
- microwave:Synset('microwave.n.02')
- pen:Synset('pen.n.01')
- cradle:Synset('cradle.n.01')
- saucepan:Synset('saucepan.n.01')
- knife:Synset('knife.n.01')
- paper_cup:Synset('cup.n.01')
- drawer_handle:Synset('handle.n.01')
- corn:Synset('corn.n.03')
- trampoline:Synset('trampoline.n.01')
- skate_board:Synset('skateboard.n.01')
- rug:Synset('rug.n.01')
- topiary:Synset('topiary.n.01')
- headstone:Synset('gravestone.n.01')
- stage:Synset('stage.n.03')

- trophy:No Synset
- stone:Synset('stone.n.02')
- placard:No Synset
- necklace:Synset('necklace.n.01')
- altarpiece:Synset('altarpiece.n.01')
- newspapers:No Synset
- whiteboard:No Synset
- magazine:Synset('magazine.n.02')
- road:Synset('road.n.01')
- rocky_mountain:Synset('mountain.n.01')
- heater:Synset('heater.n.01')
- dishcloth:Synset('dishrag.n.01')
- brush:Synset('brush.n.02')
- cubicle:Synset('booth.n.02')
- van:Synset('van.n.05')
- vegetables:Synset('vegetable.n.01')
- column:Synset('column.n.06')
- synthesizer:Synset('synthesizer.n.02')
- windmill:Synset('windmill.n.02')
- hen:Synset('hen.n.02')
- valley:Synset('valley.n.01')

- globe:Synset('globe.n.03')
- fork:Synset('fork.n.01')
- building:Synset('building.n.01')
- head:Synset('head.n.01')
- person_skating:Synset('person.n.01')
- jug:Synset('jug.n.01')
- bus:Synset('bus.n.01')
- pitcher:Synset('pitcher.n.02')
- cabinet:Synset('cabinet.n.01')
- door_frame:Synset('doorframe.n.01')
- cloth:Synset('fabric.n.01')
- bar_of_soap:Synset('soap.n.01')
- helicopter:Synset('helicopter.n.01')
- bookshelf:Synset('bookshelf.n.01')
- crane:Synset('crane.n.04')
- ear:Synset('ear.n.01')
- wine_bottle:Synset('bottle.n.01')
- pavilion:Synset('pavilion.n.01')
- range_hood:Synset('hood.n.06')
- bucket:Synset('bucket.n.01')
- pool_table:Synset('table.n.02')

- suitcase:Synset('bag.n.06')
- cake:Synset('cake.n.03')
- grass:Synset('grass.n.01')
- alarm:Synset('alarm.n.02')
- window_seat:Synset('seat.n.03')
- toilet:Synset('toilet.n.02')
- sticks:Synset('stick.n.06')
- traffic_cone:Synset('cone.n.01')
- pig:Synset('hog.n.03')
- stealth_bomber:Synset('bomber.n.01')
- curb:Synset('curb.n.01')
- ship:Synset('ship.n.01')
- sports_car:Synset('car.n.01')
- bread_roll:Synset('bun.n.01')
- cream:Synset('cream.n.03')
- street_sign:Synset('signboard.n.01')
- computer_monitor:Synset('monitor.n.04')
- antenna:Synset('antenna.n.01')
- boot:Synset('boot.n.01')
- book:Synset('book.n.01')
- person_standing:Synset('person.n.02')

- branch:Synset('branch.n.02')
- papers:Synset('paper.n.01')
- saucer:Synset('saucer.n.02')
- roll:Synset('bun.n.01')
- dresser:Synset('chest_of_drawers.n.01')
- tv_stand:Synset('rack.n.05')
- door_handle:Synset('doorknob.n.01')
- fluorescent_tube:Synset('fluorescent.n.01')
- coat_hanger:Synset('hanger.n.02')
- football:Synset('football.n.02')
- swivel_chair:Synset('chair.n.01')
- console_table:Synset('console_table.n.01')
- podium:Synset('dais.n.01')
- pool:Synset('pool.n.01')
- cliff:Synset('cliff.n.01')
- pottery:Synset('pottery.n.01')
- mantel:Synset('mantel.n.01')
- bow_window:Synset('window.n.01')
- dvd:Synset('videodisk.n.01')
- standing_bird:Synset('bird.n.01')
- water_heater:Synset('heater.n.01')

- mask:Synset('mask.n.04')
- desk_lamp:Synset('lamp.n.02')
- toilet_paper_holder:Synset('holder.n.01')
- jacket:Synset('jacket.n.01')
- mast:Synset('mast.n.01')
- weights:Synset('weight.n.02')
- washing_machine:Synset('machine.n.01')
- rocking_chair:Synset('chair.n.01')

C.1.2 CalTech 101

- stop_sign:Synset('stop.n.05')
- lotus:Synset('lotus.n.03')
- laptop:Synset('laptop.n.01')
- crocodile:Synset('crocodile.n.01')
- chandelier:Synset('chandelier.n.01')
- faces_easy:Synset('face.n.01')
- chair:Synset('chair.n.01')
- joshua_tree:Synset('tree.n.01')
- sunflower:Synset('sunflower.n.01')
- accordion:Synset('accordion.n.01')
- mandolin:Synset('mandolin.n.01')

- scorpion:Synset('scorpion.n.03')
- ewer:Synset('pitcher.n.02')
- schooner:Synset('schooner.n.02')
- dragonfly:Synset('dragonfly.n.01')
- stegosaurus:Synset('stegosaur.n.01')
- menorah:Synset('menorah.n.02')
- cannon:Synset('cannon.n.04')
- watch:Synset('watch.n.01')
- sea_horse:Synset('seahorse.n.02')
- llama:Synset('llama.n.01')
- leopards:Synset('leopard.n.02')
- tick:Synset('tick.n.02')
- flamingo:Synset('flamingo.n.01')
- stapler:Synset('stapler.n.01')
- electric_guitar:Synset('guitar.n.01')
- emu:Synset('emu.n.02')
- saxophone:Synset('sax.n.02')
- brontosaurus:Synset('apatosaur.n.01')
- butterfly:Synset('butterfly.n.01')
- lobster:Synset('lobster.n.02')
- umbrella:Synset('umbrella.n.01')

- gramophone:Synset('gramophone.n.01')
- panda:Synset('giant_panda.n.01')
- car_side:Synset('car.n.01')
- pigeon:Synset('pigeon.n.01')
- revolver:Synset('revolver.n.01')
- rhino:Synset('rhinoceros.n.01')
- buddha:Synset('buddha.n.01')
- flamingo_head:Synset('flamingo.n.01')
- soccer_ball:Synset('ball.n.01')
- bass:Synset('freshwater_bass.n.01')
- trilobite:Synset('trilobite.n.01')
- kangaroo:Synset('kangaroo.n.01')
- gerenuk:Synset('gerenuk.n.01')
- pagoda:Synset('pagoda.n.01')
- ibis:Synset('ibis.n.01')
- dalmatian:Synset('dalmatian.n.02')
- cup:Synset('cup.n.01')
- faces:Synset('face.n.01')
- octopus:Synset('octopus.n.02')
- motorbikes:Synset('minibike.n.01')
- headphone:Synset('earphone.n.01')

- pyramid:Synset('pyramid.n.03')
- scissors:Synset('scissors.n.01')
- yin_yang:No Synset
- wild_cat:Synset('wildcat.n.03')
- camera:Synset('camera.n.01')
- crab:Synset('crab.n.01')
- helicopter:Synset('helicopter.n.01')
- cougar_body:Synset('cougar.n.01')
- starfish:Synset('starfish.n.01')
- grand_piano:Synset('grand_piano.n.01')
- ceiling_fan:Synset('fan.n.01')
- brain:Synset('brain.n.01')
- metronome:Synset('metronome.n.01')
- minaret:Synset('minaret.n.01')
- strawberry:Synset('strawberry.n.01')
- cougar_face:Synset('cougar.n.01')
- ferry:Synset('ferry.n.01')
- snoopy:No Synset
- anchor:Synset('anchor.n.01')
- inline_skate:Synset('skate.n.01')
- nautilus:Synset('chambered_nautilus.n.01')

- platypus:Synset('platypus.n.01')
- dolphin:Synset('dolphin.n.02')
- hedgehog:Synset('hedgehog.n.02')
- rooster:Synset('cock.n.04')
- ant:Synset('ant.n.01')
- euphonium:Synset('euphonium.n.01')
- garfield:Synset('garfield.n.01')
- water_lilly:No Synset
- beaver:Synset('beaver.n.07')
- binocular:No Synset
- dollar_bill:No Synset
- lamp:Synset('lamp.n.02')
- elephant:Synset('elephant.n.01')
- hawksbill:Synset('hawksbill_turtle.n.01')
- pizza:Synset('pizza.n.01')
- bonsai:Synset('bonsai.n.01')
- windsor_chair:Synset('chair.n.01')
- background_google:Synset('background.n.07')
- airplanes:Synset('airplane.n.01')
- cellphone:Synset('cellular_telephone.n.01')
- wheelchair:Synset('wheelchair.n.01')

- ketch:Synset('ketch.n.01')
- wrench:Synset('wrench.n.03')
- okapi:Synset('okapi.n.01')
- mayfly:Synset('mayfly.n.01')
- barrel:Synset('barrel.n.02')
- crayfish:Synset('crayfish.n.03')
- crocodile_head:Synset('crocodile.n.01')
- person_walking:Synset('person.n.01')

C.1.3 Office

- heater:Synset('heater.n.01')
- chair_guest:Synset('chair.n.01')
- light_switch:Synset('switch.n.01')
- book_mesh:Synset('book.n.02')
- paper_bin:Synset('bin.n.01')
- curtain_rod:Synset('curtain.n.01')
- part_door_glass:Synset('door.n.01')
- wall_frame:Synset('frame.n.10')
- fileholder:No Synset
- ceiling_tiles_edge:Synset('tile.n.02')
- cup:Synset('cup.n.01')

- radio:Synset('radio.n.03')
- ceiling_light:Synset('light.n.02')
- pen:Synset('pen.n.01')
- sm_god_ray:Synset('beam.n.04')
- wall_hanger:Synset('hanger.n.02')
- chair:Synset('chair.n.01')
- calendar:No Synset
- paper_box:Synset('box.n.01')
- fanblade:No Synset
- wall_award:No Synset
- fuseline:No Synset
- antenna:Synset('antenna.n.01')
- fuse_box:Synset('fuse.n.01')
- blinds_top:Synset('blind.n.03')
- paper_pile:Synset('paper.n.01')
- smoke_detector:Synset('detector.n.01')
- manilla_papers:Synset('paper.n.01')
- part_door:Synset('door.n.01')
- light_switch_wire:Synset('wire.n.01')
- window:Synset('window.n.07')
- camera:Synset('camera.n.01')

- folder_holder:Synset('folder.n.02')
- oscillo:No Synset
- papers:Synset('paper.n.01')
- flagpole:Synset('flagpole.n.02')
- curtain:Synset('curtain.n.01')
- blinds:Synset('blind.n.03')
- electro:No Synset
- hat:Synset('hat.n.01')
- fireex:No Synset
- wall_map:Synset('map.n.01')
- map:Synset('map.n.01')
- wires_desk:Synset('wire.n.01')
- wall_clock:Synset('clock.n.01')
- ceiling_fan:Synset('fan.n.01')
- fusebox:No Synset
- corkboard:Synset('corkboard.n.01')
- cart:Synset('handcart.n.01')
- wall_clock_second_hand:Synset('clock.n.01')
- phone:Synset('telephone.n.01')
- binder:Synset('binder.n.03')
- notebook:Synset('notebook.n.01')

- fan:Synset('fan.n.01')
- part_wall:Synset('wall.n.01')
- desk:Synset('desk.n.01')
- ceiling_vent:Synset('vent.n.01')
- decal_plane:Synset('airplane.n.01')
- reel:Synset('reel.n.01')
- desklamp:No Synset
- monitor:Synset('monitor.n.04')
- box:Synset('box.n.01')
- reeltoreel:No Synset
- window_blinds:Synset('blind.n.03')
- paper_pad:Synset('pad.n.01')
- trash_can:Synset('toilet.n.01')
- ceiling_fan_blade:Synset('blade.n.08')
- ashtray:Synset('ashtray.n.01')
- nameplate:Synset('nameplate.n.01')
- shelf:Synset('shelf.n.01')
- trim_straight:Synset('trimming.n.02')
- cigarettes:Synset('cigarette.n.01')
- blinds_only:Synset('blind.n.03')
- folder:Synset('folder.n.02')

- wall_speaker:Synset('loudspeaker.n.01')
- confolder:No Synset
- ceiling_tiles:Synset('tile.n.02')
- filecab:No Synset
- book:Synset('book.n.02')
- planter:Synset('planter.n.03')
- fuse_line_clamp:Synset('clamp.n.01')
- person_walking:Synset('person.n.01')

C.1.4 Pub

- m_tavern_inside_cann:No Synset
- m_tavern_inside_pillars_module:Synset('pillar.n.03')
- m_winecast:No Synset
- m_tavern_inside_woodrack:No Synset
- straw:Synset('straw.n.01')
- m_tavern_inside_candle_holders:Synset('holder.n.01')
- m_tavern_inside_floor_brick:Synset('floor.n.04')
- m_tavern_inside_ladder:Synset('ladder.n.01')
- m_tavern_inside_window:Synset('window.n.01')
- m_tavern_inside_cupboard:Synset('cupboard.n.01')
- m_cupboard:Synset('cupboard.n.01')

- m_tavern_inside_basket:Synset('basket.n.01')
- m_tavern_inside_candles:Synset('candle.n.01')
- m_tavern_inside_square_table:Synset('table.n.02')
- m_high_stool:Synset('stool.n.01')
- m_tavern_inside_fabric:Synset('fabric.n.01')
- m_tavern_inside_straw:Synset('straw.n.01')
- basic_material:Synset('material.n.01')
- m_tavern_inside_bag:Synset('bag.n.04')
- m_tavern_inside_kiwi_fruit:Synset('kiwi.n.03')
- m_tavern_inside_bench:Synset('bench.n.01')
- m_tavern_inside_bar_counter:Synset('counter.n.01')
- m_tavern_inside_apple:Synset('apple.n.02')
- m_tavern_sunshine:Synset('sunlight.n.01')
- m_tavern_inside_windows:Synset('window.n.01')
- m_tavern_inside_firewood:Synset('firewood.n.01')
- m_tavern_inside_bowl:Synset('bowl.n.03')
- m_tavern_inside_roof:Synset('roof.n.01')
- m_tavern_inside_bread:Synset('bread.n.01')
- m_tavern_inside_wincast:No Synset
- m_tavern_inside_spoon:Synset('spoon.n.01')
- m_tavern_inside_candle_sword:Synset('sword.n.01')

- m_tavern_inside_round_table:Synset('table.n.02')
- m_tavern_inside_plate:Synset('plate.n.04')
- m_tavern_inside_chain:Synset('chain.n.05')
- m_tavern_inside_stove:Synset('stove.n.01')
- m_tavern_inside_wall_brick:Synset('wall.n.01')
- m_iron_pan:Synset('pan.n.01')
- m_tavern_inside_step:Synset('step.n.04')
- m_tavern_inside_square_stool:Synset('stool.n.01')
- m_tavern_inside_candle_stick:Synset('candle.n.01')
- m_tavern_inside_winepot:No Synset
- m_tavern_inside_orange:Synset('orange.n.03')
- m_tavern_inside_pillars:Synset('pillar.n.03')
- m_tavern_inside:Synset('inside.n.02')
- m_tavern_inside_old_chair:Synset('chair.n.01')
- m_tavern_inside_door:Synset('doorway.n.01')
- m_iron_drum:Synset('drum.n.04')
- m_tavern_inside_grill:Synset('grill.n.02')
- m_tavern_inside_gavelock:No Synset
- m_tavern_inside_floor_wood:Synset('floor.n.04')
- m_tavern_inside_door_rock:Synset('door.n.01')

C.2 Actions and Frames used in our Analysis

We also show the synsets for actions used in our analysis. If the action has a FrameNet frame connected to it, the FrameNet Frame is also connected to it.

C.2.1 SmartBody

- saccade_speak:Synset('speak.v.03'):Statement
- point_at:Synset('indicate.v.02'):Telling
- run:Synset('run.v.34'):Motion
- nod:Synset('nod.v.02'):Body_movement
- toss:Synset('flip.v.06'):Cause_motion
- shake:Synset('shake.v.09'):Gesture
- pick_up:Synset('pick.v.02'):Gathering_up
- put_down:Synset('put.v.01'):Placing
- walk:Synset('walk.v.04'):Path_shape
- jump:Synset('jump.v.08'):No Frame
- wiggle:Synset('jiggle.v.01'):Body_movement
- step:Synset('step.v.07'):Self_motion
- gaze:Synset('gaze.v.01'):Perception_active
- jog:Synset('jog.v.03'):Self_motion
- touch:Synset('touch.v.01'):No Frame
- waggle:Synset('wag.v.01'):Body_movement
- saccade_talk:Synset('talk.v.01'):Chatting

- saccade_listen:Synset('listen.v.02'):Perception_active
- speak:Synset('talk.v.02'):Chatting

C.2.2 CMU

- wash_windows:Synset('wash.v.03'):Grooming
- fence:Synset('fence.v.03'):Hostile_encounter
- point:Synset('indicate.v.02'):Telling
- dance:Synset('dance.v.02'):No Frame
- lean:Synset('lean.v.01'):Posture
- resist:Synset('resist.v.04'):Quarreling
- shake:Synset('shake.v.09'):Gesture
- swing_legs:Synset('swing.v.01'):Cause_to_move_in_place
- sweep:Synset('sweep.v.03'):Placing
- walk:Synset('walk.v.01'):Self_motion
- jump:Synset('jump.v.01'):Self_motion
- catch:Synset('catch.v.04'):Manipulation
- side_twist:Synset('twist.v.03'):No Frame
- laugh:Synset('laugh.v.01'):Make_noise
- pass:Synset('pass.v.20'):Cause_motion
- egyptian_walk:Synset('walk.v.10'):Motion
- hammer:Synset('hammer.v.01'):Cause_harm

- screw:Synset('screw.v.04'):Closure
- stretch_rope:Synset('unfold.v.03'):No Frame
- squat:Synset('squat.v.01'):Posture
- lead:Synset('lead.v.04'):Cotheme
- sit:Synset('sit_down.v.01'):Posture
- stretch:Synset('stretch.v.02'):Body_movement
- pirouette_dance:Synset('pirouette.v.01'):No Frame
- eat:Synset('eat.v.01'):Ingestion
- jumping_jack:Synset('jump.v.08'):No Frame
- skip:Synset('hop.v.01'):Self_motion
- bend_over:Synset('crouch.v.01'):Posture
- kick:Synset('kick.v.01'):No Frame
- fold_arms:Synset('fold.v.01'):Reshaping
- box:Synset('box.v.02'):Cause_impact
- mop:Synset('wipe_up.v.01'):No Frame
- direct_traffic:Synset('direct.v.01'):Request
- run:Synset('run.v.01'):Self_motion
- wash_self:Synset('wash.v.02'):Grooming
- cart_wheel:Synset('cartwheel.v.01'):Moving_in_place
- dive:Synset('dive.v.02'):Path_shape
- drink:Synset('drink.v.01'):Ingestion

- hang:Synset('hang.v.01'):No Frame
- punch:Synset('punch.v.01'):Cause_harm
- jete:Synset('dance.v.02'):No Frame
- wave:Synset('beckon.v.01'):Gesture
- un_screw:Synset('unscrew.v.02'):Closure
- yawn:Synset('yawn.v.01'):Body_movement
- penalty_kick:Synset('kick.v.07'):Finish_competition
- jog:Synset('trot.v.01'):Self_motion
- hobble:Synset('limp.v.01'):Self_motion
- foward_jumps:Synset('jump.v.01'):Self_motion
- arabesque_dance:Synset('dance.v.02'):No Frame
- dribble:Synset('dribble.v.03'):No Frame
- pull:Synset('pull.v.04'):Manipulation
- shoot:Synset('shoot.v.16'):Finish_competition
- lay_up:No Synset:No Frame
- genie_dance:Synset('dance.v.02'):No Frame
- spin_rope:Synset('whirl.v.02'):Cause_to_move_in_place
- paint_wall:Synset('paint.v.02'):Filling
- wait:Synset('wait.v.01'):No Frame
- swing:Synset('swing.v.03'):Path_shape
- climb:Synset('climb.v.01'):Self_motion
- balance:Synset('poise.v.04'):No Frame

C.2.3 The ICS Action Database

- walking:Synset('walk.v.04'):Path_shape
- showing_hand:Synset('show.v.04'):No Frame
- jump:Synset('jump.v.08'):No Frame
- lying_on_back:Synset('lie_down.v.01'):Change_posture
- look_away:Synset('look.v.01'):Perception_active
- standing:Synset('stand.v.10'):Placing
- sit_down:Synset('sit.v.01'):Posture
- stand_still:Synset('stand.v.03'):No Frame
- on_four_limbs:No Synset:No Frame
- sitting_on_chair:Synset('sit.v.01'):Posture
- stand_up:Synset('stand.v.10'):Placing
- fold_arms:Synset('fold.v.01'):Reshaping
- sitting:Synset('sit_down.v.01'):Posture
- get_up:Synset('up.v.01'):No Frame
- look_down:Synset('look.v.01'):Perception_active
- reach_out:Synset('reach.v.01'):Arriving
- running:Synset('run.v.01'):Self_motion
- sitting_on_floor:Synset('sit.v.01'):Posture
- turn:Synset('turn.v.01'):No Frame
- look_up:Synset('look.v.01'):Perception_active

- lie_down:Synset('lie_down.v.01'):Change_posture
- lying_on_side:Synset('lie_down.v.01'):Change_posture
- lying_on_face:Synset('lie.v.02'):Posture
- lying:Synset('lie_down.v.01'):Change_posture
- keep_down:Synset('keep.v.01'):Activity_ongoing

C.2.4 The Human Motion Database

- chew:Synset('chew.v.01'):Grinding
- golf:Synset('golf.v.01'):Competition
- sword_exercise:Synset('practice.v.01'):Transitive_action
- fence:Synset('fence.v.03'):Hostile_encounter
- walk:Synset('walk.v.01'):Self_motion
- jump:Synset('jump.v.08'):No Frame
- pour:Synset('decant.v.01'):No Frame
- laugh:Synset('laugh.v.01'):Make_noise
- shoot_gun:Synset('gun.v.01'):Use_firearm
- run:Synset('run.v.01'):Self_motion
- turn:Synset('turn.v.01'):No Frame
- ride_bike:Synset('ride.v.10'):Operate_vehicle
- swing_baseball:Synset('swing.v.03'):Path_shape
- draw_sword:Synset('withdraw.v.09'):Removing

- sit:Synset('sit.v.01'):Posture
- dribble:Synset('dribble.v.03'):No Frame
- stand:Synset('stand.v.10'):Placing
- pushup:No Synset:No Frame
- sword:No Synset:No Frame
- pullup:No Synset:No Frame
- smile:Synset('smile.v.01'):Making_faces
- shake_hands:Synset('shake.v.09'):Gesture
- shoot_ball:Synset('shoot.v.08'):No Frame
- kick:Synset('kick.v.03'):Cause_harm
- somersault:Synset('somersault.v.01'):No Frame
- drink:Synset('drink.v.01'):Ingestion
- flic_flac:No Synset:No Frame
- hug:Synset('embrace.v.02'):Manipulation
- hit:Synset('hit.v.03'):Cause_impact
- dive:Synset('dive.v.02'):Path_shape
- kick_ball:Synset('kick.v.07'):Finish_competition
- cart_wheel:Synset('cartwheel.v.01'):Moving_in_place
- punch:Synset('punch.v.01'):Cause_harm
- wave:Synset('beckon.v.01'):Gesture
- hand_stand:Synset('stand.v.01'):Posture

- kiss:Synset('snog.v.01'):No Frame
- catch:Synset('catch.v.01'):Experiencer_obj
- eat:Synset('eat.v.02'):Ingestion
- throw:Synset('hurl.v.03'):Text_creation
- climb_stairs:Synset('climb.v.01'):Self_motion
- ride_horse:Synset('ride.v.01'):Motion
- fall_floor:Synset('fall.v.23'):No Frame
- brush_hair:Synset('brush.v.01'):Placing
- clap:Synset('clap.v.06'):Body_movement
- smoke:Synset('smoke.v.01'):Ingest_substance
- pick:Synset('pick.v.02'):Gathering_up
- push:Synset('push.v.01'):Manipulation
- climb:Synset('climb.v.01'):Self_motion
- talk:Synset('talk.v.02'):Chatting
- situp:No Synset:No Frame
- shoot_bow:Synset('shoot.v.01'):Hit_target

C.2.5 The American Time Use Survey

- food_clean_up:Synset('clean.v.01'):No Frame
- exterior_maintenance:No Synset:No Frame
- grocery_shopping:Synset('shop.v.01'):Getting

- sleeping:Synset('sleep.v.01'):Sleep
- exterior_decoration:No Synset:No Frame
- send_email:Synset('e-mail.v.01'):Contacting
- health_related_self_care:Synset('care.v.02'):Assistance
- do_research:Synset('research.v.01'):Scrutiny
- play_sports:Synset('play.v.05'):No Frame
- watch_sports:Synset('watch.v.04'):Seeking
- personal_activities:No Synset:No Frame
- attend_training:Synset('attend.v.01'):No Frame
- call_out:Synset('call.v.26'):Discussion
- household_management:No Synset:No Frame
- interior_decoration:No Synset:No Frame
- interior_maintenance:No Synset:No Frame
- house_work:Synset('work.v.01'):No Frame
- educate_child:Synset('educate.v.01'):No Frame
- travel_to_job:Synset('travel.v.05'):Travel
- garden_care:Synset('garden.v.01'):Assistance
- write_email:Synset('write.v.05'):Contacting
- food_preparation:No Synset:No Frame
- exercise:Synset('exercise.v.04'):No Frame
- eating:Synset('eat.v.02'):Ingestion

- relax:Synset('relax.v.05'):Transitive_action
- interior_repair:Synset('repair.v.01'):No Frame
- communicate:Synset('communicate.v.05'):No Frame
- socialize:Synset('socialize.v.01'):No Frame
- paint_art:Synset('paint.v.01'):No Frame
- do_home_work:Synset('solve.v.01'):Coming_to_believe
- attend_class:Synset('attend.v.01'):No Frame
- drinking:Synset('drink.v.01'):Ingestion
- watch_t_v:Synset('watch.v.01'):Perception_active
- purchase:Synset('buy.v.01'):Commerce_buy
- interview_for_job:Synset('interview.v.03'):Chatting
- call_phone:Synset('call.v.09'):Discussion
- attend_conference:Synset('attend.v.01'):No Frame
- work:Synset('work.v.08'):No Frame
- exterior_repair:Synset('repair.v.01'):No Frame
- banking:Synset('bank.v.03'):No Frame
- lawn_care:Synset('care.v.02'):Assistance
- attend_meeting:Synset('attend.v.01'):No Frame
- grooming:Synset('groom.v.03'):Grooming

C.3 The Designation of Functional Elements for Each Frame

This section shows the designation of each functional element for each action that contains a frame in our data-set. The designations correspond to operational information FE_R, Non-Interaction Functional Information FE_{NI}, and data that does not have a designation FE_U.

C.3.1 ICT SmartBody

point_at:

- FE_R = Addressee, Speaker, Place
- FE_S = Means, Manner, Time, Message
- FE_{Un} = Medium, Iteration, Topic, Descriptor, Epistemic_stance

run:

- FE_R = Goal, Source, Theme, Path, Place
- FE_S = Degree, Purpose, Containing_event, Manner, Time, Duration, Speed
- FE_{Un} = Depictive, Distance, Direction, Area, Frequency, Iteration, Carrier, Path_shape, Result

nod:

- FE_R = Goal, Body_part, Source, Path, Agent, Addressee, Place
- FE_S = Coordinated_event, Degree, Cognate_event, Purpose, Manner, Time, Duration, Message
- FE_{Un} = Area, Internal_cause, Sub-region, Re_encoding, Depictive, Result, External_cause

toss:

- FE_R = Goal, Agent, Source, Theme, Instrument, Path, Cause, Subregion, Place

- FE_S = Degree, Means, Manner, Time
- FE_{Un} = Depictive, Distance, Handle, Area, Explanation, Initial_State, Result

shake:

- FE_R = Body_part, Indicated_entity, Addressee, Communicator, Place
- FE_S = Means, Manner, Time, Message
- FE_{Un} = Instrument

pick_up:

- FE_R = Agent, Instrument, Individuals, Path, Place
- FE_S = Containing_event, Means, Manner, Time
- FE_{Un} = Co_participant, Source, Aggregate, Frequency, Goal, Purpose

put_down:

- FE_R = Goal, Agent, Source, Theme, Path, Cause, Place
- FE_S = Degree, Means, Reason, Manner, Time, Duration, Speed, Purpose
- FE_{Un} = Depictive, Distance, Area, Beneficiary, Cotheme, Result

walk:

- FE_R = Goal, Source, Path, Road, Place
- FE_S = Degree, Purpose, Means, Manner, Time, Speed
- FE_{Un} = Depictive, Distance, Direction, Area, Path_shape, Result

saccade_speak:

- FE_R = Medium, Addressee, Speaker, Place

- FE_S = Event_description, Containing_event, Means, Manner, Time, Message
- FE_{Un} = Depictive, Group, Degree, Particular_iteration, Iteration, Internal_cause, Topic, Epistemic_stance, Frequency, Occasion

wiggle:

- FE_R = Goal, Body_part, Source, Path, Agent, Addressee, Place
- FE_S = Coordinated_event, Degree, Cognate_event, Purpose, Manner, Time, Duration, Message
- FE_{Un} = Area, Internal_cause, Sub-region, Re_encoding, Depictive, Result, External_cause

step:

- FE_R = Goal, Cotheme, Area, Source, Direction, Self_mover, Path, Place
- FE_S = Coordinated_event, Result, Means, Reason, Purpose, Distance, Time, Depictive, Manner, Duration, Speed
- FE_{Un} = Internal_cause, Path_shape, Concessive, External_cause

gaze:

- FE_R = Expected_entity, Perceiver_agentive, Body_part, Phenomenon, Place
- FE_S = Means, Manner, Time, Duration, Purpose
- FE_{Un} = Depictive, Direction, State, Location_of_perceiver, Obscuring_medium, Ground

jog:

- FE_R = Goal, Cotheme, Area, Source, Direction, Self_mover, Path, Place
- FE_S = Coordinated_event, Result, Means, Reason, Purpose, Distance, Time, Depictive, Manner, Duration, Speed

- FE_{Un} = Internal_cause, Path_shape, Concessive, External_cause

waggle:

- FE_R = Goal, Body_part, Source, Path, Agent, Addressee, Place
- FE_S = Coordinated_event, Degree, Cognate_event, Purpose, Manner, Time, Duration, Message
- FE_{Un} = Area, Internal_cause, Sub-region, Re_encoding, Depictive, Result, External_cause

saccade_talk:

- FE_R = Interlocutor_2, Interlocutors, Interlocutor_1, Place
- FE_S = Means, Purpose, Time, Duration, Manner
- FE_{Un} = Depictive, Language, Topic, Means_of_communication

saccade_listen:

- FE_R = Expected_entity, Perceiver_agentive, Body_part, Phenomenon, Place
- FE_S = Means, Manner, Time, Duration, Purpose
- FE_{Un} = Depictive, Direction, State, Location_of_perceiver, Obscuring_medium, Ground

speak:

- FE_R = Interlocutor_2, Interlocutors, Interlocutor_1, Place
- FE_S = Means, Purpose, Time, Duration, Manner
- FE_{Un} = Depictive, Language, Topic, Means_of_communication

C.3.2 CMU

wash_windows:

- FE_R = Patient, Body_part, Agent, Instrument, Place
- FE_S = Means, Time, Duration, Purpose, Manner
- FE_{Un} = Medium, Sub_region, Frequency, Result

fence:

- FE_R = Side_2, Side_1, Instrument, Sides, Place
- FE_S = Degree, Means, Reason, Purpose, Time, Duration, Manner
- FE_{Un} = Depictive, Result, Particular_iteration, Internal_cause, Issue

point:

- FE_R = Addressee, Speaker, Place
- FE_S = Means, Manner, Time, Message
- FE_{Un} = Medium, Iteration, Topic, Descriptor, Epistemic_stance

skip:

- FE_R = Goal, Cotheme, Area, Source, Direction, Self_mover, Path, Place
- FE_S = Coordinated_event, Result, Means, Reason, Purpose, Distance, Time, Depictive, Manner, Duration, Speed
- FE_{Un} = Internal_cause, Path_shape, Concessive, External_cause

shake:

- FE_R = Body_part, Indicated_entity, Addressee, Communicator, Place

- FE_S = Means, Manner, Time, Message
- FE_{Un} = Instrument

swing_legs:

- FE_R = Agent, Instrument, Theme, Place
- FE_S = Degree, Means, Reason, Manner, Time, Duration, Purpose
- FE_{Un} = Bodypart_of_agent, Direction, Result, Angle, Periodicity, Locus, Fixed_location, Cause

sweep:

- FE_R = Goal, Agent, Source, Theme, Path, Cause, Place
- FE_S = Degree, Means, Reason, Manner, Time, Duration, Speed, Purpose
- FE_{Un} = Depictive, Distance, Area, Beneficiary, Cotheme, Result

walk:

- FE_R = Goal, Cotheme, Area, Source, Direction, Self_mover, Path, Place
- FE_S = Coordinated_event, Result, Means, Reason, Purpose, Distance, Time, Depictive, Manner, Duration, Speed
- FE_{Un} = Internal_cause, Path_shape, Concessive, External_cause

jump:

- FE_R = Goal, Cotheme, Area, Source, Direction, Self_mover, Path, Place
- FE_S = Coordinated_event, Result, Means, Reason, Purpose, Distance, Time, Depictive, Manner, Duration, Speed
- FE_{Un} = Internal_cause, Path_shape, Concessive, External_cause

cook:

- FE_R = Container, Heating_Instrument, Cook, Recipient, Place
- FE_S = Degree, Means, Purpose, Time, Manner
- FE_{Un} = Ingredients, Produced_food

laugh:

- FE_R = Sound_source, Path, Addressee, Subregion, Place
- FE_S = Degree, Noisy_event, Reason, Manner, Time
- FE_{Un} = Sound, Depictive, Location_of_source, Particular_iteration, Internal_cause, Iterations, Circumstances

pass:

- FE_R = Goal, Agent, Source, Theme, Instrument, Path, Cause, Subregion, Place
- FE_S = Degree, Means, Manner, Time
- FE_{Un} = Depictive, Distance, Handle, Area, Explanation, Initial_State, Result

egyptian_walk:

- FE_R = Goal, Source, Theme, Path, Place
- FE_S = Degree, Purpose, Containing_event, Manner, Time, Duration, Speed
- FE_{Un} = Depictive, Distance, Direction, Area, Frequency, Iteration, Carrier, Path_shape, Result

hammer:

- FE_R = Body_part, Instrument, Victim, Agent, Place

- FE_S = Degree, Means, Reason, Purpose, Time, Containing_event, Manner
- FE_{Un} = Cause, Result, Explanation, Circumstances, Period_of_iterations, Subregion_bodypart, Concessive, Re_encoding, Depictive, Frequency, Iterations, Duration, Patricular_iteration

screw:

- FE_R = Agent, Beneficiary, Instrument, Place
- FE_S = Degree, Means, Manner, Time
- FE_{Un} = Depictive, Manipulator, Enclosed_region, Containing_object, Fastener, Container_portal, Circumstances, Result

squat:

- FE_R = Agent, Location
- FE_S = Degree, Time, Duration, Purpose, Manner
- FE_{Un} = Depictive, Point_of_contact, Dependent_state

lead:

- FE_R = Goal, Cotheme, Area, Source, Theme, Direction, Path, Road, Place
- FE_S = Means, Explanation, Purpose, Result, Distance, Time, Depictive, Manner, Duration, Speed, Event
- FE_{Un} = Handle, Mode_of_transportation, Following_distance

sit:

- FE_R = Agent, Location
- FE_S = Degree, Time, Duration, Purpose, Manner
- FE_{Un} = Depictive, Point_of_contact, Dependent_state

stretch:

- FE_R = Goal, Body_part, Source, Path, Agent, Addressee, Place
- FE_S = Coordinated_event, Degree, Cognate_event, Purpose, Manner, Time, Duration, Message
- FE_{Un} = Area, Internal_cause, Sub-region, Re_encoding, Depictive, Result, External_cause

lean:

- FE_R = Agent, Location
- FE_S = Degree, Time, Duration, Purpose, Manner
- FE_{Un} = Depictive, Point_of_contact, Dependent_state

bend_over:

- FE_R = Agent, Location
- FE_S = Degree, Time, Duration, Purpose, Manner
- FE_{Un} = Depictive, Point_of_contact, Dependent_state

fold_arms:

- FE_R = Undergoer, Instrument, Cause, Deformer, Subregion, Place
- FE_S = Degree, Means, Reason, Manner, Time, Purpose
- FE_{Un} = Result, Resistant_surface, Locus, Iterations

shoot:

- FE_R = Competitor, Prize, Place
- FE_S = Means, Manner, Time

- FE_{Un} = Particular_iteration, Venue, Rank, Competitors, Score, Competition, Margin, Opponent

direct_traffic:

- FE_R = Medium, Addressee, Speaker
- FE_S = Containing_event, Means, Manner, Time, Message
- FE_{Un} = Depictive, Beneficiary, Iteration, Topic

run:

- FE_R = Goal, Cotheme, Area, Source, Direction, Self_mover, Path, Place
- FE_S = Coordinated_event, Result, Means, Reason, Purpose, Distance, Time, Depictive, Manner, Duration, Speed
- FE_{Un} = Internal_cause, Path_shape, Concessive, External_cause

wash_self:

- FE_R = Patient, Body_part, Agent, Instrument, Place
- FE_S = Means, Time, Duration, Purpose, Manner
- FE_{Un} = Medium, Sub_region, Frequency, Result

dive:

- FE_R = Goal, Source, Path, Road, Place
- FE_S = Degree, Purpose, Means, Manner, Time, Speed
- FE_{Un} = Depictive, Distance, Direction, Area, Path_shape, Result

drink:

- FE_R = Source, Ingestibles, Instrument, Ingestor, Place
- FE_S = Degree, Means, Manner, Time, Duration, Purpose
- FE_{Un} = None

cart_wheel:

- FE_R = Theme, Place
- FE_S = Manner, Time, Purpose
- FE_{Un} = Depictive, Direction, Angle, Fixed_location, Periodicity, Path_shape, Cause, Result

punch:

- FE_R = Body_part, Instrument, Victim, Agent, Place
- FE_S = Degree, Means, Reason, Purpose, Time, Containing_event, Manner
- FE_{Un} = Cause, Result, Explanation, Circumstances, Period_of_iterations, Subregion_bodypart, Concessive, Re_encoding, Depictive, Frequency, Iterations, Duration, Patricular_iteration

wave:

- FE_R = Body_part, Indicated_entity, Addressee, Communicator, Place
- FE_S = Means, Manner, Time, Message
- FE_{Un} = Instrument

un_screw:

- FE_R = Agent, Beneficiary, Instrument, Place
- FE_S = Degree, Means, Manner, Time
- FE_{Un} = Depictive, Manipulator, Enclosed_region, Containing_object, Fastener, Container_portal, Circumstances, Result

yawn:

- FE_R = Goal, Body_part, Source, Path, Agent, Addressee, Place
- FE_S = Coordinated_event, Degree, Cognate_event, Purpose, Manner, Time, Duration, Message
- FE_{Un} = Area, Internal_cause, Sub-region, Re_encoding, Depictive, Result, External_cause

penalty_kick:

- FE_R = Competitor, Prize, Place
- FE_S = Means, Manner, Time
- FE_{Un} = Particular_iteration, Venue, Rank, Competitors, Score, Competition, Margin, Opponent

jog:

- FE_R = Goal, Cotheme, Area, Source, Direction, Self_mover, Path, Place
- FE_S = Coordinated_event, Result, Means, Reason, Purpose, Distance, Time, Depictive, Manner, Duration, Speed
- FE_{Un} = Internal_cause, Path_shape, Concessive, External_cause

catch:

- FE_R = Agent, Bodypart_of_agent, Instrument, Entity, Place
- FE_S = Means, Reason, Manner, Time, Duration, Purpose
- FE_{Un} = Depictive, Particular_iteration, Locus, Result

foward_jumps:

- FE_R = Goal, Cotheme, Area, Source, Direction, Self_mover, Path, Place

- FE_S = Coordinated_event, Result, Means, Reason, Purpose, Distance, Time, Depictive, Manner, Duration, Speed
- FE_{Un} = Internal_cause, Path_shape, Concessive, External_cause

eat:

- FE_R = Source, Ingestibles, Instrument, Ingestor, Place
- FE_S = Degree, Means, Manner, Time, Duration, Purpose
- FE_{Un} = None

pull:

- FE_R = Agent, Bodypart_of_agent, Instrument, Entity, Place
- FE_S = Means, Reason, Manner, Time, Duration, Purpose
- FE_{Un} = Depictive, Particular_iteration, Locus, Result

box:

- FE_R = Impactee, Impactor, Agent, Instrument, Impactors, Subregion, Place
- FE_S = Means, Manner, Time, Purpose, Speed
- FE_{Un} = Result, Force, Cause, Period_of_iterations

resist:

- FE_R = Medium, Arguers, Arguer2, Arguer1, Place
- FE_S = Means, Manner, Time, Duration, Purpose
- FE_{Un} = Depictive, Amount_of_discussion, Frequency, Issue

spin_rope:

- FE_R = Agent, Instrument, Theme, Place
- FE_S = Degree, Means, Reason, Manner, Time, Duration, Purpose
- FE_{Un} = Bodypart_of_agent, Direction, Result, Angle, Periodicity, Locus, Fixed_location, Cause

paint_wall:

- FE_R = Goal, Agent, Source, Theme, Instrument, Path, Subregion, Place
- FE_S = Degree, Means, Purpose, Reason, Manner, Time
- FE_{Un} = Depictive, Cause, Result

hobble:

- FE_R = Goal, Cotheme, Area, Source, Direction, Self_mover, Path, Place
- FE_S = Coordinated_event, Result, Means, Reason, Purpose, Distance, Time, Depictive, Manner, Duration, Speed
- FE_{Un} = Internal_cause, Path_shape, Concessive, External_cause

swing:

- FE_R = Goal, Source, Path, Road, Place
- FE_S = Degree, Purpose, Means, Manner, Time, Speed
- FE_{Un} = Depictive, Distance, Direction, Area, Path_shape, Result

climb:

- FE_R = Goal, Cotheme, Area, Source, Direction, Self_mover, Path, Place
- FE_S = Coordinated_event, Result, Means, Reason, Purpose, Distance, Time, Depictive, Manner, Duration, Speed
- FE_{Un} = Internal_cause, Path_shape, Concessive, External_cause

C.3.3 ICS Action Database

standing:

- FE_R = Goal, Agent, Source, Theme, Path, Cause, Place
- FE_S = Degree, Means, Reason, Manner, Time, Duration, Speed, Purpose
- FE_{Un} = Depictive, Distance, Area, Beneficiary, Cotheme, Result

fold_arms:

- FE_R = Undergoer, Instrument, Cause, Deformer, Subregion, Place
- FE_S = Degree, Means, Reason, Manner, Time, Purpose
- FE_{Un} = Result, Resistant_surface, Locus, Iterations

walking:

- FE_R = Goal, Source, Path, Road, Place
- FE_S = Degree, Purpose, Means, Manner, Time, Speed
- FE_{Un} = Depictive, Distance, Direction, Area, Path_shape, Result

sit_down:

- FE_R = Agent, Location
- FE_S = Degree, Time, Duration, Purpose, Manner
- FE_{Un} = Depictive, Point_of_contact, Dependent_state

sitting:

- FE_R = Agent, Location
- FE_S = Degree, Time, Duration, Purpose, Manner

- FE_{Un} = Depictive, Point_of_contact, Dependent_state

reach_out:

- FE_R = Goal, Source, Theme, Place, Path
- FE_S = Event_description, Means, Purpose, Manner, Time
- FE_{Un} = Depictive, Degree, Cotheme, Period_of_iterations, Frequency, Mode_of_transportation, Re_encoding, Goal_conditions, Circumstances

sitting_on_chair:

- FE_R = Agent, Location
- FE_S = Degree, Time, Duration, Purpose, Manner
- FE_{Un} = Depictive, Point_of_contact, Dependent_state

sitting_on_floor:

- FE_R = Agent, Location
- FE_S = Degree, Time, Duration, Purpose, Manner
- FE_{Un} = Depictive, Point_of_contact, Dependent_state

stand_up:

- FE_R = Goal, Agent, Source, Theme, Path, Cause, Place
- FE_S = Degree, Means, Reason, Manner, Time, Duration, Speed, Purpose
- FE_{Un} = Depictive, Distance, Area, Beneficiary, Cotheme, Result

lying_on_side:

- FE_R = Goal, Location, Source, Protagonist, Path

- FE_S = Purpose, Means, Manner, Time
- FE_{Un} = Depictive, Distance, Direction, Point_of_contact, Result

running:

- FE_R = Goal, Cotheme, Area, Source, Direction, Self_mover, Path, Place
- FE_S = Coordinated_event, Result, Means, Reason, Purpose, Distance, Time, Depictive, Manner, Duration, Speed
- FE_{Un} = Internal_cause, Path_shape, Concessive, External_cause

look_up:

- FE_R = Expected_entity, Perceiver_agentive, Body_part, Phenomenon, Place
- FE_S = Means, Manner, Time, Duration, Purpose
- FE_{Un} = Depictive, Direction, State, Location_of_perceiver, Obscuring_medium, Ground

lie_down:

- FE_R = Goal, Location, Source, Protagonist, Path
- FE_S = Purpose, Means, Manner, Time
- FE_{Un} = Depictive, Distance, Direction, Point_of_contact, Result

lying_on_back:

- FE_R = Goal, Location, Source, Protagonist, Path
- FE_S = Purpose, Means, Manner, Time
- FE_{Un} = Depictive, Distance, Direction, Point_of_contact, Result

look_down:

- FE_R = Expected_entity, Perceiver_agentive, Body_part, Phenomenon, Place
- FE_S = Means, Manner, Time, Duration, Purpose
- FE_{Un} = Depictive, Direction, State, Location_of_perceiver, Obscuring_medium, Ground

lying_on_face:

- FE_R = Agent, Location
- FE_S = Degree, Time, Duration, Purpose, Manner
- FE_{Un} = Depictive, Point_of_contact, Dependent_state

look_away:

- FE_R = Expected_entity, Perceiver_agentive, Body_part, Phenomenon, Place
- FE_S = Means, Manner, Time, Duration, Purpose
- FE_{Un} = Depictive, Direction, State, Location_of_perceiver, Obscuring_medium, Ground

lying:

- FE_R = Goal, Location, Source, Protagonist, Path
- FE_S = Purpose, Means, Manner, Time
- FE_{Un} = Depictive, Distance, Direction, Point_of_contact, Result

keep_down:

- FE_R = Agent, Place
- FE_S = Event_description, Means, Explanation, Subevent, Purpose, Time, Manner
- FE_{Un} = Depictive, Activity, Duration, Circumstances

C.3.4 Human Motion Database

chew:

- FE_R = Goal, Undergoer, Instrument, Grinding_cause, Place
- FE_S = Means, Manner, Time, Duration, Purpose
- FE_{Un} = Result, Grinder, Locus

golf:

- FE_R = Place
- FE_S = Degree, Means, Time, Duration, Purpose, Manner
- FE_{Un} = Prize, Participant_2, Participant_1, Venue, Rank, Participants, Score, Competition, Frequency

laugh:

- FE_R = Sound_source, Path, Addressee, Subregion, Place
- FE_S = Degree, Noisy_event, Reason, Manner, Time
- FE_{Un} = Sound, Depictive, Location_of_source, Particular_iteration, Internal_cause, Iterations, Circumstances

fence:

- FE_R = Side_2, Side_1, Instrument, Sides, Place
- FE_S = Degree, Means, Reason, Purpose, Time, Duration, Manner
- FE_{Un} = Depictive, Result, Particular_iteration, Internal_cause, Issue

walk:

- FE_R = Goal, Cotheme, Area, Source, Direction, Self_mover, Path, Place

- FE_S = Coordinated_event, Result, Means, Reason, Purpose, Distance, Time, Depictive, Manner, Duration, Speed
- FE_{Un} = Internal_cause, Path_shape, Concessive, External_cause

shoot_bow:

- FE_R = Target, Agent, Instrument, Subregion, Place
- FE_S = Means, Purpose, Time, Manner
- FE_{Un} = None

sword_exercise:

- FE_R = Depictive, Patient, Agent, Place
- FE_S = Means, Time, Cause, Event, Manner
- FE_{Un} = Result

shoot_gun:

- FE_R = Goal, Agent, Source, Path, Firearm, Place
- FE_S = Means, Purpose, Time, Manner
- FE_{Un} = Depictive, Area, Iteration

run:

- FE_R = Goal, Cotheme, Area, Source, Direction, Self_mover, Path, Place
- FE_S = Coordinated_event, Result, Means, Reason, Purpose, Distance, Time, Depictive, Manner, Duration, Speed
- FE_{Un} = Internal_cause, Path_shape, Concessive, External_cause

ride_bike:

- FE_R = Goal, Source, Driver, Path, Place
- FE_S = Degree, Means, Purpose, Time, Manner, Duration, Speed, Event
- FE_{Un} = Cotheme, Area, Vehicle, Result, Explanation, Circumstances, Distance, Route, Depictive, Particular_iteration, Frequency

swing_baseball:

- FE_R = Goal, Source, Path, Road, Place
- FE_S = Degree, Purpose, Means, Manner, Time, Speed
- FE_{Un} = Depictive, Distance, Direction, Area, Path_shape, Result

draw_sword:

- FE_R = Goal, Theme, Cause, Path, Source, Agent, Place
- FE_S = Degree, Means, Time, Subevent, Manner
- FE_{Un} = Cotheme, Instrument, Vehicle, Direction, Result, Explanation, Purpose, Circumstances, Distance, Depictive, Particular_iteration

sit:

- FE_R = Agent, Location
- FE_S = Degree, Time, Duration, Purpose, Manner
- FE_{Un} = Depictive, Point_of_contact, Dependent_state

throw:

- FE_R = Honoree, Author, Text, Instrument, Addressee, Place

- FE_S = Means, Reason, Manner, Time, Purpose
- FE_{Un} = Depictive, Form, Components

smile:

- FE_R = Body_part, Agent
- FE_S = Degree, Cognate_event, Manner, Time
- FE_{Un} = Depictive, Intended_perceiver, Internal_cause, Location_of_expressor, Path_of_gaze, External_cause

shake_hands:

- FE_R = Body_part, Indicated_entity, Addressee, Communicator, Place
- FE_S = Means, Manner, Time, Message
- FE_{Un} = Instrument

kick:

- FE_R = Body_part, Instrument, Victim, Agent, Place
- FE_S = Degree, Means, Reason, Purpose, Time, Containing_event, Manner
- FE_{Un} = Cause, Result, Explanation, Circumstances, Period_of_iterations, Subregion_bodypart, Concessive, Re_encoding, Depictive, Frequency, Iterations, Duration, Patricular_iteration

hug:

- FE_R = Agent, Bodypart_of_agent, Instrument, Entity, Place
- FE_S = Means, Reason, Manner, Time, Duration, Purpose
- FE_{Un} = Depictive, Particular_iteration, Locus, Result

hit:

- FE_R = Impactee, Impactor, Agent, Instrument, Impactors, Subregion, Place
- FE_S = Means, Manner, Time, Purpose, Speed
- FE_{Un} = Result, Force, Cause, Period_of_iterations

dive:

- FE_R = Goal, Source, Path, Road, Place
- FE_S = Degree, Purpose, Means, Manner, Time, Speed
- FE_{Un} = Depictive, Distance, Direction, Area, Path_shape, Result

drink:

- FE_R = Source, Ingestibles, Instrument, Ingestor, Place
- FE_S = Degree, Means, Manner, Time, Duration, Purpose
- FE_{Un} = None

cart_wheel:

- FE_R = Theme, Place
- FE_S = Manner, Time, Purpose
- FE_{Un} = Depictive, Direction, Angle, Fixed_location, Periodicity, Path_shape, Cause, Result

punch:

- FE_R = Body_part, Instrument, Victim, Agent, Place
- FE_S = Degree, Means, Reason, Purpose, Time, Containing_event, Manner

- FE_{Un} = Cause, Result, Explanation, Circumstances, Period_of_iterations, Subregion_bodypart, Concessive, Re_encoding, Depictive, Frequency, Iterations, Duration, Patricular_iteration

wave:

- FE_R = Body_part, Indicated_entity, Addressee, Communicator, Place
- FE_S = Means, Manner, Time, Message
- FE_{Un} = Instrument

hand_stand:

- FE_R = Agent, Location
- FE_S = Degree, Time, Duration, Purpose, Manner
- FE_{Un} = Depictive, Point_of_contact, Dependent_state

catch:

- FE_R = Stimulus, Experiencer
- FE_S = Degree, Means, Manner
- FE_{Un} = Depictive, Result, Explanation, Time, Circumstances

eat:

- FE_R = Source, Ingestibles, Instrument, Ingestor, Place
- FE_S = Degree, Means, Manner, Time, Duration, Purpose
- FE_{Un} = None

climb_stairs:

- FE_R = Goal, Cotheme, Area, Source, Direction, Self_mover, Path, Place

- FE_S = Coordinated_event, Result, Means, Reason, Purpose, Distance, Time, Depictive, Manner, Duration, Speed
- FE_{Un} = Internal_cause, Path_shape, Concessive, External_cause

kick_ball:

- FE_R = Competitor, Prize, Place
- FE_S = Means, Manner, Time
- FE_{Un} = Particular_iteration, Venue, Rank, Competitors, Score, Competition, Margin, Opponent

ride_horse:

- FE_R = Goal, Source, Theme, Path, Place
- FE_S = Degree, Purpose, Containing_event, Manner, Time, Duration, Speed
- FE_{Un} = Depictive, Distance, Direction, Area, Frequency, Iteration, Carrier, Path_shape, Result

brush_hair:

- FE_R = Goal, Agent, Source, Theme, Path, Cause, Place
- FE_S = Degree, Means, Reason, Manner, Time, Duration, Speed, Purpose
- FE_{Un} = Depictive, Distance, Area, Beneficiary, Cotheme, Result

stand:

- FE_R = Goal, Agent, Source, Theme, Path, Cause, Place
- FE_S = Degree, Means, Reason, Manner, Time, Duration, Speed, Purpose
- FE_{Un} = Depictive, Distance, Area, Beneficiary, Cotheme, Result

smoke:

- FE_R = Ingestor, Place
- FE_S = Means, Purpose, Time, Duration, Manner
- FE_{Un} = Substance, Entry_path, Delivery_device, Frequency

pick:

- FE_R = Agent, Instrument, Individuals, Path, Place
- FE_S = Containing_event, Means, Manner, Time
- FE_{Un} = Co_participant, Source, Aggregate, Frequency, Goal, Purpose

push:

- FE_R = Agent, Bodypart_of_agent, Instrument, Entity, Place
- FE_S = Means, Reason, Manner, Time, Duration, Purpose
- FE_{Un} = Depictive, Particular_iteration, Locus, Result

climb:

- FE_R = Goal, Cotheme, Area, Source, Direction, Self_mover, Path, Place
- FE_S = Coordinated_event, Result, Means, Reason, Purpose, Distance, Time, Depictive, Manner, Duration, Speed
- FE_{Un} = Internal_cause, Path_shape, Concessive, External_cause

talk:

- FE_R = Interlocutor_2, Interlocutors, Interlocutor_1, Place
- FE_S = Means, Purpose, Time, Duration, Manner

- FE_{Un} = Depictive, Language, Topic, Means_of_communication

clap:

- FE_R = Goal, Body_part, Source, Path, Agent, Addressee, Place
- FE_S = Coordinated_event, Degree, Cognate_event, Purpose, Manner, Time, Duration, Message
- FE_{Un} = Area, Internal_cause, Sub-region, Re_encoding, Depictive, Result, External_cause

C.3.5 BLS

watch_sports:

- FE_R = Sought_entity, Cognizer_agent, Ground, Place
- FE_S = Means, Manner, Time, Purpose
- FE_{Un} = Degree, Outcome

eating:

- FE_R = Source, Ingestibles, Instrument, Ingestor, Place
- FE_S = Degree, Means, Manner, Time, Duration, Purpose
- FE_{Un} = None

call_out:

- FE_R = Interlocutors, Interlocutor_2, Interlocutor_1, Place
- FE_S = Means, Containing_event, Manner, Time, Duration, Purpose
- FE_{Un} = Depictive, Domain, Language, Period_of_iterations, Topic, Amount_of_discussion, Means_of_communication, Frequency

lawn_care:

- FE_R = Goal, Instrument, Focal_entity, Place
- FE_S = Degree, Means, Duration, Manner, Time, Purpose
- FE_{Un} = Domain, Helper, Explanation, Frequency, Benefited_party

travel_to_job:

- FE_R = Goal, Area, Travel_means, Source, Direction, Path, Co_participant, Traveler, Place
- FE_S = Means, Reason, Purpose, Manner, Distance, Time, Depictive, Result, Duration, Speed
- FE_{Un} = Period_of_Iterations, Baggage, Mode_of_transportation, Descriptor, Frequency, Iterations

relax:

- FE_R = Depictive, Patient, Agent, Place
- FE_S = Means, Time, Cause, Event, Manner
- FE_{Un} = Result

grocery_shopping:

- FE_R = Beneficiary, Source, Theme, Recipient, Place
- FE_S = Means, Reason, Purpose, Time, Manner
- FE_{Un} = Result, Concessive

sleeping:

- FE_R = Sleeper, Place
- FE_S = Degree, Manner, Time, Duration

- FE_{Un} = None

purchase:

- FE_R = Seller, Place
- FE_S = Means, Reason, Purpose, Time, Manner
- FE_{Un} = Goods, Money, Period_of_iteration, Purpose_of_goods, Rate, Buyer, Recipient, Unit

send_email:

- FE_R = Addressee, Communicator, Intermediary, Place
- FE_S = Time, Reason
- FE_{Un} = Depictive, Medium, Location_of_communicator, Communication, Topic, Address, Frequency

garden_care:

- FE_R = Goal, Instrument, Focal_entity, Place
- FE_S = Degree, Means, Duration, Manner, Time, Purpose
- FE_{Un} = Domain, Helper, Explanation, Frequency, Benefited_party

write_email:

- FE_R = Addressee, Communicator, Intermediary, Place
- FE_S = Time, Reason
- FE_{Un} = Depictive, Medium, Location_of_communicator, Communication, Topic, Address, Frequency

do_home_work:

- FE_R = Cognizer, Place
- FE_S = Degree, Means, Content, Manner, Time
- FE_{Un} = Depictive, Topic, Medium, Evidence

call_phone:

- FE_R = Interlocutors, Interlocutor_2, Interlocutor_1, Place
- FE_S = Means, Containing_event, Manner, Time, Duration, Purpose
- FE_{Un} = Depictive, Domain, Language, Period_of_iterations, Topic, Amount_of_discussion, Means_of_communication, Frequency

interview_for_job:

- FE_R = Interlocutor_2, Interlocutors, Interlocutor_1, Place
- FE_S = Means, Purpose, Time, Duration, Manner
- FE_{Un} = Depictive, Language, Topic, Means_of_communication

drinking:

- FE_R = Source, Ingestibles, Instrument, Ingestor, Place
- FE_S = Degree, Means, Manner, Time, Duration, Purpose
- FE_{Un} = None

watch_t_v:

- FE_R = Expected_entity, Perceiver_agentive, Body_part, Phenomenon, Place
- FE_S = Means, Manner, Time, Duration, Purpose
- FE_{Un} = Depictive, Direction, State, Location_of_perceiver, Obscuring_medium, Ground

health_related_self_care:

- FE_R = Goal, Instrument, Focal_entity, Place
- FE_S = Degree, Means, Duration, Manner, Time, Purpose
- FE_{Un} = Domain, Helper, Explanation, Frequency, Benefited_party

do_research:

- FE_R = Instrument, Medium, Phenomenon, Cognizer
- FE_S = Degree, Means, Manner, Purpose
- FE_{Un} = Direction, Time, Ground

grooming:

- FE_R = Patient, Body_part, Agent, Instrument, Place
- FE_S = Means, Time, Duration, Purpose, Manner
- FE_{Un} = Medium, Sub_region, Frequency, Result

Bibliography

[1] A. Shoulson, M. L. Gilbert, M. Kapadia, and N. I. Badler, "An event-centric planning approach for dynamic real-time narrative," in *Motion in Games*. Doublin, Ireland: ACM, 2013, pp. 121–130.

[2] B. Sunshine-Hill and N. I. Badler, "Perceptually realistic behavior through alibi generation," *AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment; Tenth Artificial Intelligence and Interactive Digital Entertainment Conference*, 2010.

[3] N. Kraayenbrink, J. Kessing, T. Tutenel, G. de Haan, and R. Bidarra, "Semantic crowds," *Entertainment Computing*, vol. 5, no. 4, pp. 297–312, 2014.

[4] A. Normoyle, M. Likhachev, and A. Safonova, "Stochastic Activity Authoring with Direct User Control," in *Proceedings of the 18th Meeting of the ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games*, ser. I3D '14. New York, NY, USA: ACM, 2014, pp. 31–38.

[5] M. Slater, "Grand challenges in virtual environments," *Frontiers in Robotics and AI*, vol. 1, p. 3, 2014.

[6] I. Millington and J. Funge, *Artificial Intelligence for Games, Second Edition*, 2nd ed. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2009.

[7] M. Cavazza, S. Hartley, J.-L. Lugrin, and M. Le Bras, "Qualitative Physics in Virtual Environments," in *Proceedings of the 9th International Conference on Intelligent User Interfaces*, ser. IUI '04. New York, NY, USA: ACM, 2004, pp. 54–61. [Online]. Available: http://doi.acm.org/10.1145/964442.964454

[8] K. V. Hindriks, F. S. De Boer, W. Van Der Hoek, and J.-J. Ch. Meyer, "Agent Programming in 3apl," *Autonomous Agents and Multi-Agent Systems*, vol. 2, no. 4, pp. 357–401, Nov. 1999.

[9] J. T. Balint and J. M. Allbeck, "Generating Semantic Information for Virtual Environments." in *Workshop on Virtual Humans and Crowds for Immersive Environments*. Greenville: IEEE, Mar. 2016, p. 5.

[10] J. T. Balint, J. M. Allbeck, and M. R. Hieb, "Automated Simulation Creation from Military Operations Documents," in *Interservice/Industry Training, Simulation and Education Conference*. Orlando, Florida: I/ITSEC, 2015, p. Paper 15227.

[11] J. T. Balint and J. Allbeck, "ALET: Agents Learning their Environment through Text," *Computer Animation and Virtual Worlds*, vol. 28, no. 3-4, 2017.

[12] E. Yoshida, J.-P. Laumond, C. Esteves, O. Kanoun, T. Sakaguchi, and K. Yokoi, "Whole-Body Locomotion, Manipulation and Reaching for Humanoids," in *Motion in Games*, ser. Lecture Notes in Computer Science, A. Egges, A. Kamphuis, and M. Overmars, Eds. Springer Berlin Heidelberg, Jan. 2008, vol. 5277, pp. 210–221.

[13] M. Kallmann, "Analytical inverse kinematics with body posture control," *Computer Animation and Virtual Worlds*, vol. 19, no. 2, pp. 79–91, 2008.

[14] T. Tutenel, R. Bidarra, R. M. Smelik, and K. J. D. Kraker, "The Role of Semantics in Games and Simulations," *Comput. Entertain.*, vol. 6, no. 4, pp. 57:1–57:35, Dec. 2008.

[15] J.-L. Lugrin and M. Cavazza, "Making Sense of Virtual Environments: Action Representation, Grounding and Common Sense," in *IUI*. New York, NY, USA: ACM, 2007, pp. 225–234.

[16] T. Tutenel, R. Bidarra, R. M. Smelik, and K. J. De Kraker, "Rule-based layout solving and its application to procedural interior generation," in *Proceedings of the CASA Workshop on 3D Advanced Media In Gaming And Simulation*. Amsterdam, Netherlands: CASA, 2009, p. 10.

[17] D. Camozzato, L. Dihl, I. Silveira, F. Marson, and S. R. Musse, "Procedural floor plan generation from building sketches," *The Visual Computer*, vol. 31, no. 6, pp. 753–763, 2015. [Online]. Available: http://dx.doi.org/10.1007/s00371-015-1102-2

[18] M. Kallmann and D. Thalmann, "Direct 3d Interaction with Smart Objects," in *Proceedings of the ACM Symposium on Virtual Reality Software and Technology*, ser. VRST '99. New York, NY, USA: ACM, 1999, pp. 124–130.

[19] N. Farenc, S. R. Musse, E. Schweiss, M. Kallmann, O. Aune, R. Boulic, and D. Thalmann, "A paradigm for controlling virtual humans in urban environment simulations," *Applied Artificial Intelligence*, vol. 14, no. 1, pp. 69–91, 2000.

[20] C. Peters, S. Dobbyn, B. MacNamee, and C. O'Sullivan, "Smart Objects for Attentive Agents," in *Proceedings of 11th International Conference in Central Europe on Computer Graphics*, Plzen - Bory, Czech Republic, 2003, p. 11.

[21] S. Donikian and S. Paris, "Towards Embodied and Situated Virtual Humans," in *Motion in Games*, A. Egges, A. Kamphuis, and M. Overmars, Eds. Springer Berlin Heidelberg, 2008, pp. 51–62.

[22] F. Heckel and G. Youngblood, "Contextual Affordances for Intelligent Virtual Characters," in *Intelligent Virtual Agents*, ser. Lecture Notes in Computer Science, H. Vilhjlmsson, S. Kopp, S. Marsella, and K. Thrisson, Eds. Springer Berlin Heidelberg, 2011, vol. 6895, pp. 202–208.

[23] M. Kapadia, J. Falk, F. Znd, M. Marti, R. W. Sumner, and M. Gross, "Computer-assisted Authoring of Interactive Narratives," in *Proceedings of the 19th Symposium on Interactive 3D Graphics and Games*, ser. i3D '15. New York, NY, USA: ACM, 2015, pp. 85–92.

[24] J. Gibson, "Perceiving, Acting and Knowing," R. Shaw and J. Bransford, Eds. Lawrence Erlbaum, 1977.

[25] P. Sequeira, M. Vala, and A. Paiva, "What Can I Do with This?: Finding Possible Interactions Between Characters and Objects," in *Proceedings of the 6th International Joint Conference on Autonomous Agents and Multiagent Systems*, ser. AAMAS '07. New York, NY, USA: ACM, 2007, pp. 5:1–5:7.

[26] R. E. Fikes and N. J. Nilsson, "STRIPS: A new approach to the application of theorem proving to problem solving," *Artificial intelligence*, vol. 2, no. 3, pp. 189–208, 1972.

[27] R. Bindiganavale, W. Schuler, J. M. Allbeck, N. I. Badler, A. K. Joshi, and M. Palmer, "Dynamically Altering Agent Behaviors Using Natural Language Instructions," in *Proceedings of the Fourth International Conference on Autonomous Agents*. New York, NY, USA: ACM, 2000, pp. 293–300. [Online]. Available: http://doi.acm.org/10.1145/336595.337503

[28] O. van Kaick, A. Tagliasacchi, O. Sidi, H. Zhang, D. Cohen-Or, L. Wolf, and G. Hamarneh, "Prior Knowledge for Part Correspondence," *Computer Graphics Forum*, vol. Computer Graphics Forum, no. 2, pp. 553–562, 2011. [Online]. Available: http://dx.doi.org/10.1111/j.1467-8659.2011.01893.x

[29] Y. Wang, S. Asafi, O. van Kaick, H. Zhang, D. Cohen-Or, and B. Chen, "Active Co-analysis of a Set of Shapes," *ACM Trans. Graph.*, vol. 31, no. 6, pp. 165:1–165:10, Nov. 2012.

[30] X. Zhang, T. Tutenel, R. Mo, R. Bidarra, and W. F. Bronsvoort, "A Method for Specifying Semantics of Large Sets of 3d Models." in *GRAPP/IVAPP*, 2012, pp. 97–106.

[31] M. Gutierrez, D. Thalmann, and F. Vexo, "Semantic virtual environments with adaptive multimodal interfaces," in *Proceedings of the 11th International Multimedia Modelling Conference, 2005. MMM 2005.* IEEE, 2005, pp. 277–283.

[32] F. Worgotter, E. Aksoy, N. Kruger, J. Piater, A. Ude, and M. Tamosiunaite, "A Simple Ontology of Manipulation Actions Based on Hand-Object Relations," *Autonomous Mental Development, IEEE Transactions on*, vol. 5, no. 2, pp. 117–134, Jun. 2013.

[33] L. Drumond and R. Girardi, "A Survey of Ontology Learning Procedures." *WONTO*, vol. 427, pp. 1–13, 2008.

[34] P. Buitelaar, P. Cimiano, and B. Magnini, *Ontology learning from text: methods, evaluation and applications*. IOS press, 2005, vol. 123.

[35] P. Cimiano, *Ontology Learning and Population from Text: Algorithms, Evaluation and Applications*. Secaucus, NJ, USA: Springer-Verlag New York, Inc., 2006.

[36] P. University, "About WordNet," Princeton University, Tech. Rep., 2010. [Online]. Available: http://wordnet.princeton.edu

[37] C. Pelkey and J. M. Allbeck, "Populating Virtual Semantic Environments," *Computer Animation and Virtual Worlds*, vol. 24, no. 3, pp. 405–414, May 2014.

[38] R. H. Bordini, L. Braubach, M. Dastani, A. E. F. Seghrouchni, J. J. Gomez-Sanz, J. Leite, G. O'Hare, A. Pokahr, and A. Ricci, "A survey of programming languages and platforms for multi-agent systems," *Informatica*, vol. 30, no. 1, 2006.

[39] M. Thielscher, "FLUX: A Logic Programming Method for Reasoning Agents," *Theory Pract. Log. Program.*, vol. 5, no. 4-5, pp. 533–565, Jul. 2005. [Online]. Available: http://dx.doi.org/10.1017/S1471068405002358

[40] M. J. Wooldridge, *Reasoning about rational agents*. Cambridge, MA: MIT press, 2000.

[41] K. Vikhorev, N. Alechina, and B. Logan, "Agent programming with priorities and deadlines." International Foundation for Autonomous Agents and Multiagent Systems, 2011, pp. 397–404.

[42] K. Erol, J. A. Hendler, and D. S. Nau, "UMCP: A Sound and Complete Procedure for Hierarchical Task-network Planning." vol. 94, 1994, pp. 249–254.

[43] K. Erol, J. Hendler, and D. S. Nau, "Semantics for Hierarchical Task-Network Planning," University of Maryland,College Park,MD,20742, Computer Science Department,Institute for Systems Research,, Tech. Rep. T.R. 95-9, 1995.

[44] N. Nejati, T. Knik, and U. Kuter, "A Goal- and Dependency-directed Algorithm for Learning Hierarchical Task Networks," in *Proceedings of the Fifth International Conference on Knowledge Capture*, ser. K-CAP '09. New York, NY, USA: ACM, 2009, pp. 113–120. [Online]. Available: http://doi.acm.org/10.1145/1597735.1597755

[45] N. Nejati, P. Langley, and T. Konik, "Learning Hierarchical Task Networks by Observation," in *Proceedings of the 23rd International Conference on Machine Learning*, ser. ICML '06. New York, NY, USA: ACM, 2006, pp. 665–672. [Online]. Available: http://doi.acm.org/10.1145/1143844.1143928

[46] A. Mohseni-Kabir, C. Rich, S. Chernova, C. L. Sidner, and D. Miller, "Interactive Hierarchical Task Learning from a Single Demonstration," in *Proceedings of the Tenth Annual ACM/IEEE International Conference on Human-Robot Interaction*, ser. HRI '15. New York, NY, USA: ACM, 2015, pp. 205–212. [Online]. Available: http://doi.acm.org/10.1145/2696454.2696474

[47] M. Tenorth and M. Beetz, "A unified representation for reasoning about robot actions, processes, and their effects on objects," in *Intelligent Robots and Systems (IROS), 2012 IEEE/RSJ International Conference on*, Oct. 2012, pp. 1351–1358.

[48] R. Schwitter, "Controlled Natural Languages for Knowledge Representation," in *Proceedings of the 23rd International Conference on Computational Linguistics: Posters*, ser. COLING '10. Stroudsburg, PA, USA: Association for Computational Linguistics, 2010, pp. 1113–1121. [Online]. Available: http://dl.acm.org/citation.cfm?id=1944566.1944694

[49] W. Schuler, L. Zhao, and M. Palmer, "Parameterized action representation for virtual human agents," *Embodied conversational agents*, p. 256, 2000.

[50] H. Kress-Gazit, G. Fainekos, and G. Pappas, "From Structured English to Robot Motion," *Intelligent Robots and Systems, 2007. IROS 2007. IEEE/RSJ International Conference on*, pp. 2717–2722, Oct. 2007.

[51] H. Kress-Gazit, G. E. Fainekos, and G. J. Pappas, "Translating structured english to robot controllers," *Advanced Robotics*, vol. 22, no. 12, pp. 1343–1359, 2008.

[52] J. M. Allbeck and H. Kress-Gazit, "Constraints-based Complex Behavior in Rich Environments," in *Proceedings of the 10th International Conference on Intelligent Virtual Agents*, ser. IVA'10. Berlin, Heidelberg: Springer-Verlag, 2010, pp. 1–14. [Online]. Available: http://dl.acm.org/citation.cfm?id=1889075.1889077

[53] P. Langley, N. Trivedi, and M. Banister, "A Command Language for Taskable Virtual Agents." 2010.

[54] B. Coyne and R. Sproat, "WordsEye: an automatic text-to-scene conversion system." ACM, 2001, pp. 487–496.

[55] M. Ma, "Automatic conversion of natural language to 3d animation," 2006.

[56] B. Hayes and B. Scassellati, "Autonomously constructing hierarchical task networks for planning and human-robot collaboration." IEEE, 2016, pp. 5469–5476.

[57] M. Kapadia, F. Znd, J. Falk, M. Marti, R. W. Sumner, and M. Gross, "Evaluating the authoring complexity of interactive narratives with interactive behaviour trees," *Foundations of Digital Games*, 2015.

[58] N. Badler, B. Webber, M. Palmer, T. Noma, M. Stone, J. Rosenzweig, S. Chopra, K. Stanley, H. Dang, and R. Bindiganavale, "Natural language text generation from Task networks," Technical Report, CIS, University of Pennsylvania, Philadelphia, USA, Tech. Rep., 1997.

[59] J. C. Bourne, "Generating effective natural language instructions based on agent expertise," 1999.

[60] L. Kunze, T. Roehm, and M. Beetz, "Towards semantic robot description languages," in *Robotics and Automation (ICRA), 2011 IEEE International Conference on*, May 2011, pp. 5589–5595.

[61] A. Marzinotto, M. Colledanchise, C. Smith, and P. gren, "Towards a unified behavior trees framework for robot control." IEEE, 2014, pp. 5420–5427.

[62] D. Chi, M. Costa, L. Zhao, and N. Badler, "The EMOTE model for effort and shape." ACM Press/Addison-Wesley Publishing Co., 2000, pp. 173–182.

[63] N. Badler, J. Allbeck, L. Zhao, and M. Byun, "Representing and parameterizing agent behaviors." IEEE, 2002, pp. 133–143.

[64] F. Durupinar, M. Kapadia, S. Deutsch, M. Neff, and N. I. Badler, "PERFORM: Perceptual Approach for Adding OCEAN Personality to Human Motion Using Laban Movement Analysis," *ACM Trans. Graph.*, vol. 36, no. 1, pp. 6:1–6:16, Oct. 2016. [Online]. Available: http://doi.acm.org/10.1145/2983620

[65] H. Vilhjlmsson, N. Cantelmo, J. Cassell, N. E. Chafai, M. Kipp, S. Kopp, M. Mancini, S. Marsella, A. N. Marshall, C. Pelachaud, Z. Ruttkay, K. R. Thrisson, H. van Welbergen, and R. J. van der Werf, "The Behavior Markup Language: Recent Developments and Challenges," in *Intelligent Virtual Agents: 7th International Conference, IVA 2007 Paris, France, September 17-19, 2007 Proceedings*, C. Pelachaud, J.-C. Martin, E. Andr, G. Chollet, K. Karpouzis, and D. Pel, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2007, pp. 99–111.

[66] M. Thiebaux, S. Marsella, A. N. Marshall, and M. Kallmann, "SmartBody: Behavior Realization for Embodied Conversational Agents," in *AAMAS*, Richland, SC, 2008, pp. 151–158. [Online]. Available: http://dl.acm.org/citation.cfm?id=1402383.1402409

[67] K. Liu, J. Tolins, J. E. Fox Tree, M. Neff, and M. A. Walker, "Two Techniques for Assessing Virtual Agent Personality," *IEEE Trans. Affect. Comput.*, vol. 7, no. 1, pp. 94–105, Jan. 2016. [Online]. Available: http://dx.doi.org/10.1109/TAFFC.2015.2435780

[68] A. Shoulson, M. Kapadia, and N. I. Badler, "Paste: A platform for adaptive storytelling with events," *Intelligent Narrative Technologies*, vol. 6, p. 216, 2013.

[69] J. Kessing, T. Tutenel, and R. Bidarra, "Services in game worlds: A semantic approach to improve object interaction," in *Entertainment ComputingICEC 2009*, ser. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer, 2009, vol. 5709, pp. 276–281.

[70] Z. Wu and M. Palmer, "Verbs Semantics and Lexical Selection," in *Proceedings of the 32Nd Annual Meeting on Association for Computational Linguistics*, ser. ACL '94. Stroudsburg, PA, USA: Association for Computational Linguistics, 1994, pp. 133–138. [Online]. Available: http://dx.doi.org/10.3115/981732.981751

[71] A. Normoyle and N. I. Badler, "How Do Stylistic Motions Differ Numerically from Neutral Ones?" in *Proceedings of the Seventh International Conference on Motion in Games*, ser. MIG '14. New York, NY, USA: ACM, 2014, pp. 184–184. [Online]. Available: http://doi.acm.org/10.1145/2668064.2677080

[72] M. Paolucci, D. Kalp, A. Pannu, O. Sheholy, and K. Sycara, "A Planning Component for RETSINA Agents," *Intelligent Agents VI. Agent Theories, Architectures, and Languages*, vol. 1757, pp. 147–161, 2000. [Online]. Available: http://www.cs.cmu.edu/afs/cs.cmu.edu/project/pleiades-1/papers/atal99-pln.pdf

[73] E. Kontopoulos, D. Vrakas, F. Kokkoras, N. Bassiliades, and I. Vlahavas, "An Ontology-based Planning System for e-Course Generation," *Expert Syst. Appl.*, vol. 35, no. 1-2, pp. 398–406, Jul. 2008. [Online]. Available: http://dx.doi.org/10.1016/j.eswa.2007.07.034

[74] T. Sugawara, S. Kurihara, T. Hirotsu, K. Fukuda, and T. Takada, "Predicting Possible Conflicts in Hierarchical Planning for Multi-agent Systems," in *Proceedings of the Fourth International Joint Conference on Autonomous Agents and Multiagent Systems*, ser. AAMAS '05. New York, NY, USA: ACM, 2005, pp. 813–820. [Online]. Available: http://doi.acm.org/10.1145/1082473.1082597

[75] M. Kapadia, A. Shoulson, C. Steimer, S. Oberholzer, R. W. Sumner, and M. Gross, "An Event-centric Approach to Authoring Stories in Crowds," in *Proceedings of the 9th International Conference on Motion in Games*, ser. MIG '16. New York, NY, USA: ACM, 2016, pp. 15–24. [Online]. Available: http://doi.acm.org/10.1145/2994258.2994265

[76] A. Shoulson, F. M. Garcia, M. Jones, R. Mead, and N. I. Badler, "Parameterizing Behavior Trees," in *Proceedings of the 4th International Conference on Motion in Games*, ser. MIG'11. Berlin, Heidelberg: Springer-Verlag, 2011, pp. 144–155.

[77] B. Kartal, J. Koenig, and S. J. Guy, "User-driven Narrative Variation in Large Story Domains Using Monte Carlo Tree Search," in *Proceedings of the 2014 International Conference on Autonomous Agents and Multi-agent Systems*, ser. AAMAS '14. Richland, SC: International Foundation for Autonomous Agents and Multiagent Systems, 2014, pp. 69–76. [Online]. Available: http://dl.acm.org/citation.cfm?id=2615731.2615746

[78] N. Avradinis, T. Panayiotopoulos, and R. Aylett, "Continuous planning for virtual environments," *Intelligent Techniques for Planning*, pp. 162–193, 2005.

[79] J. Dormans, "Integrating emergence and progression," Hilversum, the Netherlands, 2011, p. 15.

[80] J. M. Allbeck, "CAROSA: A tool for authoring NPCs," in *Motion in Games*. Springer, 2010, pp. 182–193.

[81] P. R. Telang, F. Meneguzzi, and M. P. Singh, "Hierarchical Planning About Goals and Commitments," in *Proceedings of the 2013 International Conference on Autonomous Agents and Multi-agent Systems*, ser. AAMAS '13. Richland, SC: International Foundation for Autonomous Agents and Multiagent Systems, 2013, pp. 877–884. [Online]. Available: http://dl.acm.org/citation.cfm?id=2484920.2485059

[82] M. Sheehan and I. Watson, "On the Usefulness of Interactive Computer Game Logs for Agent Modelling," in *PRICAI 2008: Trends in Artificial Intelligence*, ser. Lecture Notes in Computer Science, T.-B. Ho and Z.-H. Zhou, Eds. Springer Berlin Heidelberg, Jan. 2008, vol. 5351, pp. 1059–1064. [Online]. Available: http://dx.doi.org/10.1007/978-3-540-89197-0_107

[83] M. Dastani, M. B. van Riemsdijk, and M. Winikoff, "Rich Goal Types in Agent Programming," in *The 10th International Conference on Autonomous Agents and Multiagent Systems - Volume 1*, ser. AAMAS '11. Richland, SC: International Foundation for Autonomous Agents and Multiagent Systems, 2011, pp. 405–412.

[84] M. O. Riedl and R. M. Young, "Narrative Planning: Balancing Plot and Character," *J. Artif. Int. Res.*, vol. 39, no. 1, pp. 217–268, Sep. 2010. [Online]. Available: http://dl.acm.org/citation.cfm?id=1946417.1946422

[85] J. Porteous, A. Lindsay, J. Read, M. Truran, and M. Cavazza, "Automated Extension of Narrative Planning Domains with Antonymic Operators," in *Proceedings of the 2015 International Conference on Autonomous Agents and Multiagent Systems*, ser. AAMAS '15. Richland, SC: International Foundation for Autonomous Agents and Multiagent Systems, 2015, pp. 1547–1555. [Online]. Available: http://dl.acm.org/citation.cfm?id=2772879.2773349

[86] J. Thangarajah, L. Padgham, and S. Sardina, "Modelling Situations in Intelligent Agents," in *Proceedings of the Fifth International Joint Conference on Autonomous Agents and Multiagent Systems*, ser. AAMAS '06. New York, NY, USA: ACM, 2006, pp. 1049–1051. [Online]. Available: http://doi.acm.org/10.1145/1160633.1160819

[87] W. Li and J. M. Allbeck, "The Virtual Apprentice," in *Proceedings of the 12th International Conference on Intelligent Virtual Agents*, ser. IVA'12. Berlin, Heidelberg: Springer-Verlag, 2012, pp. 15–27.

[88] P. Buitelaar and P. Cimiano, *Ontology learning and population: bridging the gap between text and knowledge*. Ios Press, 2008, vol. 167.

[89] R. Navigli, "Word Sense Disambiguation: A Survey," *ACM Comput. Surv.*, vol. 41, no. 2, pp. 10:1–10:69, Feb. 2009.

[90] K. Frger and T. Takala, "Animating with style: defining expressive semantics of motion," *The Visual Computer*, vol. 32, no. 2, pp. 191–203, 2015.

[91] D. Bouchard and N. I. Badler, "Segmenting Motion Capture Data Using a Qualitative Analysis," in *Proceedings of the 8th ACM SIGGRAPH Conference on Motion in Games*, ser. MIG '15. New York, NY, USA: ACM, 2015, pp. 23–30. [Online]. Available: http://doi.acm.org/10.1145/2822013.2822039

[92] C. F. Baker, C. J. Fillmore, and J. B. Lowe, "The Berkeley FrameNet Project," in *COLING*, vol. 1. Stroudsburg, PA, USA: Association for Computational Linguistics, 1998, pp. 86–90. [Online]. Available: http://dx.doi.org/10.3115/980451.980860

[93] L. Shi and R. Mihalcea, "Putting Pieces Together: Combining FrameNet, VerbNet and WordNet for Robust Semantic Parsing," in *Computational Linguistics and Intelligent Text Processing*, ser. Lecture Notes in Computer Science, A. Gelbukh, Ed. Springer Berlin Heidelberg, 2005, vol. 3406, pp. 100–111.

[94] M. M. Botvinick, Y. Niv, and A. C. Barto, "Hierarchically organized behavior and its neural foundations: A reinforcement learning perspective," *Cognition*, vol. abs/1109.2130, 2008.

[95] T. Balint and J. Allbeck, "Automated Generation of Plausible Agent Object Interactions," in *Intelligent Virtual Agents*, ser. Lecture Notes in Computer Science, W.-P. Brinkman, J. Broekens, and D. Heylen, Eds. Springer International Publishing, 2015, vol. 9238, pp. 295–309.

[96] C. Fernndez, P. Baiget, F. X. Roca, and J. Gonzlez, "Augmenting Video Surveillance Footage with Virtual Agents for Incremental Event Evaluation," *Pattern Recogn. Lett.*, vol. 32, no. 6, pp. 878–889, Apr. 2011. [Online]. Available: http://dx.doi.org/10.1016/j.patrec.2010.09.027

[97] E. Laparra and G. Rigau, "Integrating WordNet and FrameNet using a Knowledge-based Word Sense Disambiguation Algorithm," in *Proceedings of the International Conference RANLP-2009*. Borovets, Bulgaria: Association for Computational Linguistics, Sep. 2009, pp. 208–213. [Online]. Available: http://www.aclweb.org/anthology/R09-1039

[98] W. Loh, "Classification and regression trees," *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, vol. 1, no. 1, pp. 14–23, 2011.

[99] G. Salton and M. J. McGill, *Introduction to Modern Information Retrieval*. New York, NY, USA: McGraw-Hill, Inc., 1986.

[100] O. Levy and Y. Goldberg, "Dependency-Based Word Embeddings." 2014, pp. 302–308. [Online]. Available: http://www.aclweb.org/anthology/P14-2050.pdf

[101] T. Mikolov, K. Chen, G. Corrado, and J. Dean, "Efficient estimation of word representations in vector space," *arXiv preprint arXiv:1301.3781*, 2013.

[102] A. Trask, P. Michalak, and J. Liu, "sense2vec-A fast and accurate method for word sense disambiguation in neural word embeddings," *arXiv preprint arXiv:1511.06388*, 2015.

[103] F. P. Tasse and N. Dodgson, "Shape2vec: Semantic-based Descriptors for 3d Shapes, Sketches and Images," *ACM Trans. Graph.*, vol. 35, no. 6, pp. 208:1–208:12, Nov. 2016. [Online]. Available: http://doi.acm.org/10.1145/2980179.2980253

[104] O. Ludwig, Q. Do, C. Smith, M. Cavazza, and M. F. Moens, "Learning to Extract Action Descriptions from Narrative Text," *IEEE Transactions on Computational Intelligence and AI in Games*, vol. PP, no. 99, pp. 1–1, 2017.

[105] M. Ester, H.-P. Kriegel, J. Sander, and X. Xu, "A density-based algorithm for discovering clusters in large spatial databases with noise." vol. 96, 1996, pp. 226–231.

[106] Y. Liu, Z. Liu, T.-S. Chua, and M. Sun, "Topical Word Embeddings." 2015, pp. 2418–2424.

[107] D. M. Blei, A. Y. Ng, and M. I. Jordan, "Latent dirichlet allocation," *Journal of machine Learning research*, vol. 3, no. Jan, pp. 993–1022, 2003.

[108] G. Guerra-Filho and A. Biswas, "The human motion database: A cognitive and parametric sampling of human motion," *Image and Vision Computing*, vol. 30, no. 3, pp. 251–261, 2012.

[109] B. o. L. Statistics, "American Time Use Survey," Tech. Rep., 2010. [Online]. Available: http://www.bls.gov/news.release/atus.nr0.htm

[110] M. Hart, "Project Gutenberg." [Online]. Available: http://www.gutenberg.org/

[111] L. Fei-Fei, R. Fergus, and P. Perona, "Learning Generative Visual Models from Few Training Examples: An Incremental Bayesian Approach Tested on 101 Object Categories," in *Computer Vision and Pattern Recognition Workshop, 2004. CVPRW '04. Conference on*. Amsterdam, Netherlands: Elsevier, Jun. 2004, pp. 178–178.

[112] Z. Wu, S. Song, A. Khosla, F. Yu, L. Zhang, X. Tang, and J. Xiao, "3d shapenets: A deep representation for volumetric shapes," in *the IEEE Conference on Computer Vision and Pattern Recognition*, 2015, pp. 1912–1920.

[113] R. ehek and P. Sojka, "Software Framework for Topic Modelling with Large Corpora," in *Proceedings of the LREC 2010 Workshop on New Challenges for NLP Frameworks*. Valletta, Malta: ELRA, May 2010, pp. 45–50.

[114] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Machine Learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[115] H. Lee, "mergeCGF AI," Dec. 2015. [Online]. Available: realtimevisual.com

[116] E. Fast, W. McGrath, P. Rajpurkar, and M. S. Bernstein, "Augur: Mining Human Behaviors from Fiction to Power Interactive Systems." Santa Clara, California: ACM, 2016, pp. 237–247.

[117] A. J. Quinn and B. B. Bederson, "Human computation: a survey and taxonomy of a growing field." Vancouver, BC, Canada: ACM, 2011, pp. 1403–1412.

[118] M. Guzdial, B. Harrison, B. Li, and M. O. Riedl, "Crowdsourcing Open Interactive Narrative," in *International Conference on the Foundations of Digital Games*. Pacific Grove, CA: FDG, 2015, p. 9.

[119] M. Rouhizadeh, M. Bowler, R. Sproat, and B. Coyne, "Collecting semantic data by Mechanical Turk for the lexical knowledge resource of a text-to-picture generating system," in *Ninth International Conference on Computational Semantics*. Association for Computational Linguistics, 2011, pp. 380–384.

[120] M. Palmer, D. Gildea, and P. Kingsbury, "The proposition bank: An annotated corpus of semantic roles," *Computational linguistics*, vol. 31, no. 1, pp. 71–106, 2005.

[121] M. Palmer, D. Gildea, and N. Xue, "Semantic role labeling," *Synthesis Lectures on Human Language Technologies*, vol. 3, no. 1, pp. 1–103, 2010.

[122] Y. Hahm, Y. Kim, Y. Won, J. Woo, J. Seo, J. Kim, S. Park, D. Hwang, and K.-S. Choi, "Toward Matching the Relation Instantiation from DBpedia Ontology to Wikipedia Text: Fusing FrameNet to Korean," in *Proceedings of the 10th International Conference on Semantic Systems*, ser. SEM '14. New York, NY, USA: ACM, 2014, pp. 13–19. [Online]. Available: http://doi.acm.org/10.1145/2660517.2660534

[123] S. Kbler, R. McDonald, and J. Nivre, "Dependency parsing," *Synthesis Lectures on Human Language Technologies*, vol. 1, no. 1, pp. 1–127, 2009.

[124] Y. Goldberg and J. Nivre, "Training deterministic parsers with non-deterministic oracles," *Transactions of the association for Computational Linguistics*, vol. 1, pp. 403–414, 2013.

[125] S. Banerjee and T. Pedersen, "An Adapted Lesk Algorithm for Word Sense Disambiguation Using WordNet," in *Proceedings of the Third International Conference on Computational Linguistics and Intelligent Text Processing*, ser. CICLing '02. London, UK, UK: Springer-Verlag, 2002, pp. 136–145. [Online]. Available: http://dl.acm.org/citation.cfm?id=647344.724142

[126] T. U. of Tokyo, "ICS Action Database," 2003. [Online]. Available: http://www.ics.t.u-tokyo.ac.jp/action/

[127] H. Kuehne, H. Jhuang, E. Garrote, T. Poggio, and T. Serre, "HMDB: A large video database for human motion recognition," in *2011 International Conference on Computer Vision*. Barcelona, Spain: IEEE, Nov. 2011, pp. 2556–2563.

[128] K. Yordanova and T. Kirste, "A Process for Systematic Development of Symbolic Models for Activity Recognition," *ACM Trans. Interact. Intell. Syst.*, vol. 5, no. 4, pp. 20:1–20:35, Dec. 2015. [Online]. Available: http://doi.acm.org/10.1145/2806893

[129] J. Ye, G. Stevenson, and S. Dobson, "USMART: An Unsupervised Semantic Mining Activity Recognition Technique," *ACM Trans. Interact. Intell. Syst.*, vol. 4, no. 4, pp. 16:1–16:27, Nov. 2014. [Online]. Available: http://doi.acm.org/10.1145/2662870

[130] S. R. K. Branavan, N. Kushman, T. Lei, and R. Barzilay, "Learning High-level Planning from Text," in *Proceedings of the 50th Annual Meeting of the Association for Computational Linguistics: Long Papers - Volume 1*, ser. ACL '12. Stroudsburg, PA, USA: Association for Computational Linguistics, 2012, pp. 126–135. [Online]. Available: http://dl.acm.org/citation.cfm?id=2390524.2390543

[131] R. Lu and S. Zhang, *Automatic Generation of Computeranimation: Using AI for Movie Animation*. Berlin, Heidelberg: Springer-Verlag, 2002.

[132] P. Merrell, E. Schkufza, Z. Li, M. Agrawala, and V. Koltun, "Interactive Furniture Layout Using Interior Design Guidelines," *ACM Trans. Graph.*, vol. 30, no. 4, pp. 87:1–87:10, Jul. 2011. [Online]. Available: http://doi.acm.org/10.1145/2010324.1964982

[133] L.-F. Yu, S.-K. Yeung, C.-K. Tang, D. Terzopoulos, T. F. Chan, and S. J. Osher, "Make It Home: Automatic Optimization of Furniture Arrangement," *ACM Trans. Graph.*, vol. 30, no. 4, pp. 86:1–86:12, Jul. 2011. [Online]. Available: http://doi.acm.org/10.1145/2010324.1964981

[134] A. Farhadi, M. Hejrati, M. A. Sadeghi, P. Young, C. Rashtchian, J. Hockenmaier, and D. Forsyth, "Every picture tells a story: Generating sentences from images." Springer, 2010, pp. 15–29.

Curriculum Vitae

D.1 Education

- Ph.D in Computer Science George Mason University, July 2017
- Ph.D Candidate in Computer Science, George Mason University, May 2016.
- Masters of Science in Computer Science, George Mason University May 2014
- Bachelors of Science in Physics with honors, Roanoke College May 2011

D.2 Awards, Grants, and Professional Memberships

- Provost Dissertation Completion Grant Recipient, George Mason University, 2016
- Graduate Student Travel Fund Recipient, George Mason University, 2016
- Distinguished Graduate Teaching Award, George Mason University, 2015-2016
- Graduate Student Travel Fund Recipient, George Mason University, 2015
- Outstanding Graduate Teaching Award, George Mason University, 2011-2012
- President, Sigma Pi Sigma, Roanoke College, 2010-2011 Secretary, Pi Mu Epsilon, Roanoke College, 20110-2011

D.3 Appointments

- Graduate Research Assistant, Games and Intelligent Animations Lab, George Mason University, 2017
- Graduate Teaching Assistant, Introduction to Computer Science (SPARC), George Mason University, Fall 2016
- Ph.D Intern, Visual Analytics, Pacific Northwest National Laboratory, Summer 2016
- Graduate Teaching Assistant, Introduction to Computer Science (SPARC), George Mason University, 2015-2016

- Repperger Intern, Air Force Research Laboratories, Wright Patterson Air Force Base, Summer 2015
- Graduate Research Assistant, Games and Intelligent Animations Lab, George Mason University, 2012-2015
- Graduate Teaching Assistant, Introduction to Computer Science, George Mason University, 2011-2012
- Undergraduate Tutor, Physics I and II, Roanoke College, 2010-2011
- Undergraduate Research Assistant, Roanoke College, 2007-2011

D.4 Journal Papers and Book Chapters

- **J. Timothy Balint** and Jan M. Allbeck. 2017. ALET: Agents Learning their Environment through Text. Computer Animation and Virtual Worlds. 28.3-4 Best Paper Nominee
- **J. Timothy Balint**, Brad Reynolds, Leslie M. Blaha, Tim Halverson. Visualizing Eye Movements in Formal Cognitive Models, In Eye Tracking and Visualization: Foundations, Techniques, and Applications. ETVIS 2015, Springer International Publishing, 2017, pp. 93-111
- **John T. Balint** and Jan M. Allbeck, Multi-sense Attention for Autonomous Agents, in Virtual Crowds: Steps Toward Behavioral Realism, 1st ed., vol. 8, Morgan and Claypool, pp. 117130.
- Weizi Li, **John T. Balint**, and Jan M. Allbeck, Using a Parameterized Memory Model to Modulate NPC AI, in Virtual Crowds: Steps Toward Behavioral Realism, 1st ed., vol. 8, Morgan and Claypool, pp. 149157.

D.5 Conference Papers

- **J. Timothy Balint**, Jan M. Allbeck, Michael R. Hieb. 2015. Automated Simulation Creation from Military Operations Documents. In Proceedings of I/ITSEC 2015. Paper 15227
- **J. Timothy Balint** and Jan M. Allbeck. 2015. Automatic Generation of Plausible Agent Object Interactions. In Intelligent Virtual Agents (IVA 15). 295-309
- **Tim Balint** and Jan M. Allbeck. 2014. Is That How Everyone Really Feels? Emotional Contagion with Masking for Virtual Crowds. In Intelligent Virtual Agents (IVA 14). 26-35
- **Tim Balint** and Jan M. Allbeck. 2013. Whats Going on? Multi-sense Attention for Virtual Agents. In Intelligent Virtual Agents (IVA 13). 349-357
- Weizi Li, **Tim Balint**, and Jan M. Allbeck. 2013. Using a Parameterized Memory Model to Modulate NPC AI. In Intelligent Virtual Agents (IVA 13). 1-14

D.6 Workshop Papers and Posters

- **J. Timothy Balint**, Dustin Arendt, and Leslie M. Blaha. 2016. Storyline Visualizations of Eye Tracking in Movie Viewing. In Proceedings of the Second Workshop on Eye Tracking and Visualization (ETVIS 2016). Best Paper Award
- **J. Timothy Balint** and Jan M. Allbeck. 2016. Generating Semantic Information for Virtual Environments. In Proceedings of the Workshop on Virtual Humans and Crowds for Immersive Environments (VHCIE 2016).
- **J. Timothy Balint**, Brad Reynolds, Leslie M. Blaha, Tim Halverson. 2015. Visualizing Eye Movements in Formal Cognitive Models. In Proceedings of the First Workshop on Eye Tracking and Visualization (ETVIS 2015)
- **Tim Balint**, Yotam Gingold, and Jan M. Allbeck. 2014. Agent Script Generation using Descriptive Text Documents. In Proceedings of the Seventh International Conference on Motion in Games (MIG '14). 181-181.
- **John T. Balint** and Jan M. Allbeck. 2013. MacGyver Virtual Agents: using Ontologies and Hierarchies for Resourceful Virtual Human Decision-Making. In Proceedings of the 2013 international conference on Autonomous agents and multi-agent systems (AAMAS '13). 1153-1154

D.7 Presentations and Invited Talks

- ALET: Agents Learning their Environment through Text at Computer Animation and Social Agents, Seoul, South Korea, 2017. – Also presented at George Mason University, 2017
- Knowing the World: Semantics in Video Games at East Coast Gaming Conference 2017, Raleigh, North Carolina, 2017. Invited Talk – Also presented at George Mason University, 2017
- Storyline Visualizations of Eye Tracking in Movie Viewing at ETVIS, Baltimore, Maryland, 2016
- Automated Simulation Creation from Military Operations Documents at I/ITSEC, Orlando, Florida, 2015
- Automatic Generation of Plausible Agent Object Interactions at Intelligent Virtual Agents, Delft, The Netherlands, 2015
- Making Cortanas Friends: Embodying Video Game Characters at Roanoke College, Salem VA, 2015. Invited Talk
- Agent Script Generation using Descriptive Text Documents (Poster Presentation) at Motion in Games, Playa Vista, CA 2014
- Is that How Everyone Really Feels? Emotional Contagion with Masking for Virtual Crowds at Intelligent Virtual Agents, Boston MA 2014

- Using a Parameterized Memory Model to Modulate NPC AI at Intelligent Virtual Agents, Edinburgh U.K. 2013 – Also presented at George Mason University 2013
- What's going on? Multi-Sense Attention for Virtual Agents at Intelligent Virtual Agents, Edinburgh U.K. 2013 – Also presented at George Mason University 2013
- Microsoft Kinect and Motion Capture at Roanoke College, Salem VA, 2010 Shared talk with Dr. Durrell Bouchard

D.8 Service

- Reviewed for the Following Conferences and Journals
  - Journal of Graphics Tools
  - Transactions on Affective Computing
  - IEEE Transactions on Computational Intelligence and AI in Games
  - Computational Animation and Social Agents 2017
  - Autonomous Agents and Multi Agent Systems 2017
  - Motion in Games 2015
  - Motion in Games 2014
  - Intelligent Virtual Agents 2014
  - Motion in Games 2013
  - Intelligent Virtual Agents 2013
  - Autonomous Agents and Multi Agent Systems 2013
- Assisted in University Recruiting Activities
  - Appeared in a promotional video for George Mason Universitys Computer Science Department
  - The first participant presenting their graduate experience to perspective students in the Volgenau School of Engineering Perspective Applicant Session
  - Student Recruitment Czar at George Mason Universitys Computer Science Department for perspective Ph.D. students
- Mentored undergraduate students in virtual agent research