Computerized Film Directing

by

Carl Schroeder

Submitted to the Department of
Electrical Engineering and Computer Science
in Partial Fulfillment of the Requirements
for the Degree of
Bachelor of Science in Computer Science and Engineering
at the

Massachusetts Institute of Technology

May 1987

Signature of Author .....
Department of Electrical Engineering and Computer Science
May 18, 1987

Certified by .....
Thesis Supervisor

Accepted by .....
Leonard Gould
Chairman, Department Committee on Undergraduate Theses

Computerized Film Directing

by

Carl Schroeder

Submitted to the Department of
Electrical Engineering and Computer Science
on May 18, 1987 in partial fulfillment of the requirements
for the Degree of
Bachelor of Science in Computer Science and Engineering

Abstract

The intent of this thesis is to demystify the process of film creation, a domain of unfathomed human creativity, by making significant progress toward a realization of the computer as a tool for film creation. To what extent can the computer, by following its programming, aid a user in the direction and editing of a film? For that matter, what is a film? Theories based on algorithmic methods for story generation, film representation, and editing technique have been developed. Implementation of a valid subset of these ideas involved the production of a shot library, the representation thereof, and rudimentary procedures for film generation. Programming tools consisted of Common Lisp and HPRL, a representation language developed at Hewlett-Packard and beta-sited at the MIT Media lab.

Computerized Film Directing

by

Carl Schroeder

Submitted to the Department of
Electrical Engineering and Computer Science
on May 18, 1987 in partial fulfillment of the requirements
for the Degree of
Bachelor of Science in Computer Science and Engineering

Abstract

The intent of this thesis is to demystify the process of film creation, a domain of unfathomed human creativity, by making significant progress toward a realization of the computer as a tool for film creation. To what extent can the computer, by following its programming, aid a user in the direction and editing of a film? For that matter, what is a film? Theories based on algorithmic methods for story generation, film representation, and editing technique have been developed. Implementation of a valid subset of these ideas involved the production of a shot library, the representation thereof, and rudimentary procedures for film generation. Programming tools consisted of Common Lisp and HPRL, a representation language developed at Hewlett-Packard and beta-sited at the MIT Media lab.

Chapter One

Prothesis

1.1 Manifesto

I like movies. Their potential to communicate (as well as manipulate) is limitless. The experience of immersing oneself in a good movie can be ineffable.

I like Ingmar Bergman. He was an outstanding communicator through the media of film. This presumes a great deal of talent, and even the best directors are stronger in some areas than others. Bergman was foremostly a brilliant playwright , whose visions translated well, albeit bleakly, into the cinematic domains. Fellini, on the other hand, excelled in a more dynamic use of imagery to communicate more frenzied emotional states. The multiplicity of directing styles is due to the many symbiotic parts of which the cinema is comprised (story construction, image transition, image recording, etc.) . But, for all the considerations of film, might it not be possible to extract minimal rules for each knowledge domain, weight them according to "style", and thereby formulate automatic movie generation? Of course. TV writers do it all the time. I mean, sure they're good at it, but why not just get a machine to do some level of the work?

This is really more feasible than it sounds. A Russian director named Kuleschov helped prove the flexibility of narrative given a limited source of imagery. His experiments are now legendary. He was able to construct a single, smooth film from shots taken in wildy different locations, as when he depicted an ascension of the White House steps with footage of the Kremlin. With shots of different women's bodies he created a composite person who existed only in film. In his most famous test, he is said to have intercut views of such things as food, a coffin, and a child with the expressionless close-up of the actor Mozhukhin. Audiences reacted with comments of how adept the character was at communicating subtle emotions associated with the

interwoven scenes. [Fell 79]

Think about the Encyclopedia Brittanica, and then consider just how small the domains might be for a self-contained narrative like a sitcom. By far the most space allocated would be for the description knowledge of a complex multitude of shots of the entire cast doing various and sundry silly things, as they are wont to do on sitcoms.

Actually, I have no intention of making a computer generate sitcoms. I value movies for their ability to communicate on a more intellectual and aesthetic level. What are the minimal domains to maximally permit the beauty of such cinematic movements as the German Expressionists, the Italian Neorealists, or the French New Wave? A program must exhibit a good number of higher film capabilities, like the evaluation of an edit for emotional impact (assuming that such impact is generalizable to a level of universality). A director thus knows how to build powerful metaphors and moods that can work subliminally on the uneducated. Keeping in mind the work of the great Russian directors, whose methods invoked semiotics and propaganda, one may candidly anticipate the cinematic achievements of a computerized Big Brother.

My goal is to whittle down the problem of automatic movie directing to a small number of related knowledge domains and rules, while still retaining the capability for aesthetic cinematic judgement. The degree of knowledge needed to do even basic directing is staggering, but only when not limited to small domains. My attempt is named Ingmar, in homage to the namesake, with no explicit deference to the original, because the dissimilarity between works produced will be obvious.

1.2 Film

1.2.1 What's in a Film?

A film is the portrayal of a story.

However, it is not the story itself, and from this distinction many important relationships between film understanding and story understanding can be derived. While film understanding may be a barely nascent topic of computer application, story understanding is a field alive and well, with respected proponents vanguarding some very attractive research.

The film is not exactly the story because the story is not narrated in language (unless in part by the speech of a character during the film, which is story within film). The story is implied to the viewer by the presentation of evidence for his recognition of events, whether they be physical or emotional. Story summarization by language (a fruitful route to story understanding) may be useful in locating the pivotal events of the story for which depicting scenes must be sufficiently obvious for a known viewer to understand. But the film itself consists of the non-summarized linear presentation of the visible events of a story.

Notice my choice of the description 'known viewer'. A filmmaker must know his audience in order to suceed in his mode of communication. He must know what singular depictions will result in the recognition of a story event by his viewers'. The filmaker is hence relying upon many assumptions, most of the form that his audience is like himself in its understanding of the world.

Today, the basic creation of film is a practised and defined skill. There are many accepted and reliable rules. Of course, the rules may always be broken, which is why the field is not stagnant and devoid of character. But the fact is that some of the most important assumptions made by the filmmaker are taken for granted by modern

viewers. There is a filmic language, which is culture specific and vital to film, the portrayal of a story. This language includes cinematic conventions, like the use of wide to close shots for the introduction of a new setting, forms which are not understood by members of cultures which do not have films, like African tribes. For an instance of this simple example's importance, one need only consider the works of Resnais (like "Hiroshima mon Amour") where the violation of the wide to close convention confuses the viewer and thus marks ensuing events as worthy of special attention (other emotional attributes can be derived, but not without sailing between the Scylla and Charybdis of some film semiotics)

So, the filmmaker not only needs to understand a story but also requires the knowledge of how to communicate it in the ordering of particular scenes. He relies upon the recognition, but not necessarily the precise use, of many cinematic forms to help structure the scenes. He hopes that his choice of scenes is correct in the sense that the story they are part of will be similarly recognized by the viewer. The more complex the communicated events of a film, the greater the likelihood that many details will not be understood by any one viewer. For example, the works of Federico Fellini (like "8 1/2") are epic in complexity and contain many elements unique or personal to Fellini. The only person who can ever truly understand a Fellini film is Fellini. This is however okay, since any film may be good by the viewer's judgement if only it has communicated something of value, even if this was unintentional on the part of the director. Though I do not understand Fellini's iconography of Catholicism, I have my own appreciation of it.

So there are two broad types of decision on the director's part. The first is the selection of the representative events of a story. These events must then be decided for visualisation in terms of the filmic language. The former decision involves a general to specific, top down, story generation approach. This subsumes much real world comprehension, stuff about what people do and why they do it, which every viewer works out for himself during a film. The second involves a constructive, rule based,

film generation scheme. Here are included those tricks of the trade which are often not even consciously recognized by the viewer.

1.2.2 Story Summarization vs. Story Generation

As has been mentioned, one method for story understanding is that of summary. The events of the story are collected in abstractions of language. Heading the list of such research is the work of Wendy Lehnart and associates. Though their efforts were never meant to apply to film, there are connections which make the work most beguiling. And the differences between summary type research and the problems of filmic story can lead to critical insight.

Lehnart's approach of summarization represented an ability to go from the specific to the general and thereby exclude detail. Many useful concepts were developed, including the plot unit, a high level association between a pattern of events and the generally recognized plot. This knowledge is crucial for the filmmaker in locating the key events of a story. But the filmmaker is equally interested in working out the details within a story, for it is from the details that specific images will be chosen. A film cannot be made on the level of a summary alone.

The COMSYS story is one of the demonstration instances of the narrative summary work [Lehnart 84].

> John and Mike were competing for the same job at IBM. John got the job and Mike decided to start his own consulting firm, COMSYS. Within three years, COMSYS was flourishing. By that time, John had become dissatisfied with IBM so he asked Mike for a job. Mike spitefully turned him down.

Within this story is a critical plot unit called competition. It involves the mutually exclusive goals of John and Mike, who both wish to be hired for a position which is available only for one person. Summarization can identify this competition, and thereby generate a summary by what is called conceptual ellipsis, or the omission of details inferrable from the plot unit. Summarization should generate a statement for

this competition like "Mike wanted to work for IBM, but they hired John". There is a wealth of syntactic as well as semantic understanding going on to realize this.

The filmmaker may have been supplied with the COMSYS story as a script, in which case he will need to summarize it only so far as identifying the competition. He will then want to choose the scenes which communicate this competition. Conceptual ellipsis is dangerous because what might have been implied in language still has to be explicit in image. How many images and of what impact might be chosen can be determined by the inferrable nature of the plot unit event. There are very different goals going on here!

The filmmaker may communicate the COMSYS story by focusing on the competition. He may recognize the events of competition as being obvious with certain images. In this case, he wishes to portray the competition with more drama to reinforce the importance of the competition. He may choose to violate certain filmic rules, like using strange camera angles in a scene of Mike and John waiting to be interviewed. He may play out events with as much time and detail as would be alloted a more obtuse plot, yielding a bleak quality of realism. Whatever he does, he cannot just summarize the plot in one or two key scenes because the plot would be de-emphasized.

With reference to his audience, the filmmaker may also expand the plot with unrelated scenes to satisfy other goals. He may wish the audience to empathize with John more than Mike, therefore scenes of John anxiously eating breakfast before his interview would be relevant. He may wish to create an anti-story, in the style of the French New Wave, in which the competition scenes might be a bit scrambled, or too short, or dreamlike with contradictions, in order to confuse the viewer and thus give a different kind of emphasis to the competition.

The filmmaker will almost always have to expand more than summarize the story. Suppose he was given a script that called for competition between Mike and John. The filmmaker has to then consider the constraints of the competition plot unit and work

backward for a sequence of events which summarize to competition. This may not result in the same story, but this skill is needed whenever an event of great plot unit importance requires more detail for the selection of images. This is what was involved above, when a scene of John breakfasting was called for. Some working models in a story generation program for what one does in situations responded that to get a job, one must be interviewed, and that on that day one will eat breakfast with no small amount of apprehension.

But still one does not apply such a scenario immediatly, because for every scene of an expanded story conceptual conflicts may arise which require explanation. For example, suppose John was a devil-may-care sort of guy, which might explain why it took him three years to become dissatisfied with what is later depicted as a very tedious job at IBM. But such a John wouldn't be that nervous about some job interview, unless a new sub-story was presented, like John needs the work fast to pay back gambling debts. Constraint propagation is an important story driver.

There are contained in the story summarization research many ideas agreeable to the needs of film generation. Plot units are vital. Their work also relied upon a 'realization module for language generation' called Mumble. This ability to make sense of language descriptions in ways useful to other programs is invoked in filmmaking. All the algorithms for plot connectivity and representation are conceptually useful.

The bottom line is only that the results of story summarization work, as embodied in the results of Lehnart et. al, must not be taken too literally in relation to filmmaking. Despite many conceptual similarities the research goals are fundamentally different.

1.3 My Approach

A computer acting as a film director might work in one of two general ways. In both cases, it is supplied with some kind of story script, defined at an abstract (non-shot) level.

Ultimately, it might be desirable for the output to be in a non-viewable, shot specification form. In this case, the user would then still have to supply those shots by going out and filming them. Such a tool would marvelous as guidance for the rapid planning of a film by even one unfamiliar with the methods of film creation. Any film could be described in full detail directly from ideas, without the necessity of years of directing experience. However, the computer director involved is also of the most general and powerful type. There is no guiding light for the programmer of such a tool.

More pragmatic is the goal of a director which does its best to present a film built from a library of previously recorded shots. As the library grows, the usefulness of this director approaches that of the more general case. When a film simply cannot be made for lack of the right shot, a detailed failure message could be used a suggestion for the recording of new material. Or as complex video processing methods are perfected, one can imagine options to build a better shot from the existing near misses. Parts of the image that may have rendered the shot useless for simple constraint reasons could be painted out. (For example, a clock whose time conflicts with the story time.) Filming attributes could be adjusted. (Like the colorization of a black and white shot for placement in a color film.) Or, getting really fancy, one might be able to make a shot of a certain character doing something from a generic shot which contains a character whose face must be filled in. (For now, such processing abilities are of course dreams. The only capability feasible for the maximization of the library's usefulness is the specification of exactly when in the shot attributes and views of possible constraints appear, so that the program can at least make an attempt to subdivide any shot to dispose of conflicts.)

A director of the second type is the one for which I strive. An implementation would go something like this...

Library production
: The creation of a filmic source, to be accesible by the program on videodisc. This source must contain depictions of possible stories as well as limit itself to a manageable story domain. The total story must be staged, recorded, edited, and transferred to videodisc.

Film representation
: The specification of objects and rules of film which are minimally required for the generation of films from the library.

Story representation
: The specification of objects and rules of stories which are minimally required for the generation of stories representable by the library.

Library manipulation
: The minimal specifications for a program which will "direct" films built from the library. It has as input user constraints, and outputs the in-out points of shots on the videodisc in the order of intended viewing for the communication of a story.

1.4 Confessions

Given the many limitations of the second type of director, both good in terms of workability and bad in terms of myopia, one might be surprised at all that is still required to build it. The problem is that while the database of all the knowledge needed by Ingmar (from now on he will be fully personified) has been reasonably pruned, the processing implied remains dispiritingly intractable. The nemesis of AI has reared its ugly head yet again, namely, the sheer vastness of what is required to actually *do* even the simplest of tasks. (Humility is, as always, the most important lesson for the would-be god.)

And so, my project rapidly cleaved itself into two pieces: that which should be done, and that which could be done. The second category denotes the extent of the thesis that will be completed and demonstratable, whereas the first indicates all that need be

completed for that which would be done to satisfy all my original goals. The two cannot be one in the same. So an important goal is simply to minimize their disparity. It is my hope that what is done is a valid simplification of what should be done, and what should be done is sufficiently detailed on some more abstract level to support eventual realization.

What should be done is addressed in the second chapter of this document, the Synthesis. It is presented in the form of theories, abstract, open to interpretation, and of philosophical consequence. Most of the issues approached fall within the following scheme of Ingmar's anatomy...

1. Story Representation

    a. Objects

        i. Settings & Events - A priori knowledge.

        ii. Characters - Microcosms of human behavior that can generate plots, which are sequences of events that are humanly believable.

    Rules

        i. Consequence - The rules for selection of new settings or events as dictated by knowledge or character simulation.

        ii. Goodness - The rules for what makes a good story, i.e., noting the parts to be emphasized and deleting parts that are extraneous.

Story Generation

    a. Constraint - The minimal definition of a story story generation is set into motion by the user's constraints.

    b. Construction - The program builds a complete story within the constraints of the user input. This may involve substories that aid in the portrayal of plots.

    c. Editing - the story must be edited down again to retain goodness

Film Representation

a. Objects

b. Rules

i. Visual - The rules for selection of a shot based on aesthetic consequences ex. no jump shots unless jump shot has story value.

ii. Story - The rules for selection of a shot based on story consequences ex. faster cutting for action point of story.

Film Generation

a. Identification - The location of all footage which satisfies given story constraints.

b. Selection - The selection of a best next shot based on filmic rules.

c. Redefinition - If selection is not possible, then invoke story generation to change contraints (without changing ultimate story!).

Finally, the chapter entitled Prosthesis grafts some amount of respectability onto the Synthesis by recounting exactly what I did and didn't do in attempting to birth Ingmar. It is mired in matters of reality, and points the way most explicitly for further research.

The reader should be forewarned of the implications of my presentation. The order of the chapters denotes a transition from the general to the specific. The second chapter takes the form of hypothesis which are often unfettered with the innumerable examples which might be required to make the claims seem relevant. If the reader should become uncomfortable or disoriented in his digestion of the theory, then he is encouraged to skip ahead to the third chapter, wherein he should become anchored in the details which are now so familiar to myself.

My reason for this structural decision is simple. I wished to operate in a deductive rather than inductive mode. Once the reader can accept the statements of the second chapter (which he is invited to dispute), then the efforts of the implementation will

follow as cases of the theory's application. The third chapter takes on the responsibility of justification, in the form of detailed accounts of my work and its pertinance to my general motivations. This format will make sense when the reader can return to the second chapter and peruse its hypotheses independent of a clutter of implementation technicalities.

Chapter Two

Synthesis

2.1 Story Understanding

Ingmar's movie construction will be initiated by the input of a story script from a human user. This script contains the chronological constraints in which Ingmar must work to portray the story in film. The film will be culled from those images that Ingmar has available to him in the shot library. To absolve the user from an exact knowledge of the shot library, his script must contain generalities. These abstract definitions will include matters of staging, like a dinner, as well as emotive sequence, like a happy dinner, whose ending must be in some sense a happy one. Ingmar's story knowledge must be vast to support the deciphering of the abstract script requirements into the ordering of specific images from the shot library.

A story consists of a believable sequence of descriptions. The most fundamental level of believability is cause and effect, the phenomenon that there is a reason for everything that happens. If this is defied in limited ways, the story can have value as a genre type. But something must be cause and effect to make the story hold as a story.

Stories consist of settings and plots. Settings are made of simple fill-in-the-slot knowledge. There is no real reason for *why* except that that is the way we know it to be. Substantives and their many parts are settings. A dinner is a prime example, which has parts of serving, eating, drinking, and usually talking. Only a partial ordering is derivable from a dinner, since though serving must precede, eating, drinking and talking can reoccur in any sequence.

Plots are representable as settings, but not without simplifying them. Plots are the believable ordering of unordered settings. Since all possible plots cannot be made explicit, settings alone cannot represent all plots. The distinctions can be blurred,

especially since settings and plots can be nested.

The best things to represent as settings are things like a dinner, a conversation, an exit. There are actions within the settings to be sure, and associated conditional statements may be triggered, but they are expandable to some final level of detail and fall into a framelike representation. The settings do not make a story; rather they constrain the stories that might be portrayed. So a simple interactive story built from branching patterns alone is always finitely determined and therefore composed of settings.

Plots are the life of the story. There are an infinite number of plots which must be generated, not just enumerated. My assumption is that there is at a fundamental level a working model that causes a finite number of settings to be ordered in an infitite number of patterns.

For a story, the plot deciders are physical laws and character motivation. It would be uninteresting to generate stories of physical law, since these are simulations and do not pass in our culture for stories. Settings can embody physical law without too much constraint. But character motivation is absolutely necessary to a story. Why did the characters do what they did? What will the characters do next? These questions are asked relative to the audience as well as the story, and no amount of enumeration will cover all possibilities. What is needed is a model of human motivation which not only describes with summaries (as in the work of Wendy Lehnart) but also generates individual plots.

2.1.1 Settings

2.1.1.1 What is a Setting?

Settings are real world knowledge. They are expressed in terms of language.

Settings are inter-related. One setting can be expanded into a whole list of settings. The ordering of these subsettings is then determined by the plot, which can be derived from the constraints of other settings, as well as calculated dynamically. For example, the order of events in a dinner setting, each a setting in itself, can be determined by other settings, like a fight (a behavioral setting with motivational truths for humans, like one will stop eating at the end of the fight because one is so angry) or a barbacue (a physcial setting with "real" truths for humans, like the burgers are ready to be eaten after they are cooked and before they are cold). But in each case these plots, these orders of events/settings, could have been calculated from the fundamental plots, those being human reality and physical reality (each with only relative truths!) In fact, they must be so calculated if one is to be able to derive all possible plots.

2.1.1.2 What Does a Setting Look Like?

The setting is made up of statements of language. The setting may be divisible if it is a compound statement, in which case the multiple settings generated must be resolved for consequences, or it may be at a fundamental level of a single word. For example, a dinner can be expanded just by the structure of dinner, but a happy dinner is a dinner where, when the syntax allows, subsettings are happy, and ultimatley the plot is happy. (A happy dinner can thus have happy conversation, meaning the events in the conversation must end happily for the characters). Or if a happy dinner exists as an explicit setting, then that setting's structure could be followed.

The structure of settings can vary. For example, a setting with descriptive consequences, like happiness, has antonyms (sadness), synonyms (joy), an adjective form (happy), and attributes that relate to the mood of the characters. Substantive settings, like that of a dinner, have a different structure as is suggested below. This multiplicity must be handled within a class-instance representation system and

carefully strategized for the correct use by Ingmar's story generator. The class-instance system also organizes the settings for all sorts of inheritance.

An example of substantve setting structure follows:

```
name - a statement
       ex. snack (a subclass of a meal, which has
                  other subclasses like dinner)
       ex. eat
syntax - how the setting is correctly invoked, which comes
         from the rules of the language. Since language
         contains ambiguities, this syntax must express
         options, defaults, and generalities, which are
         exemplified later in this thesis.
       ex. characters have a snack
       ex. characters eat food from plate
context - higher settings which correctly include this
          setting, and can aid in disambiguating the syntax.
          Also should lead to an indication of the setting's
          relative truth.
       ex. party (indirectly a context of western culture,
                  where people always like to eat and drink)
       ex. snack, dinner
precedents - those settings which logically precede the
             setting. Settings can thus be chained to derive
             a higher setting.
       ex. char is hungry, char is thirsty
       ex. char is hungry, char get food
parts - those settings which are implied in the setting.
        Settings can thus be expanded.
       ex. serve food/drink, eat, drink, converse
       ex. swallow food
consequences - those settings that logically result
               from the setting.
       ex. character is full
       ex. plate is less full
```

As one can see, even at this level of definition there is a lot of room for ambiguity. Part of the ambiguity stems from the fact that plots are being implied, but all plots cannot be predetermined. Settings are thus useful only as starting points for generating plots, and depending on the setting, a whole grammar for the

representation of defaults, options, set relations, ordering, etc., must be invented for the setting's attributes to make sense. The goal is thus consistency within a microcosm of explicit settings for the generation of plots at the level of detail desired.

2.1.2 Plot and Characters

Ingmar's fundamental plot capability will be for the believeable ordering of shots which convey character emotion. He must have a method for simulating characters whenever he needs to justify the choice for a shot with emotive consequence.

The example which will orient the reader is that of a conversation. A conversation is a setting of many parts, like statements, questions, and replies. Some ordering of these parts is derivable from the setting itself. For instance, replies are appropriate for following questions. Obviously, the exact nature of the reply is semantically determined by the content of the question. But supposing Ingmar had at his disposal questions and replies of a generic nature, then his primary concern will be for establishing the sequence of emotional content in the conversation. A statement by a character about how happy he is with the other character is not believably answered by a replying view of the other expressing great anger, unless the rapport is within a plot that deals with the second character's overwhelming depression.

Ingmar will be driven to simulate character motivations that are consistent with the overall story for two reasons. Whenever the setting does not sufficiently determine a character's next emotional state, it will be necessary to calculate that state from the character's relationship to his surroundings. Secondly, when the next setting of a story is not determinable, Ingmar will simulate the character to yield a correct story choice. In the conversation scenario, the first option represents the determination of whether or not a character will be angry in reply to a question. Whether or not the character will even make the reply at all is a function of what the character will want to do.

2.1.2.1 What is a Character?

A character has a personality, which determines what he does as a unique human being. The personality of a character can be initially constrained by the story, as well as be devolped by the events of the generated story.

The personality is expressed in terms of the character's beliefs, and his patterns of response consistent with those beliefs. My fundamental premise is that of the pursuit of happiness, as is compatible with the utilitarian philosophies of the likes of John Stuart Mill. Happiness falls on an emotional scale from negative to positive, and represents the latter end. When a character is unhappy, he will do those things consistent with his beliefs that will bring him to a happier state. Thus, happiness is an ultimate goal, and the quantifiable result of all subgoals.

Happiness exhibits some continual rate of decay. When people have experienced something which makes them happy, this happiness will not placate them for the remainder of their existence. Even if nothing unhappy occurs, if the happiness is not repeatedly contributed to with the realization of new goals the character becomes progressively less happy due to a phenomenon we call boredom.

2.1.2.2 Beliefs

Each character has an immense system of beliefs which come about from many sources. As a living creature, the character has beliefs which include the knowledge that survival is a fundamental goal for happiness. As a member of the human race, beliefs specify what it means to stay alive, in terms of what is required by the human body to stay healty. Humans also have needs that maintain mental health. These include companionship and an understanding of the world that results in goals that work to increase happiness.

Society demands much of the individual, and as a member of the society the character wants to satisfy these requirements. He feels the pressure to conform to the laws of a

culture. Any one of these laws may conflict with the final source of beliefs, which are those of the individual. These beliefs come directly from what the character has experienced in his lifetime, and are the final determination for personality.

Beliefs have tense, which can be past, present, or future. Future beliefs lead to expectations. I propose that the frustration of expectations, even those with unhappy consequence, results in a contributing degree of unhappines. This arises from the conflict between what happens and what the character thought would happen relative to his need for understanding the world. Conflict, whether external from the frustration of expectation or internal due to mutually exclusive goals, is a cause for unhappiness.

Beliefs can be of a composite nature, comprised of any number of statements of the form *if something then something.* The fundamental pursuit of happiness is expressible as: if (if something then happy) and (something is) then happy. The character then tries to make this belief always true.

2.1.2.3 Goals

Character goals are conditional statements which link patterns of events to the amount of happiness which the character will feel when the events are realized. A shorthand for this would be to say that the character wants the premise of a goal. When the premise of a goal occurs independent of the character's actions, the emotional state of the character is augmented by the goal's conclusion. When the character has the opportunity to do whatever he wishes, he will choose the action that triggers via his goals a maximum amount of happiness.

Goals can interrelate in any number of ways, triggering other goals in complex succession. The character makes use of reasoning to follow the chain of goals to their conclusions for happiness. The character will exhibit the amount of happiness which is the sum of all the parts of happiness which he is lead to in reasoning.

2.1.2.4 Reasoning

Humans do not always act logically. This must be explained in terms of the pattern of goal recognition which lead to his conclusions for happiness. Reasoning that is incorrect in terms of all the character's beliefs can lead to the variety of human motivations which make for interesting stories. If every character did only precisely what would maximize his happiness at any given moment, the story as a whole would be an excercise in logic and unbelievable.

A character can think logically, which involves the exact following of a goal as it invokes any number of subgoals. The depth to which a character recognizes his complete goals is limited by two quantities. First, the amount of time for reaction to an event limits a character's logical reasoning. Secondly, his mental state indicates just how precisely he can follow his goals in the time available. Thus, even a logical character can be expected to act logically only so far as he can reason.

Suppose in a story a character is doing something very important to him for his happiness (like writing a thesis), and a fire breaks out. If he has time, he will make the effort to save his work before his life is truly threatened. However if he is of an extenuating mental state (like that of being under great pressure to finish the thesis by a deadline), he may not think clearly enough to get out of the burning building in time while still trying to save the work. This example suggests that when logical reasoning is curtailed, the goals closest to the character in the belief heirarchy get expanded the most. The human need to stay alive was overlooked in the immediacy of the individual demands.

Other types of reasoning which do not follow logically also influence a character. The fallacy of inductive reasoning is necessary in a world of the senses. As the character encounters life's experiences, he will develop beliefs that are not universally true. This can lead to invalid expectations, which trigger the phenomena of fear and hope. Say a character witnesses the death of a pedestrian by a hit and run driver. The event is

most traumatic, for any number of reasons which come from the character's beliefs. For some time afterward, depending on the character's exact personality, he will be afraid to cross a street. He induces an exagerrated probability of death, and operates under this pretense to suppose that he will also die.

The unconscious mind is a governing factor in much human behavior. I postulate that the phenomena is one of triggering goals by only a superficial, semantic similarity in premise. An associative pattern is followed in the unconscience, which will lead to new goals which are not warranted by reality. An illuminating example can be derived from the auto accident above. If the pedestrian was not wounded in a near miss, the witnessing character would nonetheless reconsider his own safety in crossing the street. His fear would be one of death by a reasoning process that bears resemblance to the unconscious process I have described. He foresees injury and death by associations through beliefs about accidents. He will fear his own death by induction, even though the accident he witnessed was far from fatal.

2.1.2.5 Memory

People can forget things, which can lead to fallacious behavior in the pursuit for happiness. Some forgetting is allowed by the limits of his reasoning process.

More generally, I propose the necessity for Ingmar to maintain records of individual character states linked through time. Whenever a character experiences a new event, his previous state is copied imperfectly into a new one, in which any number of beliefs may change in tense and form. The copying is done in increments of the character's reasoning, and each copy is labelled with a time stamp.

When a character reasons, he can remember his past states to augment the recognition of goal patterns. The amount of happiness expressed in the goals copied to the new state will be a function of how well he remembers matching past goals. Memory will be both objective, in terms of how far into the past the considered goal is located, and

subjective. The subjective quality of goal memory will be very important, having to do with the goal's participation in the conditions that lead to the next emotional state. Thus, the goal events which are directly connected to an emotional extreme like ecstasy are more clearly remembered than the goals which were not triggered by the ecstasy. The character will then focus by virtue of a subjective memory on a critical subset of past goals, and pursue new goals built accordingly.

An example of this phenomenon would be when even a lowly laboratory rat continually tries to receive food by pushing a button because such action was succesful in the past. If no more food arrives, the rat will still push for some time, remembering simply that the action was crucial for a past happiness. The rat will be happy to keep pushing until his mounting unhappiness causes it to give up on the premise of the fallacious goal.

2.1.2.6 The Relevance of it All

My approach to character simulation is admittedly intuitive and general. Its relevance my thesis will be obvious when the reader discovers the vast number of emotive descriptors that became necessary for my implementation in chapter 3. Therein, I mapped the emotional overtones of character shots onto numerical scales in preparation for pursuits of happiness. I pursued a number of story possibilities, and tried to emulate them within the theory of character simulation. I found myself defining such a wealth of vocabulary in terms of motivations that I was unable to tie together sufficient story knowledge for Ingmar to generate films from my particular shot library.

The reader is invited to examine one instance of plot knowledge that follows. I hope only to communicate the nature of my pursuit, and do not claim the completeness of even this illustration.

Friendship.

I had to consider the nature of friendship because of its impact upon the emotional consequences for a conversation between friends versus strangers. Put simply, when friends are seen to fight, it is understood that the fight will usually be of a temporay nature. Friends can argue, and perhaps should since no two persons can ever be identical in their beliefs, and yet friends tend to explain their beliefs in conversations of great detail. If the friendship is of any depth, a fight will be followed by a period of making up, which can include any number of apologies. Conversely, if friends do not resolve their conflicts, then the friendship was of no great extent to begin with. Ingmar will need such an understanding if he is asked to believably construct a film about, say, a happy dinner between good friends from footage that indicates a fight.

My belief assumptions are as follows:

Friends will be happy when each is happy, thus they want to be nice and remain friends. Friends trust one another, so they believe whatever each one says they believe. If what is said conflicts with individual beliefs, then there is an occasion for argument. If the conflict stems from an insincerity (a statement made by a character that conflicts with his own beliefs) then the other character will have an instance for distrust, and the friendship will deteriorate. Friends also respect one another, so if the conflict is found to be true, then the friends will note this difference of personality but will wish to remain friends. However, people want friends who are similar to themselves, so if the differences compound for important beliefs then the friendship will again begin to dissolve.

The reader is invited to consider a scenario between two friends, Ingmar and Federico. It is written in a psuedocode form for brevity, but should be intelligible. In this story, Ingmar begins by wanting to do something nice for Federico.

```
ingmar make cake1 for federico
 then cake1 (owner ingmar)
 then ingmar wants (federico like cake1)
 then ingmar wants (federico own cake1)
ingmar gives cake1 to federico
```

```
 then cake (owner federico)
  then federico is happy ;; people like to get gifts
  then ingmar is happy
  then federico is happy ;; reflexivity
federico eats cake1
 if federico says (likes cake1) and (federico hates cakes
                  or ingmar believes federico hates cakes)
  then federico says (likes cake1) insincerely
   then ingmar believes this ;friends can often
                             ;tell when each is lying
    then ingmar believes (federico hates cake1)
     then ingmar is unhappy with federico ;he betrayed trust
      if ingmar wants (like federico) most of all
        then ingmar apologizes to federico about cake1
      if ingmar wants (federico like cake) more
        then ingmar is unhappy with federico
        ;; federico can apologize or start a fight
      if ingmar believes in sincerity
       ;;(is happy with (char verb-phrase sincerely))
       then ingmar is unhappy with federico
```

2.2 Film Understanding

Assuming that Ingmar has an understanding of stories amenable to the needs of a filmmaker, he now needs an understanding of the filmic language to decide how to portray a story. His comprehension stems directly from the many rules for film construction that have been invented by human filmmakers in the long and complex history of the cinema. Film theory is a rich and subjective domain, and increases in detail with every new book published and every new film produced. Ingmar will need an appreciation of all this, but in as simplified and algorithmic a form as possible.

Ingmar's filmic perceptions must not be of an absolute nature, any more than his grasp of what motivates people in stories can be based on universal arguments. Everything is relative. Ingmar's success will depend not upon adherence to specific rules, but the dialectically motivated judgement of when and when not to use any one rule. This can only be accomplished by the ongoing permutation of fundamental

variables of film and the attachment of names and results of past usage. The direct input of existing filmic rules must be made in terms of these fundamental attributes so that Ingmar may recognize their inversions. Between films, Ingmar can even contemplate new filmic rules which no human director has ever used and store these for subsequent experimental use, success of which would necessarily be evaluated at some point in human feedback.

To make this argument a bit more concrete, let us consider an example. A fundamental variable which Ingmar might have no guideline for setting could be the movement toward or away from the camera of a character in a scene where either is acceptable by story evaluation. Ingmar notes this distinction, and generates two films, one of each possibility. He then asks us what consequence his choice had for human viewing. We would answer that when the action came toward us, we felt more involved in the story, and genuinely threatened if the image was a frightening one. When the action moved away, we felt distanced and relieved, as one might wish to feel in a denoument. Ingmar then stores this knowledge, probably in condensed forms of psychological attribute versus a pattern of film values. Next time, Ingmar will want refer to these new rules in many ways. He may be forced to choose a shot which he will now decide is inappropriate because of these rules, and thus go back to changing the story. For example, the only image of kindly Grandpa making tea at the end of a story propels his rotten teeth toward the viewer, so it would be better not to have him make tea at all. Or Ingmar may choose an image for the effect suggested by these rules, as in selecting toward movements in shots for the climactic point of a story. Lastly, Ingmar may be asked to use a rule by name, as with an invocation of Hitchcock's additive principle, for such is often called a usage of toward movements to escalate threat in a story, in recollection of that director's common practice [Scott 75].

Note that even on the level of these simple illustrations, an issue of great portent has been discovered. How did Ingmar decide when an image was frightening, as with Grandpa's rotten teeth, to know that if moving toward the viewer it would result in

threat? This is nothing less than a story in itself, the story of how the viewer will interpret an image. Ingmar must have some generic character knowledge of his viewers so that he can run simulations in the style of the previous section. The evaluation of a shot's effect often will incur story knowledge. This is a special interdependence between story and film.

Of course, these examples are also very simple in the sense that they occur all within just one shot. Many of the most important filmic conventions operate across many shots, involving the sequences of many attributes. Rules so derived can not only affect just the psychological impact of the images, but even their very comprehensibility. Thus we address the prescriptions for overall film construction, like the interplay of wide and close shots to logically introduce new image information.

In the sections to follow, I will suggest rules for film construction as derived from the many common precedents of cinema. I will attempt to keep them all in the context of the fundamental film values which they reference. Above all, it is my hope that my presentation communicates an approach of methodical definition which will support ultimate realization in Ingmar's programming.

The reader is to be reminded of the exact role of any and all fimic rules which Ingmar will recognize in his ability to construct movies. Ingmar seeks a shot based on story content to fulfill his story script. From the range of shots which roughly satisfy this need, he then considers any filmic rules as will apply to each shot if it were to be appended to his film under construction. Any one shot may trigger a wide variety of filmic consequences. Depending on Ingmar's need for these factors in his movie, he may be motivated to select a certain shot. This shot can in fact impose more constraint than Ingmar genuinely requires, as with a shot that communicates a strict sense of space for a film in which spatial perception is by far secondary to abstract impact. If no shot sufficiently satisfies those requirements that the script calls for, either filmic or story derived, then Ingmar will return to the script and apply his story understanding to change the selection criteria. In this way, Ingmar oscillates between

script/story knowledge and shots available/filmic knowledge to ultimately produce the best possible movie that portrays the user's original script.

2.2.1 Perspective

Perspective is one of the fundamental attributes of a shot. It is context dependent, so only in exceptional instances can it be explicitly constrained in the shot's content definition.

Perspective refers to whose eyes one is seeing the image through. It can be one of two basic types; a character or omniscient. A perspective is determined from the camera's position respective to the characters known to be present in a scene. When the view could not be through the eyes of any known character, the perspective is omnscient. An omniscient perspective could be the viewer himself (as when a character adresses the viewer via the camera), or no-one (as when, although a character adresses the camera, he is speaking his private thoughts and no person is implicated).

Perspective is a critical tool for film narrative. It can change from shot to shot to reveal the different parts of a story that are not visible from any single perspective, as we are doomed with in reality. There are rules for the establishment of new perspectives, and the perspective chosen dictates what shots are logically possible. The choice of perspective can force the viewer's emotional response by making him feel what the character feels (empathy and subjectivity) or giving the viewer psychological space in which to judge a scene (objectivity). The interplay of perspectives can change the portrayal of a single story, and thus the story itself, in a number of ways.

2.2.1.1 Establishing Perspective

I propose that the very first shot of any film is assumed by the viewer to be omniscient. As that shot progresses, it may be revealed that the perspective is that of a character. One way to do this would be to have a character address the camera by

another character's name. (see Character Perspective)

Another method that one might consider would be to swing the camera around to reveal a character whose line of sight is that of the opening view. But in fact, the perspective remains omniscient. The effect would be only to align the viewer closer emotionally to a character. This trick falls into a category of omniscient shots which nonetheless communicate empathy for a character. Another such shot would be of the over the shoulder variety, as Bergman used effectively in many of his films, notably Persona, to help the viewer maintain objectivty while keeping him emotionally involved in the story. (Already, useful rules are suggested by example.)

As subsequent shots reveal to the viewer the spatial locations of characters, what would have been omniscient views are evaluated by the viewer as candidates for character perspective. The viewer judges by considering the limitations of the character's being: his position with respect to what he might see (particularly in reference to landmarks), how the character feels (as might be communicated in the mood of a shot), etc. The viewer assumes that characters stay about where he saw them last, taking roughly into account any attributes they may have exhibited, like walking. See the section to follow on details of character perspectives to get a sense for all the qualifications involved. When a new view could not be a character's, then it is again omniscient.

There are forms that cause the viewer to anticipate a possible change to character perspective. The most explicit and reliable is by eye following. In this case, a clear view of the character's eyes is followed immediately by what they see. (This second shot could still be omniscient, framing the character and some viewed object , but the point is that the character's perspective succeeds unambiguously.) The qualifier 'clear view' is relative to the character's emphasis (see story section to follow).

Of course, there are still many constraints involved in eye following. The two shots must be consistent in all spatial and character implications, like surroundings and line

of sight. When these are not consistent the viewer will try to infer a passage of time. The viewer may become confused, and could interpret the view as the character's thought, particularly when the view is very inconsistent with the character's present setting. The thought is then judged to be a memory or a dream (wishes), depending on what the viewer knows about the character and the nature of the view. If the view is a logical near miss, inexplicable with a small passage of time, a psychotic state may be implied.

The mention of dreams brings up the reinterpretability of perspectives. Everyone is familiar with the hackneyed plot device of, well, it was all just a dream. This may be accomplished by seeing a character wake up just when the viewer was about to give up on making sense of what had been seen. Other devices can cause the viewer to reevaluate the perspectives of past shots. One example is what I call the introduction of a spy. Suppose a series of omniscient shots spatially suggest a character's perspective, but no such character has been introduced. Then when a character is seen in that location, (perhaps by eye following the spy's victim to the spy) the previous shots are suddenly understood by the viewer as those of the new character. With this realization may come types of discomfort on the part of the viewer for having been so duped, especially if the spy's views were not overtly suggested as in, say, a tawdry flick where the camera moves slowly in the window and through the drapes. Of course, this more obvious construction has its value precisely for the reason that an unknown character was implied. Then the viewer feels suspense and excitement as he awaits the spy's discovery.

Notice that while in theory the succesion of perspectives can result in very complex story interpretations, in practice the viewer limits what possibilities can work. When there is story or film precedent in the viewer's domain, strange tricks like the 'it was all just a dream' can be played. But the average viewer will become easily confused if too much is expected of him, and Ingmar should always be prepared to mark a rule as currently impractical for lack of familiarity.

2.2.1.2 Character Perspective

There is no clearer method for distinguishing between what the character knows and what the viewer and possibly other characters know than by exhibiting shots from a character's perspective. When something is seen in a character's eye, the viewer knows *exactly* what the character sees of that something.

The most important limitation for the view of a character is that it can only move with the character. The perspective is spatially determined by the character's position, movement, and line of sight. Many rules are necessitated by this simple fact. If a character's perspective changes a lot with respect to the surroundings, the viewer will try to make sense of it as movement with a passage of time. If no amount of time allowable by the story could account for the change, the viewer will assume a new perspective. Thus a simple simulation of the character's movement is necessary to recognize, for example, what views are possible with a simple turn of the head. But when even the smallest movement is implied, there is still room for confusion and the best remedy is always to eye follow from views of the character making the implied movement.

For example, in the film "Kamikazi '89" (the last film Fassbinder starred in before he died), Fassbinder's character is moving downward in an open freight elevator. The director alternates views of him in the elevator with views from a downward moving camera of factory floors. Even with the simplicity of the deduction that the factory views are in real time and from Fassbinder's perspective, many clues make this as recognizable as possible. Lights play upward on Fassbinder to suggest his downward movement, and one hears the sound of an elevator. Each factory shot is preceeded by this view of him, and begins from a height greater than that of a standing human being. Fassbinder's eyes are clearly visible and follow the floor shots. It is these details which make the sequence work, despite the fact that the factory shots *are visibly not from where an elevator would be.*

Other facts of a character's perspective can serve as immediate clues for the viewer. A previously mentioned example was the verbal identification, as by name, of one character by another. Another is the visibility of body parts, as in a hand writing a letter (a truly omniscient viewer has no body).

Of course, the latter image could as well as have been explicitly introduced by a shot of the character picking up pen and pad, or otherwise indicating that he was about to write a letter, creating an expectation more of story than image. Story expectation can always help to identify a character perspective. An important case frequently occurs in dialogue. The normal alternation of character speech as a kind of story anticipation harmonizes perfectly with the most natural of eye following. The result is the ability to dart back and forth between the characters' views of one another with perfect intelligibility.

Lastly, consider the usefulness of camera attributes that are consistent with *how* the character would be seeing. Such perspective clues are best in conjunction with other constraints, and probably wouldn't be available to Ingmar unless he had some rather advanced video processing resources. A good example is given in "Mother", by Pudovkin, when a shot of the mother waking up is eye followed by a shot of the room around her which is initially unfocussed and then clears.

A shot from a character's perspective is moored in the realities of what it means to be that character, and more generally, a human being. Any one of the limitations of the character's perspective can then serve as a clue to the viewer for the linking of shot perspectives to characters. The violation of the perspective limitations in a shot which is overall distinguishable as being from the character's viewpoint is a clue to mental states (like flashback, planning, hallucination, etc.) or the passage of time.

2.2.1.3 Omniscient Perspective

The omniscient perspective is the wellspring of the magic that is cinema.

The omniscient perspective ostensibly imposes no restrictions, for it is not imprisoned in any body. Anything photographable can be and is displayed to a viewer who accepts the image with none of the earthly qualifications to which he is otherwise bound. Every portrait of the story dodges corporeal law to explode in the imagination, and in exactly the detail which the filmmaker intended.

That our minds should be able to accept the input of only two senses, sight and sound, to so vividly construct an alternate reality is almost incredible. It can also be dangerous.

2.2.2 Synedoche

> Synedoche - a figure of speech in which a part is used for a whole, an individual for a class, a material for a thing, or the reverse of any of these. (Ex.: *bread* for *food*, the *army* for a *soldier*, or *copper* for *a penny*) [Webster's New World Dictionary]

There is a usage of synedoche in the filmic language which is critical for the communication of any story. Since the viewer has only the window of the filmmaker's choosing with which to look out upon the story, it is the burden of the creator to use synedoche effectively to maintain the viewer's unquestioning fidelity to that window. Every shot must have a purpose in keeping alive the illusion of an entire world. When a small part of a whole is thrust in the viewer's face, he must be able to induce the whole. And when a whole is presented, its parts must continue to be suggested. When this kind of synedoche is violated, the viewer is at best frustrated with a feeling of having missed something, and at worst completely lost.

A story told entirely through character perspectives would be automatically strong in synedoche. As long as the viewer is not denied critical character views, every scene will be lucidly presented by virtue of how we, as human beings, take in our world.

The problem arises when the film, to be of any real interest, condenses the story in strategically selected omniscient views spiced with the occasional revealing or comfortable character perspective. To maintain good synedoche, filmmakers have invented many clever cinematic prescripts.

2.2.2.1 The Establishing Shot

To communicate the spatial arrangement of any scene it is vital to use critically timed establishing shots. These shots explicitly depict as much of the whole as possible so as to guide the viewer in understanding the context of the parts. Establishing shots are necessarily wide, and have little emotional value due to their inevitable distancing effect. It could be said that all theatre is made of establishing shots, which is why so many films which rely on close-ups cannot translate to the stage.

Establishing shots are needed by the film viewer whenever the spatial relationship between two or more shots is unclear. Though there are techniques to delay the need for an establishing shot (these are reviewed in the next section), sooner or later the viewer will conciously demand an overview of what is going on. Even when the viewer has firmly in mind the layout of a setting, it is good film practice to reaffirm the viewer's spatial expectations with the prudent use of visually interesting establishing shots.

Exactly when to use establishing shots is extremely difficult to define. Ingmar could very well precede every scene's action with its establishing shot, but this would be predictable, boring, and even unrealistic. So unless the use of wide shots has psychological value, a good film makes judicious use of minimal area establishing shots. Such shots can be postponed to lend impact to closer shots, and this delay will cause tension in the viewer between the desire to see more of the part and the need to see the whole. Ingmar needs a decision process to recognize when an establishing shot is absolutely necessary, by spatial as well as story constraints. Ingmar needs to appreciate precedents, like those of the induction principle of Alain Resnais [Scott 75].

In "Hiroshima mon Amour", the very first shots are those of the oily skin of intwined lovers, over which is dubbed their enigmatic conversation of strange and disturbing memories. The skin is so close that one is struck by the abstract beauty of its pores and folds and beads of sweat. So powerful is the image that for a few hypnotic moments the viewer does not even crave an establishing shot for this mysterious movie that has only just begun.

2.2.2.2 Preserving the Whole

There are countless proven methods for maintaining the relationship of part to whole in close shots. Many interact synergistically, and have direct consequences for the continuity of the film (see Continuity). Whenever spatial relationships *do* change significantly, as may be demanded by the story, or even suggested by strong combinations of camera movement and gesture, only an interesting establishing shot can alleviate the confusion of the viewer who has lost his orientation in the space.

A progression of succesively closer shots can busy the viewer with increasing detail, and postpone spatial relationship. This trick cannot last, and the viewer must eventually be pulled back either to the same scene or an entirely new setting.

Another formula might be to follow character perspectives through some action. Such shots could be substituted with more omniscient shots that serve a similar narrative purpose, like of the over-the-shoulder variety. It feels natural to evesdrop on a story in this fashion, and the viewer's spatial models are reinforced with the orderly presentation. A conversation built in this fashion can be sustained indefinitely without the need for more than one introductory establishing shot (though it might get tedious).

The more flexible usage of omniscient shots which have little relation to the characters' lines of sight demands careful technique. The more close-up the shots are, the more potentially disorienting they will be, and their unambiguous presence will

require stricter adherence to the rule(s).

The simplest such technique would be to conduct the viewer through the setting in an orderly fashion, taking small steps much as the viewer would himself if he were in the space. If the mental leap between omniscient perspectives is logical and predictable, then a surveillance sequence so constructed can precisely suggest the entire scene and obviate the need for a single panoramic establishing shot. This approach would be necessary when the setting is so huge that no coherent overview could be photographed. Use of this maxim can also result in strong continuity.

Sound as a sensory information source in film may be secondary to image, but is not to be underestimated. Artisans of radio drama are familiar with the potential of sound for synedoche. In a movie, otherwise disconnected images can work together to advance a coherent space if only they are accompanied by unifying background noises. This approach is particularly useful in a visually complex environment. A confusing sequence of machinary can be established as located on a single factory floor when their combined sound is maintained across the shots. Then for each closeup a mechanism's specific sound may be accentuated to re-enforce the viewer's perception of married space.

Another technique which is often planned into the shooting of a film is the use of visually unique objects to serve as landmarks for the spatial arrangement of a setting. Every non-establishing shot in the scene must include a sufficiently recognizable glimpse of a landmark. Since such elements are unique to a specific shot library, Ingmar could only apply this method as an afterthought in his analysis of the shot representations. Ingmar has to make some tricky judgements about what people will remember seeing, and he might have to do close ups just to point out the landmarks. Executed properly, this routine could lead the viewer unambiguously about a setting for even long sequences.

A different crutch would be to use the screen as a representation of the setting. The

dimensions of the screen become roughly translatable to a larger plane of view. This scheme is remarkable interpretable by the trained viewer as long as the planar position of the view is not grossly violated. (And most people today are well domesticated viewers.)

The effect is to preserve the directions as well as positions of the parts of an image, as well as to impose rigorous continuity. For an example, consider a conversation portrayed only in side shots, rather than in a previously mentioned style of character perspective volleying. Whenever Ingmar shows a close up of a character, that character is on his or her side of the screen, and facing in the direction of the other character. This powerful presentation mode is typical of dialogue sequences in early sound films, and is addressed in [Bloch 87]. The style is probably a result of a theatrical approach to staging, and is rarely used in current movies due to its dated appearance.

Perhaps the most intelligent option to moving about in a space is also the most general and difficult to analyze. It involves the guiding of the viewer by means of the story. Story understanding, ie. the understanding by recognition of all the parts of the story, will lead the viewer to anticipate a number of views. Real world knowledge allows the viewer to expect many details, and he looks for these details to confirm his knowledge. The viewer gratefully accepts new images so long as they do not contradict the expected. Establishing shots are then useful when they reaffirm the expectations, however they must not be too blatent for fear of destroying the viewing adventure.

Several illustrations will make this clearer.

a. Suppose a fragmentary establishing shot depicts a woman standing in a plain and bare corner, next to a window. Ingmar can display a shot of a bare wall and the viewer will assume that this was the woman's view of the other side of a room. An amazing amount of induction has taken place, without the benifit of a real overview! Because the woman was the only character visible, eye following was possible from a rather

wide shot. The corner fit into the viewer's mental model of a room. And because the room was so bare, it was easy to make the wall view inferrable as opposite from the woman in her line of sight. If that wall had had, say a painting, the same cut would have still implied the opposite wall, but a story expectation might have been triggered.

b. Imagine a classic case of the match cut (editing to follow an event in time, though usually not real time). A character enters a room through a door. Because the viewer knows exactly what to expect in terms of actions and views, the scene can be pieced together from any number of shots. Omniscient perspective can jump all around, through the door, to the character's face, onto the door handle, and back again with perfect trenchancy. The only limitations might be of continuity and time. The action is so familiar that a lot of time can be cut out and the viewer won't even think twice. In fact, a sequence that is too long would be *less* believable (see Time and its condensation). A classic approach is one in which we briefly see the character reach for a door handle, followed immediately by a shot from inside of the door already opening as the character steps through it. The soundtrack swiftly overlaps the sound of the handle turning, the door opening, and the character's footsteps.

c. A character's narration can carry the viewer around a setting with perfect clarity. If the character is, for example, describing the room about him, as in train of thought, a sequence of images of that room can be used to reflect the description. The viewer will need an establishing shot if he is ever to know how the room is layed out, but at the time of narrative the viewer will accept whatever is shown as being in the room.

Lastly, it should be noted that within the filmic synedoche lies the entire problem of set consistency. Obviously, Ingmar has to keep running track of what the viewer has seen to know when a new shot contains a contradiction that would destroy the synedoche. The confict could come in any number of mismatched spatial or photographic attributes. For example, in case a above, if Ingmar shows the painting he must remember that the viewer thinks a painting is across from the woman, even though the wall shot might not even have been from the same room. If the wall shot is

in fact from a different room, Ingmar better have ascertained that things like the lighting and color made it compatible and thus believable with the shot of the woman in the corner.

2.2.2.3 The Vignette

There are times when the environment for the story events is ill-defined, and no number of establishing shots can ever locate an activity. Such would be the case of a crowd scene. In this instance, establishing shots might be used to communicate the phenonomenon of a crowd; however, our knowledge of crowds alleviates the *necessity* for such a shot.

Close ups in abstract environments take on the quality of vignettes, whose only relationship to the whole is as some part. Any number of sequences can be strung together in various orders to depict the whole and make it conceptually manageable for the viewer. Local spatial constraints become relaxed. The viewer becomes interested in what the individual vignettes have to say; as the vignettes are presented, inductive reasoning causes the viewer to form judgements about the meaning of the whole.

While the viewer will readily accept the chaotic presentation of vignettes, the environment they are part of becomes in turn an object which is desirable to locate. Hence, establishing shots of a crowd in relation to its surroundings become relevent and necessary as the crowd moves toward, say, the capitol.

2.2.3 Continuity

Apart from the adherence to rules of perspective and synedoche, many film editing decisions are made based entirely on aesthetic constraints. There are guidelines for maintaining the fluidity of a film. When these are broken, the story may remain intelligible but an illusion of reality has been shattered. The response of the viewer will

be one of surprise and frustration, which may or may not be desirable in a larger story context.

Ingmar must have knowledge of many criteria for continuity, including the following.

2.2.3.1 Cut Points

The selection of the exact point at which to cut into or out of a shot almost involves more art than science.Ingmar must also understand the concept of sequence, that is the series of shots which form a visual unit, versus the ordering of sequences to form a complete narrative. Most of the comments which follow pertain to the creation of a sequence. As they apply to the beginning of the first shot and the end of the last shot in a sequence,they also relate to the joining of sequences. With a few precedents, and a lot of real world knowledge, Ingmar might fare acceptably at the art of cut points.

Most cuts are best made on gesture, or to paraphrase for generality, the pauses between completed movements. To decide in which movements to look for pause, Ingmar will have to consider the relative emphasis of the parts of any specific shot. And to know when the pause occurs Ingmar needs an understanding of the movement.

The formula can apply to the movement of the camera itself. When an image is relatively static, as with a wide shot of a formal dinner, cutting should be made when the camera is as still as possible. Of course, looking for the camera's pause can include or exclude important detail, or otherwise interfere with the story presentation, so this rule must be frequently overridden. For intance, if the only establishing shot of the dinner was a pan across the table, it might be far more important to cut at the view of a certain character than complete the shot, perhaps to eye follow to their perspective. The speed of the camera movement will be an important factor for cutting, indicating how recognizable the last frame will be. A swish movement is one that is so quick that the view is blurred. Cutting from a swish is almost always good, simply because the viewer wants to stop the image to discern detail. Cutting to a swish will have

psychological value, as the viewer will feel surprised and accelerated.

Cutting for scene movement involves looking for the completion of some most visible and completable activity. Such completion may imply finality or propose more movement, and in the latter case Ingmar may wish to consult thrust matching to absorb the momentum of the shot. (see section to follow)

A few classic shots will provide insight. In the first, a man posessed with emotion delivers an ultimatum which he punctuates with a blow to his desk of his fist. The cut is made just as the fist slams against the desk. Any shot at all could follow. In another shot, the man walks toward the camara with animated stride. The cut is made as either foot hits the floor. But now there remains a momentum, because the activity of walking continues. If this momentum is ignored, some succesor options might unpleasantly jar the viewer.

Ingmar will routinely encounter very tricky decisions in his quest for the best cut point. The interactions of many scene and camera movements will make the problem one of choosing the lesser evil. In all cases, cutting must be kept in perspective of more important issues, like what should and shouldn't be displayed to serve the story. Even in the best of movies from human cinematographers, awkward cut points can be forgivable, desirable, or even unnoticeable in the larger narrative context.

2.2.3.2 Jump Cuts

Jump cuts occur whenever a visual similarity at the boundary of two shots causes the viewer to feel unpleasantly yanked to the second image. This effect is dominant for the straight cut, and may be diminished with the use of other transitions which effectively communicate the passage of time (see Time under Story). Jump cuts can have important stylistic value. The work of Godard, quintessential in evaluating the filmic intent of the French New Wave, is repleat with jump cutting. In "Breathless", a woman riding in a convertible is shown in a series of views all taken from the same

angle and distance. No visual transitions are offered between the shots which, what with the woman's gestures, her sweeping hair, and the passing street, cannot possibly match.

When Ingmar determines that two shots constitute a jump, his most direct option will be to reselect the second view. To stay within the story requirements, this route may entail just the identification of another shot which portrays the same subject from a perspective of greater spatial difference. In fact, it is good style to keep moving the viewer about a space in distinct and appreciable increments. It will be a challenge to Ingmar to maximize this practice while staying within the confines of forms for synedoche and other types of continuity.

Jumps between necessary shots are easily avoided with the use of unrelated images as buffers. Whenever two shots indicate approximately the same distance and angle to the same subject, Ingmar can find a view of a different subject (usually in the proximity) to insert. Of course, the image in the buffer shot selected can have disasterous story consequences if not properly chosen. Furthermore, the buffer must serve the needs of perspective, synedoche and continuity. For all the trouble to fix them, some jumps might be best left as is.

Consider the Godard jump example above. To repair the situation, Ingmar would look for other shots related to the car setting. A shot from outside of the car speeding past would be effective, but distracts from the woman. A closer shot of the woman would break the jump, but then the shot would relate less to the driver's view of her. A jump to her perspective would be nice except that now the viewer is in her perspective. This would suggest views of the driver, or build viewer empathy for the woman. This jump cutting has very definite story repercussions, enforcing realism and objectivity, and any attempt to change the sequence will destroy the effects. With a tip of the hat to the likes of Godard, Ingmar must realize that sometimes a jump cut is just the trick to make a sequence work.

2.2.3.3 Limits of Rotation

A continuity consideration which also has correlations to synedoche is that of the degree of rotation implied between two shots. It is an interesting fact that the human mind is far more adept at making linear than rotational displacements through an imaginary space. If the perspective movement involved in the transition to a new view includes rotation, the amount of image porcessing required by the viewer to orient himself increases as the rotation approaches 180 degrees. The effect of a 180 degree shift is to possibly push the viewer into a state of confusion. This confusion may not ultimately destroy the synedoche as the viewer reorients. But a most unpleasant feeling of having been jerked about will diminish the viewer's sense of filmic continuity.

One kind of rotation Ingmar must look out for is that of the camera around a subject. Whenver a single subject of the overall image is transcendent, an edit must not take the viewer to its opposite side. This should especially be true for similar views which are distinguishable only in direction. The front and back views of the human are are far more differentiable than the sides, which are mirror images and present opposite sight trajectories.If rotation about something is required, Ingmar must lead the viewer in steps of less than 180 degrees. The succession of shots thus derived will work especially well if, in avoiding jump cuts, Ingmar also tries to vary the distance to the subject.

When the subject of a scene becomes a space strongly oriented by opposing directions, Ingmar must again beware of rotations that bring the viewer through 180 degrees. Examples include conversations between two people and battles between two armies. In these cases, the intelligibity of the story as well as the synedoche can hinge upon the preservation of directions. Ingmar must be dissuaded from extreme rotations even by comfortable increments until such scenes have been logically concluded.

A more fundamental rotation problem arises when Ingmar contemplates his viewer's

models of real world knowledge. Since our world is one of gravity, every example we have ever known of objects has an orientation to downward. While many things were small enough to allow rotation and the construction of mental maps independent of gravity, other iconic knowledge, like that of buildings, is most familiar at its 'normal' views of street level and headon.

The consequence is twofold, and relate directly to synedoche.

Firstly, when an image is presented in its realistic, headon/eye-level perspective, the viewer expects down to be the bottom of the screen. This is obvious really. When we see a character walking toward us in a film, if he is not right side up we'll think the director is up to something strange. A little tilt might be stylistic, but a lot, like up side down, would be laughable.

Secondly, whenever images are displayed from other than the normal, down oriented viewing angle, Ingmar must realize that there is potential for confusion as the viewer compensates. A new sense of downwardness, independent of the screen or even the viewer, must be recognized. The subject of the image may even involve image processing for a model with another, different down. Thus, a view from a ceiling down onto a standing character is confusing, but not as confusing as the same perspective of the character lying on the floor, especially if that character's feet direct upward on the screen! A shot upward of a tall building is not even that abnormal in terms of daily perspectives, but since the viewer has gained a new down, one that points out at him from the screen, it also takes a moment, however small, to acclimate. In this latter example, if Ingmar is lead to the upward shot of the building by a story requirement (say, to see an open window out of which someone will fall), he notes the discontinuous change of rotation and seeks transitions to the view. The simplest would be to find a street level shot which tilts upward. If this is not available, Ingmar searches for a street scene in which a person looks upward. The transition to make the sudden rotation would then be an unambiguous case of eye following.

As Ingmar builds sequences of shots, flavoring the mix with any number of interesting perspectives, he must decide how the shots will affect the viewer's sense of orientation. When sudden leaps of rotation are made, Ingmar must choose either to make the rotation more comfortable by adding shots, keep the transition for stylistic reasons and perhaps display the image longer to allow the viewer to adjust, or rework the sequence at a story level.

2.2.3.4 Thrust Matching for Momentum

Thrust matching should be considered whenever the cut point of a shot is such that movement is suggested. Since film is presented (currently) in two dimensions, it is those movements which cross the screen that hold the greatest momentum. In thrust matching, a strong movement expectation is answered in the next shot with another movement that may begin about from where (on the screen) the suggested movement would have begun. The thrust match builds continuity across the shots and makes the edit appear more precise and filmic.

If the movements to be matched are of the same action in time, by the same subject, then the matching is critical to a sense of progress. A familiar case is that of a character striding across the screen. If the direction of movement is not all too literally retained through a sequence, the viewer may get the impression that the character has gone in circles. Such a frustration of advancement may or may not have story value. The thrust matching of this and other single actions will also challenge Ingmar in the eschewal of nearly unavoidable jump cutting. In the instance of the man walking, methods like waiting for the character to walk off the screen and then starting him again on the other side (a sort of wrap-around effect) address the problematic interactions of thrust and jump.

If the suggested and matched movements are similar but arise from different subjects, the edit will be fluid and inviting. Also, the viewer will seek metaphor between the dissimilar images, and the enforced vector of momentum can have allegorical

significance. An excellent instance of this is found in "Aguirre, Wrath of God", by Werner Herzog. The entire story is one of a Spanish expedition moving inexorably toward doom in the Amazon jungle. A collection of metaphors are used throughout the movie to evoke this fate: movement downward (out of the Andes and down the river) toward death (in Herzog's symbology, circles and the elemental). As the film opens, we begin from a great distance in the sky and move down to follow the expedition, complete with weapons including most unwieldy canons, as it winds its way down a steep mountainside. The struggle is slow, dogged, and dreamlike. We close in increasing detail, up until we are behind a soldier whose gaudy helmet drops before us as he descends the precarious path. Suddenly, Herzog thrust matches to a cannon falling from a cliff and exploding into the river. The viewer is riveted with the action, and the edit exquisitely withstands analysis.

2.2.3.5 Audio Flow

When some violation of strict visual continuity is inevitable, methods for controlling the flow of audio may ameliorate the situation. Whether it be theme music or, more relevantly, the sounds of the story, audio can have a transcendent unifying effect on a sequence of shots.

Sound is generally used to accompany image, but it can be an unfaithful partner. When the audio is that of a greater scene, it is natural to preserve the location's ambient sound , even for those shots of silent subjects. This is a simple maxim for synedoche as well as continuity. In most narrative movies, sound tracks are built in the studio using 'clean' ambient, dialogue, and music tracks. This is necessary to maintain continuity by avoiding such undesirable sounds as that of a passing plane, which is particulary problematic at a cut point.

Matters get really interesting when the sounds of distant images is overlapped to express continuity. Consider the case of a character hurrying out of a building to catch a taxi. The door is seen to be closing as the character rushes through it, but the actual

sound of its slam is not contained in the time condensed shot. The slam must be heard however, and will be when the character has already vaulted into the street. Thus, the sound of the door must unnaturally loud in comparison to the street noise of the next shot in order to be discernable.

The use of sound introduces innumerable complications to Ingmar. Many of the filmic forms so useful to this day were developed in the silent era of cinema. Without the burden of sound, innovators like Sergei Eisenstein were free to nurture film theory in purely visual terms. Eisenstein himself referred to his discoveries as being within a realm of optical counterpoint, and fairly dreaded the time when the optical and acoustic would interract [Scott 75]. It may be observed from the lag in technology that the use of sound still constitutes a frontier, wherein directors like David Lynch experiment, as with the amplification and distortion of the otherwise subaudible to justify dream-like sequences. It will be enough for Ingmar to use sound in the simplest of forms to support the visual phenomena of synedoche and continuity in the portrayal of story.

2.2.4 Story

Once Ingmar has acquired the more fundamental techniques of film to smoothly depict sequences, he will require a more general knowledge of how to edit for story effect. His directing accomplishments will not be acceptable until he can apply the vast number of precedents for story representation which render film the most subjective and powerful of media forms.

2.2.4.1 Emphasis

The wonder of cinema stems from the filmmaker's mandate to show the viewer exactly what he wishes. It is vital to display the events of a story as clearly as possible, to the exclusion of extraneous material. Because the viewer is forced to witness the story via the screen, every image will be interpreted in the context of the story. Each view will

be assimilated for its significance, as if the viewer himself were within the story and willfuly chose to look in the predetermined direction.

It seems obvious then that Ingmar will need an acute sense of what his audience will see in every shot. It will not be enough to select those shots which contain the image required by the story. Ingmar needs to find the views which best emphasize the script details. He will consider the degree to which a subject is framed by the camera, the nature of the subject's optical impression (like its movement and color), and the extent to which the surroundings may distract from that subject. Ingmar may use the consequences of many other filmic rules to decide when an image has been emphasized, as by the subject of a thrust match. From the need to emphasize parts of a whole, Ingmar must generate a suitable pattern of perspectives, for example with the closing in on a subject from an original establishing shot.

Ingmar must repeatedly tap his capacity for story understanding to know when an image deemed important in initial shot selection is condusive to the story's portrayal. The viewer will notably feel empathy for a character that is repeatedly emphasized more than others. If the shots available to Ingmar present, say, the bad guy in closer view than the good guy, Ingmar may need to skip valuable shots simply to avoid misplaced emphasis. When considering a shot of some outstanding emphasis, Ingmar should be able to evaluate its consequence for the story as portrayed in the film to that point. When he has been lead irrevocably to a choice for inconsistent emphasis, Ingmar will need to backtrack and replan the film for a better shot selection.

2.2.4.2 Time

Time is remarkably pliable in film. Aside from the director's obvious ability to alter the chronology of events, for each such event the amount of time passed in the telling of the story and in the viewing can be two very different entities. The time may be expanded, compressed, or sustained, with very different story consequences. The handling of time in film can cause the viewing experience to be one of complete and

unquestioned immersion, or utter disgust for the movie's lack of believability due to its monotony and redundancy.

Every shot has an minimal assimilation time in the context of a story. This is a duration of view for which the audience discerns the details, working first from the most obvious/emphasized images to the finer points. As long as the action, be it verbal or visual, of a shot continues the assimilation time increases. It is reached when the shot becomes static relative to its story content.

When an image is perfectly consistent with its surroundings, the assimilation time may be surpassed as a sense of realism works upon the viewer. If action resumes then the extension has been justified. If little else occurs in the shot, the viewer may become bored as filmic continuity is lost. Of course, it is in that subjective interim between cognition and boredom that the perspicacious moviegoer will contemplate what he has seen and make the associations that lead to deep allegorical insight. Ingmar should make judgements about the extent of his audience's patience, for it will limit his narrative potential. American audiences, weaned on Hollywood, will have less patience than the fans of the so called foreign films.

A worst case for extended viewing time will arise when the shot is not entirely within the constraints of synedoche. Then the viewer will most certainly notice the flaws and suspend his belief in the film.

Posing a shot below its assimilation time can have marvelous effects. As long as important chronology is not lost, the viewer will see the story information but will be rushed in his understanding. Minor inconsistencies of setting will be ignored. The viewer will feel elevated to a new level of activity. In his excitement, the story will be almost certainly accepted at face value. Incredibly, the story may become *more* believable. The plot will thicken, and the viewer will anticipate a climactic event.

These effects, perhaps benificial to the story, will decrease as the shot's emphatic

content becomes unintelligible. The limit is perhaps best illustrated in the experimental extremes of Charles Braverman. His "American Time Capsule" of 1968 invoked the controversies of subliminal manipulation with its lightning presentation of images from American history. But Braverman's genius, so often imitated in vain, lay not with coherent narrative but rather the ability to create a disturbing cumilative effect with subtle matching across the images. The technique should well be noted by Ingmar.

The transition between two shots can also communicate the passage of time. The filmmaker has several editing options available for the introduction of a new shot. The straight cut is the most familiar to the average viewer. A straight cut in itself implies no time passage, which is why its use for two shots of the same subject can result in a most annoying jump cut. If time is to pass with straight cuts, it must come about with the sequence of shots and the state changes they relate. Other cuts however can work to show temporal isolation. The dissolve is great for the compare and contrast of two images, thus holding important story potential. Wipes limit the degree of image comparison possible, but their use is a bit arcane. A fade will seperate two shots most explicitly, and can convey any amount of time passage.

Lastly, the filmmaker wields his time control by virtue of just how much of an action in time he displays across a sequence of shots. Important story events must not of course be neglected. But the complete portrayal of mundane, commonplace activities has no part in a good film, except perhaps to goad the viewer for some psychological effect. As with single shots, the virtue of believability falls on the side of time contraction. Thus, for familiar events which are required by the story but serve no other function, like a character entering a room through a door, the condensation of time has priority.

There are certainly occasions when time must be maintained, most notably to support the soundtrack. Dialogue cannot be condensed. It will be up to Ingmar to edit creatively and maintain viewer interest when the visually dull phenomenon of

something like conversation must be constructed. Among his talents will be the ability to crop shots as closely as possible and move the viewer about a space with speed as well as accurate synedoche. He should also appreciate that when the conversation is nonetheless of great portent too much visual trickery will distract.

When an event is of some complexity, it may even be advantageous to expand time. This skill arises in comedy [Reisz 68], wherein it is important to see the expressions of several characters as they react to brief events. Wide shots cannot convey the laughable details that close ups can, so the viewer is strategically treated to more view than is consistent with story time. In comedy, and cases of similar wide close conflict, Ingmar must be able to show all the important views without making the time expansion painfully obvious to his audience.

2.2.4.3 Psychology

It is in this final, catch-all category of editing for story psychology that Ingmar will need to store vast amounts of precedent knowledge, as well as a thorough understanding of the English language. To direct with any style at all, and thus rival his human counterparts, Ingmar must realize how to whip the viewer into emotional states condusive to his narrative. To this end, he must'study' the techniques of all the giants of cinematic history.

The fundamental principle at work in editing for psychological impact is that of the viewer's tendency to correlate phenomena in his understanding of a story from the sequence of images that is a film. Story appreciation flows from the construction of summaries. Since every shot contains limitles detail, the infinite details of an entire *movie* must be lumped together by the recognition of common attributes. Semantic interpretations are made which build analogies from comparisons. Thus film has a gestalt quality, in which elements are forced together to yield an understanding that is not necessarily indicated by any one shot or its contents. The term montage might be relevant, but its employment has been so varied as to make the word a vehicle for

more confusion than illumination.

Lev Kuleshov was the earliest theorist in this domain of an interpretable reality as portrayed by film. He belonged to the founding school of Russian filmmakers, who were motivated to explore the expressive potential of cinema as a medium for communicating the ineffable triumphs of the socialist revolution. Cinema's latent power for propaganda was recognized by Lenin, who in 1919 had declared "Of all the arts the most important for us in my opinion is the film" [Casty 73].

Kuleshov expounded a principle of continuity, which was to be applied faithfully by Vsevolod Pudovkin in such brilliant works as the 1914 masterpiece entitled "Mother". In "Mother", the sequence of images constantly and consistently cooperate to establish the righteousness of a proletarian uprising. Among the many visual metaphors employed is that of a frozen river thawing and releasing its power in an explosion of ice, even as the downtrodden workers unite in frightening force to release themselves from their unjust bondage as slaves to the factory owners. But just as the river's melt is an annual, cyclic event, so too is the rebellion quashed, only to await an inevitable resurgence of revolution.

The consistency of the imagery in examples of the continuity principle was challenged by Sergei Eisenstein, who saw the burden of maintaining literal coherency as an obstacle to the more general expressive needs of the filmmaker. He postulated a principle of collision, born from the application of dialectical materialism [Scott 75]. Kuleshov's continuity became only a special case in which all imagery lead to direct and shallow interpretation. Eisenstein argued that equally coherent communication was possible with the use of contrasting elements, whose differences could suggest even more powerful themes by the route of synthesis from thesis and antithesis. While perhaps more visually challenging, in practice the works of Eisenstein were perfectly comparable in expressive content to those of previous Russian filmmakers who had assumed only the continuity principle.

The consequence of this cinematic history for Ingmar has already been intimated. Whenever a thematic communication is discovered to follow from an existing rule, Ingmar should equally consider an instance of the precise violation of that rule as a means for establishing a consistent and perhaps stronger psychological effect. This is not to say that such effect will be acceptable to any one viewer, since the violation of filmic expectations can result in suspension of belief. Ingmar should follow known precedent and only experiment to a lesser degree with the spontaneous invention of Eisensteinian collision.

Precedents for the psychological re-enforcement of story, which are both familiar and effective for a majority of viewers, exist in abundance. In all cases, they deal with the establishment of semantic analogy between film elements. When the analogy is continuous, a smoothness of communication is perceived. When the analogy is one of disparity, tension and emphasis are produced, which can rivet the perceptive viewer to the film.

When Ingmar follows strictly rules for synedoche and filmic continuity, the viewer can be guided smoothly and directly into story interpretation. An example previously mentioned was that of the thrust match in Herzog's film "Aguirre, Wrath of God". Thrust matching between the different objects of helmet and cannon lead the viewer to seek continuous analogies. The soldier wearing the helmet could therefore be expected to meet a violent and sorry fate just like that of the exploded and wrecked cannon.

The works of Alain Resnais often exploit a violation of synedoche. Resnais' induction principle [Scott 75] involves the presentation of close images prior to their establishing shots. The effect if one of anxiety and claustrophobia, as well as the emphasis of the subjects in close.

Alfred Hitchcock perfected the use of those filmic techniques which semantically translate directly to a point in the flow of classical narrative. This he called his

additive principle [Scott 75]. When the actions of the story are quick and threatening, as they would be in building to climax, Hitchcock would increase the rate of cutting (using shorter duration shots), thrust the viewer inward with closer shots, and likewise use shots whose subjects moved toward the camera. These effects would increase until the climactic point, whence everything could reverse. The viewer then feels relief as the cutting slows and the subjects recede.

Another method of Hitchcock, which would be interwoven with the general additive scheme, was that of establishing suspense versus shock [Scott 75]. Suspense occurs when the viewer is shown in close something that the characters are seen not to be aware of. The viewer anticipates the conflict that will occur when the characters encouter the something. Suspense is thus enforced when, say, an axe-murderer is known by the viewer to be lurking in the corner. Shock is the reverse effect, when the viewer discovers something at the same time that the characters do. If an axe-murderer leaps out to attack a character, the viewer is shocked, and will feel empathy for the character because he was similarly surprised. Mitigated shock with less empathy could occur if the character was first seen crouched in expectation, but the viewer was still not informed of the exact danger.

Even a still frame from a single shot can suggest relationships of analogy by means of visual composition. Many such rules for instantaneous metaphor were implemented by the great Russian directors so long ago that they now seem obvious and cliche, but their validity holds.

The angle with which a character is depicted yields semantic interpretation. High views down onto a subject suggest the subject's weakness, subserviance, and desperation. A shot upward implies that the character is in a position of power and confidence.

The film attribute of distance has direct English significance. If a character is portrayed in a majority of wide shots, the viewer will feel that distance as alienation,

and the character will seem inconsequential or impotent. Movement seen from afar will likewise be subdued. When the viewer is held close to the subject, the effect is one of subjectivity. If the character is threatening, the viewer will be threatened. When the image is inviting, the viewer will feel comforted and empathetic. Any motion in close up will be exagerrated and also add to a sense of threat. The consequences of wide versus close shots must be weighed carefully by Ingmar not only for the resultant emphasis of the parts but also the psychologial effects incurred.

Shot content in terms of line and color can locate comparisons between elements. A beautiful example comes from a film by Alexander Dovzhenko entitled "Earth" [Vogel 74]. In a sequence commonly censored by Soviet authorities, the despair and helplesness of a woman whose lover has just been murdered is communicated in a single image. A room is dark and confining, with a preponderence of perpendicular lines, and drapes and tablecloth visible. The woman is bright and naked, perched to the side at an angle, poised as if seeking escape. Whenever Ingmar can discern visual contrasts such as these, he should consider their exploitation through semantic association for story.

Unusual conditions of photography can have drastic psychological corollary. Some such effects could even be generated by Ingmar from normal shots with a bit of video processing. Overtones of color can have clear import, as with Bergman's "The Passion of Anna", wherein a story of violence and primitive emotion is bolstered by the dull red hue of the entire film. Andrei Tarkovsky made use of literally negative images to communicate dream sequences, as in "My Name is Ivan". A boy buffeted by the savagery of war escapes his predicament in an inverse scene where he rides in the back of a horse drawn cart, eating apples across from a pretty young girl. The sexual connotations of this sequence could also be recognized by Ingmar with a bit of Freudian vocabulary.

Some directors are concerned with the tendency of their audience to believe too strongly in the alternative reality of film. They like to remind the viewer of the nature

of cinema with sequences that depict the medium itself. Both Fellini and Bergman have been known to practise this trick, either by interrupting the story itself with scenes of the actors and equipment or by including objects of photography, like cameras, in the shots. Ingmar could consider this effect of movies about movies whenever he had such scenes at his disposal.

Chapter Three

Prosthesis

The details of my work constitute Ingmar's first assignment in film direction. My project grew entirely from a single scenario, which was filmed and edited into a shot library on videodisc. The necessity of developing from a specific case was obvious; I could not even begin to realize Ingmar without the context of a certain story and its shots. What was less obvious but became critical for the project's ultimate validity was the appreciation of just how much filmic knowledge was incorporated in the staging and photography of this first application. A great number of filmic constraints were automatically satisified. As a result, the Ingmar of this implementation, even in a postulated complete form, remains critically germinal. With the introduction of new images for which the implicit constraints no longer hold, Ingmar will require more powerful filmic understanding. For this simple reason alone have I shaped the exceedingly general, and necessarily less defensible, hypotheses of the preceeding chapter. By developing a framework for Ingmar's directing knowledge, I hope now to cogently indicate those more universal filmic considerations which might otherwise be disregarded with the myopia of my exact accomplishments.

The Ingmar of my work, for all its limitations, is in itself frightfully involved. I had sincerely hoped to succeed with a stage of implementation that at least included the linear presentation of library shots in response to some minimal story specification. This generated movie was to have exhibited some number of basic constraints of story chronology and editing technique. For various restrictions of time and programming environment this goal was not reached. I remain convinced of the value of my efforts, particularly in their context of the transcendent directing knowledge which a generalized Ingmar will require. To the end of communicating my progress, I will discuss my pursuit in the order with which it developed.

3.1 Library Production

3.1.1 The Making of the Shot Library

To make the shot library program accesible, it was necessary to put it on videodisc. Economic reality, as well as an appreciation of directing complexity, deemed that Ingmar could have at his disposal no more than about 15 minutes of image material. Within this claustrophobic space, I wished to include as many narrative possibilities as a single scenario could justify. Any one story plot would be constructed from a subset of the shots available, keeping in mind the basic narrative requirements for an introduction, climax, and denouement.

To satisfy these constraints, the scene needed to be almost archetypal in extent. There were also demands for the feasibility of shooting; the entire production had to be filmable in one location with a minimum of props. To be amenable to representation and eventual movie generation, the entire work also had to be simplistic. There would be a minimum of characters manipulating a minumum of objects with a minimum of movement.

For inspiration, I turned to the legacy of Spanish director Luis Bunuel. His works reek of the complex and often disturbing psychological interactions of stereotyped characters. Although his films often transport the viewer about many locations, the most powerful and revealing sequences are usually rooted in simple and static settings, making his stories frequently translatable to a theatrical environment. His best offerings are among the most powerful in cinematic history, so his staging practices are far from limiting. I contemplated such masterpieces as "The Discrete Charm of the Bourgouesie", "The Exterminating Angel", and "Viridiana", wherein the primeval forces of the characters' interplay occur in the familiar pretext of a dinner.

This was just the insight I required! I would stage a dinner between two characters, and loosely script a number of story possibilities. The filming took place in one evening

at a straightfowardly arranged apartment with the invaluable expertise of Glorianna Davenport and Brian Bradley. The cast consisted of myself, my friend Bianca Philippi, and a cameo cat. We filmed several permutations of ourselves entering the apartment, serving wine and blue spaghetti, consuming said materials at a sparsely set table, conversing, and playing through several story events involving the breakage of wine glasses. With Glorianna's patient assistance, the footage was edited down from almost two hours to 16 minutes. The final result was pressed to videodisc and entitled True Dinners.

3.1.2 The Challenge of True Dinners

True Dinners was slanted toward the possiblities for narrative. For Ingmar, the logical ordering of events will computationally overshadow his fundamental skills for fluid editing.

Several cases for the chronology of physical states arise in True Dinners. A clock on the wall was carefully set and filmed in many shots, so that an absolute temporal consistency would be demanded. The amount of food and wine visible on the table also constrains the possibilities for shot sequence, though less rigorously since views of the glasses and plates being refilled were deliberately included. The breakage of the glass is not necessarily final, because along with an expected scene of either character sweeping up the mess there is available a shot of each character apologizing, to which the other replies that all is well and a new glass may be obtained.

For every state combination possible, there are any number of shots which can lead to that state. An empty plate was eaten from after being served to from a bowl of spaghetti which was taken from an oven and placed on the table. Each empty glass was drunken from after the wine was uncorked after it was placed on the table after it was given to the host as a gift by the entering character. The characters enter and exit the dinner scene under various pretenses, which include leaving in a huff partway through the dinner or helping one another clear the table after a completed meal.

Emotional states are also entirely optional under a wide range of permutations. For many scenes there was shot the context of either character as host, and therefore more obliged to matters of seating, serving, cleaning up, and bidding the guest a potentially hostile farewell. In the introductory shots, those of the characters entering and setting the table, two distinct contingencies were anticipated; either the characters are close friends, and thus given to touching and boisterous speech, or they are dining in a more formal context, and hence act shy and reserved. As the dinner progresses, the gamut of emotional displays is available. The characters exhibit various degrees of anger, pleasure, or boredom, and address one another accordingly. The event of the broken glass, which could very well serve as the climactic point of the story, is portrayed with either character as having accidently knocked the glass over or done so quite deliberately, with appropriate reactions of regret or self-satisfaction to follow. When either character breaks off the dinner in some annoyance, the other may be apologetic and sad or quite eager to see them go.

The True Dinners library is embodied in no less than 108 seperate shots. A single story built from these images might contain far fewer shots, and could not include all 108 since many are mutually exclusive. Even with the number of possibilities, there are enough shots consistent with any same plot to generate a variety of movies, some of which could be most dramatic while others fail for tedium and lack of believablity. Ingmar will have much to consider in his succesful generation of film from this one tiny but versatile library.

3.1.3 A Bittersweet Taste of the Implicit

True Dinners provides a splendid environment for which Ingmar can exercise his story understanding (see Chapter 2). The logical chronology of physical states may be obtained from a database of rather static knowledge. This representation is of the sort I refer to as setting. To order a believable series of character emotional states, Ingmar will resort to the generation of plots. This entails some simulation, however crude, of

the characters with their beliefs and goals in the pursuit of happiness. My general theory for this method is admittedly complex, but for True Dinners only a few considerations are requisite. In particular, the phenomenon of friendship must be approached. The friendly, informal context for the characters can change the severity of any argument that may ensue, because though friends may fight they really want to stay friends and are more likely to make up than strangers. Other relevant plot factors are less compulsory for basic simulation, like drunkeness (there are scenes in which the characters appear inebriated, in which case events could get more emotional with augmented consequence), humor (to really decide when to use a view of one laughing), and pride (when one of several abstruse and intellectual statements is made and the other character does not respond favorably, how seriously insulted should the first person be?). Genuine plot production, down to the last intentional detail, is vastly entangled, but there's no reason why Ingmar couldn't wing it with setting type knowledge, and with perhaps unexpected human interpretable results.

What is disturbing to the intent of a robust Ingmar is the amount of editing skill that has been obviated with the filming of True Dinners. In general, Ingmar should make few, if any, presumptions concerning the nature of his shot library. He should be prepared to analyze shots for every little detail as he constructs a movie for perspective, synedoche, continuity, and story. But because True Dinners was filmed in a small apartment, for activity that was planned to be relatively static, many a nasty editing quandry was eliminated from the start.

Most notably, Ingmar will not need to maintain complete spatial models for the position of characters and objects in the dinner environment. The camera was located in only one of two general areas- the entryway outside the kitchen, and a limited space in front of the dining table and seated characters. These two sites are easily distinguishable by the presence of the entryway door through which the characters enter and exit. Synedoche is remarkably easy to maintain, involving no crucial mappings. The environment is so small that almost any shot wider than an extreme

close gives an instant impression of the space. The only view which could lead to spatial confusion is that of a character obtaining the blue spaghetti from the oven, and even those include pans to and from the table area.

The two continuity questions which also relate to synedoche, and are therefore strategic, are the limits of rotation and thrust matching. Both are answered with the camera placement, since no views are made available of the opposite side of the table, or of the same motion completable from different directions. The photography in fact imposes the continuity rich situation where the screen translates to the space, as discussed under synedoche as a stern formula for maintaining the whole in conversations.

As far as spatial maintenance for the intelligibilty of transitions to character perspective, there are few instances where a character's view is even conceivable. In only a couple instances was the camera able to maneuver behind Bianca and thus give a potential for her perspective. In these cases, barring their unacceptability for other reasons, the view of myself seated and opposite marks the shot unambiguously in the space, and disposes again of any need for a general coordinate system.

Ingmar will in fact need coordinates to subjects, but only from the camera. Jump cuts will be readily avoidable with this information. Planning for interesting and smooth transitions between wide and close shots will also be sufficiently facilitated.

Thus, the mixed blessing of True Dinners involves its gross simplifications for image representation. Content, both spatial (movement, distance, etc., to subjects) and semantic (what's happening in story terms), has been reduced primarily to an issue of character specification. Objects are either in the hands of the character, thus part of the character description, or simply visible in a small space. The environment, and the exact location of its contents, is irrelevant except for the most minimal of descriptions. The only other knowledge Ingmar will demand is that of the camera's attributes, like its motion in the search for cut points.

The category for filmic understanding that is most strongly supported by the True Dinners scenario is that of editing for story. Ingmar needs the insights of rules for emphasis, time, and psychology to make the final movie generation choices that determine whether his work will be cinematically good or not. Even in this domain, it should be noted that some precedents, like those of toward/away motion for climax/denouement, may still never apply due to the confined quarters of the photography.

3.2 Film Representation

3.2.1 The Application of HPRL

HPRL stands for the Heuristic Programming and Representation Language which was recently developed at Hewlett Packard [HewlettPackard 86]. This 'expert system' shell runs on Common Lisp, and at the time of my thesis' planning had been presented to the MIT Media Lab for beta-siting. It was suggested that I use and evaluate this programming tool as an alternative to the creation from scratch of my own semantic description system for True Dinners.

HPRL is a large scale implementation of many database management facilities as would be valuable for research in Artificial Intelligence. The language proved useful for the immediate structuring of the True Dinners representation. My experience with HPRL also pointed out many of the language's limitations, at least with respect to my needs. In fact, HPRL ultimatley hindered my progress, as will be discussed in a section to follow. Most of the problems encountered will also be described as I recount my programming efforts.

While much of HPRL's power is contained in the complex and convoluted commands it offers for the manipulation of a knowledge base, the format for data specification is simple and remarkably readable. The reason for this is that information is stored in

the intuitive structure of class heirarchies. The quintessential example used throughout the reference manual is that of restaurants. Classes of restaurants are constructed and connected by links that semantically translate into is-a-kind-of's. Say, McWeeger's is a kind of Fast-Food-Restaurant, which is a kind of American-Restaurant as well as a kind of Restaurant, which is a kind of Place-to-Eat, ad infinitum. In this example, McWeeger's is called an *instance* of the higher specifications, which are *classes*. One problem with this approach can be never quite knowing when to stop at the instance, which can only belong to one class while classes can belong to several classes (like the Fast-Food-Restaurant pointing to American-Restaurant as well as Restaurant). If McWeeger's became popular and sold franchises, then the knowledge base would have to change, with McWeeger's being now a class and subsuming instances with clumsy names like McWeeger's-in-Watertown and McWeeger's-at-Kenmore.

For each instance, the user can define any number of *slots*, which are just the attachment of a name to any number of details. McWeeger's thus would have a food slot, in which is stored values like Chicken-el-Wacko. If slots are specified for classes, the contents are stored on the *generic-instance*, which is like a template for any real instance. When an instance lacks the values of a higher generic-instance, they could be *inherited*. If the generic-instance for Fast-Food-Restaurants has Grease listed in the food slot, then *asking* what kind of food McWeeger's offers might return Grease as well as Chicken-el-Wacko. Whether or not Grease is really returned must be determinable, so the *control of inheritance* is one of the reasons why HPRL gets so complicated. Inheritance can serve as a means for obtaining default values. Most Fast-Food-Restaurants do serve Grease, but actually McWeeger's doesn't (they use something *much worse*).

Instructing the reader in HPRL is hopefully not that necessary, since with a little acclimation he should have no trouble in getting past the syntax to the real meaning. The appendices of this thesis contain all the code which was written in my pursuit of

Ingmar, and liberal reference to the HPRL located there (as distinguishable from Lisp code by HPRL's *define-class* and *new-instance* commands) should prove illuminating.

3.2.2 Representing True Dinners

For each shot, there will be several things, or rather instances of those things. These things will represent entities like characters, objects, the camera and the surroundings. In slots for these things will go knowledge of specific states, like what the character is doing or how the camera moves. This all requires much definition and explanation, which will be dealt with shortly.

An initial point I wish to make has to do with the requirements for representing knowledge in HPRL. Consider the things and their relationship to the shot. They do not translate to HPRL's sense of a-kind-of. A character is not a-kind-of shot, but rather a-part-of a shot. To fit this association to HPRL's syntax demands a circuitous approach which may not seem intuitive. Shots and their parts are defined equally as subclasses of a single class, which I denoted as the Film-Noun. Since the parts-of affiliation is not succinctly supported, the connection from shots to parts is made in appropriately entitled slots. In each shot is a slot for, say, characters, and in the characters slot go the names of the character instances of that shot. That much seems logical. But to make the reverse linkage, for each character instance there must be an explicit slot designated as the parents. In this slot are deposited the names of the shots for which that character definition is present. Because a character's definition requires a lot of knowledge, a character instance is usually attributable to only one shot. But other things which are less emphasized with the narrative priority of True Dinners, things like the environment, may be common to many shots and thus point to all the possibilities in the parent slot.

Appendix B contains the HPRL code for the representation of True Dinners. The reader should look to this and other appendices to accompany his understanding of the thesis from here on. The opening section of this second appendix contains class

definitions for True Dinners, but after a while instances are specified which exhibit the arguments of the previous paragraph. Appendix A holds the even more general Film-Noun demarcations which apply to all of appendix B.

That point I wanted to make concerns the limitations of the is a-kind-of depiction of HPRL. Parts-of relations must be mutated to fit the syntax, and explicit backpointers must often be provided. While HPRL's heirarchical scheme is agreeable to many knowledge base constructions, matters of syntax make the tool less than ideal. There is in fact a mechanism for establishing the parts of connection, but it involves the verbose specification of an entirely new abstraction, the referencing environment. If I had used referencing environments, my code would have been far less intelligible.

One more issue must be deliberated. The perceptive reader may have wondered in his examination of the appendices about the terms single-valued and multiple-valued. Slot can have one of these two important possible content types. When the slot is multiple valued, this means that it is like a bag of things. Asking for the contents of a multiple valued slot results in an unordered pile of stuff being dumped from the bag. Defaults from above in the heirarchy can accompany this mess. HPRL's power to access slot values hinges upon this bag-like nature. Returning to the restaurant illustration, the food served in restaurants was stored in multiple valued slots. For True Dinners, that parent slot on each thing is also of the multiple variety, and hence is spelled as parents even when there is only one value.

Single valued slots contain only one thing. Inheritance for more things is not allowed. This one thing could be a list in which an ordered sequence of values is specified, but in this case the ability to locate these short of dissecting the whole list is non-exisitant. Thus, HPRL does not support ordered multiple values. This is a problem identifiable for those single valued slots, denoted by a non-plural name, which nontheless have more than one value.

Now lest the reader become too disoriented with this technical preamble, I will proceed

to detail the True Dinners film representation. It is of a very explicit nature, because Ingmar can never actually 'see' the library, for he lacks the capabilities for any image processing. If intricate algorithms for such processing ever become viable, Ingmar might conceivably build his own representations for the shots at his disposal.

3.2.2.1 The Character

The narrative potential of True Dinners will most directly result from sufficient character specification, since it is the characters that really make a story.

Following the definitions of the first appendix, a character instance begins with the slot of *structure*. This slot has to do with english grammar and the usage of the word character. This is required in story representation, and is not relevant here.

The *parents* slot has already been explained as a multiple valued, explicit backpointer to the containing shot(s).

The character has a *name*, which can be traced from above any instance as the class name of the character, but is just as well also stated for every instance. The default character name, as will appear on the generic instance and therefore may be inherited, has been piously established as God.

The *frame* slot deserves some background, since it is not unique to characters and will be encountered again. It is not to be confused with HPRL terminology, for it refers to the frame numbers of a videodisc. Every shot in the library is delineated by frame numbers. It is Ingmar's ability to index the library by frame numbers which makes the the videodisc ideal for the generation of new movies. Every shot starts at a frame number, called the in-point, and ends at the out-point.

For Ingmar to be able to subdivide any shot, and thus maximize the applicability of the library to any one story, important shot contents should also have in and out points. The frame slot provides the means for deducing these points for any given

decriptor within a shot part, including characters. The values of the frame slot are like headings across a chart, specifying the frame number to which the ordered value of other multiple-content (but necessarily single-valued) slots is valid . The reader should seek example in the second appendix at the place where shot 1 and the first instance of Carl is defined. See how slot values line up beneath the frame slot. The second 'face' term of 'right-side' applies to this Carl from frame 22687 to 22728. The first terms of all the slots are valid to 22687, and begin from the containing shot's in-point, seen later to be 22364. Abbreviations for this syntax include end, which stands for the shot's out-point. There is no third value in the face slot, so it is assumed to be na, or not-applicable. Na must be used as a place holder whenever the nth term of ordered, timed slots is unimportant, but is not the last and optional term.

The *visibility* slot literally defines how visible the character is at one time. Possibilities are- Out when the character is not seen, Obstructed and (Obstructed by thing) where thing might be a door, Partial and (Partial parts) where a part might be the hand, Full when the character is seen enough to be completely recognizable, Entire when the character is seen from head to toe, and specifiers for the degree to which the character is in focus (should default to In-Focus). There could very well be other useful cases for visibility but the point is only to get a preliminary range. This attitude applies to most timed slots. The reader should also notice that visibility, like several slots, could be a list of mutually applicable terms.

The *face* slot refers to the identification of which of six faces, like those of a cube,is most visible. The value is one of Top, Bottom, Front, Back, Left-side or just Left, Right-side or just Right.

Distance is borrowed straight from film terminology, and is one of a number of arguably relative tokens- Long, Medium-Long, Medium, Medium-Close, Close, Extreme-Close.

The *angle* refers to that inclination with which the face is visible. Its specification

should be flexible to connote many exact views, and to this end I chose the potential for adverb value pairs. The voluntary use of an adverb is noted by its enclosure with <>'s in the definition's comment. This syntax, and its combination with the obligatory contents of parantheses, will reappear in the code. Example angle values at a single time include- Left, (High Headon), and ((Very Low) Right). (Ambiguities which the clever reader might cast, like the similarity between a High Front view and a Low Top view, are resolved in interpretive Lisp procedures.)

Location means literally where on the screen the character is located. This slot's value definition is as leniant as that of angle.

The character's *movement* is an absolute judegement indicative of activity in a setting. A seated character shown with a moving camera is not defined as moving. Along with adverb/direction pairs, the movement could be- None when the character is very still, Small when the character is moving minimally, and Complex when so much movement has been made that Ingmar should make no presumptions as to what went where. Movement can be a very relative thing. An extreme close of a hand, identifiable with among other things, a visibilty term of (Partial Hand), can yield movement that would not be tagged as more than Small for a wider shot.

As could be rightly inferred by the reader, the worth of the six slots of visibility, face, distance, angle, location, and movement is in the spatial coordinates relative to the camera that are implied. The accurate mapping from these intuitive slot values to numerical implications, in spite of ambiguity and relativity, is approached in Library Manipulation.

The final and critical character slot is entitled *activity*. Here are stored all the semantic descriptions for what the character is actually doing. The foundation is laid in the form of sentences, which are English descriptors crucial to Ingmar's story 'understanding'. Sentences are defined in the section for story representation. For now, the reader can get a sense of the possibilities, as well as discover more about the shot

library, from browsing the activity slots in the second appendix. A syntax that warrants explanation is that of the specification for chronology. At any one description time, simultaneous sentences are connected by +'s while sequential activities are divided with the usage of /'s. The frame numbers for the exact shot appearance of particular sentences are citable in accompanying in out pairs, for which the mnemonics start and end may be used when their implied values are unambiguously derivable. (This latter shot time affixment syntax will reappear in other state specifiers, as for objects.)

3.2.2.2 The Camera

Some degree of definition for the camera's attributes will be necessary to understand the overall effects upon the shot. In the case of True Dinners, the most important descriptor will be for the movements of the camera. As the camera moves, all the visual elements of the shot move, but only because of imparted motion. Accordingly, as mentioned in the Character section above, while the elements must be specified as static for a static scene, the derived movement should enter in Ingmar's calculations as explicitly camera accredited.

The designation for camera has a *movement* slot whose values are timed to an aforementioned *frame* slot. Any one movement can be resultant of both linear displacements (Dolly-in, Dolly-out, Dolly-left, Dolly-right, Rise, and Drop) and angular variations (in analogous order, Zoom-in, Zoom-out, Pan-left, Pan-right, Tilt-up, and Tilt-down). These values can be combined, along with just one adverb at this time, namely Swish, to imply great speed which probably resulted in blurring. None, Small, and Complex have the same meanings as defined for character movement.

One other slot is currently supported, the *effect*. The terms herein may be timed to frames, and should denote special effects like Color Filters, Handheld and thus jiggly photography as is often associated with a Cinema Verite impact, and Slow or Fast Motion. No such effects were used in True Dinners, so although the blue color of the

spaghetti should ultimatley violate Ingmar's real-world expectations, he can deduce that the spaghetti was in fact quite blue.

3.2.2.3 The Object

There are innumerable objects to be seen in True Dinners, and those of primary story importance can be found in the class definitions at the onset of appendix B.

Objects, however important, are secondary to the characters and are assumed not to be of interest beyond their visibility and use by the characters as evinced by appearance in the activity descriptors. Only two new kinds of slots are defined for objects, and these presume the burden of great flexibility to account for several kinds of state knowledge.

The *descriptions* slot is multiple valued, and thus not orderable to any timing. A description term is meant to be a single word state specification or predicate value pair. Examples include broken for a glass, and (time 7:45) for a clock. To allow for a definition of precise frame appearance time, in out point pairs may be coupled. It is assumed that such frame exactitude will be less than common, because Ingmar will regularly and correctly infer visibility from the object's manipulation by a character, or even more simply from the existence of the object in the parent shot's objects slot. Only the clock will routinely require the time stamp, except when the lack thereof implies default to the beginning and end of the entire shot.

The *state-desc* slot is an early attempt to get down some real world knowledge. It crudely encodes state sequences, and will be attended to in Story Representation.

3.2.2.4 The Environment

With the amount of spatial information derviable from characters, camera, and objects, and the constrained space of the True Dinners secario, there seemed to be no need for further such description in the environment instances. In general application however, this will be the repository for complex maps in which all other shot elements must be properly located.

Each environment has an *objects* slot, which contains those objects that somehow belong to the environment, as opposed to having been transported to the scene by the characters. Such specification will often be redundant with the objects slot of the shot, so for the environment these values are best inherited by instances from the generic instance. In fact, every environment instance will frequently be identical to the generic instance, so the environment slot of a shot will regularly hold (a name) rather than 'name', where name is the generic environment instance, and a/an is HPRL's syntax for the generic instance. But because I wished to avoid cluttering the class definition with the lengthy listing of each generic case's parents, the reader will note in the appendix that explicit default instances with a name suffix of 0 were created when logically the generic instance is referenced. Such explicit defaults as kitchen0 also obviate the potential for inheritance confusion.

The two environments of True Dinners are the Doorway and the Kitchen. Both are similar in the contents of three other slots, *light*, *lighting*, and *descriptions*. These slots are of obvious importance in more general footage. The only interesting point to make is that it is in the descriptions slots that I stored an owner predicate, as is implied in the opening scenes where one character plays host by admitting the guest, seating them, and setting the food on the table.

3.2.2.5 The Context

For higher story knowledge which can only be interpreted from a shot by humans, and which has no place in the slots of other Film-Nouns, it seemed natural to create an object known as the context. *Metaphors*, *themes*, and abstract *actions* reside in the context. Ingmar will resort to the these values for ultimate judgements as to the applicability of any particular shot to the cause of a story script.

In the representation code for True Dinners, I initially built a great number of contexts, only to later realize their relevance to a final level of story manipulation which I would most likely not attain. Consequently, I pushed much of the context down into the other Film-Nouns, thereby complicating my state descriptors and character activity slots. The balance between what might be reconstructable by Ingmar as context from low level definitions and what must be a priori as context at the higher context class level is a cause for careful planning. A clear insight will not be obtainable until quite a bit of film generation has been realized and evaluated. The reader is invited to look at some of context instances which I have retained, although not used, and speculate on the problem.

3.2.2.6 The Shot

The shot is where the proverbial buck stops. It has slots for each of the Film-Nouns considered above, some of which are multiple-valued as when such designation makes sense- *Characters* and *objects*, *camera*, *environment*, and *context*.

Each shot also contains its *in-point* and *out-point*, referring to the numerical indices that border the shot on the videodisc.

The shot has a *perspective*. As was discussed in the theory of the second chapter, perspectives are rarely identifiable without context in a film construction. All of the shots of True Dinners inherit the default value of Omniscient.

The slots *precedent*, *resultant*, *concurrent*, and *exclusive* map those sections of the shot to other shots which just happen to work perfectly in the manner that the name indicates. These slots steer Ingmar to possibilities for shot selection that he might not otherwise have made. Two important examples arose in True Dinners.

Shot number 19 is of myself serving blue spaghetti. The shot worked fine except for a photographic error in the middle of the action. Thus there are really two shots, labelled 1901 and 1902. These point to one another through the precedent and resultant slots, and when Ingmar is lead to the sequence he should notice the strong potential for jump cutting and insert some other shot between the two.

Shot number 45 is an alternative to the beginning of shot 44. It shows Bianca from a height saying the first line of the entire text of shot 44, which is taken from table level. The resultant slot of 45 should point to the exact entry location of 44, while 44's precedent slot should map to shot 45 from the end of Bianca's first line. Both shots should designate their mutual overlap in the exclusive slots. Thus Ingmar will have at least this one clear occasion for stylistic decision based on some rule for story psychology and the effect of high versus low angle.

The format for a value in any of these four slots is the name of the related shot, followed by four frame numbers, those of the current shot's in and out as required in the mapping, and the target shot's in and out. Most of these latter values will default to ends and starts for the whole shot, but as in the case of the overlap between shots 44 and 45 the precision is quite necessary.

3.2.2.7 Texts

The use of sound is of great consequence to the creation of a film. But in the case of True Dinners, the preceise representation of sound would become so involved that it would handicap the fundamental goal for image manipulation. Thus, even when the soundtrack contains verbal data, as in the many statements made by the characters

that suggest the construction of dialogues, it was of primary concern to keep the audio content within a generic scheme. Whatever a character says falls into simple representation categories of emotional overtone and agreement or disagreement with a previous statement or the quality of the meal. This range of content is addressed in the sentences that describe the character's activity.

A few more complex character remarks are of an equally general nature. To provide a place to store these, text objects are supported. A text object has a *transciption* slot, in which are placed the recitations in verbatim. This slot provides the reader with a meaningful description, and these statements can be found in the second appendix. Ingmar has no real understanding of natural language however, so for his benifit the *descriptions* slot is defined. Into this slot go any english language depictions which are both correct in summary and plausible for the simplisitic language knowledge that Ingmar might have at his disposal. When Ingmar chooses a shot in which a character announces one of these texts, which are philosophical and admittedly in the style of Bergman himself, the selection of a good next shot could well be guided by the key-word style contents of this latter slot.

3.2.3 The Woes of Representation

3.2.3.1 Construction

The above representation scheme is all well and good in theory, but even its application to the 108 shots of True Dinners proved to be discouragingly time consuming.

Interfaces to aid in the composition of shot library specification are most desirable, and a topic for extensive research. The representation was designed to be intuitively approachable by humans, but there remains so much room for error as well as ambiguity that only computerized tools can amend the difficulties of actual

construction. Ultimately, systems for the rapid human directed analysis of shot content using the most advanced of interface designs will be demanded. As is suggested in [Davenport 87], all manner of graphics, mice, digitizing tablets, and controls for viewing modes become applicable.

In the case of my project, I could not afford to become sidetracked in the ancillary exploitation of such luxuries. I transcribed the content of True Dinners in a painfully slow process of pen and paper while sitting before a monitor with the videodisc controller in hand. Recognizing the prominence as well as complexity of the character objects, I wrote only the most philistine of Lisp utility code, which may be found in appendix C, to commit the characters to HPRL format. As my work progressed, I returned to the True Dinners code to complete as many shot defintions as were needed to debug later programs. At the time of this writing, the True Dinners library representation remains incomplete, though it does sufficiently suggest the shot content for the reader to discern those details which illustrate my project.

3.2.3.2 Semantics

Apart from the unwieldy volume of even a crude representation for just 16 minutes of shots, True Dinners proved the necessity for vast amounts of English language description. Because film deliberately works in the domain of the specific and thus viewable, attempts to generalize vocabulary into a set of primitives reminiscent of Schank's research cannot result in an intelligible specification for any one shot. The best that can be done is to work within the confines of a standardized English hybrid which Ingmar might then be able to convert to more general definition so as to facilitate matching for abstract story content. This judgement motivated the precise format of my film representation, and just how Ingmar can hope to decipher the vast number of distinct words to be found in the descriptions is further explored in the sections to follow.

3.3 Story Representation

Film stories are built from illimitable quantities of language knowledge. Much of the knowledge is of a static variety that is perfectly representable in heiarchical structures like those of HPRL. These I have referred to in chapter two as settings. Other cognizance can only come from the dynamic interactions of knowledge, as with character simulation to derive the motives, or plots, that justify any particular shot selection by Ingmar.

The fundamental level of story representation which was accomplished in the limited domain of True Dinners was that of the establishment of settings for the basic vocabulary utilized in character activity specification. Understanding of abstract entities, like dinner, or conversation, was not even thinkable until the constituent parts, like eating, drinking, and talking, were defined.

The primitive heirarchy realized may be perused by the reader in appendix D. Herein may be seen an appreciation of minimal rules for English grammar, as well as the contextual constraints of what transpired in True Dinners.

3.3.1 Class Knowledge

In understanding my heirarchy for English language, the reader must recall HPRL's inability to succinctly define the parts-of relation. Nouns, verbs, adjectives, and adverbs are all parts of the sentence, and these in turn are all parts of what is called English. To facilitate these distinctions, all definitions stem from what I call the *Thought*. Thoughts constitute everything known to Ingmar, including every shot instance which is traceable via the Film-Noun. There is a tidy philosophical implication in this arrangement. Since every instance of knowledge, particularly each image of life, is distinguished in its slots by the very same language of which it is an example, all knowledge has a kind of self reference. Such circularity may very well be at the core of sentient existence, and could explain why logic alone never seems to provide justification for the experiences of being alive.

Every thought has a *structure*. This is a grammatical template with which Ingmar can expand a fragmentary sentence, as would be comfortable for the specification of stories and shots by we implication-happy humans, into a fully standardized form, complete with all default states. For example, when a character's activity is said to be eating, what is really meant (at least in True Dinners) is the sentence 'character eats food from plate', with the food being the blue spaghetti, and the plate defaulting to the character's plate, which is at least not empty, having been served with blue spaghetti. The excruciating detail of this expansion is absolutely necessary for Ingmar, since he is made up of nothing more than moronic computer procedures which can take nothing for granted. When a story script calls for a shot of 'Bianca eating', and there exists a shot wherein Bianca's activity is spelled out to be 'eat spaghetti', Ingmar mustn't ignore the match for lack of higher language understanding.

Structures are defined in a very precise manner. The structure is read left to right, with parantheses clustering the knowledge for a single target word. When any one of a number of words is acceptable, they are seperated by *or*. An *option* indicates that what is to immediately follow may or may not be present in any one sentence. Adverbs are always optional, and have an effect of scaling some consequence. *Defaults* specify values which should be assumed when not otherwise derivable. *Requires* list those states which must be, as in the negation of empty for a plate when someone wishes to eat from it. The reader may be entertained with a self-guided tour of my structures, and anticipate how they will be later used in the actual standardization of a sentence in just one procedure.

The use of *synonyms* and *antonyms* marks an attempt to map many forms of language together which are functionally equivalent in the pretense of True Dinners. The brevity of the database is largely due to the interrelationships of English defintions through these slots. Synonyms proved to be more immediately supportable than antonyms; when a word is definable by it's opposite, all the invertable parts of the antonym's knowledge must be identified for the specification to hold. Adjectives have

a *predicate* slot to serve a similar intent for absolving Ingmar of superficial conflicts of english usage.

Befores, *durings*, and *afters* are slots in which to store any and all thoughts concerning appropriate activity and state implications which are not so universally applicable as to warrant placement in the structure slot. These kinds of knowledge are particularly interpretable as a primordial source for Ingmar's story construction, and if used to expand a script would impose strict chronology, as well as much redundancy.

A *context* slot should be familiar now to the reader as a handy place to deposit more abstract associations.

Some classes of language lead to numerical scaling, and thus should be so defined to facilitate Ingmar's comprehension in terms of magnitudes of transition. The two examples that were encountered were those of *mood-adjectives* (a class of adjective) and *adverbs*.

Mood-adjectives have a *mood* slot, which places the mood on an arbitrary intuitive scale from negative to positive. It is perhaps remarkable that the reader will note no great distortion of connotation for the mood-adjectives defined in True Dinners. An interesting case is that of being drunk, for in such instances the attribute does not specify a single mood so much as a potential to multiply the extent of the previous mood. For example, when one is happy, getting drunk could make one very happy, and thus make plausible Ingmar's decision for a shot combination that shows this progression. A less convincing case is made for the possibly misnamed mood of intelligent, by which I simply meant that one is thoughtfully considering a situation.

These potentially controversial definitions of mood must be understood as nothing less than first steps toward my elusive and distant goal of character simulation, and should be judged accordingly.

Adverbs have a *multiplier* slot. The value here is used to scale any other scalable

extensive knowledge about only a few sematically distinct things the choice of names can become confusing (as the reader will have noticed by now as he has attempted to trace his way through the code for True Dinners, following a myriad of numerical suffixes).

The *state-desc* slot contains any number of state description triples. The first element is *enforced* or *lax*, depending on whether or not any shot instance of this object absolutely must have one of the associated state values. The second term specifies the connection to other state values. It is nil for enforced states. The final value is the list of allowable states, in order of a chronology which Ingmar could discover in related language definitions. This order may restart with some action, especially for lax states.

An example which will make this all clear is that of the food. The bowl of spaghetti in True Dinners is either unserved or served, and irreversibly in that order because once the food is served the bowl will be messy and can never be shown unserved again. In the served state, the food will go from full to half-full to empty as it is served from. If there were a shot of a character refilling the bowl, this order could restart, but Ingmar will not find such a view in True Dinners.

3.3.3 The Relationship of Story to the Shots Available

In my explanations of knowledge definitions for the first and most basic level of Ingmar's story understanding, the reader may have concocted any number of extenuating circumstances in which the representations became hopelessly invalid.

It is exactly true that everything I have defined is limited to the scope of True Dinners. I would very much have liked to make a generalized knowledge base, even one which contained only those details which apply to True Dinners. But the fact is that the task of building an Ingmar can proceed only relative to one film library at a time. It is impossible to foresee all the cases that might arise in order to program for generality. Ingmar's knowledge will evolve from each learning assignment in much the

same way humans grow in their understanding of the world. Each new experience forces us to rework all that we have learned, until at last one day we have seen enough to fall into routines, and possibly court evil by refusing new circumstance that threatens what we know, and thereby stagnate until we die.

3.4 Library Manipulation

For all the little occasions of incompleteness or inconsistency in Ingmar's rudimentary knowledge, the time had come to work toward his initial attempts at film generation. The first crude procedures had to be sculpted that would tie Ingmar together just enough to allow a low level script, defined at very nearly a shot level, to be translated into a movie of some minimal coherence.

3.4.1 Extraction of Spatial Parameters

As has been justified previously, True Dinners requires only the calculation of coordinate information relative to characters and the camera. Appendix E contains those Lisp procedures that translate terms of spatial representation into the arbitrarily referenced numbers which they imply.

3.4.1.1 The Character

The *view-vector* procedure takes the character's face and angle values for a single frame time and returns a vector of the camera's view. The character is oriented in a three-dimensional space at (0,0,0), with front facing in the +z direction and the top in the +y direction. The vector returned is in the form of the coordinates for the point that is at a distance of one from the origin. The point is where the camera is located such that in being turned to the origin the perspective of the character in the shot is the same.

Location-vector computes a vector for the character's location from the distance and

location slots. The camera is located in 3-space at the orgin, looking down the -z axis. The procedure returns a point ahead of the camera that represents the location of that character.

Char-movement calculates a vector of movement from the character's compound movement descriptors. A point from the character at origin in the direction of movement is returned with distance from the character being indicative of velocity. The movement is *not* relative to the character's orientation, and is absolute with respect to the camera.

Additional procedures return judgements for the centrality of a character with respect to the screen, the amount of angular difference between two character movements, and the effective acceleration between two movements. These and other routines will be useful to Ingmar in ruling for jump cuts and thrust matches between two shots.

3.4.1.2 The Camera

A single procedure named *camera-move* computes a six-dimensional representation of the camera's movement. The camera is situated at the origin and looks down the -z axis of a three-space. The first three coordinates define linear movement as (x,y,z) while the last three denote angular change. The distinction is important for Ingmar to understand how the visibile content of a view changes.

3.4.2 Shot Analysis for Editing Decisions

When Ingmar has sought and discovered an activity within a shot that is required as a next image for the story, he must 'look around' the entire shot, making judgements first about cut points and then about the matching characteristics for an edit between his last shot selection and this new potential choice.

Two procedures, *slot-at-time* and *time-borders*, help Ingmar to move through those slots of a shot element that are timed to a frame slot descriptor. Characters and

cameras have the timed sequences of slot attributes.

Inter-location will return a character's location value as interpolated between the bordering definitions of a character's movement when the frame time Ingmar is considering is not explicitly described. This procedure assumes a linear and constant displacement.

The *adjust-cut* procedure will return a best cut point following on camera movement to whichever side of a character activity is desired. This is the method by which Ingmar begins a cut point search. After using this function, Ingmar will wish to consider what other activities may have begun in the adjusted viewing time, and whether these are completed or even consistent with the story. Such latter decisions will involve semantic, story knowledge.

I began writing the procedure *jump-potential*, which was to decide when the best edit point between two shots constituted a jump cut, and to what degree. Ingmar would run this analysis for the entire bag of shots that he was considering as candidates for next in the story, picking the best shot, if any. Matching characteristics would also clue Ingmar into the strength of an edit that exhibited thrust continuations. The reader is reminded that thrust matching can be very important for synedoche, continuity, and psychological repercussions (like the enforcement of a metaphor between different objects). When more than one shot satisfied minimal matching constraints, Ingmar would consider the distances involved to chose the shot which would move the camera foward or back from the subjects in the shot sequence so far with maximum smoothness. Angle and distance both enter into the calculations for psychological as well as spatial consistency.

An important supporting procedure immediately came into play. For all sorts of matching decisions, it will be vital for Ingmar to decide where the visual focus of a shot lies. In the character driven narratives of True Dinners, this will involve choosing which of the two characters has the more important story role in a shot at any one

time. *Main-char* thus required some strategic semantic analysis based on character activity. My attention shifted to the story level as I pondered the writing of grammar manipulation functions.

3.4.3 Story and Grammar

In appendix F are located those procedures which address the problem of actually using the versatile grammar definitions of the English language file to manipulate fragmented and abbreviated thoughts into standardized story sentences. Ingmar requires standardized expansion for the accurate matching between story script targets and the character activities of True Dinners.

The procedure *expand-phrase* seeks to eliminate the confusions that would arise from the usage of one kind of sentence shorthand. Both the script target sentences and the library sentences allow *and*s and *or*s to build compound depictions. In natural English, and and or can lead to ambiguities, so it is required that their usage be delineated within parentheses. Thus, Ingmar should recognize '(carl and bianca) (eat or drink)' as convertible to four seperate sentences, 'carl eat', 'carl drink', 'bianca eat', 'bianca drink', from which any subset including both carl and bianca is valid.

A story target, without higher levels of language interface, is now definable as a sentence or list of sentences, which may include ands and ors, and a list of state descriptors, each of which is a subject plus a state or predicate state pair. The exact and consistent handling of the optional relationships between sentences expanded from imbedded ors demands that Ingmar retain the pre-expanded form for later reference.

Once Ingmar has derived a single thought from expansion, he needs to standardize the thought by following the structure slots retrieved for each word of the thought. This process will dredge up a number of defaults and requires, which then must be interwoven with the modifiers that came with the script sentence. Or in the case of standardizing a sentence from the shot library, Ingmar has to look in the slots of all

objects referenced by the sentence for the given shot, stringing together the total modifier list and then resolving that with the grammar derived requires and defaults. While defaults merely indicate that a state is implied, requires will pinpoint logical conflicts. Such conflicts may be of a shot in relation to the compiled states to date of a sequence for which the shot is a possible next view, in which case the shot is simply not good for immediate use, and the nature of the conflict should guide Ingmar's choice for the right shot. A conflict from requires with a script means that Ingmar is missing something, and a thought for a scene of the missing state transition should be generated. The reader must recall another option for both script and shot conflicts, however. The filmmaker's mandate to contract time translates to the ability to skip state transitions when a brief passage of time could explain the phenomenon, and inclusion of the missing state would interrupt the emphasis of the story. So, with a next level of sophistication, Ingmar should know better than to, for example, show a serving shot every single time two shots of the characters in passionate argument display a slight and unexplained increase in the amount of blue spaghetti on the plate.

These are the general motivations behind the programming details of sentence standardization. The specific next step is taken in the function *fit-to-structure*. The procedure is very recursive and entangled, but it works beautifully. It expands a phrase into its full sentence. It is passed a noun, a phrase, a structure, a control boolean called options, and a list called expandeds. Expandeds should be nil for the first iteration, and is used in recursion to prevent very subtle forms of looping that can otherwise occur due to what an expert in parsing would no doubt quickly identify as a fault of my grammer (in a precise sense of the word). Options is nil only when any and all optional words, like adverbs, should be dropped in the expanded form. Ingmar will want to do just this when making a positive match between, say, eating and eating fast. The structure is most generally provided as 'sentence'. The noun is the subject of the phrase, as would be passed in expanding a sentence found in a character's slot, which commonly lacks the implied character. The noun is not currently used, but will be when Ingmar needs to decipher the relativized structure

specifications of 'self','subject' and 'other'. The entire procedure returns in a list the expanded phrase, any phrase leftover (also used in recursion), the requires, and defaults.

Now! It was at this point that I was halted by a sorry fact of HPRL, or to be fairer, a fact of my system's configuration for HPRL. The thing is really big. I was left with a minimum of running memory, so my system was spending a lot of time garbage collecting. Every execution of fit-to-structure took a noticeable pause. I couldn't be sure how much of that pause was due to garbage collection and how much was actual run time. I did know that fit-to-structure was still a very low level procedure for Ingmar, and it would be run innumerable times just for the selection of a single next shot in movie generation. More imminently critical was the overhead of running any kind of debugging tools. I spent five hours just chasing down one last bug of fit-to-structure in a single monitored execution, or so I thought, because five hours later the bug had vanished. Thereafter, fit-to-structure worked on all my test cases.

The facts are as follows. I was working on an HP Bobcat, for which the sum of static and dynamic heap space is a maximum of 10.9 Megabytes without kernel reconfiguration (something I could only accomplish with the infinite patience of a system guru). Static space is where all programming resides, and dynamic is the working space leftover. I was using the Nmode environment, which works for a default configuration of 60% static and 40% dynamic out of a total 8 Meg. To accomodate HPRL, Hewlett Packard advised bumping the total to at least 10, and this was done for the same heap ratios.

It is impossible to know exactly what modifications to the memory configuration might have solved my memory problems, since Lisp tends to hide such detail. A friendly hacker in the know changed the specs to 10.5 Meg and ratios of 50/50, but loading problems arose. These were fixed, but not in time to justify further experimentation. I had to write my thesis. And even such a solution could have proved quite temporary, since at any time, without warning, my ever increasing programming

demands might have again taxed the cluttered space. Not that my code was even anywhere close to the size that would eventually be called for by Ingmar. Lastly, it should be noted that the final runtime for Ingmar to construct a movie remains inestimable, but even with unlimited memory could prove to be so long as to further hinder progress. Options for the resolution of such difficulties include- the reconfiguration of the system, development of a representation language specific to True Dinners, and the simplification of Ingmar. The latter choice is of course the least feasible, since Ingmar can only increase in complexity to even approach the requirements for directing.

Computerized movie directing is a goal that remains frustrated.

3.5 An Evaluation of HPRL

In this section, I wish to summarize the faults of HPRL, many of which have been previously described in some detail. I in no way wish to slander the language. I benifitted immensely from my experience with it, for HPRL is an elite state-of-the-art AI tool whose time has come. Only in its application to my specific project did it fall short, and then often not because it didn't do what was advertised, but that it didn't fulfill enough of my needs, and what it could do became irrelevant. I list the problems in order of priority.

1) The Memory Requirement - HPRL should be loadable in discrete subsections, so that the size might be pruned for an application such as mine. I did not need much of HPRL, and relied primarily upon its most basic of knowledge construction and retrieval commands. If I could have honed in on that code, I might have alleviated the space problems which I all too soon ran into.

2) The Parts-of Relation - HPRL's referencing environments did not satisfy my needs for a simple and effective implementation of a bidirectional parts-of relationship. I had to relate such objects explicitly through instance slots.

3) Ordered Multiple Values - For most of my representation needs, I required multiple valued slots whose contents could be retrieved in a guaranteed order. In some cases, procedural work had to be done on the slot values anyway, so easy access was not possible. But in all cases, whenever an order had to be retained I was forced to do so in single-valued slots of lengthy lists, whose dissection for value referencing was left to my programming.

4) Uniqueness of Names - Every HPRL definition requires a different name. In semantically oriented applications like my own, this can lead to a usage of clumsy and confusing suffixes. Perhaps a mechanism for the sharing of names might be feaasible, whereby the instances could be distinguished with their position in the knowledge heirarchy, or in combination with a user specifiable pattern of slot contents.

5) Identification of Frames - This is a minor sort of bug. Everything in HPRL is stored in a system of things called frames, not to be confused with videodisc frames. The frame-p function returns true when its argument is a frame. Frame returns the frame when its argument can be parsed to the name of a frame. I had to use a combination of both to tell when something was a generic frame, which is specified with two words, a (or an) and the name. Frame-p returns nil for 'a carl'. Frame returns the frame for 'a carl', but gives an error on incorrect two word combinations, like 'one carl'. Frame-p will correctly yield nil for 'one carl'.

Acknowledgements

Undying gratitude goes to Glorianna Davenport, who supervised this thesis. Her faith in my ability to chart a course through great waters allowed me to pusue my vast project as I saw fit. But she also steered me with wise and timely encouragement, comment, and suggestion. My experience under her auspices was on the whole wonderful, and cause for my new enthusiasm for academic research. I wish to continue the trying project of realizing an Ingmar under her direction.

My dear friend Bianca Philippi provided boundless moral support for the immense daily expenditure in energy that this thesis represented. She also performed her role in True Dinners with a true sense of acting professionalism.

A number of the residents of the media lab earned my humble respect. Among those were Patrick Purcell, Brian Anderson, and Brian Bradley. Innumerable others contributed to the pleasant and supportive atmosphere of the lab.

Lastly, I thank my loving parents who made *everything* possible. I can never say enough on their behalf, and only hope to reward them with my simple existence in the years to come.

Appendix A

Film Representation

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;this file contains Ingmar's knowledge of film representation;;
;;>Carl Schroeder<                                            ;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

;; default single-valued, plural slots are multiple

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

     ;;character class of shot descriptors
(define-class CHARACTER {FILM-NOUN}
  :instance-slots
  ((structure :v (character))
   (parents
    :declare (multiple-valued))
   (name
    :v God)
   (frame
    :v end)
     ;; VISIBILITY,
     ;; one or list of following    out obstructed/( by)
     ;;                         partial/( parts)  full  entire
     ;;                         past-focus in-focus before-focus
     ;; full=fully recognizable not necessarily entire
   (visibility
    :v out)
     ;; FACE,
     ;; one of following na
     ;;                  top bottom
     ;;                  front back
     ;;                  left-side/left right-side/right
   (face
    :v na)
     ;; DISTANCE,
     ;; one of following na
     ;;                  long med-long med
```

```
    ;;                   med-close close ex-close
   (distance
    :v na)
    ;; ANGLE.
    ;; one or list of following  na
    ;;                  ( < little/somewhat very/quite extremely >
    ;;                    ( left headon right high low )        )
    ;;
   (angle
    :v na)
    ;; LOCATION.
    ;; one or list of following     na
    ;;                                ( < little/somewhat very/far >
    ;;                         ( left center right
    ;;                           upper/up/top  lower/low/bottom)  )
    ;; for some reasonable centerpoint
   (location
    :v na)
    ;; MOVEMENT.
    ;; one or list of following    na  none  small complex
    ;;     ( <slow fast swish> (up down left right toward away) )
    ;;  ex: (left (slow right))
   (movement
    :v small)
    ;; chronological textual phrases built from keywords
   (activity
    :v na)
    ))

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

     ;;camera class of shot descriptors
(define-class CAMERA {FILM-NOUN}
  :instance-slots
  ((parents
    :declare (multiple-valued))
    ;; list of frames to which desc is appropriate
   (frame
    :v end )
    ;; one or list of following   none small complex swish
    ;;                            zoom-in zoom-out
    ;;                            dolly-left dolly-right
    ;;                            pan-left pan-right
    ;;                            dolly-in dolly-out
```

```
      ;;                              tilt-up  tilt-down
      ;;                              rise   drop
      ;; as with characters, can build with + (simultaneous)
      ;;                             and / (sequence)
    (movement
     :v small )
      ;; i.e. filters, slow/fast motion, double-exposure,
      ;;         negative, freeze, handheld, black&white
    (effect
     :v none )
    ))


;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

     ;;object class of shot descriptors
(define-class OBJECT {FILM-NOUN}
  :instance-slots
  ((structure :v (object))
   (name)
   (parents
    :declare (multiple-valued))
     ;;textual modifier including appearance (start end)
   (descriptions
    :declare (multiple-valued))
   (state-desc)))

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

     ;;environment class of shot descriptors
(define-class ENVIRONMENT {FILM-NOUN}
  :instance-slots
  ((structure :v (environment))
   (parents
    :declare (multiple-valued))
   (name
    :v space)
   (objects
    :declare (multiple-valued))
   (light
    :type (one-of natural artificial)
    :v natural)
   (lighting
   :type (one-of front back top bottom right-side left-side all)
```

```
    :v all)
     ;; i.e. colors, lines, composition
   (descriptions
    :v (owner God)
    :declare (multiple-valued))
   ))

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

     ;;context class of shot descriptors
(define-class CONTEXT {FILM-NOUN}
  :instance-slots
  ((parents
    :declare (multiple-valued))
     ;; i.e. pairs like (bird freedom)
   (metaphors
    :declare (multiple-valued))
     ;; i.e. concepts like escape
   (themes
    :declare (multiple-valued))
     ;; i.e. concepts like coup
   (action)))


;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;


(define-class SHOT {FILM-NOUN}
  :instance-slots
  ((in-point
    :type number)
   (out-point
    :type number)
   (perspective
    :type (one-of character omniscient viewer)
    :v omniscient)
     ;; shots which specifically relate
     ;; format ({shot} thisin thisout thatin thatout
     ;;                    [default to starts and ends])
   (precedent)
   (resultant)
   (concurrent)
   (exclusive)
     ;;content specifications
```

```
    ;;;camera
   (camera)
   (characters
    :declare (multiple-valued))
    ;;;objects
   (objects
    :declare (multiple-valued))
    ;;;environment
   (environment)
    ;;;context
   (context)
    ))

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

;; a text instance is used as the object of
;; a say sentence in several shots' character objects.
;; since sound is not specifically addressed
;; in this implementation, text is the one
;; place to store summary descriptors for
;; particularly involved spoken sentences.

(define-class TEXT {FILM-NOUN}
  :instance-slots
  ((parents
    :declare (multiple-valued))
   (transcription)
   (descriptions
    :declare (multiple-valued))))
```

Appendix B

True Dinners

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;; this file contains Ingmar's representation ;;
;; of the True-Dinners film                   ;;
;; >Carl Schroeder<                           ;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

;; default single-valued , plural slots are multiple

;;;*characters

(define-class BIANCA {CHARACTER}
  :instance-slots
  ((name :v bianca)))

(define-class CARL {CHARACTER}
  :instance-slots
  ((name :v carl)))

(define-class CAT {CHARACTER}
  :instance-slots
  ((name :v cat)))

;;;*objects

(define-class DOOR {OBJECT}
  :instance-slots
  ((name :v door)
   (descriptions :v closed)
;; the associated states, in sequence of values,
;; first value is enforced or lax, second is implied states,
;; third a list ordered values
   (state-desc :v ( lax nil (closed open)) )))


(define-class WINE {OBJECT}
  :instance-slots
  ((name :v wine)
```

```
   (descriptions :vs ((type French) (color white) closed))
   (state-desc :vs ( ( enforced nil (closed open))
                     ( lax open (full half-full empty))))))

(define-class CORKSCREW {OBJECT}
  :instance-slots
  ((name :v corkscrew)))

(define-class TABLE {OBJECT}
  :instance-slots
  ((name :v table)))

(define-class FOOD {OBJECT}
  :instance-slots
  ((name :v food)
   (descriptions :vs ((type spaghetti) (color blue) unserved))
   (state-desc :vs ( (enforced nil (unserved served))
                     ( lax served  (full half-full empty))))))


(define-class CLOCK {OBJECT}
  :instance-slots
  ((name :v clock)
   (descriptions :v (time unknown)  )))

(define-class BROOM {OBJECT}
  :instance-slots
  ((name :v broom)))

(define-class GLASS {OBJECT}
  :instance-slots
  ((name :v glass)
   (descriptions :vs (clean whole) )
   (state-desc :vs ( (enforced nil (whole broken))
                     (enforced nil (clean dirty))
                 (lax (dirty whole) (full half-full empty))))))

(define-class HIS-GLASS {GLASS}
  :instance-slots
  ((descriptions :v  (user Carl)  )
   ))

(define-class HER-GLASS {GLASS}
  :instance-slots
```

```
  ((descriptions :v  (user Bianca) )
   ))

(define-class PLATE {OBJECT}
  :instance-slots
  ((name :v plate)
   (descriptions :vs (clean whole))
   (state-desc :vs ( (enforced nil (whole broken))
                     (enforced nil (clean dirty))
                (lax (dirty whole) (full half-full empty))))))

(define-class HIS-PLATE {PLATE}
  :instance-slots
  ((descriptions :v  (user Carl)  )
   ))

(define-class HER-PLATE {PLATE}
  :instance-slots
  ((descriptions :v  (user Bianca)  )
   ))

;;;*environments

(define-class DOORWAY {ENVIRONMENT}
  :instance-slots
  ((name :v doorway)
   (objects :v (a door))
   (light :v artificial)
   (lighting :v top)
   (descriptions :v  (color white) )
   ))

(define-class KITCHEN {ENVIRONMENT}
   :instance-slots
   ((name :v kitchen)
    (objects :vs ((a his-plate) (a her-plate) (a his-glass)
                  (a her-glass) (a food) (a corkscrew)
                  (a clock) (a broom) (a table)))
    (light :v artificial)
    (lighting :v top)
    (descriptions :v (color white))))

;;;*contexts = not completable until higher level
;;;            of story generation is defined
```

```
(define-class ENTRANCE {CONTEXT}
  :instance-slots
  ((themes :vs (entrance beginning introduction))
   (action :v enter)))


(define-class DINNER {CONTEXT}
  :instance-slots
  ((action :v (eat drink talk))))

(define-class BEFORE-DINNER {DINNER}
  :instance-slots
  ((action :v (enter (get food) (get wine) sit toast ))))


(define-class MOOD {CONTEXT})

(define-class FORMAL {MOOD}
  :instance-slots
  ((themes :vs (formal unfamiliar introduction nervous))))

(define-class INFORMAL {MOOD}
  :instance-slots
  ((themes :vs (informal familiar friendly relaxed))))


(define-class WHOSE-PLACE {CONTEXT})

(define-class HER-PLACE {WHOSE-PLACE}
  :instance-slots
  ((themes :v (owner bianca))))

(define-class HIS-PLACE {WHOSE-PLACE}
  :instance-slots
  ((themes :v (owner carl))))

(define-class SHE-ENTERS-FORMAL
              ({ENTRANCE} {FORMAL} {HIS-PLACE})
  :instance-slots
  ((action :v (bianca enters))))

(define-class HE-ENTERS-FORMAL
              ({HER-PLACE} {ENTRANCE} {FORMAL})
  :instance-slots
```

```
  ((action :v (carl enters))))

(define-class SHE-ENTERS-INFORMAL
              ({INFORMAL} {HIS-PLACE} {ENTRANCE})
  :instance-slots
  ((action :v (bianca enters))))

(define-class HE-ENTERS-INFORMAL
              ({INFORMAL} {HER-PLACE} {ENTRANCE})
  :instance-slots
  ((action :v (carl enters))))

;;; make explicit defaults rather than relying on generic
;;; instances so don't have to worry about inheritance of
;;; the default's parents slot (anyway, an instance of a
;;; generic instance is not the same as the genereic instance)

(new-instance {DOOR}
  :name door0
  :slots ((parents :vs ())))

(new-instance {WINE}
  :name wine0
  :slots ((parents :vs (shot6 shot14 shot15 shot17))))

(new-instance {CORKSCREW}
  :name corkscrew0
  :slots ((parents :vs (shot11 shot14 shot15))))

(new-instance {FOOD}
  :name food0
  :slots ((parents :vs (shot16 shot17 shot1901 shot20))))

(new-instance {BROOM}
  :name broom0
  :slots ((parents :vs ())))

(new-instance {HIS-GLASS}
  :name his-glass0
  :slots ((parents :vs (shot15 shot16 shot17))))

(new-instance {HER-GLASS}
  :name her-glass0
  :slots ((parents :vs (shot15 shot16 shot17))))
```

```
(new-instance {HIS-PLATE}
  :name his-plate0
  :slots ((parents :vs (shot17 shot1901 shot1902 shot20))))

(new-instance {HER-PLATE}
  :name her-plate0
  :slots ((parents :vs (shot1901 shot1902 shot20))))

(new-instance {KITCHEN}
  :name kitchen0
  :slots ((parents :vs (shot9 shot14 shot15 shot16 shot17
           shot18 shot1901 shot1902 shot20 shot21 shot22
           shot23 shot24 shot25 shot26 shot27 shot28 shot29
           shot30 shot31 shot32 shot33 shot34 shot35 shot36
           shot37 shot38 shot39 shot40 shot41 shot42 shot43
           shot44 shot45 shot46 shot47 shot48 shot49 shot50))))


(new-instance {CAMERA}
  :name camera0
  :slots ((parents :vs (shot16 shot17 shot18 shot1901 shot1902
                        shot21 shot22 shot24 shot25 shot26 shot27
                          shot30 shot32 shot33 shot34 shot35))))

;;;;;;shot 1
(new-instance {CARL}
  :name carl1
  :slots ((parents :v shot1)
          (frame :v              (22687          22728     end))
          (visibility :v        (partial         partial   out))
          (face :v               (back           right-side))
          (angle :v             (low             headon))
          (distance :v          (med-close       close))
          (location :v           (right          center))
          (movement :v           (complex        left))
          (activity :v  ((open door / take wine / close door)
                                  (exit left)))
          ))

(new-instance {BIANCA}
  :name bianca1
  :slots ((parents :v shot1)
          (frame :v              (22450          22573
                                   22634          end))
```

```
        (visibility :v        (obstructed     partial
                              partial         out))
        (face :v              (na             front
                                              front))
        (angle :v             (na             headon
                                        (low right)))
        (distance :v          (na             medium
                                      med-close))
        (location :v          (na             center
                                     center))
        (movement :v          (na             toward
                                              left))
        (activity :v          (na
                        (enter door with wine / give wine)
                        (exit left)))
        ))

(new-instance {SHE-ENTERS-FORMAL}
  :name context1
  :slots ((parents :v shot1)
          (themes :vs (happy hopeful shy ))))

(new-instance {WINE}
  :name wine1
  :slots ((parents :vs (shot1 shot3 shot9))
          (descriptions :vs ( closed (owner bianca)))))

(new-instance {DOOR}
  :name door1
  :slots ((parents :vs (shot1 shot3))
          (descriptions :v (owner carl) )))

(new-instance {DOORWAY}
  :name doorway1
  :slots ((parents :vs (shot1 shot3))
          (descriptions :v (owner carl))))

(new-instance {SHOT}
  :name shot1
  :slots ((in-point :v 22364)
          (out-point :v 22741)
          (characters :vs (bianca1 carl1))
          (objects :vs (wine1 door1))
          (environment :v doorway1)
```

```
          (context :v context1)))


;;;;;;shot 2

(new-instance {CARL}
  :name carl2
  :slots ((parents :v shot2)
          (frame :v              (22800          22974
                                  23030          end))
          (visibility :v         (obstructed     full
                                  full           out))
          (face :v               (na             front
                                                front))
          (angle :v              (na               headon
                                                headon))
          (distance :v           (na               medium
                                                med-close))
          (location :v           (na                center
                                                center))
          (movement :v          (na                toward
                                                   left))
          (activity :v          (na
                          (enter door with wine / give wine)
                                (exit left)))))

(new-instance {BIANCA}
  :name bianca2
  :slots ((parents :v     shot2)
          (frame :v     (22848          22918          end))
          (visibility :v (full          out          partial))
          (face :v       (back          na             back))
          (angle :v      ((high left)   na      (high left)))
          (distance :v   (med-close     na             close))
          (location :v   (right         na             right))
          (movement :v   (small         na             small))
          (activity :v   ((open door)   na       (take wine)))
          ))

(new-instance {HE-ENTERS-FORMAL}
  :name context2
  :slots ((parents :v shot2)
          (themes :vs (happy hopeful shy))))
```

```
(new-instance {WINE}
  :name wine2
  :slots ((parents :vs (shot2 shot4 shot5 shot6))
          (descriptions :vs ( closed (owner carl)))))

(new-instance {DOOR}
  :name door2
  :slots ((parents :vs (shot2 shot4))
          (descriptions :v (owner bianca) )))

(new-instance {DOORWAY}
  :name doorway2
  :slots ((parents :vs (shot2 shot4))
          (descriptions :v (owner bianca))))

(new-instance {SHOT}
  :name shot2
  :slots ((in-point :v 22742)
          (out-point :v 23034)
          (characters :vs (bianca2 carl2))
          (objects :vs (wine2 door2))
          (environment :v doorway2)
          (context :v context2)))

;;;;;;shot 3

(new-instance {CARL}
  :name carl3
  :slots ((parents :v      shot3)
          (frame :v      (23203          end))
          (visibility :v (full           partial))
          (face :v       (back           left-side))
          (angle :v      (left           right))
          (distance :v   (med-close              med))
          (location :v   (right          right))
          (movement :v   (small          left))
          (activity :v   ((open door / hug bianca)
                          (close door / exit left)))
          ))


(new-instance {BIANCA}
  :name bianca3
  :slots ((parents :v      shot3)
```

```
          (frame :v     (23057          23204
                                        23284          end))
          (visibility :v (obstructed          full
                                      partial          out))
          (face :v       (na           front          front))
          (angle :v      (na           headon         left))
          (distance :v   (na           med        med-close))
          (location :v   (na           (center right)
                                       (low left)))
          (movement :v   (na           small          left))
          (activity :v   (na   (enter door with wine / hug carl)
                               (exit left)))
          ))

(new-instance {SHE-ENTERS-INFORMAL}
  :name context3
  :slots ((parents :v shot3)
          (themes :v happy)))

(new-instance {SHOT}
  :name shot3
  :slots ((in-point :v 23035)
          (out-point :v 23389)
          (characters :vs (bianca3 carl3))
          (objects :vs (wine1 door1)))
          (environment :v doorway1))
          (context :v context3)))


;;;;;;shot 4

(new-instance {BIANCA}
  :name bianca4
  :slots ((parents :v     shot4)
          (frame :v     (end))
          (visibility :v (partial))
          (face :v       (back))
          (angle :v      (high))
          (distance :v   (med))
          (location :v   (right))
          (movement :v   (complex))
          (activity :v   ((open door / hug carl / close door)))
          ))
```

```
(new-instance {CARL}
  :name carl4
  :slots ((parents :v     shot4)
          (frame :v     (23460          23675
                                        23727          end))
          (visibility :v (obstructed    full           full
                                                       out))
          (face :v       (na            front        front))
          (angle :v      (na            headon       headon))
          (distance :v   (na            med        med-close))
          (location :v   (na            center       center))
          (movement :v   (na            toward         left))
          (activity :v   (na
                         (enter door with wine / hug bianca)
                         (exit left)))
          ))

(new-instance {HE-ENTERS-INFORMAL}
  :name context4
  :slots ((parents :v shot4)
          (themes :vs (happy))))

(new-instance {SHOT}
  :name shot4
  :slots ((in-point :v 23390)
          (out-point :v 23804)
          (characters :vs (bianca4 carl4))
          (objects :vs (wine2 door2)))
          (environment :v doorway2))
          (context :v context4)))

;;;;;;shot 5

(new-instance {BIANCA}
  :name bianca5
  :slots ((parents :v     shot5)
          (frame :v     (23920          23990        end))
          (visibility :v (full          out         full))
          (face :v       (left-side     na        right-side))
          (angle :v      (headon        na          high))
          (distance :v   (med           na          med))
          (location :v   (left          na          center))
          (movement :v   (right         na          small))
          (activity :v   ((enter kitchen)
```

```
                        sit))
          ))


(new-instance {CARL}
  :name carl5
  :slots ((parents :v     shot5)
          (frame :v     (23920          23997          end))
          (visibility :v (out           partial        out))
          (face :v       (na            left-side))
          (angle :v      (na            low))
          (distance :v   (na            med-close))
          (location :v   (na            left))
          (movement :v   (na            right))
          (activity :v   (na
                     (enter kitchen / put wine / exit right)))
          ))

(new-instance {CAMERA}
  :name camera5
  :slots ((parents :v shot5)
          (frame :v (23900 33958 end))
          (movement :v (pan-right
                        (pan-left + tilt-down) small))))

(new-instance {CLOCK}
  :name clock5
  :slots ((parents :v shot5)
          (descriptions :v (time 7:50 (23840 23930)))))

(new-instance {HIS-GLASS}
  :name his-glass5
  :slots ((parents :vs (shot5 shot8))
          (descriptions :v (owner carl))))

(new-instance {HER-GLASS}
  :name her-glass5
  :slots ((parents :vs (shot5 shot8))
          (descriptions :v (owner carl))))

(new-instance {HIS-PLATE}
  :name his-plate5
  :slots ((parents :vs (shot5 shot6 shot8))
          (descriptions :v (owner carl))))
```

```
(new-instance {HER-PLATE}
  :name her-plate5
  :slots ((parents :vs (shot5 shot6 shot8))
          (descriptions :v (owner carl))))

(new-instance {KITCHEN}
  :name kitchen5
  :slots ((parents :vs (shot5 shot6 shot7 shot8))
          (descriptions :v (owner carl))))

;; note that carl entered with the wine, so it seems to be his,
;; though if we prelude with shot1, we see it is bianca's and
;; she gave it to him
(new-instance {SHOT}
  :name shot5
  :slots ((in-point :v 23805)
          (out-point :v 24234)
          (characters :vs (bianca4 carl4))
          (objects :vs (clock5 wine2 his-glass5 her-glass5
                               her-plate5 his-plate5))
          (environment :v kitchen5))
          (camera :v camera5)))


;; shot 6

(new-instance {CARL}
  :name carl6
  :slots ((parents :v     shot6)
          (frame :v     (24457          24566
                                        24712          end))
          (visibility :v (full          full
                                        partial        out))
          (face :v       (left-side     front          back))
          (angle :v      (headon        low            low))
          (distance :v   (med-long      med-long     med-close))
          (location :v   (center        center         right))
          (movement :v   (small         toward         left))
          (activity :v   ((get food)
                         (get corkscrew)
                         (put (food & corkscrew) / sit)))
          ))
```

```
(new-instance {BIANCA}
  :name bianca6
  :slots ((parents :v     shot6)
          (frame :v     (24540          end))
          (visibility :v (out           full))
          (face :v       (na            front))
          (angle :v      (na            headon))
          (distance :v   (na            med))
          (location :v   (na            (center left)))
          (movement :v   (na            small))
          (activity :v   (na
                         seated))
          ))


(new-instance {CAMERA}
  :name camera6
  :slots ((parents :v shot6)
          (frame :v (24517 24577 end))
          (movement :v (small pan-left small))))

(new-instance {CLOCK}
  :name clock6
  :slots ((parents :v shot6)
          (descriptions :v (time 7:50 (24542 end)))))

(new-instance {FOOD}
  :name food6
  :slots ((parents :vs (shot6 shot8))
          (descriptions :vs (unserved (owner carl)) )))

(new-instance {CORKSCREW}
  :name corkscrew6
  :slots ((parents :v shot6)
          (descriptions :v (owner carl))))

(new-instance {SHOT}
  :name shot6
  :slots ((in-point :v 24235)
          (out-point :v 24943)
          (characters :vs (bianca6 carl6))
          (objects :vs (clock6 wine0 food6
                               her-plate5 his-plate5))
          (environment :v kitchen5))
```

```
          (camera :v camera6)))

;; shot 7

(new-instance {BIANCA}
  :name bianca7
  :slots ((parents :v     shot7)
          (frame :v     (25052          25281          end))
          (visibility :v (partial       full           out))
          (face :v       (left-side             front))
          (angle :v      (headon                high))
          (distance :v   (med-close             med))
          (location :v   (left                  (low right)))
          (movement :v   (right         small))
          (activity :v   ((enter kitchen / sit)))
          ))


(new-instance {CARL}
  :name carl7
  :slots ((parents :v     shot7)
          (frame :v     (25050          25305          end))
          (visibility :v (out           partial        out))
          (face :v       (na            right-side))
          (angle :v      (na            low))
          (distance :v   (na            med-close))
          (location :v   (na            left))
          (movement :v   (na            right))
          (activity :v   (na
                    (enter kitchen / put wine / exit right)))
          ))

(new-instance {CAMERA}
  :name camera7
  :slots ((parents :v shot7)
          (frame :v (25010 end)
          (movement :v (pan-right small)))))

(new-instance {CLOCK}
  :name clock7
  :slots ((parents :v shot7)
          (descriptions :v (time 7:50))))

(new-instance {SHOT}
```

```
  :name shot7
  :slots ((in-point :v 24944)
          (out-point :v 25325)
          (characters :vs (bianca7 carl7))
          (objects :vs (clock7 wine2)
          (environment :v kitchen5))
          (camera :v camera7)))

;; shot 8

(new-instance {CARL}
  :name carl8
  :slots ((parents :v     shot8)
          (frame :v     (25508          25639       end))
          (visibility :v (full          full        partial))
          (face :v       (left-side     front           back))
          (angle :v      ((right headon)   low           low))
          (distance :v   (med-long     med-long     med-close))
          (location :v   (center        center          right))
          (movement :v   (small         toward          left))
          (activity :v   ((get food)
                     (get corkscrew / put (food & corkscrew))
                        sit))
          ))


(new-instance {BIANCA}
  :name bianca8
  :slots ((parents :v     shot8)
          (frame :v     (25580          end))
          (visibility :v (out           full))
          (face :v       (na            front))
          (angle :v      (na            headon))
          (distance :v   (na            med-close))
          (location :v   (na            (center left)))
          (movement :v   (na            small))
          (activity :v   (na
                        seated))
          ))

 (new-instance {CAMERA}
   :name camera8
   :slots ((parents :v shot8)
           (frame :v (25572 25672 end)
```

```
          (movement :v (small pan-left small)))))

(new-instance {CLOCK}
  :name clock8
  :slots ((parents :v shot5)
          (descriptions :v (time 7:50 (25603 end)))))

(new-instance {SHOT}
  :name shot8
  :slots ((in-point :v 25444)
          (out-point :v 25844)
          (characters :vs (bianca8 carl8))
          (objects :vs (clock8 food6 corkscrew6 his-glass5
                              her-glass5 her-plate5 his-plate5))
          (environment :v kitchen5))
          (camera :v camera8)))


;; shot 9

(new-instance {CARL}
  :name carl9
  :slots ((parents :v     shot9)
          (frame :v     (25967          end))
          (visibility :v (full          out))
          (face :v       (left-side))
          (angle :v      (headon))
          (distance :v   (med-close))
          (location :v   (left))
          (movement :v   (right))
          (activity :v   ((enter kitchen / sit)))
          ))


(new-instance {BIANCA}
  :name bianca9
  :slots ((parents :v     shot9)
          (frame :v     (25913          end))
          (visibility :v (out           partial))
          (face :v       (na            back))
          (angle :v      (na            right))
          (distance :v   (na            med-close))
          (location :v   (na            left))
          (movement :v   (na            right))
```

```
          (activity :v   (na
                         (enter kitchen / put wine)))
          ))

(new-instance {CAMERA}
  :name camera9
  :slots ((parents :v shot9)
          (frame :v (25928 end))
          (movement :v (small pan-right))))

(new-instance {SHOT}
  :name shot9
  :slots ((in-point :v 25845)
          (out-point :v 25984)
          (characters :vs (bianca9 carl9))
          (objects :v wine1)
          (environment :v kitchen0)
          (camera :v camera9)))


;; shot 10

(new-instance {CARL}
  :name carl10
  :slots ((parents :v     shot10)
          (frame :v     (end))
          (visibility :v (full))
          (face :v       (front))
          (angle :v      (left))
          (distance :v   (med))
          (location :v   (center))
          (movement :v   (small))
       (activity :v ((seated / ask that bianca get corkscrew)))
          ))

(new-instance {KITCHEN}
  :name kitchen10
  :slots ((parents :vs (shot10 shot11 shot12 shot13))
          (descriptions :v (owner bianca))))

(new-instance {SHOT}
  :name shot10
  :slots ((in-point :v 25985)
          (out-point :v 26097)
```

```
          (characters :v carl10)
          (environment :v kitchen10)))


;; shot 11

(new-instance {CARL}
  :name carl11
  :slots ((parents :v     shot11)
          (frame :v     (26500          26650          end))
          (visibility :v (out           full           out))
          (face :v       (na            front))
          (angle :v      (na            left))
          (distance :v   (na            med-close))
          (location :v   (na            left))
          (movement :v   (na            small))
          (activity :v   (na
                         seated))
          ))


(new-instance {BIANCA}
  :name bianca11
  :slots ((parents :v     shot11)
          (frame :v     (26342          26490
                                        26677          end))
          (visibility :v (full          full
                                        out          full))
          (face :v       (front         front
                                        na        right-side))
          (angle :v      (left          low
                                        na             left))
          (distance :v   (med-long      med
                                        na         med-close))
          (location :v   (center        center
                                        na             right))
          (movement :v   (small         left
                                        na             small))
          (activity :v   ((get food)
                         (put food / get corkscrew)
                         na
                         sit))
          ))
```

```
(new-instance {CAMERA}
  :name camera11
  :slots ((parents :v shot11)
          (frame :v (26353 26500 26633 end))
          (movement :v (small (pan-left + tilt-down)
                              (pan-left + tilt-up)
      (pan-right + tilt-down / small / pan-right + tilt-up)))))

(new-instance {CLOCK}
  :name clock11
  :slots ((parents :v shot11)
          (descriptions :v (time 7:50 (26410 26434)))))

(new-instance {FOOD}
  :name food11
  :slots ((parents :vs (shot11 shot13))
          (descriptions :vs (unserved (owner bianca)))))


(new-instance {SHOT}
  :name shot11
  :slots ((in-point :v 26098)
          (out-point :v 26762)
          (characters :vs (bianca11 carl11))
          (objects :vs (clock11 corkscrew0 food11))
          (environment :v kitchen10))
          (camera :v camera11)))

;; shot 12

(new-instance {CARL}
  :name carl12
  :slots ((parents :v     shot12)
          (frame :v     (end))
          (visibility :v (partial))
          (face :v       (front))
          (angle :v      (left))
          (distance :v   (med))
          (location :v   (left))
          (movement :v   (complex))
          (activity :v   ((enter kitchen / sit)))
          ))
```

```
(new-instance {BIANCA}
  :name bianca12
  :slots ((parents :v     shot12)
          (frame :v     (26815          26872       end))
          (visibility :v (out           partial     out))
          (face :v       (na            back))
          (angle :v      (na            right))
          (distance :v   (na            med-close))
          (location :v   (na            left))
          (movement :v   (na            right))
          (activity :v   (na
                  (enter kitchen / put wine / exit right)))
          ))

(new-instance {CAMERA}
  :name camera12
  :slots ((parents :v shot12)
          (frame :v end)
          (movement :v complex)))

(new-instance {HIS-GLASS}
  :name his-glass12
  :slots ((parents :vs (shot12 shot13))
          (descriptions :v (owner bianca))))

(new-instance {HER-GLASS}
  :name her-glass12
  :slots ((parents :vs (shot12 shot13))
          (descriptions :v (owner bianca))))

(new-instance {HIS-PLATE}
  :name his-plate12
  :slots ((parents :vs (shot12 shot13))
          (descriptions :v (owner bianca))))

(new-instance {HER-PLATE}
  :name her-plate12
  :slots ((parents :vs (shot12 shot13))
          (descriptions :v (owner bianca))))

(new-instance {SHOT}
  :name shot12
  :slots ((in-point :v 26763)
          (out-point :v 27095)
```

```
          (characters :vs (bianca12 carl12))
          (objects :vs (wine1 his-glass12 her-glass12
                              his-plate12 her-plate12))
          (environment :v kitchen10)
          (camera :v camera12)))


;; shot 13

(new-instance {BIANCA}
  :name bianca13
  :slots ((parents :v     shot13)
          (frame :v     (27427          27543
                                        27830          end))
          (visibility :v (full          full
                                        full           out))
          (face :v       (front         front        front))
          (angle :v      (left          headon       right))
          (distance :v   (med-long      med        med-close))
          (location :v   (center        center       right))
          (movement :v   (small         left         complex))
          (activity :v   ((get food)
                         (put food)
                         ((get & put) corkscrew / sit)))
          ))


(new-instance {CARL}
  :name carl13
  :slots ((parents :v     shot13)
          (frame :v     (27838          end))
          (visibility :v (out           partial))
          (face :v       (na            left-side))
          (angle :v      (na            low))
          (distance :v   (na            med-close))
          (location :v   (na            left))
          (movement :v   (na            small))
          (activity :v   (na
                         seated))
          ))

(new-instance {CAMERA}
  :name camera13
  :slots ((parents :v shot13)
```

```
          (frame :v (27435 27559 27828 end))
          (movement :v (small pan-left
                        (pan-right / pan-left) pan-left))))

(new-instance {CORKSCREW}
  :name corkscrew13
  :slots ((parents :v shot13)
          (descriptions :v (owner bianca)))

(new-instance {CLOCK}
  :name clock13
  :slots ((parents :v shot13)
          (descriptions :v (time 7:50 (27508 27582)
                                 (27691 27735)))))

(new-instance {SHOT}
  :name shot13
  :slots ((in-point :v 27096)
          (out-point :v 27879)
          (characters :vs (bianca13 carl13))
          (objects :vs (food11 corkscrew13 clock13 his-glass12
                        her-glass12 his-plate12 her-plate12))
          (environment :v kitchen10))
          (camera :v camera13)))


;; shot 14

(new-instance {BIANCA}
  :name bianca14
  :slots ((parents :v     shot14)
          (frame :v     (end))
          (visibility :v (full))
          (face :v       (front))
          (angle :v      ((high headon)))
          (distance :v   (med-close))
          (location :v   (center))
          (movement :v   (small))
          (activity :v   ((use corkscrew)))
          ))

(new-instance {CAMERA}
  :name camera14
  :slots ((parents :v shot14)
```

```
          (frame :v (27950 end)
          (movement :v ((rise + tilt-down) small)))))

(new-instance {SHOT}
  :name shot14
  :slots ((in-point :v 27880)
          (out-point :v 28101)
          (characters :vs (bianca14)
          (objects :vs (corkscrew0 wine0))
          (environment :v kitchen0)
          (camera :v camera14)))

;; shot 15

(new-instance {BIANCA}
  :name bianca15
  :slots ((parents :v     shot15)
          (frame :v     (28443          28594          end))
          (visibility :v ((partial hands) out       partial))
          (face :v       (na             na             back))
          (angle :v      (na             na       (high left)))
          (distance :v   (close          na         med-close))
          (location :v   (center         na            right))
          (movement :v   (small          na           complex))
          (activity :v   ((seated / use corkscrew / open wine)
                         na
                 (pour wine for carl / pour wine for bianca)))
          ))


(new-instance {CARL}
  :name carl15
  :slots ((parents :v     shot15)
          (frame :v     (28451          28602
                                        28843          end))
          (visibility :v (out           (partial face)
                                        (partial hands)  out))
          (face :v       (na            front))
          (angle :v      (na            (high left)))
          (distance :v   (na            close          close))
          (location :v   (na            center         left))
          (movement :v   (na            small          small))
          (activity :v   (na
                         seated
```

```
                        (receive wine)))
          ))

(new-instance {CAMERA}
  :name camera15
  :slots ((parents :v shot15)
          (frame :v (28429 28490 28502 28614 28815 end))
          (movement :v (small (tilt-up + pan-left) small
                   (tilt-down + pan-right) small pan-right))))

(new-instance {SHOT}
  :name shot15
  :slots ((in-point :v 28102)
          (out-point :v 29143)
          (characters :vs (bianca15 carl15))
      (objects :vs (corkscrew0 wine0 his-glass0 her-glass0))
          (environment :v kitchen0)
          (camera :v camera15)))


;; shot 16

(new-instance {BIANCA}
  :name bianca16
  :slots ((parents :v     shot16)
          (frame :v     (end))
          (visibility :v (full))
          (face :v       (front))
          (angle :v      (right))
          (distance :v   (med))
          (location :v   (right))
          (movement :v   (small))
          (activity :v   ((say that (bianca & carl) toast
                               / toast / drink glass)))
          ))


(new-instance {CARL}
  :name carl16
  :slots ((parents :v     shot16)
          (frame :v     (end))
          (visibility :v (partial))
          (face :v       (back))
          (angle :v      ((high right)))
```

```
          (distance :v   (med-close))
          (location :v   (left))
          (movement :v   (small))
          (activity :v   ((toast / drink glass)))
          ))

(new-instance {HIS-GLASS}
  :name his-glass16
  :slots ((parents :vs (shot16 shot18 shot1901 shot1902 shot20
                        shot21 shot25 shot26 shot27 shot28 shot29
                          shot31 shot32 shot33))
          (descriptions :v full)))

(new-instance {HER-GLASS}
  :name her-glass16
  :slots ((parents :vs (shot16 shot18 shot1901 shot1902 shot20
                        shot23 shot29 shot30 shot31 shot32 shot34
                          shot35))
          (descriptions :v full))))

(new-instance {WINE}
  :name wine16
  :slots ((parents :vs (shot16 shot18 shot1902 shot20
                           shot29 shot31 shot32 shot34 shot36))
          (descriptions :vs (open full))))

(new-instance {SHOT}
  :name shot16
  :slots ((in-point :v 29144)
          (out-point :v 2451)
          (characters :vs (bianca16 carl16))
          (objects :vs (wine16 food0 his-glass16 her-glass16))
          (environment :v kitchen0)
          (camera :v camera0)))


;; shot 17

(new-instance {CARL}
  :name carl17
  :slots ((parents :v     shot17)
          (frame :v     (29771          end))
          (visibility :v (full          partial))
          (face :v       (front         front))
```

```
          (angle :v      ((left headon) (low left)))
          (distance :v   (med-close            med-close))
          (location :v   (center               left))
          (movement :v   (small          small))
          (activity :v   ((seated / use corkscrew / open wine)
                 (pour wine for carl / pour wine for bianca)))
          ))


(new-instance {BIANCA}
  :name bianca17
  :slots ((parents :v     shot17)
          (frame :v     (29809          29990          end))
          (visibility :v (out           (partial hand)  out))
          (face :v       (na))
          (angle :v      (na))
          (distance :v   (na            med-close))
          (location :v   (na            right))
          (movement :v   (na            small))
          (activity :v   (na
                        (receive wine)))
          ))


(new-instance {SHOT}
  :name shot17
  :slots ((in-point :v 29452)
          (out-point :v 30270)
          (characters :vs (bianca17 carl17))
   (objects :vs (food0 his-plate0 wine0 her-glass0 his-glass0))
          (environment :v kitchen0)
          (camera :v camera0)))

;; shot 18

(new-instance {BIANCA}
  :name bianca18
  :slots ((parents :v     shot18)
          (frame :v     (end))
          (visibility :v ((partial hand)))
          (face :v       (na))
          (angle :v      (na))
          (distance :v   (ex-close))
          (location :v   (right))
```

```
          (movement :v   (complex))
          (activity :v   (toast))
          ))


(new-instance {CARL}
  :name carl18
  :slots ((parents :v     shot18)
          (frame :v     (end))
          (visibility :v ((partial hand)))
          (face :v       (na))
          (angle :v      (na))
          (distance :v   (ex-close))
          (location :v   (left))
          (movement :v   (complex))
          (activity :v   (toast))
          ))

(new-instance {SHOT}
  :name shot18
  :slots ((in-point :v 30271)
          (out-point :v 30391)
          (characters :vs (bianca18 carl18))
          (objects :vs (wine16 his-glass16 her-glass16))
          (environment :v kitchen0))
          (camera :v camera0)))

;; shot 19, in two parts, 1901 and 1902
;; (or could be one shot if damage descriptions supported)

(new-instance {CARL}
  :name carl1901
  :slots ((parents :v     shot1901)
          (frame :v     (end))
          (visibility :v (full))
          (face :v       (front))
          (angle :v      ((low left)))
          (distance :v   (med-close))
          (location :v   (left))
          (movement :v   (small))
          (activity :v   ((serve food for bianca)))
          ))
```

```
(new-instance {BIANCA}
  :name bianca1901
  :slots ((parents :v     shot1901)
          (frame :v     (end))
          (visibility :v (out))
          (face :v       (na))
          (angle :v      (na))
          (distance :v   (na))
          (location :v   (na))
          (movement :v   (na))
          (activity :v   ((receive food)))
          ))

(new-instance {CONTEXT}
  :name context1901
  :slots ((resultants :v shot1902)))

(new-instance {SHOT}
  :name shot1901
  :slots ((in-point :v 30392)
          (out-point :v 30589)
          (characters :vs (bianca1901 carl1901))
          (objects :vs (his-glass16 her-glass16
                        his-plate0 her-plate0 food0))
          (environment :v kitchen0)
          (context :v context1901)
          (camera :v camera0)))


(new-instance {CARL}
  :name carl1902
  :slots ((parents :v     shot1902)
          (frame :v     (end))
          (visibility :v (partial))
          (face :v       (front))
          (angle :v      ((low left)))
          (distance :v   (med-close))
          (location :v   (left))
          (movement :v   (complex))
          (activity :v   ((serve food for bianca (start 30817)
                           / serve food for carl (30817 end))))
          ))

(new-instance {BIANCA}
```

```
  :name bianca1902
  :slots ((parents :v     shot1902)
          (frame :v     (30817          end))
          (visibility :v ((partial hands)                out))
          (face :v       (na))
          (angle :v      (na))
          (distance :v   (close))
          (location :v   (right))
          (movement :v   (small))
          (activity :v   ((receive food)))
          ))

(new-instance {FOOD}
  :name food19
  :slots ((parents :vs (shot1902 shot28 shot29 shot31 shot32))
          (descriptions :vs (served full))))

(new-instance {CONTEXT}
  :name context1902
  :slots ((precedents :v shot1901)))

(new-instance {SHOT}
  :name shot1902
  :slots ((in-point :v 30590)
          (out-point :v 30950)
          (characters :vs (bianca1902 carl1902))
          (objects :vs (wine16 his-glass16 her-glass16
                       his-plate0 her-plate0 food19))
          (environment :v kitchen0)
          (context :v context1902)
          (camera :v camera0)))

;; shot 20

(new-instance {BIANCA}
  :name bianca20
  :slots ((parents :v     shot20)
          (frame :v     (end))
          (visibility :v (partial))
          (face :v       (right-side))
          (angle :v      (high))
          (distance :v   (med))
          (location :v   (right))
          (movement :v   (complex))
```

```
        (activity :v   ((serve food for carl (31006 31627)
                        / serve food for bianca (31669 end))))
        ))


(new-instance {CARL}
  :name carl20
  :slots ((parents :v     shot20)
          (frame :v     (31196          31690          end))
          (visibility :v (out           full           out))
          (face :v       (na            front))
          (angle :v      (na            left))
          (distance :v   (na            med))
          (location :v   (na            left))
          (movement :v   (na            complex))
          (activity :v   (na
                         (receive food)))
          ))

(new-instance {CAMERA}
  :name camera20
  :slots ((parents :v shot20)
          (frame :v (31049 31649 end)
          (movement :v ((tilt-down + pan-left)
                        small pan-right small)))))

(new-instance {CLOCK}
  :name clock20
  :slots ((parents :v shot20)
          (descriptions :v (time 7:54 (start 30979)))))

(new-instance {SHOT}
  :name shot20
  :slots ((in-point :v 30951)
          (out-point :v 32104)
          (characters :vs (bianca20 carl20))
          (objects :vs (food0 wine16 his-glass16 her-glass16
                              his-plate0 her-plate0 clock20))
          (environment :v kitchen0)
          (camera :v camera20)))

;; shot 21

(new-instance {CARL}
```

```
  :name carl21
  :slots ((parents :v     shot21)
          (frame :v     (end))
          (visibility :v (partial))
          (face :v       (front))
          (angle :v      ((low left)))
          (distance :v   (ex-close))
          (location :v   (left))
          (movement :v   (complex))
          (activity :v   (eat))
          ))

(new-instance {HIS-PLATE}
  :name his-plate21
  :slots ((parents :vs (shot21 shot25 shot26 shot28
                               shot29 shot31 shot32 shot33))
          (descriptions :v full)))

(new-instance {SHOT}
  :name shot21
  :slots ((in-point :v 32105)
          (out-point :v 32267)
          (characters :v carl21)
          (objects :vs (his-glass16 his-plate21))
          (environment :v kitchen0)
          (camera :v camera0)))


;; shot 22

(new-instance {BIANCA}
  :name bianca22
  :slots ((parents :v     shot22)
          (frame :v     (end))
          (visibility :v ((partial face)))
          (face :v       (front))
          (angle :v      ((high right)))
          (distance :v   (ex-close))
          (location :v   (right))
          (movement :v   (small))
          (activity :v   (eat))
          ))
```

```
(new-instance {HER-PLATE}
  :name her-plate22
  :slots ((parents :vs (shot22 shot23 shot24 shot29
                        shot30 shot31 shot32 shot34 shot35))
          (descriptions :v full)))

(new-instance {SHOT}
  :name shot22
  :slots ((in-point :v 32268)
          (out-point :v 32434)
          (characters :v bianca22)
          (objects :vs (her-plate22))
          (environment :v kitchen0)
          (camera :v camera0)))


;; shot 23

(new-instance {BIANCA}
  :name bianca23
  :slots ((parents :v     shot23)
          (frame :v     (end))
          (visibility :v (full))
          (face :v       (front))
          (angle :v      ((low right)))
          (distance :v   (close))
          (location :v   ((center right)))
          (movement :v   (small))
          (activity :v   (eat))
          ))

(new-instance {CAMERA}
  :name camera23
  :slots ((parents :v shot23)
          (frame :v (32478 end))
          (movement :v (tilt-down small))))

(new-instance {CLOCK}
  :name clock23
  :slots ((parents :v shot23)
          (descriptions :v (time 7:54 (start 32460)))))

(new-instance {HER-PLATE}
  :name her-plate22
```

```
  :slots ((parents :vs (shot22))
          (descriptions :v full)))

(new-instance {SHOT}
  :name shot23
  :slots ((in-point :v 32435)
          (out-point :v 32577)
          (characters :v bianca23)
          (objects :vs (her-plate22 her-glass16 clock23)
          (environment :v kitchen0)
          (camera :v camera23)))

;;shot 24

(new-instance {BIANCA}
  :name bianca24
  :slots ((parents :v     shot24)
          (frame :v     (end))
          (visibility :v ((partial hand)))
          (face :v       (na))
          (angle :v      (high))
          (distance :v   (ex-close))
          (location :v   (center))
          (movement :v   (small))
          (activity :v   (eat))
          ))


(new-instance {SHOT}
  :name shot24
  :slots ((in-point :v 32578)
          (out-point :v 32644)
          (characters :v bianca24)
          (objects :vs (her-plate22))
          (environment :v kitchen0)
          (camera :v camera0)))

;; shot 25

(new-instance {CARL}
  :name carl25
  :slots ((parents :v     shot25)
          (frame :v     (end))
          (visibility :v (full))
```

```
          (face :v       (front))
          (angle :v      (headon))
          (distance :v   (med-close))
          (location :v   ((center top)))
          (movement :v   (small))
          (activity :v   (eat))
          ))

(new-instance {FOOD}
  :name food25
  :slots ((parents :vs (shot25 shot26))
          (descriptions :vs (served half-full))))

(new-instance {SHOT}
  :name shot25
  :slots ((in-point :v 32645)
          (out-point :v 32801)
          (characters :v carl25)
          (objects :vs (food25 his-plate21 his-glass16))
          (environment :v kitchen0)
          (camera :v camera0)))

;;shot 26

(new-instance {CARL}
  :name carl26
  :slots ((parents :v     shot26)
          (frame :v     (end))
          (visibility :v (full))
          (face :v       (front))
          (angle :v      (left))
          (distance :v   (med-close))
          (location :v   (center))
          (movement :v   (complex))
          (activity :v   ((eat + is somewhat mad )))
          ))


(new-instance {SHOT}
  :name shot26
  :slots ((in-point :v 32802)
          (out-point :v 32915)
          (characters :v carl26)
          (objects :vs (food25 his-plate21 his-glass16))
```

```
          (environment :v kitchen0)
          (camera :v camera0)))

;; shot 27

(new-instance {CARL}
  :name carl27
  :slots ((parents :v     shot27)
          (frame :v     (end))
          (visibility :v (partial))
          (face :v       (front))
          (angle :v      ((high left)))
          (distance :v   (ex-close))
          (location :v   (center))
          (movement :v   (complex))
          (activity :v   ((eat + is mad)))
          ))

(new-instance {HIS-PLATE}
  :name his-plate27
  :slots ((parents :vs (shot27))
          (descriptions :v half-full)))

(new-instance {CAMERA}
  :name camera27
  :slots ((parents :v shot27)
          (frame :v (32953 33009 end)
          (movement :v (small tilt-up small)))))

(new-instance {SHOT}
  :name shot27
  :slots ((in-point :v 32916)
          (out-point :v 33078)
          (characters :v carl27)
          (objects :vs (his-plate27 his-glass16))
          (environment :v kitchen0)
          (camera :v camera0)))

;; shot 28

(new-instance {CARL}
  :name carl28
  :slots ((parents :v     shot28)
          (frame :v     (end))
```

```
          (visibility :v (full))
          (face :v       (left-side))
          (angle :v      (right))
          (distance :v   (close))
          (location :v   ((center left)))
          (movement :v   (complex))
          (activity :v   (eat))
          ))

(new-instance {CAMERA}
  :name camera28
  :slots ((parents :v shot28)
          (frame :v (33126 end)
          (movement :v ((tilt-down + pan-right) small)))))

(new-instance {SHOT}
  :name shot28
  :slots ((in-point :v 33079)
          (out-point :v 33167)
          (characters :v carl28)
          (objects :vs (food19 his-plate21 his-glass16))
          (environment :v kitchen0)
          (camera :v camera28)))

;; shot 29

(new-instance {BIANCA}
  :name bianca29
  :slots ((parents :v     shot29)
          (frame :v     (33296          33416          end))
          (visibility :v (full          out            full))
          (face :v       (right-side    na        right-side))
          (angle :v      (left          na             left))
          (distance :v   (close         na            close))
          (location :v   (right         na            right))
          (movement :v   (small         na            small))
          (activity :v   (eat
                                 na
                                 eat))
          ))


(new-instance {CARL}
  :name carl29
```

```
  :slots ((parents :v     shot29)
          (frame :v     (33304          33415          end))
          (visibility :v (out           full           out))
          (face :v       (na            front))
          (angle :v      (na            left))
          (distance :v   (na            close))
          (location :v   (na            left))
          (movement :v   (na            small))
          (activity :v   (na
                                eat))
          ))


(new-instance {CAMERA}
  :name camera29
  :slots ((parents :v shot29)
          (frame :v (33278 33333 33398 33456 end))
     (movement :v (small pan-left small pan-right small))))

(new-instance {SHOT}
  :name shot29
  :slots ((in-point :v 33168)
          (out-point :v 33657)
          (characters :vs (carl29 bianca29))
          (objects :vs (food19 his-plate21 his-glass16
                               her-plate22 her-glass16 wine16))
          (environment :v kitchen0)
          (camera :v camera29)))

;; shot 30

(new-instance {BIANCA}
  :name bianca30
  :slots ((parents :v     shot30)
          (frame :v     (end))
          (visibility :v (full))
          (face :v       (right-side))
          (angle :v      (headon))
          (distance :v   (close))
          (location :v   (right))
          (movement :v   (small))
          (activity :v   (eat))
          ))
```

```
(new-instance {SHOT}
  :name shot30
  :slots ((in-point :v 33658)
          (out-point :v 33887)
          (characters :vs (bianca30))
          (objects :vs (her-plate22 her-glass16))
          (environment :v kitchen0)
          (camera :v camera0)))

;; shot 31

(new-instance {BIANCA}
  :name bianca31
  :slots ((parents :v     shot31)
          (frame :v     (34062          end))
          (visibility :v (full          out))
          (face :v       (right-side))
          (angle :v      (high))
          (distance :v   (med))
          (location :v   (right))
          (movement :v   (small))
          (activity :v   (eat))
          ))


(new-instance {CARL}
  :name carl31
  :slots ((parents :v     shot31)
          (frame :v     (34056          end))
          (visibility :v (out           full))
          (face :v       (na            left-side))
          (angle :v      (na            high))
          (distance :v   (na            med))
          (location :v   (na            left))
          (movement :v   (na            small))
          (activity :v   (na
                                 eat))
          ))

(new-instance {CAMERA}
  :name camera31
  :slots ((parents :v shot31)
          (frame :v (34045 34097 end))
          (movement :v (small pan-left small))))
```

```
(new-instance {SHOT}
  :name shot31
  :slots ((in-point :v 33888)
          (out-point :v 34295)
          (characters :vs (carl31 bianca31))
          (objects :vs (food19 his-plate21 his-glass16
                               her-plate22 her-glass16 wine16))
          (environment :v kitchen0)
          (camera :v camera31)))

;; shot 32

(new-instance {CARL}
  :name carl32
  :slots ((parents :v     shot32)
          (frame :v     (end))
          (visibility :v (full))
          (face :v       (left-side))
          (angle :v      (hgih))
          (distance :v   (med-long))
          (location :v   (left))
          (movement :v   (small))
          (activity :v   ((eat + drink)))
          ))


(new-instance {BIANCA}
  :name bianca32
  :slots ((parents :v     shot32)
          (frame :v     (end))
          (visibility :v (full))
          (face :v       (right-side))
          (angle :v      (high))
          (distance :v   (med-long))
          (location :v   (right))
          (movement :v   (small))
          (activity :v   (eat))
          ))

(new-instance {SHOT}
  :name shot32
  :slots ((in-point :v 34296)
          (out-point :v 34605)
          (characters :vs (carl32 bianca32))
```

```
          (objects :vs (food19 his-plate21 his-glass16
                               her-plate22 her-glass16 wine16))
          (environment :v kitchen0)
          (camera :v camera0)))

;; shot 33

(new-instance {CARL}
  :name carl33
  :slots ((parents :v     shot33)
          (frame :v     (end))
          (visibility :v (full))
          (face :v       (front))
          (angle :v      (left))
          (distance :v   (close))
          (location :v   ((left center)))
          (movement :v   (small))
     (activity :v   ((drink (start 34731) / eat (34748 end))))
          ))

(new-instance {SHOT}
  :name shot33
  :slots ((in-point :v 34606)
          (out-point :v 34871)
          (characters :vs (carl33))
          (objects :vs (his-plate21 his-glass16 ))
          (environment :v kitchen0)
          (camera :v camera0)))

;; shot 34

(new-instance {BIANCA}
  :name bianca34
  :slots ((parents :v     shot34)
          (frame :v     (end))
          (visibility :v (full))
          (face :v       (front))
          (angle :v      (right))
          (distance :v   (close))
          (location :v   ((right center)))
          (movement :v   (small))
          (activity :v   ((drink (start 34978) / is happy
                                 / eat (34995 end))))
          ))
```

```
(new-instance {SHOT}
  :name shot34
  :slots ((in-point :v 34872)
          (out-point :v 35119)
          (characters :vs (bianca34))
          (objects :vs (her-plate22 her-glass16 wine16))
          (environment :v kitchen0)
          (camera :v camera0)))

;; shot 35

(new-instance {BIANCA}
  :name bianca35
  :slots ((parents :v     shot35)
          (frame :v     (end))
          (visibility :v (full))
          (face :v       (front))
          (angle :v      (right))
          (distance :v   (med-close))
          (location :v   (right))
          (movement :v   (complex))
          (activity :v   ((drink / is mad
                                 / is impatient (35385 end))))
          ))

(new-instance {SHOT}
  :name shot35
  :slots ((in-point :v 35120)
          (out-point :v 35466)
          (characters :vs (bianca35))
          (objects :vs (her-plate22 her-glass16))
          (environment :v kitchen0)
          (camera :v camera0)))

;;shot 36

(new-instance {BIANCA}
  :name bianca36
  :slots ((parents :v     shot36)
          (frame :v     (35525          end))
          (visibility :v (full          out))
          (face :v       (right-side))
          (angle :v      (left))
          (distance :v   (med))
```

```
          (location :v   ((right center)))
          (movement :v   (none))
          (activity :v   (seated))
          ))


(new-instance {CARL}
  :name carl36
  :slots ((parents :v     shot36)
          (frame :v     (33509          end))
          (visibility :v (out           full))
          (face :v       (na            left-side))
          (angle :v      (na            headon))
          (distance :v   (na            med-close))
          (location :v   (na            left))
          (movement :v   (na            small))
          (activity :v   (na
                (drink / is (mad or impatient) (35637 end))))
          ))

(new-instance {CAMERA}
  :name camera36
  :slots ((parents :v shot36)
          (frame :v (35504 35543 end))
          (movement :v (none pan-left small))))

(new-instance {HIS-GLASS}
  :name his-glass36
  :slots ((parents :vs (shot36)
          (descriptions :v (half-full (35514 end))))))

(new-instance {HER-GLASS}
  :name her-glass36
  :slots ((parents :vs (shot36))
          (descriptions :v (half-full (start 35540)))))

(new-instance {HIS-PLATE}
  :name his-plate36
  :slots ((parents :vs (shot36))
          (descriptions :v (half-full (35513 end)))))

(new-instance {HER-PLATE}
  :name her-plate36
  :slots ((parents :vs (shot36))
```

```
          (descriptions :v (half-full (start 35529)))))

(new-instance {FOOD}
  :name food36
  :slots ((parents :vs (shot36))
          (descriptions :vs (half-full (35510 end)))))

(new-instance {SHOT}
  :name shot36
  :slots ((in-point :v 35467)
          (out-point :v 35720)
          (characters :vs (bianca36 carl36))
          (objects :vs (wine16 food36 his-plate36
                       her-plate36 his-glass36 her-glass36))
          (environment :v kitchen0)
          (camera :v camera36)))

;; shot 37

(new-instance {BIANCA}
  :name bianca37
  :slots ((parents :v      shot37)
          (frame :v      (35753          end))
          (visibility :v (out            full))
          (face :v       (na             right-side))
          (angle :v      (na             headon))
          (distance :v   (na             close))
          (location :v   (na             center))
          (movement :v   (na             none))
          (activity :v   (na
                                 seated))
          ))

;; shot 38

(new-instance {CARL}
  :name carl38
  :slots ((parents :v      shot38)
          (frame :v      (end))
          (visibility :v (full))
          (face :v       (front))
          (angle :v      ((high left)))
          (distance :v   (med))
          (location :v   (center))
```

```
          (movement :v   (small))
          (activity :v   ((sigh (start 35869)
                                / drink + is mad)))
          ))

;; shot 39

(new-instance {BIANCA}
  :name bianca39
  :slots ((parents :v     shot39)
          (frame :v     (end))
          (visibility :v (full))
          (face :v       (front))
          (angle :v      ((low headon right)))
          (distance :v   (med))
          (location :v   ((bottom right)))
          (movement :v   (small))
   (activity :v   ((say text1) + is mad / sigh (36366 end))))
          ))

(new-instance {TEXT}
  :name text1
  :slots ((parents :vs (shot39 shot42))
 (transcription :v (the true artist is one who can make you say
                   'wow!', to which he replies 'so what?'))
          (descriptions :vs (opinion art truth vanity))))

;; shot 40

(new-instance {CARL}
  :name carl40
  :slots ((parents :v     shot40)
          (frame :v     (end))
          (visibility :v (full))
          (face :v       (front))
          (angle :v      ((high left)))
          (distance :v   (med))
          (location :v   (center))
          (movement :v   (small))
          (activity :v   ((say text2 / drink (36667 end))))
          ))

(new-instance {TEXT}
  :name text2
```

```
  :slots ((parents :vs (shot40 shot41))
 (transcription :v (in this country, education is obtained
                   only at a price. Indoctrination is free,
                   but often disguised as education.))
    (descriptions :vs (opinion problem education freedom))))

;; shot 41

(new-instance {CARL}
  :name carl41
  :slots ((parents :v     shot41)
          (frame :v     (end))
          (visibility :v (full))
          (face :v       (front))
          (angle :v      ((low left)))
          (distance :v   (med-close))
          (location :v   (center))
          (movement :v   (small))
          (activity :v   ((say text2 + is mad)))
          ))

;; shot 42

(new-instance {BIANCA}
  :name bianca42
  :slots ((parents :v     shot42)
          (frame :v     (end))
          (visibility :v (full))
          (face :v       (front))
          (angle :v      ((high right)))
          (distance :v   (med))
          (location :v   (center))
          (movement :v   (small))
          (activity :v   ((say text1 + is thoughtful)))
          ))

;; shot 43

(new-instance {CARL}
  :name carl43
  :slots ((parents :v     shot43)
          (frame :v     (end))
          (visibility :v (full))
          (face :v       (front))
```

```
          (angle :v      (headon))
          (distance :v   (close))
          (location :v   (center))
          (movement :v   (small))
          (activity :v   ((say text3 + is thoughtful
                               / drink (37559 end))))
          ))

(new-instance {TEXT}
   :name text3
   :slots ((parents :v shot43)
           (transcription :v
            (where is the coninuity of self? I have only some
              rapidly decaying flesh in common with yesterday.
                          Soon, I will become unrecognizable.))
           (descriptions :vs (life self fear))))

;; shot 44

(new-instance {BIANCA}
  :name bianca44
  :slots ((parents :v     shot44)
          (frame :v     (end))
          (visibility :v ((partial face)))
          (face :v       (front))
          (angle :v      ((low right)))
          (distance :v   (ex-close))
          (location :v   ((center right)))
          (movement :v   (small))
          (activity :v   ((say text4 (start 38087)
                  / is thoughtful / drink slow (38040 38199)
                  / drink fast all (38192 end))))
          ))

(new-instance {TEXT}
  :name text4
  :slots ((parents :v shot44)
          (transcription :v
       (I have become superstitious. History is reducible
       to a numinous sequence. We spend our lives preparing
              for a single moment of symbolic relevance.))
          (descriptions :vs (self life history meaning))))

;; shot 45
```

```
(new-instance {BIANCA}
  :name bianca45
  :slots ((parents :v     shot45)
          (frame :v     (end))
          (visibility :v (full))
          (face :v       (left-side))
          (angle :v      ((very high)))
          (distance :v   (med))
          (location :v   (center))
          (movement :v   (small))
          (activity :v   ((play with food (start 38507)
                                / say text4a)))
          ))

(new-instance {TEXT}
  :name text4a
  :slots ((parents :v shot45)
          (transcription :v (I have become superstitious.))
          (descriptions :vs (self magic))))

;; shot 46

(new-instance {CARL}
  :name carl46
  :slots ((parents :v     shot46)
          (frame :v     (end))
          (visibility :v (full))
          (face :v       (left-side))
          (angle :v      ((very high)))
          (distance :v   (med))
          (location :v   ((left center)))
          (movement :v   (small))
          (activity :v   ((say text5)))
          ))

(new-instance {TEXT}
  :name text5
  :slots ((parents :v shot46)
          (transcription :v (this is the age of polymorphism!
               the whims of nature invite whimsical response.))
          (descriptions :vs (opinion present mankind))))

;; shot 47
```

```
(new-instance {BIANCA}
  :name bianca47
  :slots ((parents :v     shot47)
          (frame :v     (end))
          (visibility :v (full))
          (face :v       (front))
          (angle :v      (right))
          (distance :v   (close))
          (location :v   (right))
          (movement :v   (small))
          (activity :v   ((play with food
                        / show that bianca is uninterested)))
          ))

;; shot 48

(new-instance {BIANCA}
  :name bianca48
  :slots ((parents :v     shot48)
          (frame :v     (end))
          (visibility :v (full))
          (face :v       (front))
          (angle :v      (right))
          (distance :v   (close))
          (location :v   ((right center)))
          (movement :v   (small))
          (activity :v   ((show agreement (start 39066)
                          / show impatience (39066 end))))
          ))

;; shot 49

(new-instance {BIANCA}
  :name bianca49
  :slots ((parents :v     shot49)
          (frame :v     (end))
          (visibility :v (full))
          (face :v       (front))
          (angle :v      (right))
          (distance :v   (close))
          (location :v   ((center right)))
          (movement :v   (right))
          (activity :v   ((show anger (39291 end)
                          / show agreement)))
```

```
        ))

;; shot 50

(new-instance {CARL}
  :name carl50
  :slots ((parents :v     shot50)
          (frame :v     (end))
          (visibility :v (full))
          (face :v       (front))
          (angle :v      (left))
          (distance :v   (close))
          (location :v   (center))
          (movement :v   (small))
          (activity :v   ((show disinterest (start 39468)
                  / show agreement & is thoughtful (39453 39563)
          / drink (39563 39622) / show impatience (39620 end))))
          ))


(new-instance {CARL}
  :name carl51
  :slots ((parents :v     shot51)
          (frame :v     (end))
          (visibility :v (full))
          (face :v       (front))
          (angle :v      (left))
          (distance :v   (med))
          (location :v   (center))
          (movement :v   (small))
          (activity :v   ((show interest (start 39887)
                            / show impatience (39887 end))))
          ))


(new-instance {BIANCA}
  :name bianca52
  :slots ((parents :v     shot52)
          (frame :v     (end))
          (visibility :v (full))
          (face :v       (right-side))
          (angle :v      (left))
          (distance :v   (close))
          (location :v   ((center left)))
```

```
        (movement :v   (small))
     (activity :v ((drink + show (impatience or disagreement)
             (start 40179) / drink + show anger (40085 end))))
        ))


(new-instance {BIANCA}
  :name bianca53
  :slots ((parents :v     shot53)
        (frame :v     (end))
        (visibility :v (full))
        (face :v       (front))
        (angle :v      (right))
        (distance :v   (med-close))
        (location :v   (right))
        (movement :v   (small))
        (activity :v   ((show anger (start 40515)
        / show (disagreement or impatience) (40505 end))))
        ))


(new-instance {BIANCA}
  :name bianca54
  :slots ((parents :v     shot54)
        (frame :v     (end))
        (visibility :v (full))
        (face :v       (front))
        (angle :v      (right))
        (distance :v   (med))
        (location :v   ((center right)))
        (movement :v   (small))
        (activity :v   ((show impatience)))
        ))


(new-instance {BIANCA}
  :name bianca55
  :slots ((parents :v     shot55)
        (frame :v     (end))
        (visibility :v ((partial face)))
        (face :v       (front))
        (angle :v      (right))
        (distance :v   (close))
        (location :v   (center))
```

```
          (movement :v   (small))
          (activity :v   ((eat + show (agreement or happy))))
          ))


(new-instance {CARL}
  :name carl56
  :slots ((parents :v     shot56)
          (frame :v     (end))
          (visibility :v (full))
          (face :v       (front))
          (angle :v      (left))
          (distance :v   (med-close))
          (location :v   (center))
          (movement :v   (small))
          (activity :v   ((show disagreement / drink)))
          ))


(new-instance {CARL}
  :name carl57
  :slots ((parents :v     shot57)
          (frame :v     (end))
          (visibility :v (full))
          (face :v       (front))
          (angle :v      (left))
          (distance :v   (med-close))
          (location :v   (center))
          (movement :v   (small))
          (activity :v   ((show agreement + is thoughtful)))
          ))


(new-instance {CARL}
  :name carl58
  :slots ((parents :v     shot58)
          (frame :v     (end))
          (visibility :v (full))
          (face :v       (front))
          (angle :v      (left))
          (distance :v   (med-close))
          (location :v   (center))
          (movement :v   (small))
          (activity :v   ((show agreement)))
```

```
        ))


(new-instance {CARL}
  :name carl59
  :slots ((parents :v     shot59)
          (frame :v     (end))
          (visibility :v (full))
          (face :v       (front))
          (angle :v      (left))
          (distance :v   (med-close))
          (location :v   (center))
          (movement :v   (small))
          (activity :v   ((show (anger or impatience)
                                / reply is very mad)))
          ))


(new-instance {CARL}
  :name carl60
  :slots ((parents :v     shot60)
          (frame :v     (end))
          (visibility :v ((partial face)))
          (face :v       (front))
          (angle :v      (left))
          (distance :v   (close))
          (location :v   (center))
          (movement :v   (small))
          (activity :v   ((reply disinterest)))
          ))


(new-instance {BIANCA}
  :name bianca61
  :slots ((parents :v     shot61)
          (frame :v     (end))
          (visibility :v (full))
          (face :v       (right-side))
          (angle :v      (high))
          (distance :v   (med-close))
          (location :v   (right))
          (movement :v   (small))
          (activity :v   ((show that bianca like food)))
          ))
```

```
(new-instance {BIANCA}
  :name bianca62
  :slots ((parents :v      shot62)
          (frame :v      (end))
          (visibility :v (full))
          (face :v       (front))
          (angle :v      (right))
          (distance :v   (close))
          (location :v   (right))
          (movement :v   (small))
          (activity :v   ((drink wine from glass
                                 / show dislike wine)))
          ))


(new-instance {BIANCA}
  :name bianca63
  :slots ((parents :v      shot63)
          (frame :v      (end))
          (visibility :v (full))
          (face :v       (right-side))
          (angle :v      (high))
          (distance :v   (med-close))
          (location :v   (right))
          (movement :v   (small))
          (activity :v   ((show dislike food)))
          ))


(new-instance {BIANCA}
  :name bianca64
  :slots ((parents :v      shot64)
          (frame :v      (end))
          (visibility :v (full))
          (face :v       (front))
          (angle :v      (right))
          (distance :v   (close))
          (location :v   ((center right)))
          (movement :v   (right))
          (activity :v   ((show dislike food / is impatient)))
          ))


(new-instance {BIANCA}
```

```
  :name bianca65
  :slots ((parents :v     shot65)
          (frame :v     (end))
          (visibility :v (full))
          (face :v       (front))
          (angle :v      (headon))
          (distance :v   (med-close))
          (location :v   (center))
          (movement :v   (small))
          (activity :v   ((drink / say like wine)))
          ))


(new-instance {BIANCA}
  :name bianca66
  :slots ((parents :v     shot66)
          (frame :v     (42076          end))
          (visibility :v (partial               full))
          (face :v       (front         front))
          (angle :v      (low           headon))
          (distance :v   (close         close))
          (location :v   ((top right)           center))
          (movement :v   (small         small))
          (activity :v
  ((is mad + ask that carl pour wine for bianca / receive wine)
                                (drink + is thoughtful)))
          ))


(new-instance {CARL}
  :name carl66
  :slots ((parents :v     shot66)
          (frame :v     (42075          end))
          (visibility :v ((partial hand)                out))
          (face :v       (na))
          (angle :v      (na))
          (distance :v   (na))
          (location :v   (na))
          (movement :v   (na))
          (activity :v   ((pour wine for bianca)))
          ))


(new-instance {BIANCA}
```

```
  :name bianca67
  :slots ((parents :v     shot67)
          (frame :v     (42308          end))
          (visibility :v (full          partial))
          (face :v       (front))
          (angle :v      (headon))
          (distance :v   (close))
          (location :v   (center))
          (movement :v   (small))
          (activity :v
            ((ask that carl (serve or pour) for bianca)
                                (receive wine)))
          ))


(new-instance {CARL}
  :name carl67
  :slots ((parents :v     shot67)
          (frame :v     (out))
          (visibility :v (na))
          (face :v       (na))
          (angle :v      (na))
          (distance :v   (na))
          (location :v   (na))
          (movement :v   (na))
          (activity :v   ((pour wine for bianca)))
          ))


(new-instance {CARL}
  :name carl68
  :slots ((parents :v     shot68)
          (frame :v     (end))
          (visibility :v (full))
          (face :v       (front))
          (angle :v      (headon))
          (distance :v   (close))
          (location :v   (center))
          (movement :v   (small))
          (activity :v   ((show dislike food + is mad
                                / is sick)))
          ))
```

```
(new-instance {CARL}
  :name carl69
  :slots ((parents :v     shot69)
          (frame :v     (end))
          (visibility :v (full))
          (face :v       (front))
          (angle :v      (headon))
          (distance :v   (close))
          (location :v   (center))
          (movement :v   (small))
          (activity :v   ((say is happy)))
          ))


(new-instance {CARL}
  :name carl70
  :slots ((parents :v     shot70)
          (frame :v     (end))
          (visibility :v (full))
          (face :v       (front))
          (angle :v      ((high headon)))
          (distance :v   (close))
          (location :v   (center))
          (movement :v   (small))
          (activity :v   ((say is happy / drink)))
          ))


(new-instance {CARL}
  :name carl71
  :slots ((parents :v     shot71)
          (frame :v     (end))
          (visibility :v (full))
          (face :v       (front))
          (angle :v      (headon))
          (distance :v   (med-close))
          (location :v   (center))
          (movement :v   (small))
          (activity :v   ((say dislike wine)))
          ))


(new-instance {CARL}
  :name carl72
```

```
  :slots ((parents :v     shot72)
          (frame :v     (43081          end))
          (visibility :v (partial               full))
          (face :v       (front          front))
          (angle :v      ((low left)    (headon left)))
          (distance :v   (close          close))
          (location :v   (left           left))
          (movement :v   (small          small))
          (activity :v   ((ask that bianca pour / receive wine)
                                (say like wine / drink)))
          ))


(new-instance {BIANCA}
  :name bianca72
  :slots ((parents :v     shot72)
          (frame :v     (42939          43080          end))
          (visibility :v (out           partial        out))
          (face :v       (na))
          (angle :v      (na))
          (distance :v   (na))
          (location :v   (na))
          (movement :v   (na))
          (activity :v   (na
                                (pour wine for carl)))
          ))


(new-instance {CARL}
  :name carl73
  :slots ((parents :v     shot73)
          (frame :v     (end))
          (visibility :v (full))
          (face :v       (front))
          (angle :v      (headon))
          (distance :v   (close))
          (location :v   (center))
          (movement :v   (small))
          (activity :v   ((say is sick)))
          ))


(new-instance {BIANCA}
  :name bianca74
```

```
  :slots ((parents :v     shot74)
          (frame :v     (end))
          (visibility :v (full))
          (face :v       (front))
          (angle :v      ((high right)))
          (distance :v   (med-close))
          (location :v   ((bottom center)))
          (movement :v   (small))
          (activity :v   ((say is happy)))
          ))


(new-instance {CARL}
  :name carl75
  :slots ((parents :v     shot75)
          (frame :v     (end))
          (visibility :v (full))
          (face :v       (front))
          (angle :v      (left))
          (distance :v   (close))
          (location :v   (center))
          (movement :v   (small))
          (activity :v   ((say is unhappy)))
          ))


(new-instance {CARL}
  :name carl76
  :slots ((parents :v     shot76)
          (frame :v     (end))
          (visibility :v (full))
          (face :v       (front))
          (angle :v      (left))
          (distance :v   (med))
          (location :v   (center))
          (movement :v   (small))
          (activity :v   ((say is mad)))
          ))


(new-instance {BIANCA}
  :name bianca77
  :slots ((parents :v     shot77)
          (frame :v     (end))
```

```
          (visibility :v (partial))
          (face :v       (na))
          (angle :v      (na))
          (distance :v   (med-close))
          (location :v   (left))
          (movement :v   (small))
          (activity :v   ((seated / play with cat)))
          ))


(new-instance {CAT}
  :name cat77
  :slots ((parents :v     shot77)
          (frame :v     (end))
          (visibility :v (full))
          (face :v       (front))
          (angle :v      ((high right)))
          (distance :v   (med-close))
          (location :v   ((top right)))
          (movement :v   ((down left)))
          (activity :v   (walk))
          ))


(new-instance {CARL}
  :name carl78
  :slots ((parents :v     shot78)
          (frame :v     (43692          end))
          (visibility :v (out           full))
          (face :v       (na            front))
          (angle :v      (na            headon))
          (distance :v   (na            med-close))
          (location :v   (na            center))
          (movement :v   (na            small))
          (activity :v   (na
                                (is happy (43271 end))))
          ))


(new-instance {BIANCA}
  :name bianca79
  :slots ((parents :v     shot79)
          (frame :v     (43930          end))
          (visibility :v (full          partial))
```

```
          (face :v       (back          left-side))
          (angle :v      ((low headon)         low))
          (distance :v   (med-close            close))
          (location :v   (center               right))
          (movement :v   (small         (left down)))
          (activity :v   ((get broom)
                                walk))
          ))


(new-instance {BIANCA}
  :name bianca80
  :slots ((parents :v     shot80)
          (frame :v     (end))
          (visibility :v (full))
          (face :v       (front))
          (angle :v      (high))
          (distance :v   (med))
          (location :v   ((center top)))
          (movement :v   (small))
          (activity :v   ((use broom / get glass)))
          ))


(new-instance {CARL}
  :name carl81
  :slots ((parents :v     shot81)
          (frame :v     (end))
          (visibility :v (full))
          (face :v       (front))
          (angle :v      (headon))
          (distance :v   (close))
          (location :v   (center))
          (movement :v   (small))
          (activity :v   ((is thoughtful)))
          ))


(new-instance {BIANCA}
  :name bianca81
  :slots ((parents :v     shot81)
          (frame :v     (end))
          (visibility :v (out))
          (face :v       (na))
```

```
          (angle :v      (na))
          (distance :v   (na))
          (location :v   (na))
          (movement :v   (na))
          (activity :v   ((put glass )))   ;; glass is broken
          ))


(new-instance {BIANCA}
  :name bianca82
  :slots ((parents :v     shot82)
          (frame :v     (end))
          (visibility :v (full))
          (face :v       (back))
          (angle :v      (low))
          (distance :v   (med-close))
          (location :v   (center))
          (movement :v   (small))
          (activity :v   (sit))
          ))


(new-instance {CARL}
  :name carl83
  :slots ((parents :v     shot83)
          (frame :v     (44612    44692          end))
          (visibility :v (full     out            full))
          (face :v       (front     na           front))
          (angle :v      (left      na           left))
          (distance :v   (med       na           med))
          (location :v   (center    na           center))
          (movement :v   (small     na           small))
          (activity :v   ((drop glass1  accidently)
                          ;; his glass
                               na
                               (say is sorry + show sad)))
          ))


(new-instance {CARL}
  :name carl84
  :slots ((parents :v     shot84)
          (frame :v     (44917          44972
                                        45222          end))
```

```
        (visibility :v (full          full
                                      full        partial))
        (face :v       (front         front        back))
        (angle :v      (headon       headon       headon))
        (distance :v   (med           med-close
                                    med-close        close))
        (location :v   (left          left
                                   right           center))
        (movement :v   (up             right
                                  complex         complex))
        (activity :v   (stand
                               walk
                               (get broom)
  (use broom / get glass / put (glass & broom) / exit left)))
        ))


(new-instance {BIANCA}
  :name bianca84
  :slots ((parents :v     shot84)
        (frame :v     (44977          45213
                                      45325           end))
        (visibility :v (out           partial
                                      out         partial))
        (face :v       (na            right-side
                                      na        right-side))
        (angle :v      (na            headon
                                      na           headon))
        (distance :v   (na            med
                                      na              med))
        (location :v   (na            center
                                      na           center))
        (movement :v   (na            small
                                      na            small))
        (activity :v   (na
                               seated))
        ))


(new-instance {BIANCA}
  :name bianca85
  :slots ((parents :v     shot85)
        (frame :v     (45632          45675       end))
        (visibility :v (full          partial     full))
```

```
        (face :v       (front          na          front))
        (angle :v      ((headon right)  na  (headon right)))
        (distance :v   (close          na          close))
        (location :v   (center         na          center))
        (movement :v   (small          na          small))
        (activity :v   ((drink / drop glass accidently)
                              na
                              (is sorry)))
        ))


(new-instance {BIANCA}
  :name bianca86
  :slots ((parents :v     shot86)
        (frame :v     (45843          end))
        (visibility :v (partial                full))
        (face :v       (na             front))
        (angle :v      (na             (low right)))
        (distance :v   (na             med-long))
        (location :v   (na             (up left)))
        (movement :v   (na             small))
        (activity :v   (na
                              (is happy)))
        ))


(new-instance {BIANCA}
  :name bianca87
  :slots ((parents :v     shot87)
        (frame :v     (end))
        (visibility :v (full))
        (face :v       (right-side))
        (angle :v      (left))
        (distance :v   (med-close))
        (location :v   (center))
        (movement :v   (small))
        (activity :v   ((throw glass)))
        ))


(new-instance {CARL}
  :name carl88
  :slots ((parents :v     shot88)
        (frame :v     (end))
```

```
          (visibility :v (partial))
          (face :v       (front))
          (angle :v      (headon))
          (distance :v   (ex-close))
          (location :v   (center))
          (movement :v   (complex))
          (activity :v   ((throw glass)))
          ))


(new-instance {CARL}
  :name carl89
  :slots ((parents :v     shot89)
          (frame :v     (end))
          (visibility :v ((partial face)))
          (face :v       (front))
          (angle :v      ((headon left)))
          (distance :v   (ex-close))
          (location :v   (center))
          (movement :v   (small))
          (activity :v   ((drink + is drunk (start 46335)
                          / drink + is impatient (46335 end))))
          ))


(new-instance {CARL}
  :name carl90
  :slots ((parents :v     shot90)
          (frame :v     (end))
          (visibility :v ((partial face)))
          (face :v       (front))
          (angle :v      ((headon left)))
          (distance :v   (ex-close))
          (location :v   (center))
          (movement :v   (small))
          (activity :v ((ask about bianca is drunk / is mad)))
          ))


(new-instance {CARL}
  :name carl91
  :slots ((parents :v     shot91)
          (frame :v     (end))
          (visibility :v (full))
```

```
          (face :v       (front))
          (angle :v      ((high headon left)))
          (distance :v   (med))
          (location :v   (center))
          (movement :v   (small))
          (activity :v   ((drink + is (drunk or happy))))
          ))


(new-instance {BIANCA}
  :name bianca92
  :slots ((parents :v     shot92)
          (frame :v     (end))
          (visibility :v (full))
          (face :v       (front))
          (angle :v      ((high right)))
          (distance :v   (med-close))
          (location :v   ((center right)))
          (movement :v   (small))
          (activity :v   ((ask about carl is drunk (start 46723)
            / laugh (46723 46818) / drink + laugh (46818 end))))
          ))


(new-instance {BIANCA}
  :name bianca93
  :slots ((parents :v     shot93)
          (frame :v     (end))
          (visibility :v (full))
          (face :v       (right-side))
          (angle :v      (headon))
          (distance :v   (close))
          (location :v   (center))
          (movement :v   (small))
          (activity :v   ((is (sick or unhappy or impatient))))
          ))


(new-instance {BIANCA}
  :name bianca94
  :slots ((parents :v     shot94)
          (frame :v     (end))
          (visibility :v ((partial face)))
          (face :v       (front))
```

```
          (angle :v      (headon))
          (distance :v   (ex-close))
          (location :v   (center))
          (movement :v   (small))
          (activity :v ((say is (unhappy or mad) (start 47450)
                                / sigh + is mad (47450 end))))
          ))


(new-instance {CARL}
  :name carl95
  :slots ((parents :v     shot95)
          (frame :v     (47620          end))
          (visibility :v (partial               out))
          (face :v       (back))
          (angle :v      ((high right)))
          (distance :v   (med-close))
          (location :v   ((low left)))
          (movement :v   (small))
          (activity :v   (eat))
          ))


(new-instance {BIANCA}
  :name bianca95
  :slots ((parents :v     shot95)
          (frame :v     (47620          end))
          (visibility :v (full          out))
          (face :v       (front))
          (angle :v      ((high right)))
          (distance :v   (med))
          (location :v   ((top right)))
          (movement :v   (small))
          (activity :v   (eat))
          ))


(new-instance {CARL}
  :name carl96
  :slots ((parents :v     shot96)
          (frame :v     (47729          end))
          (visibility :v (full          out))
          (face :v       (front))
          (angle :v      ((high left)))
```

```
          (distance :v   (med))
          (location :v   (left))
          (movement :v   (small))
          (activity :v   (eat))
          ))


(new-instance {BIANCA}
  :name bianca96
  :slots ((parents :v     shot96)
          (frame :v     (47750          end))
          (visibility :v (partial               out))
          (face :v       (back))
          (angle :v      ((high left)))
          (distance :v   (med-close))
          (location :v   ((low right)))
          (movement :v   (small))
          (activity :v   (eat))
          ))


(new-instance {BIANCA}
  :name bianca97
  :slots ((parents :v     shot97)
          (frame :v     (47837          end))
          (visibility :v (out           full))
          (face :v       (na            front))
          (angle :v      (na      (high center right)))
          (distance :v   (na            med-close))
          (location :v   (na            right))
          (movement :v   (na            small))
          (activity :v   (na
                                eat))
          ))


(new-instance {CARL}
  :name carl98
  :slots ((parents :v     shot98)
          (frame :v     (47989          48498          end))
          (visibility :v (out           partial        out))
          (face :v       (na            front))
          (angle :v      (na            left))
          (distance :v   (na            med))
```

```
        (location :v   (na              left))
        (movement :v   (na              complex))
        (activity :v   (na
                    (stands / clear table / exit right)))
        ))


(new-instance {BIANCA}
  :name bianca98
  :slots ((parents :v     shot98)
        (frame :v     (47977          48568       end))
        (visibility :v (out           partial      out))
        (face :v       (na            right-side))
        (angle :v      (na            headon))
        (distance :v   (na            med))
        (location :v   (na            right))
        (movement :v   (na            complex))
        (activity :v   (na
              (stand / help carl clear table / exit right)))
        ))

(new-instance {BIANCA}
  :name bianca99
  :slots ((parents :v     shot99)
        (frame :v     (48607          48911          end))
        (visibility :v (out           partial        out))
        (face :v       (na            right-side))
        (angle :v      (na            (high headon)))
        (distance :v   (na            med))
        (location :v   (na            right))
        (movement :v   (na            complex))
        (activity :v   (na
                       (stand / clear table / exit right)))
        ))


(new-instance {CARL}
  :name carl99
  :slots ((parents :v     shot99)
        (frame :v     (48626          end))
        (visibility :v (out           partial))
        (face :v       (front))
        (angle :v      (left))
        (distance :v   (med-long))
```

```
          (location :v   (left))
          (movement :v   (complex))
          (activity :v   ((stand
                     / help bianca clear table / exit right)))
          ))


(new-instance {CARL}
  :name carl100
  :slots ((parents :v     shot100)
          (frame :v     (49089          end))
          (visibility :v (full          out))
          (face :v       (front))
          (angle :v      (left))
          (distance :v   (med-close))
          (location :v   (center))
          (movement :v   (complex))
          (activity :v   ((show is mad / exit left)))
          ))


(new-instance {BIANCA}
  :name bianca100
  :slots ((parents :v     shot100)
          (frame :v     (49099          end))
          (visibility :v (out           full))
          (face :v       (na            right-side))
          (angle :v      (na            headon))
          (distance :v   (na            med-close))
          (location :v   (na            (center right)))
          (movement :v   (na            small))
          (activity :v   (na
                                (is (thoughtful or sad))))
          ))


(new-instance {BIANCA}
  :name bianca101
  :slots ((parents :v     shot101)
          (frame :v     (end))
          (visibility :v (full))
          (face :v       (front))
          (angle :v      (right))
          (distance :v   (med-close))
```

```
          (location :v   ((center right)))
          (movement :v   (small))
          (activity :v   ((say is sorry / is sad)))
          ))


(new-instance {CARL}
  :name carl101
  :slots ((parents :v     shot101)
          (frame :v     (end))
          (visibility :v (out))
          (face :v       (na))
          (angle :v      (na))
          (distance :v   (na))
          (location :v   (na))
          (movement :v   (na))
          (activity :v   ((is mad / stand)))
          ))


(new-instance {BIANCA}
  :name bianca102
  :slots ((parents :v     shot102)
          (frame :v     (end))
          (visibility :v (full))
          (face :v       (front))
          (angle :v      (right))
          (distance :v   (med-close))
          (location :v   ((center right)))
          (movement :v   (small))
          (activity :v   ((say is mad)))
          ))


(new-instance {CARL}
  :name carl102
  :slots ((parents :v     shot102)
          (frame :v     (end))
          (visibility :v (out))
          (face :v       (na))
          (angle :v      (na))
          (distance :v   (na))
          (location :v   (na))
          (movement :v   (na))
```

```
          (activity :v
            ((say that carl exit kitchen + is mad / stand)))
          ))


(new-instance {CARL}
  :name carl103
  :slots ((parents :v     shot103)
          (frame :v     (end))
          (visibility :v (full))
          (face :v       (left-side))
          (angle :v      (headon))
          (distance :v   (close))
          (location :v   (center))
          (movement :v   (small))
          (activity :v
            ((say is sorry about glass is broken)))
          ))


(new-instance {BIANCA}
  :name bianca103
  :slots ((parents :v     shot103)
          (frame :v     (end))
          (visibility :v (out))
          (face :v       (na))
          (angle :v      (na))
          (distance :v   (na))
          (location :v   (na))
          (movement :v   (na))
          (activity :v   ((say is (happy & agreeing))))
          ))


(new-instance {BIANCA}
  :name bianca104
  :slots ((parents :v     shot104)
          (frame :v     (end))
          (visibility :v (full))
          (face :v       (right-side))
          (angle :v      (headon))
          (distance :v   (close))
          (location :v   (center))
          (movement :v   (small))
```

```
          (activity :v
                 ((say is sorry about glass is broken)))
          ))


(new-instance {CARL}
  :name carl104
  :slots ((parents :v     shot104)
          (frame :v     (end))
          (visibility :v (out))
          (face :v       (na))
          (angle :v      (na))
          (distance :v   (na))
          (location :v   (na))
          (movement :v   (na))
          (activity :v   ((say is (happy & agreeing))))
          ))


(new-instance {BIANCA}
  :name bianca105
  :slots ((parents :v     shot105)
          (frame :v     (50526          end))
          (visibility :v (full          out))
          (face :v       (front))
          (angle :v      (right))
          (distance :v   (med-close))
          (location :v   (right))
          (movement :v   ((complex left)))
          (activity :v ((say is mad / stand / exit kitchen)
                                (is mad)))
          ))


(new-instance {CARL}
  :name carl105
  :slots ((parents :v     shot105)
          (frame :v     (50530          end))
          (visibility :v (out           (partial face)))
          (face :v       (na            front))
          (angle :v      (na            headon))
          (distance :v   (na            close))
          (location :v   (na            center))
          (movement :v   (na            small))
```

```
        (activity :v   (na
                                (say is sorry + unhappy)))
        ))


(new-instance {CARL}
  :name carl106
  :slots ((parents :v     shot106)
          (frame :v     (end))
          (visibility :v ((partial face)))
          (face :v       (front))
          (angle :v      (headon))
          (distance :v   (ex-close))
          (location :v   (center))
          (movement :v   (small))
          (activity :v   ((say is mad)))
          ))


(new-instance {BIANCA}
  :name bianca106
  :slots ((parents :v     shot106)
          (frame :v     (end))
          (visibility :v (out))
          (face :v       (na))
          (angle :v      (na))
          (distance :v   (na))
          (location :v   (na))
          (movement :v   (na))
          (activity :v   ((exit kitchen)))
          ))


(new-instance {BIANCA}
  :name bianca107
  :slots ((parents :v     shot107)
          (frame :v     (51022          end))
          (visibility :v (full          full))
          (face :v       (left-side             back))
          (angle :v      (right         headon))
          (distance :v   (med           med-close))
          (location :v   (left          center))
          (movement :v   (right         away))
          (activity :v   ((walk fast + is mad)
```

```
                                    (open door / exit door)))
          ))


(new-instance {CARL}
  :name carl108
  :slots ((parents :v     shot108)
          (frame :v     (51175          end))
          (visibility :v (full          full))
          (face :v       (left-side             back))
          (angle :v      (right         headon))
          (distance :v   (med           med-close))
          (location :v   (left          center))
          (movement :v   (right         away))
          (activity :v   ((walk fast + is mad)
                                 (open door / exit door)))
          ))
```

Appendix C

Representation Utility

```
;;;;time saving hprl constructor

(defun build ()
  (format t "~% i mindlessly format character instances")
  (format t "~% what shot number do i start at?~%")
  (setq shotnum (parse-integer (read-line)))
  (loop
      (char-format shotnum)
      (format t
    "~% done. increment shot? (default yes - quit to stop)~%")
      (setq incrmp (read-line))
      (cond  ((string= incrmp "quit") (return))
             (t
          (setq shotnum (cond ((string= incrmp "no") shotnum)
                              (t (1+ shotnum))))))))


(defun char-format (shotnum)
  (with-open-file
   (stream "true-dinners/images.l"
           :direction :output :if-exists :append)
    (format t "~% which character (1=carl 2=bianca 3=cat) ~%")
      (setq char (read-line))
      (format t "~% frames?~%")
      (setq frames (read-delimited-list #\.))
      (format t "~% visibility?~%")
      (setq visibility (read-delimited-list #\.))
      (format t "~% face?~%")
      (setq face (read-delimited-list #\.))
      (format t "~% angle?~%")
      (setq angle (read-delimited-list #\.))
      (format t "~% distance?~%")
      (setq distance (read-delimited-list #\.))
      (format t "~% location?~%")
      (setq location (read-delimited-list #\.))
      (format t "~% movement?~%")
      (setq movement (read-delimited-list #\.))
```

```
      (format t "~% activity?~%")
      (setq activity (read-delimited-list #\.))
      (setq name (cond ((string= char "1") "carl")
                       ((string= char "2") "bianca")
                       (t "cat")))
      (format stream
        "~%(new-instance {~A}~%"  (string-upcase name))
      (format stream "  :name ~A~A~%" name shotnum)
      (format stream
        "  :slots ((parents :v     {shot~A})~%" shotnum)
      (format stream
        "           (frames :vs     ~A)~%" (hprlformat frames))
      (format stream
   "          (visibility :vs ~A)~%" (hprlformat visibility))
      (format stream
     "          (face :vs       ~A)~%" (hprlformat face))
      (format stream
     "          (angle :vs      ~A)~%" (hprlformat angle))
      (format stream
     "          (distance :vs   ~A)~%" (hprlformat distance))
      (format stream
     "          (location :vs   ~A)~%" (hprlformat location))
      (format stream
     "          (movement :vs   ~A)~%" (hprlformat movement))
      (format stream
     "          (activity :vs   ~A)~%" (formatbig activity))
      (format stream "          ))~%~%")
      ))

(defun hprlformat (list)
  (cond ((> (length list) 5) (formatbig list))
        (t (formatsmall list))))

(defun formatlist (list padder)
  (do ((linelist (concatenate 'string "("
                 (string-downcase (princ-to-string (car list))))
                   (concatenate 'string linelist padder
             (string-downcase (princ-to-string (car list))))))
      (nil)
      (setq list (cdr list))
      (if (equal list ())
        (return (concatenate 'string linelist ")")))
      ))
```

```
(defun formatsmall (list)
  (setq padder '(#\Tab #\Tab))
  (formatlist list padder))

(defun formatbig (list)
  (setq padder '(#\Newline #\Tab #\Tab #\Tab #\Tab))
  (formatlist list padder))
```

Appendix D

English Representation

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;; this file contains the "meta-knowledge" of the english ;;
;; langauge that INGMAR needs to understand film.         ;;
;; the heirarchical structure of all hprl knowledge       ;;
;; begins here.                                           ;;
;; >Carl Schroeder<                                       ;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

;; default = single-valued, plural slots are multiple valued

(define-class THOUGHT ()
  :instance-slots
  ((structure)
   (synonyms
    :declare (multiple-valued))
   (antonyms
    :declare (multiple-valued))
   (contexts
    :declare (multiple-valued))))

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;sentences are homes of specific ;;
;;knowledge sequences              ;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

(define-class SENTENCE {THOUGHT}
  :instance-slots
  ((structure :v (noun
     ((verb-phrase or pred-phrase) (default pred-phrase)) ))
   (befores
    :declare (multiple-valued))
   (afters
    :declare (multiple-valued))))


(new-instance {SENTENCE}
  :name open-wine
```

```
  :slots  ((structure :v (noun open wine))
           (befores :v (noun has corkscrew))))

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

(define-class NOUN {THOUGHT}
  :instance-slots
  ((structure :v (((character or object) (default self))))))

(define-class PRED-PHRASE {THOUGHT}
  :instance-slots
  ((structure :v ( (((is or are) adjective) or
                   ( (has-the or have-the) pred-noun)) ))))

(define-class PRED-NOUN {NOUN}
  :instance-slots
  ((structure)))

(new-instance {PRED-NOUN}
  :name owner
  :slots ((structure :v ( owner character ))))

;; distinguish between who uses and who owns
(new-instance {PRED-NOUN}
  :name user
  :slots ((structure :v (user character ))))

(new-instance {PRED-NOUN}
  :name mood
  :slots ((structure :v ( mood mood-adjective))))

(new-instance {PRED-NOUN}
  :name time
  :slots ((structure :v (time number))))

(define-class STORY-NOUN {NOUN})

(define-class FILM-NOUN {NOUN})

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

(define-class VERB-PHRASE {THOUGHT}
  :instance-slots
  ((structure :v ( verb-phrase  ))
```

```
   (befores
     :declare (multiple-valued))
   (durings
     :declare (multiple-valued))
   (afters
     :declare (multiple-valued))))


(new-instance {VERB-PHRASE}
  :name eat
  :slots ((structure :v (eat (option (fast or slow or all))
              food from (plate1  (require (state (not empty)))
                                 (default (owner self)))))

          (befores :v (serve food))
          (synonyms :vs (eats eating ate))
          (afters :v (plate1 is (not full)))
          (contexts :v dinner)))

(new-instance {VERB-PHRASE}
  :name drink
  :slots ((structure :v (drink (option (fast or slow or all))
              wine from (glass1 (require (state (not empty)))
                                (default (owner self)))))
          (befores :v (pour wine))
          (afters :v ((glass1 is (not full))));;become drunk?
          (synonyms :vs (drinks drinking drank))
          (contexts :v dinner)))

(new-instance {VERB-PHRASE}
  :name serve
  :slots ((structure :v (serve (option (fast or slow or all))
                         (food (require (state (not empty))))
                              (on or onto or to)
                         (plate1 (require (state (not full)))
                                 (default (owner character1)))
                         for (character1 (default other))))
          (afters :vs ((food (not full)) (plate1 (not empty))))
          (contexts :v dinner)
          (synonyms :vs (serves serving served))
          (antonyms :v receive)))

(new-instance {VERB-PHRASE}
  :name pour
```

```
  :slots ((structure :v (pour (option (fast or slow or all))
                              (wine (require (state (not empty))))
                                  (to or into or in)
                              (glass1 (require (state (not full)))
                                      (default (owner character1)))
                              for (character1 (default other))))
          (befores :v (wine opened))
          (afters :vs ((wine (not full)) (glass1 (not empty))))
          (contexts :v dinner)
          (antonyms :v receive)
          (synonyms :vs (pours pouring poured))))

(new-instance {VERB-PHRASE}
  :name receive
  :slots ((structure :v (receive
               ((wine or food) (require (state (not empty))))
                        (in or on)
               ((glass or plate) (require (state (not full)))
                                 (default (owner character1)))
                        from (character1 (default other))))
          (durings :v (character2 (pour or serve) for subject))
          (contexts :v dinner)))

(new-instance {VERB-PHRASE}
  :name enter
  :slots ((structure :v (enter
  (left or right or door or environment (default environment))
                        (option with object)))
          (antonyms :v exit)
          (befores :v (not (present subject)))
          (afters :v (present subject))
          (contexts :v entrance)
          (synonyms :vs (enters entering entered))))

(new-instance {VERB-PHRASE}
  :name close
  :slots ((structure :v (close
                   ((door or wine) (require (state opened)))))
          (synonyms :vs (closes closing closed))
          (antonyms :v open)))

(new-instance {VERB-PHRASE}
  :name give
  :slots ((structure :v (give object1
```

```
                        (to or on or in)
                        ((object2 or environment or character)
                         (default environment))))
          (synonyms :vs (gives gave giving put puts putting))
          (antonyms :vs (take get))
          (befores :vs ( (subject has object1)
   ((object2 or environment or character) (not have) object1)))
          (afters :vs ( (subject (not have) object1)
   ((object2 or environment or character) has object1)))))

(new-instance {VERB-PHRASE}
  :name has
  :slots ((structure :v (has object))
          (synonyms :vs (have having had))
          (befores :v (subject get object))))

(new-instance {VERB-PHRASE}
  :name show
  :slots ((structure :v (show that sentence))
          (synonyms :vs (shows showing showed demonstrate
                   demonstrates demonstrating demonstrated))))

(new-instance {VERB-PHRASE}
  :name say
  :slots ((structure :v (say that sentence))
          (synonyms :vs (said says saying claim claims
                         claiming remark remarks remarking))))

(new-instance {VERB-PHRASE}
  :name ask
  :slots ((structure :v (ask that sentence))
          (synonyms :vs (asks asking asked))
          (afters :v (sentence or reply))))

(new-instance {VERB-PHRASE}
  :name reply
  :slots ((structure :v (reply that sentence))
          (synonyms :vs (replies replying replied
                           answer answers answering))
          (befores :v ask)))


(new-instance {VERB-PHRASE}
  :name hug
```

```
  :slots ((structure :v (hug (other or object)))
          (synonyms :vs (hugs hugging hugged))
          (durings :v ((other or object) hugs subject))
          (afters :vs ( (subject is happy)
                        ((other or object) is happy)) )))

(new-instance {VERB-PHRASE}
  :name use
  :slots ((structure :v (use object to verb-phrase))
          (synonyms :vs (uses using used))
          (befores :v (subject has object))))

(new-instance {VERB-PHRASE}
  :name toast
  :slots ((structure :v (toast with other))
          (synonyms :vs (toasts toasting))
          (durings :v (other toasts with subject))
          (befores :v (pour wine))
          (afters :v (drink wine))
          (contexts :v dinner)))

(new-instance {VERB-PHRASE}
  :name break
  :slots ((structure :v (break object))
          (durings :v (subject (throw or drop) object))
          (befores :vs ( (subject has object)
                         (subject is very (happy or mad))))
          (afters :vs ( (subject (not have) object)
                        (object is broken)))
          (synonyms :vs (breaks breaking broke))))

(new-instance {VERB-PHRASE}
  :name throw
  :slots ((structure :v (throw object))
          (synonyms :vs (throws threw throwing))
          (befores :vs ( (subject has object)
                         (subject is very (happy or mad))))
          (afters :vs ( (subject (not have) object)
                        (object broken)))))

(new-instance {VERB-PHRASE}
  :name drop
  :slots ((structure :v (drop object))
          (synonyms :vs (drops dropped dropping))
```

```
          (befores :v (subject has object))
          (afters :v (subject is sorry))
          (contexts :v accident)))

(new-instance {VERB-PHRASE}
  :name clear
  :slots ((structure :v (clear table))
          (synonyms :vs (clears clearing cleared))
          (durings :v (get (glass & plate & wine & food)))
          (contexts :v dinner)))

(new-instance {VERB-PHRASE}
  :name help
  :slots ((structure :v (help
                          (sentence (default other clean))))
          (synonyms :vs (helps helping helped))))

(new-instance {VERB-PHRASE}
  :name laugh
  :slots ((structure :v (laugh
                         (option at (object or character))))
          (synonyms :vs (laughs laughing laughed))
          (contexts :vs (happy  mad  drunk))))

(new-instance {VERB-PHRASE}
  :name sigh
  :slots ((structure :v sigh)
          (synonyms :vs (sighs sighed sighing))
          (contexts :vs (sad  bored))))

(new-instance {VERB-PHRASE}
  :name play
  :slots ((structure :v (play with (object or character)))
          (synonyms :vs (plays playing played))
          (contexts :vs (happy  bored))))

(new-instance {VERB-PHRASE}
  :name like
  :slots ((structure :v (like (object or character)))
          (synonyms :v (likes liking liked))
          (durings :v (subject is happy
                               with (object or character)))
          (contexts :v happy)))
```

```
(new-instance {VERB-PHRASE}
  :name agree
  :slots ((structure :v (agree with character))
          (synonyms :v (agrees agreed agreeing agreement))))

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

(define-class ADJECTIVE {THOUGHT}
  :instance-slots
  ((structure :v ( ((state-adjective or mood-adjective)
                    (default mood-adjective)) ))
   (predicates
    :declare (multiple-valued))))

(define-class STATE-ADJECTIVE {ADJECTIVE}
  :instance-slots
  ((structure :v ( (state-adjective (default present) )))))

(new-instance {STATE-ADJECTIVE}
  :name color)

(new-instance {STATE-ADJECTIVE}
  :name open
  :slots ((antonyms :v closed)
          (synonyms :v opened)))

(new-instance {STATE-ADJECTIVE}
  :name closed
  :slots ((antonyms :vs  (opened open))
          (synonyms :v shut)))

(new-instance {STATE-ADJECTIVE}
  :name present
  :slots ((synonyms :v existing)))

(new-instance {STATE-ADJECTIVE}
  :name unserved
  :slots ((antonyms :v (served touched))))

(new-instance {STATE-ADJECTIVE}
  :name served
  :slots ((synonyms :v touched)
          (antonyms :v unserved)))
```

```
(new-instance {STATE-ADJECTIVE}
  :name untouched
  :slots ((synonyms :v untouched)))

(new-instance {STATE-ADJECTIVE}
  :name whole
  :slots ((synonyms :v unbroken)
          (antonyms :v broken)))

(new-instance {STATE-ADJECTIVE}
  :name clean
  :slots ((antonyms :v dirty)))



(define-class MOOD-ADJECTIVE {ADJECTIVE}
  :instance-slots
  ((structure :v ((option adverb)
          (mood-adjective (default happy))
          (option (about or with) (noun or sentence))))
   ;; one of very (-10 to 10)
   (mood :v 0)))

(new-instance {MOOD-ADJECTIVE}
  :name happy
  :slots ((predicates :v happiness)
          (synonyms :v like)
          (antonyms :v (mad sad))
          (mood :v 7)))

(new-instance {MOOD-ADJECTIVE}
  :name mad
  :slots ((predicates :v anger)
          (synonyms :v (very impatient) )
          (antonyms :v happy)
          (mood :v -7)))

(new-instance {MOOD-ADJECTIVE}
  :name sad
  :slots ((predicates :v regret)
          (synonyms :v sorry)
          (antonyms :v happy)
          (mood :v -2)))
```

```
(new-instance {MOOD-ADJECTIVE}
  :name impatient
  :slots ((predicates :v impatience)
          (synonyms :vs (unhappy (somewhat mad)))
      ;; the antonym patient is caught in the antonym rule
          (mood :v -4)))

(new-instance {MOOD-ADJECTIVE}
  :name interested
  :slots ((predicates :v interest)
          (synonyms :vs (thoughtful patient))
          (antonyms :v bored)
          (mood :v 2)))

(new-instance {MOOD-ADJECTIVE}
  :name agreeable
  :slots ((predicates :v agreement)
          (synonyms :vs (agree agreeing))
          (mood :v 4)))

(new-instance {MOOD-ADJECTIVE}
  :name intelligent
  :slots ((predicates :v intelligence)
          (synonyms :v (very interested))
          (antonyms :vs (stupid dumb))
          (mood :v very)))

(new-instance {MOOD-ADJECTIVE}
  :name drunk
  :slots ((predicates :v drunkeness)
          (antonyms :v sober)
          (mood :v very)))

(new-instance {MOOD-ADJECTIVE}
  :name sick
  :slots ((synonyms :v ill)
          (predicates :vs (illness sickness))
          (mood :v -5)))

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

(define-class ADVERB {THOUGHT}
  :instance-slots
    ;; assume that adverb modifies some
```

```
    ;; scaleable property (like mood)
  ((structure :v (adverb))
   (multiplier :v 1)))

;;adverbs of degree

(new-instance {ADVERB}
  :name somewhat
  :slots ((multiplier :v .5)))

(new-instance {ADVERB}
  :name very
  :slots ((synonyms :v strongly)
          (multiplier :v 1.5)))

(new-instance {ADVERB}
  :name extremely
  :slots ((multiplier :v 2)))

(new-instance {ADVERB}
  :name all
  :slots ((multiplier :v 0)))

;;;adverbs of rate

(new-instance {ADVERB}
  :name slow
  :slots ((multiplier :v .5)))

(new-instance {ADVERB}
  :name fast
  :slots ((multiplier :v 1.5)
          (synonyms :v quickly)))

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

;; (define-class CONNECTOR {THOUGHT})
;; with to from about ...
;; appear in structures but don't merit definition right now
```

Appendix E

Editing Procedures

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;; this file contains Ingmar's lowest level of  ;;
;; editing knowledge, simple image decoding and ;;
;; comparison procedures.                       ;;
;; >Carl Schroeder<                             ;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

;; general note: lists are the preferred processing form but
;; the listed function is often used to protect against errors

;; makes a list if needed
(defun listed (thing)
  (cond ((listp thing) thing)
        (t (list thing))))

;**;;;;;;;;;;;;;;;;;;;
; functions on vectors
;;;;;;;;;;;;;;;;;;;;;;

;; adds two xyz lists
(defun vadd (these those)
  (mapcar #'+ these those))

(defun vsub (these those)
  (mapcar #'- these those))

;; multiplies two xyz lists
(defun vmult (these those)
  (mapcar #'* these those))

;; multiplies scalar over xyz list
(defun smult (this those)
  (list (* this (first those))
        (* this (second those))
        (* this (third those))))

;; returns a vector x fractions
```

```
;; between v1 and v2 for each elt
(defun vscale (v1 v2 x)
  (mapcar #'sscale v1 v2 '(x x x)))

(defun sscale (a b c)
  (+ a (* (- b a) c)))

;; returns an angle of change
;; between two vectors (points from 0,0,0)
;;  in : two x,y,z lists
;; out : an absolute angle between
(defun angle-between (v1 v2)
  (progn
     (setq side1 (vlength v1))
     (setq side2 (vlength v2))
     (setq side3 (vdist v1 v2))
       ;; side-side-side derivation
     (setq x (/ (* (- side2 side1)
                   (+ side2 side1)) side3))
     (setq d2 (/ (+ side3 x) 2))
     (setq d1 (/ (- side3 x) 2))
     (- 180 (+ (rad-deg (acos (/ d2 side2)))
               (rad-deg (acos (/ d1 side1))) ))))

;; converts radians to degrees
(defun rad-deg (radians)
  (* radians (/ 180 pi)))

;; returns the distance from origin of (x y z)
(defun vlength (vector)
  (vdist '(0 0 0) vector))

;; returns the distance between two vector endpoints
(defun vdist (v1 v2)
  (sqrt (+ (expt (- (first v2) (first v1)) 2)
           (expt (- (second v2) (second v1)) 2)
           (expt (- (third v2) (third v1)) 2))))

;**;;;;;;;;;;;;;;;;
; fuctions on lists
;;;;;;;;;;;;;;;;;;;

(defun inlist (that these)
  (member that these :test 'equal))
```

```
(defun comlist (these those)
  (dolist (this these)
    (if (inlist this those) (return t))))

;**;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
; the procedures that deal with char angle view
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

;; given a char's face and angles,
;; derive in coordinates the vector of view
;;  in : char-face, list of a oneof
;;     : char-angles, list of oneof(s), allowing adverbs
;; out : 3 elt list of x,y,z view for char at 0,0,0
;;       nil = na,nil
(defun view-vector (char-face char-angles)
  (do ((fpos (face-coord (first (listed char-face))))
 ;; fangle adjusts the aangle vector according to the face seen
 ;; this is needed because , for example,
 ;; a low top view is close to a high front view
       (fangle (face-angle (first (listed char-face))))
       (new-angle '(0 0 0))
       (char-angles (listed char-angles)))
      (nil)
      (if (inlist (car char-angles)
             '(little somewhat very quite extreme extremely))
            ;protect against one list too few
          (setq char-angles (list char-angles)))
      (cond ((or (equal char-angles '())
                 (equal char-angles '(na)))
             (return nil))
            (t
             (dolist (angle char-angles)
               (cond ((listp angle)
                                     ; ex. high very
                   (cond ((listp (ch-angle-case (car angle)))
                             (setq new-angle (vadd new-angle
                      (vmult fangle
                        (smult (ch-angle-case (second angle))
                    (if (equal (car angle) 'headon)
                        fpos
                        (ch-angle-case (first angle)) ))))))
                              (t
                               (setq new-angle (vadd new-angle
                                                 (vmult fangle
```

```
                (smult (ch-angle-case (first angle))
                       (if (equal (second angle) 'headon)
                       fpos
                       (ch-angle-case (second angle)) ))))))))
                     (t
                      (setq new-angle (vadd new-angle
                                            (vmult fangle
                                 (if (equal angle 'headon)
                                     fpos
                                     (ch-angle-case angle))))))))
            (return (vadd new-angle fpos))))))

(defun face-coord (char-face)
  (case char-face
          (top    '( 0  1  0))
          (bottom '( 0 -1  0))
          (front  '( 0  0  1))
          (back   '( 0  0 -1))
          ((left left-side)   '(-1  0  0))
          ((right right-side) '( 1  0  0))))

(defun face-angle (char-face)
  (case char-face
    (top    '(-1  0 -1))
    (bottom '( 1  0  1))
    (front  '( 1  1  0))
    (back   '(-1  1  0))
    (left   '( 0  1  1))
    (right  '( 0  1 -1))))

(defun ch-angle-case (char-angle)
   (case char-angle
        ((little somewhat) .5)
        ((very quite)       2)
        ((extreme extremely) 3)
        (high   '( 0  1  1))
        (low    '( 0 -1 -1))
        (left   '(-1  0 -1))
        (right  '( 1  0  1))))


(defun d-view-angle (v1 v2)
  (angle-between v1 v2))
```

```
;**;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
; the procedures that deal with char location view
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

;; given a char's distance and location,
;; derive in coordinates the vector of location (3 dimensional)
;;  in : char-distance, list of a oneof
;;     : char-location, list of oneof(s)
;; out : 3 elt x,y,z list
;;     : nil = na or nil
(defun location-vector (char-distance char-location)
  (progn
    (setq depth (depth-vector (listed char-distance)))
    (when (null depth) nil)
    (setq pos (position-vector char-location))
    (when (null pos) nil)
    (vadd depth (position-vector char-location))))

;; given a char's distance,
;; derive in coordinates the vector of depth
;;  in : char-distance, list of a oneof
;; out : 3 elt x,y,z list
;;     : nil = na or nil
(defun depth-vector (char-distance)
  (case (car char-distance)
    ((na nil) nil)
    ((extreme-close ex-close) '(0 0 -1))
    (close                    '(0 0 -2))
    ((medium-close med-close) '(0 0 -3))
    ((medium med)             '(0 0 -4))
    ((medium-long med-long)   '(0 0 -5))
    (long                     '(0 0 -6))))

;; given a char's position, derive in coordinates
;; the vector of position (2 dimensional)
;;  in : char-positions, list of oneof(s) allowing adverbs
;; out : 3 elt x,y,z list
;;     : nil = na or nil
(defun position-vector (char-positions)
  (do ((char-positions (listed char-positions))
       (new-pos '(0 0 0))
       (div 1))
      (nil)
     (when (inlist (car char-positions)
```

```
                '(nil na)) (return nil))
    (if (in-list (car char-positions)
                 '(somewhat little very far))
      ;protect against one list too few
        (setq char-positions (list char-positions)))
    (cond ((or (equal char-positions '())
               (equal char-positions '(na)))
           (return '()))
          (t
           (dolist (pos char-positions)
             (cond ((equal pos 'center)
                    (setq div (1+ div)))
                   ((listp pos)
                    (cond ((listp (ch-pos-case (car pos)))
                           (setq new-pos (vadd new-pos
                       (smult (ch-pos-case (second pos))
                              (ch-pos-case (first pos))))))
                          (t
                           (setq new-pos (vadd new-pos
                       (smult (ch-pos-case (first pos))
                              (ch-pos-case (second pos))))))))
                   (t
                    (steq new-pos (vadd new-pos
                                        (ch-pos-case pos))))))
           (return (smult (/ 1 div) new-pos))))))

(defun ch-pos-case (pos)
  (case pos
    ((somewhat little) .5)
    ((very far) 2)
    ((upper up top)        '( 0  5  0))
    ((lower low bottom)    '( 0 -5  0))
    (left                  '(-5  0  0))
    (right                 '( 5  0  0))))


;; returns a measure of centrality of the location
;;  in : location vector
;; out : number
(defun centrality (loc)
  (vlength (vmult loc '(1 1 0))))

;; returns the difference in two locations
;; (apparent motion vector)
```

```
(defun d-loc-v (loc1 loc2)
  (vsub loc2 loc1))

;; returns the difference in depth
(defun d-depth-dist (loc1 loc2)
  (- (third loc2) (third loc1)))

;; returns the differnce in position
(defun d-pos-v (loc1 loc2)
  (vsub (vmult loc2 '(1 1 0)) (vmult loc1 '(1 1 0))))

;; returns the distance between positions
(defun d-pos-dist (loc1 loc2)
  (vdist (vmult loc1 '(1 1 0)) (vmult loc2 '(1 1 0))))

;**;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
; procedures that deal with char movement
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

;; given a char's movement, derive in
;; coordinates the vector of movement or modifiers
;;  in : char-move, list of oneof(s)
;; out : 3 elt list of x,y,z movement for char at 0,0,0
;;        (speed proportional to length, >0 , 1 is "average")
;;         (0,0,0)= none or small
;;          nil   = na, nil or complex
(defun char-movement (char-moves)
  (do ((char-moves (listed char-moves))
       (new-move '(0 0 0)))
      (nil)
    (if (comlist char-moves '(fast slow swish))
      ;protect against one list too few
        (setq char-moves (list char-moves)))
    (cond ((or (equal char-moves '())
               (equal char-moves '(na))
               (inlist 'complex char-moves))
           (return nil))
          (t
           (dolist (move char-moves)
             (cond ((listp move)
                    (cond ((listp (ch-move-case (car move)))
                           (setq new-move (vadd new-move
                                 (smult (ch-move-case (cadr move))
                                        (ch-move-case (car move))))))))
```

```
                        (t
                         (setq new-move (vadd new-move
                      (smult (ch-move-case (car move))
                             (ch-move-case (cadr move))))))))
                    (t
                     (setq new-move (vadd new-move
                                          (ch-move-case move))))))
            (return new-move)))))

(defun ch-move-case (move)
  (case move
    (slow .5)
    (fast  2)
    (swish 3)
    (up     '( 0  1  0))
    (down   '( 0 -1  0))
    (left   '(-1  0  0))
    (right  '( 1  0  0))
    (toward '( 0  0  1))
    (away   '( 0  0 -1))
    ((none small) '( 0 0 0))))

(defun d-char-move-angle (m1 m2)
  (angle-between m1 m2))

;; returns a measure of the apparent
;; acceleration between two move vectors
(defun d-char-move-accel (m1 m2)
  (- (vlength m1) (vlength m2)))

;**;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
; procedures that deal with camera movements
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

;; returns two vectors for a camera's movement
;;  in : camera-moves
;; out : 2 3 elt xyz lists, move-vector and angle-vector
;;       ((000)(000)) = none or small
;;               nil  = na, nil or complex
(defun camera-move (cam-moves)
  (do ((cam-moves (listed cam-moves))
       (new-move '((0 0 0) (0 0 0))))
      (nil)
    (if (comlist (list (car cam-moves)
```

```
                    (cadr cam-moves)) '(fast slow swish))
        ;protect against one list too few
          (setq cam-moves (list cam-moves)))
    (cond ((or (equal cam-moves '())
               (equal cam-moves '(na))
               (inlist 'complex cam-moves))
           (return nil))
          (t
           (dolist (move cam-moves)
             (cond ((listp move)
                    (cond ((listp (cm-move-case (car move)))
                           (setq new-move (vadd new-move
                       (smult (cm-move-case (second move))
                              (cm-move-case (first move))))))
                          (t
                           (setq new-move (vadd new-move
                       (smult (cm-move-case (first move))
                           (cm-move-case (second move))))))))
                   (t
                    (setq new-move (vadd new-move
                                    (cm-move-case move))))))
           (return (list (butlast new-move 3)
                         (cdddr new-move))))))

(defun cm-move-case (move)
  (case move
    (slow .5)
    (fast  2)
    (swish 3)
    (zoom-in      '( 0  0  0  0  0 -1) )
    (zoom-out     '( 0  0  0  0  0  1) )
    (pan-left     '( 0  0  0 -1  0  0) )
    (pan-right    '( 0  0  0  1  0  0) )
    (tilt-up      '( 0  0  0  0  1  0) )
    (tilt-down    '( 0  0  0  0 -1  0) )
    (dolly-in     '( 0  0 -1  0  0  0) )
    (dolly-out    '( 0  0  1  0  0  0) )
    (dolly-left   '(-1  0  0  0  0  0) )
    (dolly-right  '( 1  0  0  0  0  0) )
    (rise         '( 0  1  0  0  0  0) )
    (drop         '( 0 -1  0  0  0  0) )
    ((none small) '( 0  0  0  0  0  0) )))

;**;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
```

```
; procedures that deal with analyzing a single shot
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;


;; returns the sublist of the film-object's slot
;; that corresponds to the frame time
;; does not make judgements for applicability in
;; case much time has elapsed from sublist to frame time!!
;;  in : film-object, slot, frame-time (could be 'end, 'start)
;; out : sublist (could be nil)
;; req : film-obj has slots slot and 'frame
(defun slot-at-time (film-object slot frame-time)
  (do ((values (listed (ask (frame film-object) slot)))
       (times  (listed (ask (frame film-object) 'frame)))
       (index 0 (1+ index)))
      (nil)
      (cond ((equal frame-time 'start)
             (cond ((equal (first values) 'na)
                    (return nil))
                   (t (return (listed (first values))))))
            ((or (equal frame-time 'end)
                 (>= index (length times)))
             (cond ((or (> (length times) (length values))
                        (equal (last values) 'na))
                    (return nil))
                   (t (listed (last values)))))
            ((<= frame-time (nth index times))
             (cond ((or (> index (length values))
                        (equal (nth index values) 'na))
                    (return nil))
                   (t (listed (nth index values))))))))


;; returns the frame numbers of desc's on the
;; two sides of a frame number
;; ex. 0 1 2 3 4  frames in shot
;;         2 end  frames in obj
;;     0 or 1 => (0 2)
;;     2,3 or 4 => (2 nil)
;;   (used to interpolate a location between desc borders)
;;  in : frame-time (number, start or end)
;; out : (before-time after-time), (numbers or nil)
;; req : frame-time is in range of obj's slot frame-times
```

```
(defun time-borders (shot obj time)
  (do* ((shot (frame shot))
        (times (ask (frame obj) 'frame))
        (in  (ask  shot 'in-point))
        (out (ask  shot 'out-point))
        (index 0 (1+ index))
        (tme (nth index times) (nth index times)))
       (nil)
       (when (equal time 'start) (setq time in))
       (when (equal time 'end )  (setq time out))
       (when (equal tme 'end) (setq tme out))
       (when (<= time tme)
             (cond ((= index 1)
                    (return (list in
                                  (cond ((= (length times) 1)
                                         nil)
                                        (t
                                         (1+ tme))))))
                   (t
                    (return (list (1+ (nth (1- index) times))
                                  (cond ((= index (length times))
                                         nil)
                                        (t
                                         (1+ tme)))))) ))))


;; returns a location vector averaged between frame-time
;; positions to approximate the character's movement
;;  in : shot,char,time
;; out : (x y z) location
;;     :  nil = na, or was last descriptor time
;;              (inability to project location for last time
;;             should not be a problem because assume that
;;available shots were ended on small movement or char was out)
(defun inter-location (shot char time)
  (progn
     (setq borders (time-borders shot char time))
     (setq time1 (first borders))
     (setq time2 (second borders))
     (when (null time2) nil) ;last desc
     (setq loc1 (location-vector
                 (slot-at-time char 'distance time1)
                 (slot-at-time char 'location time1)))
     (setq loc2 (location-vector
```

```
                (slot-at-time char 'distance time2)
                (slot-at-time char 'location time2)))
    (when (or (null loc1) (null loc2)) null) ;can't interp
    (vscale loc1 loc2 (/ (- time time1) (- time2 time1)))))

;; returns a cut point adjusted to the nearest camera pause
;; (limited to start and end of course)
;;  adjusted backward for 'in and foward for 'out
;;  in : in/out, shot, time (number, 'start or 'end)
;; out : time or nil (if next cam move desc was na)
(defun adjust-cut (type shot time)
  (do* ((shot (frame shot))
        (start (ask shot 'in-point))
        (end  (ask shot 'out-point))
        (camera (frame (ask shot 'camera)))
        (next-time (cdr (borders  shot camera time)))
        (times (ask camera 'frame)))
       (nil)
         ;; replace mnemonics
       (when (equal time 'start)
             (setq time start))
       (when (equal time 'end)
             (setq time end))
       (when (equal (car (last times)) 'end)
             (setq times (append (butlast times) end)))
         ;; get actual time in frame list
       (setq time (if (null next-time)
                      end
                      (1- next-time)))
         ;; find next time appropriate in correct direction
       (when (equal type 'in)
             (setq times (reverse times)))
       (setq times (nthcdr (position time times) times))
       (do ((index 0 (1+ index))
            (tme (nth index times)))
           ((= index (length times)) ;as far as it goes
            (if (equal type 'in)
                start
                end))
         (setq cammove (camera-move
                        (slot-at-time camera 'movement tme)))
         (cond ((null cammove)
                (return nil))
               ((equal cammove '((0 0 0) (0 0 0)))
```

```
                (return (if (equal type 'in)
                  ;going backward need beginning of desc time
                           (1+ (nth (1+ index) times))
                           tme)))))))
;; -> incomplete code - do not evaluate
;; returns the character most visible
;; and thus to be empathized with in a shot
;;  in : shot, start, end (could be 'start and 'end)
;; out : char or nil (if no characters)
(defun main-char (shot start end)
  (do ((

;**;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
; procedures that deal with comparing shots
; for cutting potential
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

;; -> incomplete code - do not evaluate
;; determines a degree of jumpiness for the editing of two shots
;; either shot can be a (shot frame-number) pair
(defun jump-potential (shot1 shot2 quiet-cut-p)
  (do ((end) (start) (out-chars '()) (in-chars '()))
      (nil)
        ;; extract the parameters
      (if (listp shot1)
          (progn
             (setq shot1 (frame (first shot1)))
             (setq end (second shot1)))
          (progn
             (setq shot1 (frame shot1))
             (setq end 'end)))
      (if (listp shot2)
          (progn
             (setq shot2 (frame (first shot2)))
             (setq start (second shot2)))
          (progn
             (setq shot2 (frame shot2))
             (setq start 'start)))
      (if (equal end 'end)
          (setq end (ask shot1 'out-point)))
      (if (equal start 'start)
          (setq start (ask shot2 'in-point)))
        ;; adjust in out points if camera
        ;; movements yield a bad cut point
```

```
(when (and quiet-cut-p
           (camera-move-vector
            (slot-at-time ((frame (ask shot1 'camera))
                                                 'movement
                                                 end))))
      (setq end (adjust-cut 'out shot1 end)))
(when (and quiet-cut-p
           (camera-move-vector
            (slot-at-time ((frame (ask shot2 'camera))
                                                 'movement
                                                 start))))
      (setq start (adjust-cut 'in shot2 start)))
  ;; find characters of each shot
(setq out-chars (ask shot1 'characters))
(setq in-chars  (ask shot2 'characters))
(setq out-main (main-char shot1 'out end))
(setq in-main (main-char shot2 'in start))))
  ;; evaluate image-match for each char
;; to char pair (with particular emphasis on main to main)
```

Appendix F

Story Procedures

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;; the procedures in this file are used in finding  ;;
;; a shot by indexing for story elements.            ;;
;; implementation proceeded to the ability for       ;;
;; generating a standardized syntax from a general  ;;
;; english language descriptor.                      ;;
;; >Carl Schroeder<                                  ;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;


;; this procedure takes an incomplete descriptor which might
;; have compound parts, denoted by & and or, and returns
;; a list of phrases
(defun expand-phrase (phrase)
  (do* ((new-phrases '(())))
       (nil)
      (dolist (thing phrase)
        (setq new-phrases
          (cond ((symbolp thing)
                 (appendall new-phrases thing))
                (t
                 (setq new-new '())
                 (dolist (part thing)
                   (cond ((or (equal part '&)
                              (equal part 'and)))
                         (t
                          (setq new-new (append new-new
                            (appendall new-phrases part))))))
                 new-new))))
      (return new-phrases)))

;; those is ( (--) ..) and that is symbol,
;; gives ( (-- that) ..)
(defun appendall (those that)
  (cond ((null those) nil)
```

```
        (t
         (append (list (append (car those) (list that)))
                 (appendall (cdr those) that)))))

;; this frighteningly large procedure does all the work
;; to match a single incomplete sentence into its grammar
;; as defined in the english file
;;  in : noun, phrase, structure (highest is a sentence) ,
;;       options (t/nil, norm t), expandeds (list, init. nil)
;;       noun is the char of this desc, thus for filling
;;           in defaults like a missing subject
;;       phrase is matched to structure,
;;           which comes from a grammar definition
;;       by making options nil, all optional
;;           parts of the phrase will be dropped
;;          (helps to generalize in phrase to phrase matching)
;;       expandeds is used in recursion, should be nil
;;          at highest entry into procedure
;; out : ( (new-phrase) (phrase leftover)
;;         (require states) (default states) )
;;       require and default states must be returned for
;;         later analysis of implied char or object states
(defun fit-to-structure
     (noun phrase structure options expandeds)
  (do  ((new-phrase '())
        (requires '())
        (defaults '())
        (obj-guide 0)
        (word) (form) (def1) (req1)
        (flasp) (cunk) (jilk) (shorp))
       ((null structure) (list new-phrase phrase
                               requires defaults))

       (setq word (car phrase))
          ;; for now, just get rid of a suffix
       (setq form (listed (car (desuffix structure))))
                 ;; it's an option statement
       (cond ((equal (car form) 'option)
              (setq flasp (fit-to-structure noun phrase
                                (cdr form) options expandeds))
              ;; something worked for that word
              (cond ((not (equal phrase (second flasp)))
                     (if options          ;; make it official
                        (setq structure (append (cdr form)
```

```
                                    (cdr structure)))
            ;; want to trash it and whatever matched
               (progn
                (setq structure (cdr structure))
                (setq phrase (second flasp)))))
           (t        ;; forget it
            (setq structure (cdr structure)))))

         ;; get the requires and defaults
    (t
     (setq flasp (sep-things form 'default))
     (setq def1 (first flasp))
     (setq form (second flasp))
     (setq flasp (sep-things form 'require))
     (setq req1 (first flasp))
     (setq form (second flasp))
     (if (and (listp (car form))
              (equal (list (car form)) form))
       ;; just checking it's not ((thing))
       (setq form (car form)))

     (setq cunk 'lost)    ;; is it (word) ?
     (cond ((and (= (length form) 1)
                 (symbolp (car form)))
            (setq form (car form))
            (cond ((is-a word form)  ;; this is it!
                   (setq cunk 'found)
                        ;; but expandable
                   (cond ((and (expandable word)
                               (not
                                (or (equal word form)
         ;; hasn't been expanded since the last match
         ;; vital for avoiding loops!
         (inlist (expandable word) expandeds))))
                         (setq expandeds (append
                           expandeds (list
                                (expandable word))))
                         (setq structure (append
                          (get-a-structure  word)
                          (cdr structure))))
                        (t   ;; go for it
        ;; a definite match, can expand anything again
                         (setq expandeds nil)
       (setq new-phrase (append new-phrase
```

```
                                (list word)))
            (setq phrase (cdr phrase))
            (setq structure (cdr structure)))))
                  ;; expandable and nonrepeating
                      ((and (expandable form)
                            (not (inlist (expandable form)
                                         expandeds)))
                       (setq cunk 'found)
                       (setq expandeds (append expandeds
                              (list (expandable form))))
                       (setq structure (append
                          (get-a-structure form)
                          (cdr structure))))
                      (t  nil))) ;; blah

                (t   ;; a list
                 (cond ((inlist 'or form)  ;; ors
                        (do* ((index 0 (1+ index))
                              (flasp (nth index form)
                                     (nth index form)))
                             ((or (= index (length form))
                                  (equal cunk 'found))
                              nil)
                             ;; ignore those ors
                           (cond ((equal flasp 'or) nil)
                                 (t    ;; try each thing
                                  (setq jilk
(fit-to-structure noun phrase (listed flasp)
                  options expandeds))
                                  (when (not (equal
        (second jilk) phrase)) ;; something matched
                                   (setq cunk 'found)
                                   (when
       (is-a flasp 'object) (setq obj-guide index))
                                   (setq structure
      (append (listed flasp) (cdr structure))))))))
                       (t   ;; just some nested form
                        (setq jilk (fit-to-structure
     noun phrase (listed flasp) options expandeds))
                        (when (not
(equal (second jilk) phrase))  ;; and something works
                              (setq cunk 'found)
                              (setq structure
         (append (listed flasp) (cdr structure)))))))))
```

```
                ;; nothing worked, consider possibilities
          (when (equal cunk 'lost)
                          ;; case 1) an orlist of objects
;; whose choice comes from analogy to an earlier choice
                (cond ((and (listp form)
                            (inlist 'or form)
                            (symbolp (car form))
                            (is-a (car form) 'object)
                            (<= obj-guide (length form)))
                       (setq cunk 'found)
                       (setq word (nth obj-guide form))
                       (setq new-phrase (append new-phrase
                                                (list word)))
                       (setq structure (cdr structure)))
                      (t
                       ;; can we find a good default?
                       (setq shorp
                         (second (sep-symbol def1)))
                       (if (= (length shorp) 1)
                         ;; error prevention
                         (setq shorp (car shorp)))
                       (when shorp
                             (setq cunk 'found)
                             (setq structure (append
                      (listed shorp) (cdr structure)))))))
                ;; still nothing worked,
                ;; now just pick if we can
          (when (equal cunk 'lost)
            ;; from a list, put on the structure
                   (cond ((listp form)
                          (setq structure (append
                            (listed (car (listed form)))
                            (cdr structure))))
                         (t
                        ;; only choice, put on the output
                          (setq new-phrase (append
                                 new-phrase (list form)))
                          (setq structure
                            (cdr structure)))))

    ;; get rid of non-attributes in defaults or requires
         (if (and def1 (caar (sep-symbol def1)))
             (setq defaults (append defaults
              (list (list word
```

```
                                (caar (sep-symbol def1)))))))
                (if (and req1 (caar (sep-symbol req1)))
                    (setq requires (append requires
                     (list (list word
                                 (caar
                                  (sep-symbol req1))))))))))

;; numerical suffixes are allowed in a grammar
;; definition to distinguish instances
;; this returns (thing suffix)
(defun sep-suffix (thing)
    (list (intern (string-right-trim "1234567890"
                                     (string thing)))
          (intern (string-left-trim
                   "ABCDEFGHIJKLMNOPQRSTUVWXYZ"
                   (string thing)))))

;; get rid of all suffixes in any hairy list of stuff
;; with exquisite recursion
(defun desuffix (thing)
  (append (cond ((listp (car thing))
                 (list (desuffix (car thing))))
                (t
                 (list (car (sep-suffix (car thing))))))
          (cond ((cdr thing)
                 (desuffix (cdr thing)))
                (t
                 nil))))

;; what is expandable and what is not
;; for now, includes synonyms, but would include antonyms
(defun expandable (thing)
  (or (direct-expandable thing)
      (direct-expandable (name (synonym thing)))))

;; what is explicitly available
(defun direct-expandable (thing)
  (cond ((or (verb-p thing)
             (pred-noun-p thing))
         ;; isn't that stupid? it wouldn't return a value
         (car (list thing)))
        ((inlist thing '(sentence noun pred-phrase
                         verb-phrase adjective mood-adjective))
         (car (list (list 'a thing))))
```

```
        (t
         nil)))

;; what has a synonym (return the synonym frame)
(defun synonym (thing)
  (solve (list '?x 'synonyms thing)
         :returns '?x))

;; this gets the relevant structure,
;; for now includes synonyms but not antonyms
(defun get-a-structure (thing)
  (ask (frameable (expandable thing)) 'structure))


;; takes a list of lists and symbols
;; and returns ((lists) (symbols))
(defun sep-symbol (list)
  (do* ((lists '())
        (symbols '())
        (index 0 (1+ index)))
       ((= index (length list)) (list lists symbols))
       (setq this (nth index list))
       (if (listp this)
           (setq lists (append lists (list this)))
           (setq symbols (append symbols (list this))))))


;; this takes a list of form (... (key ...) ... )
;; and returns ( (...) (...))
(defun sep-things (form key)
  (do* ((thing form (cdr thing))
        (this (car thing) (car thing))
        (new-form '())
        (key-form '()))
       ((null this) (list key-form new-form))
       (cond ((symbolp this)
              (setq new-form (append new-form (list this))))
             ((equal (car this) key)
              (setq key-form (append key-form (cdr this))))
             (t
              (setq new-form (append new-form
                                     (list this)))))))
```

```
;; returns nil or index of this in the list those
(defun is-a-in (this those)
  (do* ((index 0 (1+ index))
        (that (nth index those) (nth index those)))
       ((= index (length those)) nil)
       (if (is-a this that) (return index))))

(defun is-a (this that)
  (or (is-a-dir this that)
      (is-a-dir (name (synonym this)) that)))

(defun is-a-dir (this that)
  (or (equal this that)
      ;; needed to avoid an error on '(one carl)
      (and (frameable this)
           (frameable that)
           (class-frame-p (frameable that))
                 ;; has = verb-phrase
           (or (instance-p (frameable this) (frameable that))
               (and (class-frame-p (frameable this))
                    (inlist (frameable that) (superclasses
                                    (frameable this))))))))
                                     ;; carl is-a character

;; like frame, but safer, and more general than frame-p
(defun frameable (thing)
  (and (or (symbolp thing)
           (= (length thing) 1)
           (equal (car thing) 'a))
       (frame thing)))

(defun pred-noun-p (thing)
  (and (frameable thing)
       (not (equal thing 'pred-noun))
       (is-a thing 'pred-noun)))

(defun verb-p (thing)
  (and (frameable thing)
       (not (equal thing 'verb-phrase))
       (is-a thing 'verb-phrase)))

(defun mood-p (thing)
  (and (frameable thing)
       (not (equal thing 'mood-adjective))
```

```
       (is-a thing 'mood-adjective)))

(defun char-p (thing)
  (and (frameable thing)
       (not (equal thing 'character))
       (is-a thing 'character)))

(defun object-p (thing)
  (and (frameable thing)
       (not (equal thing 'object))
       (is-a thing 'object)))

;; note that this will not work because hprl
;; syntax is not integrated into lisp
(defun names (frames)
  (mapcar #'name (listed frames)))
```

References

[Bloch 87] Bloch, G. R.
From Concepts to Film Sequences.
Research Report, Yale University, New Haven, CT, 1987.

[Casty 73] Casty, A.
Development of the Film.
Harcourt Brace Jovanovich, New York, NY, 1973.

[Davenport 87] Davenport, G., Levitt, D.
Generating Seamless Movies.
NSF Grant Proposal, M.I.T., Cambridge, MA, 1987.

[Fell 79] Fell, J. L.
A History of Films.
Holt Rinehart and Winston, New York, NY, 1979.

[HewlettPackard 86]
Hewlett Packard.
HP-RL Reference Manual.
Technical Report, , Palo Alto, CA, 1986.

[Lehnart 84] Lehnart, W. G., Cook, M. E., McDonald, D. D.
Conveying Implicit Content in Narrative Summaries.
Research Report, University of Massachusettes, Amherst, MA, 1984.

[Reisz 68] Reisz, K., Millar, G.
The Technique of Film Editing.
Hastings House, New York, NY, 1968.

[Scott 75] Scott, J. F.
Film the Medium and the Maker.
Holt Rinehart and Winston, New York, NY, 1975.

[Vogel 74] Vogel, A.
Film as a Subversive Art.
Random House, New York, NY, 1974.

Table of Contents