PUC

ISSN 0103-9741

Monografias em Ciência da Computação
n° 08/12

Early Cases of Bertillon, the Logic Programming Sleuth

Simone D. J. Barbosa
Fabio A. Guilherme da Silva
Antonio L. Furtado

Departamento de Informática

PONTIFÍCIA UNIVERSIDADE CATÓLICA DO RIO DE JANEIRO
RUA MARQUÊS DE SÃO VICENTE, 225 - CEP 22453-900
RIO DE JANEIRO - BRASIL

Monografias em Ciência da Computação, No. 08/12 ISSN 0103-9741
Editor: Prof. Carlos José Pereira de Lucena June, 2012

Early Cases of Bertillon, the Logic Programming Sleuth

Simone D. J. Barbosa
Fabio A. Guilherme da Silva
Antonio L. Furtado

fabio.guilherme@gmail.com, furtado@inf.puc-rio.br

Abstract: The present text has to do with communicative events, whose presence in reasonably realistic plots is no less vital than that of action events. A compact general-purpose package is introduced as part of the **Logtell** plot-composition project. To test the usability of the package, a first effort to specify an application domain belonging to the genre of detective stories was undertaken. Seven criminal cases are reported as examples, with a fairly successful outcome thanks to the participation of M. Maurice Bertillon, an imaginative (and imaginary) French detective and an adept of logic programming, who kindly agreed to use our SWI-Prolog plan-based implementation. In particular, the text indicates how our interactive **PlotBoard** tool allows to combine, in alternative stepwise fashion, plot sequences produced by plan-generation with choices, extensions and adaptations resulting from the user's intervention.

Keywords: Plot Composition, Communicative Speech Acts, Detective Stories, Logic Programming, Plan Generation, Plan Recognition.

Resumo: O presente texto tem a ver com eventos comunicativos, cuja presença em enredos razoavelmente realistas é não menos vital que a dos eventos de ação. Um pacote compacto de propósito geral é apresentado como parte do projeto **Logtell** de composição de enredos. Para testar a usabilidade do pacote, um primeiro esforço foi empreendido no sentido de especificar um domínio de aplicação pertencente ao gênero de estórias de detetive. Sete casos criminais são relatados como exemplo, com desfecho bastante bem sucedido graças à participação de Monsieur Maurice Bertillon, um imaginativo (e imaginário) detetive francês adepto da programação em lógica, que gentilmente se dispôs a usar nossa implementação em SWI-Prolog. Em particular, o texto indica como nossa ferramenta interativa **PlotBoard** permite combinar, em fases alternativas, trechos de enredo produzidos por geração de planos com escolhas, extensões e adaptação resultantes da intervenção do usuário.

Palavras-chave: Composição de Enredos, Atos da Fala Comunicativos, Estórias de Detetive, Programação em Lógica, Geração de Planos, Reconhecimento de Planos.

In charge of publications

Rosane Teles Lins Castilho
Assessoria de Biblioteca, Documentação e Informação
PUC-Rio Departamento de Informática
Rua Marquês de São Vicente, 225 - Gávea
22451-900 Rio de Janeiro RJ Brasil
Tel. +55 21 3527-1516 Fax: +55 21 3527-1530
E-mail: bib-di@inf.puc-rio.br

Dr. Mortimer: Recognizing, as I do, that you are the second highest expert in Europe ———"
Holmes: "Indeed, sir! May I inquire who has the honour to be the first?"
Dr. Mortimer: "To the man of precisely scientific mind the work of Monsieur Bertillon must always appeal strongly."

Sir Arthur Conan Doyle, *The Hound of the Baskervilles*

1. Introduction

When we decided to compile the memoirs of young Maurice Bertillon's incipient detective career, we had two objectives in mind:

- To start developing a formal specification of the detective stories genre
- To use this genre as benchmark to validate our package of communicative events

It has been convincingly argued [Todorov] that detective stories actually contain two stories: the story of the crime and the story of the investigation. The first story, that of the crime, would, according to him, end before the second begins; the characters of the second story, the story of the investigation, do not act, they learn ─ which is well within the scope of the package to be discussed here. In fact in many detective stories the crime and the investigation are not neatly separated in time, the two series of events being interspersed. For instance, in *Curtain* [Christie-1], Poirot was, from the beginning, aware of the identity of the culprit, who perpetrated a couple of additional crimes at the very place where the little Belgian detective was lodged.

Our attention was first called to the name Bertillon when we read the passage in epigraph, which so much irritated the great Sherlock Holmes [Doyle]. We first thought that Holmes's rival was equally fictitious – but soon a Wikipedia entry was located introducing him as real[1]:

> Alphonse Bertillon (April 24, 1853 – February 13, 1914) was a French police officer and biometrics researcher who created anthropometry, an identification system based on physical measurements. Anthropometry was the first scientific system used by police to identify criminals. Before that time, criminals could only be identified based on unreliable eyewitness accounts. The method was eventually supplanted by fingerprinting but his other contributions like the mug shot and the systematisation of crime-scene photography remain in place to this day.

In contrast, the Bertillon whose memoirs we now start to record is, as the old saying goes, "a figment of the authors' imagination". Like his possible ancestor, he should please those who claim to possess a "precisely scientific mind". His expertise, however, is related to computer science, rather than to biometrics. Specifically, he endeavours to apply the methods of logic programming. Learning about his talents, we approached him with an invitation to run experiments with the support of our prototype tools.

It so happened that Swords-and-Dragons was the first story genre treated by us in our **Logtell** interactive plot-composition project [Ciarlini-1]. Plan-generation was employed to produce the plots, which were then displayed via computer-graphics animation techniques. As expected, the plots were full of physical act scenes, including the abduction of a

[1] http://en.wikipedia.org/wiki/Alphonse_Bertillon

princess amid violent combats between the villainous dragon and the brave knights who came to rescue her. Some work [Silva] has been done after that, and will be further elaborated here, to provide a repertoire of communicative acts suitable to a broader variety of genres.

The present text is organized as follows. Section 2 briefly describes our conceptual modelling approach with an emphasis on the recently-developed communicative operations. Section 3 deals with specific features of the detective stories domain. Bertillon's early feats are reported in section 4, which shows, in addition, how our **PlotBoard** tool is used to compose the stories by stages, allowing at each stage various types of user's intervention. Section 5 presents some related work. Section 6 contains concluding remarks.

In the interest of readability a number of technical points have been omitted (for more details, cf. the SWI Prolog program reproduced in the Appendix).

2. A logic programming method

We begin by briefly reviewing our three-schema conceptual specification approach, used along the **Logtell** project [Ciarlini-1]. The next sub-sections introduce three classes of *communicative events* which are needed, besides the usual action events, to narrate reasonably realistic plots where the characters' mutual reliance for information or for action, and how they form their beliefs, are aspects to be made explicit: information-gathering events, directive and commissive events, and library-consulting events.

The communicative events were designed so as to constitute a general-purpose domain-independent package, even though in the present work we have chosen a specific genre, dealing with detective stories, to test its adequacy.

2.1. Conceptual specification

To model a chosen story genre, to which the plots to be composed should belong, we must specify at least:

a. what can exist at some state of the underlying mini-world,
b. how states can be changed, and
c. the factors driving the characters to act.

Accordingly, our conceptual design method involves three schemas – static, dynamic and behavioural

The *static schema* specifies, in terms of the *Entity-Relationship* (*ER*) model [Batini], the entity and relationship classes involved, together with the list of their attributes. A state of the world consists of all facts about the existing entity instances and their properties holding at some instant. To these basic concepts borrowed from the ER model, we add the notion of *belief*. Informally speaking, beliefs correspond to the partial view, not necessarily correct, that a character currently forms about the factual context (for a formal characterization, of the BDI model, cf. [Rao]).

The *dynamic schema* defines a fixed repertoire of operations for consistently performing state changes. Indeed, in our model, we equate the notion of event with the state change resulting

from the execution of such predefined operations. The *STRIPS* [Fikes] model is used. Each operation is defined in terms of pre-conditions, which consist of conjunctions of positive and/or negative literals, and any number of post-conditions, consisting of facts to be asserted or retracted as the effect of executing the operation.

The *behavioural schema* concerns the goals of the characters (agents) participating in the story, which motivate them to perform some operation or some sequence of operations to move from the current state to a state where their goals are achieved. Initially our behavioural schema was limited to situation-objective (a.k.a. goal-inference) rules [Ciarlini-1]. Several extensions have since then been considered, taking into consideration personality aspects such as drives, attitudes and emotions [Barbosa]. Another extension is provided by the *conditioner* clauses to be introduced in the next sections.

Being defined in terms of pre-conditions and post-conditions, the operations denoting the events to occur in a plot can be readily chained together in view of the characters' goals by a *plan-generating algorithm* [Ciarlini-1]. As a consequence, it becomes natural to equate plots (sequences of events) with plans (sequences of operations able to bring about the events).

Moreover the availability of the plan-generator in a logic-programming environment makes each three-schema conceptual specification executable. Once some compatible initial state has been set, introducing the characters and objects that will figure in a usage session, our tools can be run to produce one or more plots, possibly adapted as the result of the user's interventions but always kept consistent with the specified genre.

The **Logtell** strategy of plot composition by plan generation – combined with multistage user interaction – has proven, in the course of our experiments until now [Ciarlini-3], a suitable way for treating literary genres encompassing narratives with a high degree of regularity, such as fairy tales, and application domains of business information systems, such as banking, which are obviously constrained by providing a basically inflexible set of operations and, generally, by following strict and explicitly formulated rules.

2.2. Information-gathering events

The purpose of the *information-gathering* events [Silva] is to enable the various characters to mentally apprehend the state of the world. Without such events, one would have to assume that the characters are omniscient, being aware of all facts that currently hold and of how they change as a consequence of the action events.

Here we shall recognize a sharp distinction between the facts themselves and the sets of *beliefs* of each character about the facts that hold at the current state of the world, which constitute, so to speak, their respective *internal states*. Beliefs can be right or wrong, depending on their corresponding or not to the actual facts. Moreover, we have taken the option that acquiring a belief does not cancel a previous belief. As a consequence, we allow a character to simultaneously entertain more than one belief with respect to the same fact, possibly with a different degree of confidence which depends on the provenance of the beliefs. That said, we shall consider three types of information-gathering events, each type associated with a set of operations:

- Communication events - operations: `ask, tell, agree, ask_event, tell_event, agree_event`
- Perception events - operations: `sense, watch`
- Reasoning events - operations: `infer, suppose`.

Operations sense, ask, tell, agree, infer, and suppose refer to beliefs on facts, whereas watch, ask_event, tell_event, and agree_event, refer to some action event witnessed by a character. The operations are defined in terms of their pre-conditions and post-conditions [Fikes]. The pre-conditions are logical expressions involving affirmed or negated facts and beliefs, whereas post-conditions denote the effect of the operation in terms of beliefs that are added or deleted to/from the current internal states of the characters involved. However the specification of the operations is deliberately kept at a minimum, to be complemented, both with respect to pre-conditions and post-conditions, by separate *conditioners* that express the peculiarities of the different characters participating in the stories.

In the computer science community, communication between characters immediately brings to mind the communication processes executed by software agents in multi-agent environments. In particular, the Agent Communication Language consists of formally defined operations similarly defined by their pre-conditions and post-conditions [FIPA]; for an earlier more formal treatment, see for instance [Sadek]. Software agents differ from fictional characters (and, ironically, from human beings in general) in that they are supposed to only transmit information on which they believe, to agents that still lack such information and need it in order to play their role in the execution of some practical service.

In contrast, certain characters are prone to lie, either for their benefit or even out of habit. In general they may ignore the conversational maxims prescribed by philosophers of language, such as [Grice]. The bare specification of our tell(A,B,F) operation does not even require that A has any notion of the fact F to be transmitted to B. It is enough that both characters are at the same local L; if they are not, a current_place(A,L) sub-goal is recursively activated, which may cause the displacement of the teller (character A) to L, where B currently is. And the only necessary effect of the operation is merely that F has been told by A to B. Whether or not B will believe in F will depend on the execution of the agree operation, which in turn depends on whether or not B trusts A. Another purpose served by tell(A,B,F) is to convey merely expressive speech acts [Searle]; the F parameter can then be any arbitrary sentence. With whatever purpose the operation is used, the B parameter may remain unspecified, in which case A is addressing a general audience.

The ask operation is similarly defined, and its effect is just that A has asked F from B, who may respond or not. The fundamental character-dependent conditioners are established, respectively, by separate will_tell and will_ask conditioners.

Perception is the faculty whereby people keep contact with the world through their five senses (sight, hearing, touch, smell and taste). At the present stage of our work we do not make such distinctions, and merely consider a generic sense(C,F) operation to apprehend any sort of fact F specified in the static schema as perceptible, with a variant version that makes provision for defective sensing. For correct sensing of a positive or negative fact F, F must be successfully tested. Distorted sensing (for instance, of certain colours by a daltonic subject) is accompanied by a side-remark on the true fact. In any case, besides the sensed clause (which, similarly to the told and asked clauses of the communication operations, registers the main effect of the sense operation) a belief clause is immediately added, since direct perception does not depend on a third party who might not be trusted.

The watch(C,E) operation allows a character C to witness an event E, denoted as always in our system by some operation defined in the dynamic schema. Operations ask_event, tell_event, and agree_event – analogously to the fact-oriented ask, tell and agree – permit,

respectively, that another character C' may question C about E, that C (spontaneously or after being questioned) relates the event, and that C' effectively agrees with its occurrence.

As before, the definitions of these operations are left to be completed by conditioners, respectively sense_rule and watch_rule clauses. For sense, it is required that, to ascertain a positive or negative fact F involving a person or object currently at place L, a character C must be at L, either originally or as the result of pursuing current_place(C,L) as a sub-goal. For watch, the current_place requirements depend on what type of event is being watched, which justifies their being left to the special watch_rule clauses. For instance, the action operation go(A,L1,L2) (which, incidentally, is universally required as the basic way to update current_place(C,L) facts), can be watched partly by persons at L1 (origin) and partly by those present at L2 (destination).

We established that both the agent of an event E and a character who watches the occurrence of E are, as expected, aware of, at least, the main effects of the event. Anticipating what we shall treat in our detective stories environment, if A kills B, or if C watches A killing B, then the facts killed(A,B) and dead(B,true) will be believed by such characters (i.e. the agent A and/or any witness C). And when the agent or a witness uses tell_event to inform another character C' and C' reacts with an agree_event, this person C' will also start believing in those facts caused by the event related. The ability to transmit several facts by communicating one event seems to us a most convenient feature.

Deduction, induction and abduction are complementary reasoning strategies. For deduction, if there is a rule $A \rightarrow B$ and the antecedent A is known to hold, it is legitimate to *infer* that the consequent B holds. In the case of induction (fundamental to the natural sciences), the systematic occurrence of B whenever A occurs may justify the adoption of rule $A \rightarrow B$. Abduction (cf. [Peirce], for example) is a non-guaranteed but nevertheless most useful resource in many uncertain situations: given the rule $A \rightarrow B$, and knowing that B holds, one may *suppose* that A also holds. This is a type of reasoning habitually performed by medical doctors, who try to diagnose an illness in view of observed symptoms. The trouble is, of course, that it is often the case that more than one illness may provoke the same symptom — in other words: there may exist other applicable rules $A^1 \rightarrow B$, $A^2 \rightarrow B$, ..., $A^n \rightarrow B$, suggesting different justifications for the occurrence of B. Thus in abduction, wherein the implication arrow is followed backward, one is led to formulate hypotheses rather than the firm conclusions issuing from deduction over deterministic rules.

Our infer and suppose operations utilize, respectively, deduction and abduction. The conditioners for both can be the same rules of inference (inf_rules) to be traversed forward in the former case or backward in the latter. In our implementation of the infer operation, given a rule P=>F accepted by character A, the antecedent P furnishes the beliefs to be tested as pre-condition, whereas A's belief in F will be acquired as the added effect (another addition being an inferred clause) upon a successful evaluation of P. In contrast, in the case of the suppose operation, the belief in the consequent will just motivate the addition of a supposed clause in a fact present in the logical expression of the antecedent.

We must stress that the inference rules adopted by the characters in our story context do not pretend to be scientifically correct. Often originating from popular traditions, they may lead to far-fetched or even absurd beliefs. A curious example will suffice. As a goal that, so we thought, should fail, we enquired how could a red-haired girl called Mary come to believe that her hair was not red. To our surprise, our plan-generator found a solution using inference in a most devious way: John gives Peter the correct information, which he

faithfully transmits to Mary. However, having first noticed that Peter is daltonic, Mary is led to apply our (admittedly naive) inference rule dealing with red-green colour blindness. Here is the result, which, incidentally, provides an example of pseudo-natural language texts obtained by applying simple templates to the formal notation of the plan (assigned to variable `P`) :

```
?- plans((believes('Mary',hair_colour('Mary',C)), not (C = red)),P), narrate(P),
nl.

Mary senses that Peter is daltonic. John tells Peter: "- Mary has red hair".
Peter agrees with John. Mary asks Peter: "- What is the colour of my hair?".
Peter tells Mary: "- Your hair is red". Mary infers that she has green hair.

C = green,
P = start=>sense(Mary, daltonic(Peter, true))=>tell(John, Peter,
hair_colour(Mary, red))=>agree(Peter, John, hair_colour(Mary, red))=>ask(Mary,
Peter, hair_colour(Mary, C))=>tell(Peter, Mary, hair_colour(Mary,
red))=>infer(Mary, hair_colour(Mary, green)).
```

2.3. Directive and commissive events

In order to achieve a desired goal, a character `C` may need to resort to another character `C'` to perform an action that `C` is not able – or, for some reason, is not willing – to execute personally. Three operations were supplied to meet this requirement: `request`, `comply` , and `refuse`. Linguists [Searle] classify such communicative acts in the directive (the first) and in the commissive categories (the two last ones).

As before, the specification of pre-conditions and post-conditions is complemented by conditioner clauses, named respectively `will_request`, `will_comply` and `will_refuse`. It is normal to specify (and we took this option in the detective story example described in section 3) that `C'` always complies to the requests of `C` if an `obeys(C',C)` relationship has been declared to bind one to the other. On the other hand, compliance may be subjected to a reciprocal request: `C'` would, so to speak, negotiate with `C`, imposing a task as payment or compensation for the service to be rendered to `C` – see the example in section 3 (which will also serve to illustrate the use of conditioners). Moreover, even if a character has `complied` to perform an action it does not necessarily follow that the promise will be fulfilled, which is in consonance with the existence of characters who shamelessly lie when passing information.

The `request` operation can be either directed to a specific character or left open (denoted by an uninstantiated variable in this case). By allowing the second option we obtained the generality afforded by the `cfp` (call-for-participation) operation of the Agent Communication Language [FIPA].

2.4. Library-consulting events

As, in a sense, a dual process to plan-generation, plan-recognition is a no less invaluable resource for the composition of story plots. In principle, if we let the plan-generator run for an indefinite time, it should produce all plots consistent with the given specification, not all

of which worthy of our attention. It is therefore sensible to provide a collection of story patterns extracted from pre-existing narratives of diverse provenance.

Such collection, residing in a conveniently indexed library [Schank, Kolodner], could then contribute to create new plots by adaptation, or – as in our case – to match a few observed events against typical story patterns, thereby allowing to predict, with a fair chance of success, what the agents involved are trying to accomplish.

The items in our libraries have the format `user/main-agent-involved/conclusive-message/goals/plan/complementary-test`. For our current purposes, the `plan`, the `test` and the `message` are of special interest.

Our library-consulting operations, which are also incorporated in the plots produced by the planner, are named `collect`, `recognize`, and `try`. The first simply assembles in a list the events that have been directly `watched` by or `related` to a character. The `recognize` operation performs the pattern-matching itself, checking whether all events in the list of observations correspond to events in the `plan` component of a library item. If the pattern-matching succeeds, the logical expression in the `test` is passed to the `try` operation, with a double purpose: to exclude trivial cases of successful matches and to call for the execution of additional events; typically, in the detective stories, they are recommendations to the detective to gather further information. It should be noted that such events are then appended and thus also become part the generated plot. Finally, if both the pattern-matching and the subsequent test succeed, the `message` is composed and becomes available (in particular to the domain-oriented `expose` operation to be introduced in the next section, whereby the detective communicates his verdicts).

3. The detective stories domain

3.1. Detectives in action

One thing we do *not* propose to do is to model the action of real-life detectives, whose work is supported today by the highly sophisticated resources of forensic science. The method we are modestly beginning to develop is based on suggestions coming from some illustrious fictional detectives. We refrained, however, on Bertillon's advice, to abide by Mrs. Ariadne Oliver's preposterous claim [Christie-2] that, pursuant to the adoption of her unorthodox criterion for choosing the head of Scotland Yard, reason should yield its place to intuition.

Curiously various suggestions from the experts are in harmony with a major contribution of Semiotics, namely the characterization of the so-called *four master tropes* [Ramus, Vico, Burke, White], which have been declared to constitute "a system, indeed *the* system, by which the mind comes to grasp the world conceptually in language" [Culler]. In previous work we associated these tropes – metonymy, metaphor, synecdoche, and irony – with, respectively, four relations between events, which we have denominated [Ciarlini-2] syntagmatic, paradigmatic, meronymic, and antithetic.

According to [Chandler], metonyms are based on various indexical relationships between concepts, notably the substitution of effect for cause, and convey an idea of contiguity. Borrowing from [Saussure], we require the presence of s*yntagmatic relations* between events, to justify their being meaningfully placed in sequence. Indeed a detective must first of all see the events under investigation as a coherent cause-and-effect sequence,

wherein each event creates the conditions for what comes next. Dupin's genius is of this sort [Poe]:

> At such times I could not help remarking and admiring (although from his rich ideality I had been prepared to expect it) a peculiar analytic ability[2] in Dupin. He seemed, too, to take an eager delight in its exercise – if not exactly in its display – and did not hesitate to confess the pleasure thus derived.

The definition of events via the pre- / post-conditions of operations and the composition of plots by a backward-chaining planner is the main device we use to guarantee consistency along the syntagmatic axis. Our situation-objective rules also play an important role, functioning as *triggers* for future actions of the agents involved. Specifically for the domain of detective stories, we concentrate on motivation aspects.

The paradigmatic relations, inspired on metaphor [Lakoff], arise from similarities and analogies. Story patterns tend to repeat themselves, as Hercule Poirot so well realized when reflecting on Norton's skill to induce several other people to commit a crime in his stead [Christie-1]:

> It was amazing. But it was not new. There were parallels. And here comes in the first of the "clues" I left you. The play of *Othello*. For there, magnificently delineated, we have the original of X. Iago is the perfect murderer. The deaths of Desdemona, of Cassio – indeed of Othello himself – are all Iago's crimes, planned by him, carried out by him.

As said before, our specifications include libraries of story patterns and library-consulting events were included in our method, thus allowing detectives to access previous cases, classic or not, that seem to incorporate some familiar motif.

In [Winston], where six types of part-of links are distinguished, one reads: "We will refer to relationships that can be expressed with the term 'part' in the above frames as 'meronymic' relations after the Greek 'meros' for part". Going down to details, such as finding how many times the ash of a cigar has fallen on the soil, is one of Sherlock Holmes precautions [Doyle]:

> Before turning to those moral and mental aspects of the matter which present the greatest difficulties, let the enquirer begin by mastering more elementary problems. Let him, on meeting a fellow-mortal, learn at a glance to distinguish the history of the man, and the trade or profession to which he belongs. Puerile as such an exercise may seem, it sharpens the faculties of observation, and teaches one where to look and what to look for. By a man's finger nails, by his coat-sleeve, by his boot, by his trouser knees, by the callosities of his forefinger and thumb, by his expression, by his shirt cuffs—by each of these things a man's calling is plainly revealed. That all united should fail to enlighten the competent enquirer in any case is almost inconceivable.

Noteworthy details thus include fingerprints, footprints, likely and unlikely weapons, the fact that someone is carrying a jewel, etc. We have dealt elsewhere [Ciarlini-2], though not here, with an even more significant aspect of whole-part decomposition, namely the description of events at the level of more basic actions. For example, a homicide may comprise the obtention of a lethal poison, the act of pouring it in a glass of wine, etc., etc.

[2] The emphasis is ours

Antithetic relations express negation and opposition, such as the apparently irreducible difference between good and evil and, consequently, between the hero and the villain. And yet – ironically – shifting from one extreme to its contrary may be necessary for understanding an opponent. Father Brown, a catholic priest, a man of impeccable morals, thus explains his performance as a detective [Chesterton]:

> I had planned out each of the crimes very carefully," went on Father Brown, "I had thought out exactly how a thing like that could be done, and in what style or state of mind a man could really do it. And when I was quite sure that I felt exactly like the murderer myself, of course I knew who he was.

Quite appropriately we must remember that the learned chronicler of Father Brown's adventures was considered a master of paradox [Kenner]. But dramatic irony [Booth] can perhaps be pointed out as the most characteristic ingredient of detective stories. The character who looks more innocent-looking is in many cases found to be the sought-for criminal. And frequently the detective is compelled to change a line of investigation because it is revealed that things were "another way round". What makes a story interesting is almost always the culprit's skill as a deceiver, inducing false beliefs that until the final showdown appear to be true. Among the early cases of Bertillon to be reported in section 4.2, the two last ones would seem to have a touch of irony.

Let us consider one more point about the habitual reasoning practices of a detective. Like medical doctors, they often proceed by abduction, on which we based our `suppose` operation. Sherlock Holmes once said to his friend Dr. Watson, as the good doctor narrates in his memoirs [Doyle]:

> "... In solving a problem of this sort, the grand thing is to be able to reason backwards. That is a very useful accomplishment, and a very easy one, but people do not practise it much. In the every-day affairs of life it is more useful to reason forwards, and so the other comes to be neglected. There are fifty who can reason synthetically for one who can reason analytically."
> "I confess," said I, "that I do not quite follow you."
> "I hardly expected that you would. Let me see if I can make it clearer. Most people, if you describe a train of events to them, will tell you what the result would be. They can put those events together in their minds, and argue from them that something will come to pass. There are few people, however, who, if you told them a result, would be able to evolve from their own inner consciousness what the steps were which led up to that result. This power is what I mean when I talk of reasoning backwards, or analytically."

To finish this section, let us recall how Mrs. Oliver's intuition surprisingly seemed, while appraising Poirot's methods, to anticipate what Bertillon would propose to do with the benefit of the more advanced technology of our 21st century [Christie-3]:

> "Do you know what you sound like?" said Mrs. Oliver. "A computer. You know. You're programming yourself. That's what they call it, isn't it? I mean you're feeding all these things into yourself all day and then you're going to see what comes out."
> "It is certainly an idea you have there," said Poirot, with some interest. "Yes, yes, I play the part of the computer. One feeds in the information. "
> "And supposing you come up with all the wrong answers?" said Mrs. Oliver.
> "That would be impossible," said Hercule Poirot. "Computers do not do that sort of a thing."
> "They're not supposed to," said Mrs. Oliver, "but you'd be surprised at the things that happen sometimes. My last electric light bill, for instance. I know there's a proverb which says To err is human, but a human error is nothing to what a computer can do if it tries."

3.2. Events in the domain of detective stories

Two events correspond to crimes, namely `kill` and `steal`. Event `attack` is also an aggressive action, not necessarily criminal, whose agent can in principle be any of the characters participating in the story. For expressing the detective's provisional or final conclusions, an `expose` event is provided.

The specification of operation `kill`, the major focus of all investigations reported in section 4, is shown below. The pre-condition combines some `motivation` clause with the general requirements that the victim should not already be dead and that the killer must be at the same location as the victim currently is. Note that the requirements supplied in the involved `motivation` clause undergo a previous preparation: each fact `F` mentioned in the `Circ` parameter (the circumstances that motivate the crime) is converted into `believes(X,F)`, where `X` is the would-be criminal.

```
operation(kill(X,Y,M)).
added(killed(X,Y),kill(X,Y,M)).
added(motive(X,[kill(X,Y),M]),kill(X,Y,M)).
added(dead(Y,true),kill(X,Y,M)).
precond(kill(X,Y,M),P) :-
  motivation(X,[kill(X,Y,M),Circ]),
  prep_mot(X,Circ,Circ1),
  appc((current_place(X,L),/current_place(Y,L),/(not dead(Y,true))),Circ1,P).
```

A few typical motivation clauses follow:

```
motivation(A,[kill(A,B,greed),(owns(B,O),not (A = B),carries_object(A,O))]).
motivation(A,[kill(A,B,jealousy),(loves(A,B),loves(B,C),gender(A,M),gender(C,M),
  not (A = B), not (A = C))]).
motivation(A,[kill(A,B,request),(obeys(A,C),not (B = C),
  complied(A,[C,kill(A,B,request)]))]).
motivation(A,[kill(A,B,vengeance),(loves(A,C),not (A = B), not (A = C),
  killed(B,C), not (B = C))]).
motivation(A,[kill(A,B,'self-defense'),(attacked(B,A), not (A = B))]) :-
  A = 'Marian'.
motivation(A,[kill(A,A,lovesickness),(loves(A,B),not loves(B,A),not (A = B))]).
```

A detective is supposed to not only find the identity of the killer but also the motive of the crime. An economical way to formulate an appropriate inference clause, to be applied by the detective `D`, is, in words: "`D` will infer that, if `D` believes that `X` killed `Y` and that the circumstances described in `Circ` currently held, then the motive `M` that guided `X` was that indicated in the motivation clause corresponding to those same circumstances". Two clauses were used to express that formally, the second clause serving to introduce a bias: we anticipate that, in the case of suicide, the detective (in our example) will always assume a crisis of lovesickness as explanation of the killer/victim's desperate action:

```
inf_rule(D,(killed(X,Y),Circ) => motive(X,[kill(X,Y),M])) :-
  motivation(X,[kill(X,Y,M),Circ]), not (X == Y).
inf_rule(D,killed(X,X) => motive(X,[kill(X,X),lovesickness])).
```

As said before, detective stories make ample use of all the communicative events of section 2. Among the conditioners that provide the necessary flexibility to the operations,

those associated with the directive and commissive events deserve special attention. In the context of detective stories, such events mobilize the relationship between instigators and their accomplices. The following `will_comply` conditioners convey a possible arrangement: a character `C1` who `obeys` to `C2` will always comply with whatever `C2` may request; on the other hand, if the obedient `C1` requests that the dominating `C2` shall kill some person `C3`, `C2` will comply only if `C1` in turn complies to also get involved in the criminal aggression against `C3`, specifically by stealing an object owned by the victim (this sort of tricky negotiation guides the action in case 6 of section 4):

```
will_comply(C1,C2,Act,true) :- obeys(C1,C2).
will_comply(C2,C1,kill(C2,C3,request),
  (obeys(C1,C2),owns(C3,O),complied(C1,[C2,steal(C1,O,C3)]))).
```

4. Enters Bertillon

4.1. The context of his miniworld

Six personages are involved:

The *dramatis personae*:

Bertillon	a private French detective initiating his career in England
Marian	lovely red-haired young lady, still unmarried
Robin	Marian's suitor
Patrick	another suitor of Marian, a notorious lecher
Jane	former actress serving as Patrick's accomplice
Cogsworth	British butler, head of Marian's household

At the initial state, all characters are in London, except Jane who is in Manchester. Cogsworth, the loyal butler of Marian[3], is destined to participate as Bertillon's main witness. He is indeed ever disposed to testify and even to volunteer information, always consistent with what he believes to be true. Marian is equally sincere, but is a shade too credulous. In particular, she accepts whatever is told by Jane, who happens to be a compulsive liar. Marian plays, in most occasions, the role of the victim. Patrick is tempted to kill her, either impelled by jealousy or because he covets a precious jewel that she imprudently wears attached to a necklace. She must also beware of Jane, obsessively averse to red-haired persons and an obedient accessory to Patrick's machinations. On the bright side, she has nothing to fear from her butler, and enjoys a reciprocal love relationship with young Robin.

Cases 1 through 5 are rather straightforward; mainly relying on Cogsworth's testimony – but also thanks to his (un)fair knowledge of the meta data – Bertillon is able to infer who is the culprit and his or her motivation. Cases 6 and 7 are somewhat more involved, compelling the detective to resort to the minimal library of assorted crime patterns displayed below:

[3] For a reference, confirming his unshakable dedication to his previous employer, see: http://www.ctgso.org/actors/documents/BeautyandtheBeastJrMASTER.pdf

```
lib([

 (U/A/[A,' deceived ',B,', possibly with criminal intent']/
 (loves(B,C),told(A,[B,not loves(C,B)]),believes(B,not
  loves(C,B)),told(A,[B,loves(C,D)]),believes(B,loves(C,D)),
  killed(B,C),killed(B,B))/
 (start=>tell(A,B,not loves(C,B))=>agree(B,A,not
  loves(C,B))=>kill(B,C,jealousy)=>kill(B,B,M))/
 (/told(A,[B,not loves(C,B)]),trusts(U,X),asked(U,[X,loves(C,B)]),
  told(X,[U,loves(C,B)]),agreed(U,[X,loves(C,B)]))),

 (U/A/[A,' may have used ',B,'''s theft as cover-up,
  but the suspect is ',A,' possibly moved by jealousy']/
 (obeys(B,A),person(D),not D = A, not D = B,current_place(D,L),
  current_place(B,L),requested(A,[B,steal(B, O, V)]),
  watched(D,request(A,B,steal(B, O, V))),
  stolen(B, [V,O]),killed(A,V))/
 (start=>request(A, B, steal(B, O, V))=>steal(B, O, V)=>kill(A, V, jealousy))/
 (sensed(U,loves(A,V)),questioned(U,[D,request(A,B,steal(B, O, V))]),
  related(D,[U,request(A,B,steal(B, O, V))]))),

 (U/default/['The culprit is the butler!']/
 (true)/
 (start)/
 (true))
  ]).
```

4.2. His seven cases

For each case, we shall provide a brief synopsis, the calling sequence to the planner, and the generated narrative. The plot texts result from the application to the plans of still crude templates [Deemter], forgivably perchance if one recalls that English is not Bertillon's native language. We formatted the texts for readability, using italics and colours.

case 1: *In a fit of passion*

synopsis: Patrick kills Marian, unaware of the presence of the butler, who is later in a position to communicate the event to Bertillon, together with all information needed to establish that the murderer's motive was jealousy.

```
ex1 :-
  plans(( motive('Patrick',[kill('Patrick','Marian'),jealousy]),
          watched('Cogsworth',kill('Patrick','Marian',M)),
          related('Cogsworth',['Bertillon',kill('Patrick','Marian',M)]),
          agreed_op('Bertillon',['Cogsworth',kill('Patrick','Marian',M)]),
          agreed('Bertillon',['Cogsworth',loves('Patrick','Marian')]),
          agreed('Bertillon',['Cogsworth',loves('Marian','Robin')]),
          inferred('Bertillon',motive(S,[kill(S,'Marian'),M])),
          exposed('Bertillon',[S,'Marian',jealousy,nil])
        ),Plan),narrate(Plan),nl,nl, !.
```

Patrick senses that Marian loves Robin. Patrick kills Marian. Cogsworth watches the event: 'Patrick kills Marian'. Cogsworth relates to Bertillon the event: 'Patrick kills Marian'. Bertillon agrees with Cogsworth about the event. Cogsworth tells Bertillon: "- *Patrick loves Marian*". Bertillon agrees with Cogsworth. Cogsworth tells Bertillon: "- *Marian loves Robin*". Bertillon agrees with Cogsworth. Bertillon assumes that Patrick, in the event 'Patrick kills Marian', was motivated by jealousy. Bertillon says: "- *The suspect is Patrick and the motive is jealousy*".

case 2: *All that glitters*

synopsis: This time Patrick commits two crimes: murder and theft. The first is once again watched by the butler, who later notices that the culprit carries the object of a second crime, namely theft, thus characterizing greed as the primary motivation.

```
ex2 :-
  plans((  sensed('Patrick',owns('Marian',jewel)),
           stolen('Patrick',['Marian',jewel]),
           motive('Patrick',[kill('Patrick','Marian'),greed]),
           watched('Cogsworth',kill('Patrick','Marian',M)),
           sensed('Cogsworth',carries_object('Patrick',jewel)),
           related('Cogsworth',['Bertillon',kill('Patrick','Marian',M)]),
           agreed_op('Bertillon',['Cogsworth',kill('Patrick','Marian',M)]),
           agreed('Bertillon',['Cogsworth',owns('Marian',jewel)]),
           believes('Bertillon',
              motive('Patrick',[kill('Patrick','Marian'),greed])),
           supposed('Bertillon',carries_object('Patrick',jewel)),
           asked('Bertillon',['Cogsworth',carries_object('Patrick',jewel)]),
           agreed('Bertillon',['Cogsworth',carries_object('Patrick',jewel)]),
           inferred('Bertillon',motive('Patrick',[kill(S,'Marian'),M])),
           exposed('Bertillon',[S,'Marian',greed,nil])
        ),Plan),narrate(Plan),nl,nl, !.
```

Patrick senses that Marian has a jewel. Patrick steals jewel from Marian. Patrick kills Marian. Cogsworth watches the event: 'Patrick kills Marian'. Cogsworth senses that Patrick carries a jewel. Cogsworth relates to Bertillon the event: 'Patrick kills Marian'. Bertillon agrees with Cogsworth about the event. Cogsworth tells Bertillon: "- *Marian has a jewel*". Bertillon agrees with Cogsworth. Bertillon supposes that Patrick carries a jewel. Bertillon asks Cogsworth: "- *Does Patrick carry a jewel?*". Cogsworth tells Bertillon: "- *Patrick carries a jewel*". Bertillon agrees with Cogsworth. Bertillon assumes that Patrick, in the event 'Patrick kills Marian', was motivated by greed. Bertillon says: "- *The suspect is Patrick and the motive is greed*".

case 3: *Murder by proxy*

synopsis: The jealous Patrick procures Marian's death by ordering Jane to do the killing. The inevitable butler watches their conversation and Jane's fatal act. Learning of both scenes from Cogsworth, Bertillon establishes Jane's direct involvement as well as Patrick's role as instigator, not bothering however to disclose the latter's motive (jealousy, this time).

```
ex3 :-
  plans((  current_place('Marian','Manchester'),
           current_place('Cogsworth','Manchester'),
           current_place('Bertillon','Manchester'),
           complied('Jane',['Patrick',kill('Jane','Marian',request)]),
           watched('Cogsworth',
              request('Patrick','Jane',kill('Jane','Marian',request))),
           killed('Jane','Marian'),
           watched('Cogsworth',kill('Jane','Marian',M)),
           related('Jane',['Patrick',kill('Jane','Marian',M)]),
           related('Cogsworth',['Bertillon',kill('Jane','Marian',M)]),
           agreed_op('Bertillon',['Cogsworth',kill('Jane','Marian',M)]),
           related('Cogsworth',
           ['Bertillon',request('Patrick','Jane',kill('Jane','Marian',M))]),
           agreed_op('Bertillon',
              ['Cogsworth',request('Patrick','Jane',kill('Jane','Marian',M))]),
           inferred('Bertillon',motive('Jane',[kill(S,'Marian'),M])),
           believes('Bertillon',
              requested(I,['Jane',kill('Jane','Marian',request)])),
```

```
          exposed('Bertillon',[S,'Marian',request,I])
        ),Plan),narrate(Plan),nl,nl, !.
```

Marian goes from London to Manchester. Cogsworth goes from London to Manchester. Bertillon goes from London to Manchester. Patrick goes from London to Manchester. Patrick orders Jane: "- *Kill Marian!*". Jane complies with Patrick's request: "- *At your behest, I will kill Marian*". Cogsworth watches the event: 'Patrick orders Jane: "- *Kill Marian!*"'. Jane kills Marian. Cogsworth watches the event: 'Jane kills Marian'. Jane relates to Patrick the event: 'Jane kills Marian'. Cogsworth relates to Bertillon the event: 'Jane kills Marian'. Bertillon agrees with Cogsworth about the event. Cogsworth relates to Bertillon the event: 'Patrick orders Jane: "- *Kill Marian!*"'. Bertillon agrees with Cogsworth about the event. Bertillon assumes that Jane, in the event 'Jane kills Marian', was motivated by request. Bertillon says: "- *The suspect is Jane, with Patrick as the instigator*".

case 4: *The reluctant victim*

synopsis: But our victim is not necessarily so helpless! Jane, on her own initiative, repelled as she is at the sight of red-haired Marian, attacks her to be promptly killed in reaction. The butler watches the aggression and the counter-aggression events, which, reported to Bertillon, result in a verdict of self-defense.

```
ex4 :-
  plans((  attacked('Jane','Marian'),
           watched('Cogsworth',attack('Jane','Marian')),
           motive('Marian',[kill('Marian','Jane'),'self-defense']),
           watched('Cogsworth',kill('Marian','Jane',M)),
           related('Cogsworth',['Bertillon',attack('Jane','Marian')]),
           agreed_op('Bertillon',['Cogsworth',attack('Jane','Marian')]),
           related('Cogsworth',['Bertillon',kill('Marian','Jane',M)]),
           agreed_op('Bertillon',['Cogsworth',kill('Marian','Jane',M)]),
           inferred('Bertillon',motive('Marian',[kill(S,'Jane'),M])),
           exposed('Bertillon',[S,'Jane','self-defense',nil])
        ),Plan),narrate(Plan),nl,nl, !.
```

Jane goes from Manchester to London. Jane senses that Marian has red hair. Jane attacks Marian. Cogsworth watches the event: 'Jane attacks Marian'. Marian senses that Jane attacked her. Marian kills Jane. Cogsworth watches the event: 'Marian kills Jane'. Cogsworth relates to Bertillon the event: 'Jane attacks Marian'. Bertillon agrees with Cogsworth about the event. Cogsworth relates to Bertillon the event: 'Marian kills Jane'. Bertillon agrees with Cogsworth about the event. Bertillon assumes that Marian, in the event 'Marian kills Jane', was motivated by self-defense. Bertillon says: "- *The suspect is Marian, but the motive is just self-defense*".

case 5: *Avenging fury*

synopsis: Patrick kills Marian moved by jealousy. Two persons watch the murder: her Butler and her lover. Cogsworth limits himself to give testimony, but Robin's reaction is more effective: he kills the assassin. To many a body of jurors, one might surmise, vengeance in the aftermath of a heinous murder should at least seem admissible as extenuating circumstance.

```
ex5 :-
  plans((  killed('Patrick','Marian'),
           watched('Cogsworth',kill('Patrick','Marian',_)),
           watched('Robin',kill('Patrick','Marian',_)),
           killed('Robin','Patrick'),
           related('Cogsworth',['Bertillon',kill('Robin','Patrick',M)]),
           agreed_op('Bertillon',['Cogsworth',kill('Robin','Patrick',M)]),
           inferred('Bertillon',motive('Robin',[kill(S,'Patrick'),M])),
           believes('Bertillon',killed('Patrick',V)),
```

```
          exposed('Bertillon',[S,'Patrick',vengeance,V])
        ),Plan),narrate(Plan),nl,nl, !.
```

Patrick senses that Marian loves Robin. Patrick kills Marian. Cogsworth watches the event: 'Patrick kills Marian'. Robin watches the event: 'Patrick kills Marian'. Robin kills Patrick. Cogsworth watches the event: 'Robin kills Patrick'. Cogsworth relates to Bertillon the event: 'Robin kills Patrick'. Bertillon agrees with Cogsworth about the event. Cogsworth tells Bertillon: "- *Robin loves Marian*". Bertillon agrees with Cogsworth. Cogsworth tells Bertillon: "- *Patrick killed Marian*". Bertillon agrees with Cogsworth. Bertillon assumes that Robin, in the event 'Robin kills Patrick', was motivated by vengeance. Bertillon says: "- *The suspect is Robin and the motive is vengeance, given that Patrick had first killed Marian*".

case 6: *Framed!*

synopsis: Patrick's loose morals do not prevent him from arming a trap for the submissive Jane. Preparing to kill Marian, pressed as before by his jealous impulses, he takes advantage of Jane's own inclinations. To exterminate the detested red-haired young lady, Jane requests the aid of her superior. As a condition to comply, Patrick requires that she should also perform some aggressive act, at the same making a profit; namely, she must steal Marian's rich jewel. The butler misses Jane's request to Patrick, but watches Patrick imposing the theft of the jewel as a condition for complying, and also the theft itself. However he exclusively reports to Bertillon the latter event – which in no way incriminates Patrick. Perceiving that Marian is dead, Bertillon, like so many detectives in popular fiction, builds the hypothesis (an instance of abductive reasoning from the only rule he knows to explain death: `killed(X,Y) => dead(Y,true)`) that someone killed her. And, at a loss for anything else, he proceeds from the only clue available, i.e. the theft reported by Cogsworth, to consult his library of crime patterns. A match occurs with a pattern involving requested theft by one person as cover-up for the more drastic act of the instigator. The library item recommends checking whether such a request occurred, which leads Bertillon to question Cogsworth, who responds relating the event that he, at first, had failed to report. Once, for a change, Bertillon has not enough data for a firm inference, but he advances the possibility that Patrick murdered Marian and sought to cast all suspicion upon his accomplice, Jane.

```
ex6 :-
  plans((  sensed('Patrick',loves('Marian',L)),
           current_place('Jane','London'),
           complied('Patrick',['Jane',kill('Patrick','Marian',request)]),
           watched('Cogsworth',
              request('Patrick','Jane',steal('Jane',jewel,'Marian'))),
           stolen('Jane',['Marian',jewel]),
           watched('Cogsworth',steal('Jane',jewel,'Marian')),
           motive('Patrick',[kill('Patrick','Marian'),jealousy]),
           sensed('Bertillon',dead('Marian',true)),
           supposed('Bertillon',killed(X,'Marian')),
           related('Cogsworth',['Bertillon',steal('Jane',jewel,'Marian')]),
           obs('Bertillon',Obs),
           recognized('Bertillon','Bertillon'/Ag/Cr_type/Goals/Pl_lib/Q),
           tried('Bertillon',Q),
           exposed('Bertillon',['Patrick','Marian',lib,Cr_type])
        ),P),
  narrate(P),nl,nl,!.
```

Patrick senses that Marian loves Robin. Jane goes from Manchester to London. Jane senses that Marian has red hair. Jane requests Patrick: "- *I ask you to kill Marian*". Patrick orders Jane: "- *You should steal the jewel that Marian possesses*". Jane complies with Patrick's request. Patrick complies with Jane's request: "- *As you demanded, I will kill Marian*".

Cogsworth watches the event: 'Patrick orders Jane: "- *You should steal the jewel that Marian possesses*"'. Jane steals jewel from Marian. Cogsworth watches the event: 'Jane steals jewel from Marian'. Patrick kills Marian. Bertillon senses that Marian is dead. Bertillon supposes that someone killed Marian. Cogsworth relates to Bertillon the event: 'Jane steals jewel from Marian'. Bertillon collects observations. Bertillon recognizes a plan. Bertillon senses that Patrick loves Marian. Bertillon questions Cogsworth about the event: 'Patrick orders Jane: "- *You should steal the jewel that Marian possesses*"'. Cogsworth relates to Bertillon the event: 'Patrick orders Jane: "- *You should steal the jewel that Marian possesses*"'. Bertillon says: "- *Patrick may have used Jane's theft as cover-up, but the suspect is Patrick possibly moved by jealousy*".

case 7: *Iago syndrome*

synopsis: Though meekly consenting to the whims of Patrick, Jane surpasses him by her far richer imagination. Patrick is not even mentioned in this case, the hardest one until now in Bertillon's incipient career. Jane lies to Marian, convincing her that Robin does not love her. As a consequence, Marian is taken by lovesickness and kills herself. The ill-intentioned conversation and its fatal outcome are watched by Cogsworth, and this time he at once relates both scenes to Bertillon. The verdict of suicide is inescapable and Bertillon clearly states it. Feeling, however, that Jane's talking to the victim may have further implications, he again resorts to the library. What he finds is the same pattern that took his Belgian colleague, Hercule Poirot, to identify Norton as the wonted "X" in his last case [Christie-1]. The located library item contains a recommendation, leading him to check with the well-informed Cogsworth: does Robin love Marian contrary to what Jane had proclaimed? The butler's affirmative reply is an indication that Jane knowingly and deliberately induced Marian to commit suicide. Bertillon denounces Jane, despite his conviction that no British court of justice would condemn her.

```
ex7 :-
  plans(( told('Jane',['Marian',not loves('Robin','Marian')]),
          watched('Cogsworth',
             tell('Jane','Marian',not loves('Robin','Marian'))),
          killed('Marian','Marian'),
          watched('Cogsworth',kill('Marian','Marian',_)),
          related('Cogsworth',['Bertillon',tell('Jane','Marian',not F)]),
          agreed_op('Bertillon',['Cogsworth',tell('Jane','Marian',not F)]),
          related('Cogsworth',['Bertillon',kill('Marian','Marian',M)]),
          agreed_op('Bertillon',['Cogsworth',kill('Marian','Marian',M)]),
          inferred('Bertillon',motive('Marian',[kill(S,'Marian'),M])),
          exposed('Bertillon',[S,S,M,nil]),
          obs('Bertillon',Obs),
          recognized('Bertillon','Bertillon'/Ag/Cr_type/Goals/Pl_lib/Q),
          tried('Bertillon',Q),
          exposed('Bertillon',['Jane','Marian',lib,Cr_type])
     ),Plan),narrate(Plan),nl,nl, !.
```

Jane goes from Manchester to London. Jane tells Marian: "- *Robin does not love you now*". Cogsworth watches the event: 'Jane tells Marian: "- Robin does not love you now"'. Marian agrees with Jane. Marian commits suicide. Cogsworth watches the event: 'Marian commits suicide'. Cogsworth relates to Bertillon the event: 'Jane tells Marian: "- Robin does not love you now"'. Bertillon agrees with Cogsworth about the event. Cogsworth relates to Bertillon the event: 'Marian commits suicide'. Bertillon agrees with Cogsworth about the event. Bertillon assumes that Marian, in the event 'Marian commited suicide', was motivated by lovesickness. Bertillon says: "- *Marian commited suicide, moved by lovesickness*". Bertillon collects observations. Bertillon recognizes a plan. Bertillon asks Cogsworth: "- *Is Robin in love with Marian?*". Cogsworth tells Bertillon: "- *Robin loves Marian*". Bertillon agrees with Cogsworth. Bertillon says: "- *Jane deceived Marian, possibly with criminal intent*".

4.3. Interactive composition with the PlotBoard tool

Hopefully the cases narrated in the previous section are sufficient to give an idea of what can arise from the current specification of detective stories, and hence offer some indication of how useful is the package of communicative speech acts described before. However, generating an entire plot via a single call to the plan-generator algorithm is not satisfactory from a digital entertainment viewpoint. An environment to run plot composition interactively is essential, and now we proceed to describe how this is done with the **PlotBoard** tool, whose flow of control is depicted in figure 1 (for more details, cf. [Ciarlini-2]).

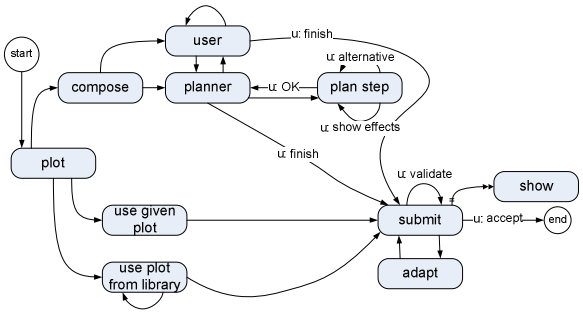

Figure 1: Flow of control of **PlotBoard**

Users only enjoy a truly immersive experience if they are allowed to guide the composition process. The planner is still welcome since it incorporates a knowledge of the domain that a casual user may not possess, but now it serves in a secondary helper's capacity. Under the user's command, the tool generates the plot in a stepwise fashion, alternating between the `user mode` and the `planner mode`.

When the `planner mode` is on, the situation-objective rules are activated to find the short-range goals that can, at the current state, be pursued by the planner. Picking one of these goals, the planner then produces an appropriate plan and displays it to the user, who may issue an `ok` or ask for an alternative plan (for the same or for another simultaneously active goal). After the chosen plan step is executed, the user is asked whether the planner may `continue`, in which case the situation-objective rules are again activated for the next step.

But the user can prefer to shift to `user mode`, wherein several types of intervention are announced in a menu, such as indicating the goal to be achieved next by the planner or even a specific operation to be executed. And after signaling `finish`, the user can still decide to perform one or more adaptations of several kinds over the events, such as inserting, deleting, replacing, reordering, summarizing, detailing, etc.

The situation-objective rules adopted in our trial run are listed below, prefixed with numbers for easy reference:

```
(1) sit_obj('Patrick',
 (loves('Patrick',Y),not dead(Y,true),not loves(Y,'Patrick'),loves(Y,Z),
  not ('Patrick' = Z)),
 (motive('Patrick',[kill('Patrick',Y),jealousy]))).

(2) sit_obj('Jane',
  (owns(Y,O),not dead(Y,true),carries_object(Y,O)),
```

```
   (sensed('Jane',owns(Y,O)),carries_object('Jane',O),
    motive('Jane',[kill('Jane',Y),greed]))).

(3) sit_obj('Jane',
   (hair_colour(V,red),not dead(V,true),loves(H,V),loves(V,H)),
   (told('Jane',[V,not loves(H,V)]),
    watched('Cogsworth',tell('Jane',V,not loves(H,V))),
    agreed(V,['Jane',not loves(H,V)]))).

(4) sit_obj('Marian',
   (not dead('Marian',true),
    loves('Marian',H),believes('Marian',not loves(H,'Marian'))),
   (loves(M,'Marian'),not (M = H),told('Marian',[M,'Let us meet someday!']))).

(5) sit_obj('Marian',
   (not dead('Marian',true),
    loves('Marian',H),believes('Marian',not loves(H,'Marian'))),
   (killed('Marian','Marian'),
    watched('Cogsworth',kill('Marian','Marian',_)),
    believes('Cogsworth',killed('Marian','Marian')))).

(6) sit_obj('Cogsworth',
   (believes('Cogsworth',killed(V,V))),
   (related('Cogsworth',['Bertillon',tell(S,V,not loves(H,V))]),
    agreed_op('Bertillon',['Cogsworth',tell(S,V,not loves(H,V))]),
    related('Cogsworth',['Bertillon',kill(V,V,M)]),
    agreed_op('Bertillon',['Cogsworth',kill(V,V,M)]))).

(7) sit_obj('Bertillon',
   (believes('Bertillon',killed(V,V))),
   (inferred('Bertillon',motive(V,[kill(S,V),M])),
    exposed('Bertillon',[S,S,M,nil]))).

(8) sit_obj('Bertillon',
   (believes('Bertillon',killed(V,V)),exposed('Bertillon',[S,S,M,nil])),
   (obs('Bertillon',Obs),
    recognized('Bertillon','Bertillon'/Ag/Cr_type/Goals/Pl_lib/Q),
    tried('Bertillon',Q),
    exposed('Bertillon',[S,V,lib,Cr_type]))).
```

At the initial state, rules (1), (2) and (3) offer alternative possibilities to the user's choice. If the goals of either (1) or (2) are pursued, Marian is murdered (by Patrick or by Jane, respectively). If (3) is preferred, Jane comes from Manchester and tells a lie to Marian in the butler's presence, coinciding with the first events of case 7, as described in the previous section. The entire plot of case 7 will indeed be generated if one chooses, at each subsequent step, the alternatives offered by rules (5) through (8).

But the story does not have to end badly. After applying (3), rules (4) and (5) become active. With (5) Marian yields to depression, but (4) allows her to recover and teach the (allegedly) disloyal Robin a lesson, by giving a chance to his rival, the ill-reputed Patrick.

When we started the trial run shown in figure 2, we took the `planner mode` and, from the alternative event sequences generated, chose the one produced by rule (3), and then shifted to the `user mode`. In a direct intervention, we added an `attack(Marian,Jane)` event, whereby Marian somehow reacted to Jane's provocation. But, returning next to the `planner mode`, we chose (5) so that the suicide scene ensued, again witnessed by the horrified Cogsworth. At this point, we indicated that the generation phase was finished. Asked by the tool whether the obtained plot should be accepted, we chose instead the `adapt` option,

and removed the fifth event (in which Marian expressed her belief in Jane's false assertion). We then selected the `show` option from the menu, thus causing the plot to be displayed via the Prolog/Java interface.

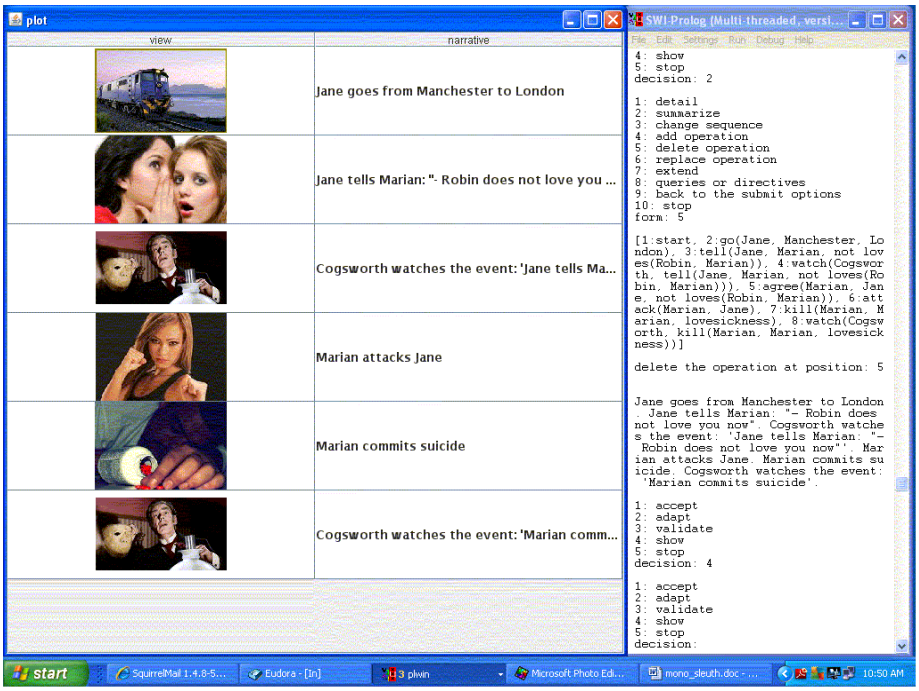

Figure 2: Interactive composition with **PlotBoard**

5. Related work

Mystery and, more specifically, detective stories do not constitute an unexplored field in digital entertainment. In fact, plot-driven detective stories (often called "whodunits", i.e. "who done [did] it?") have been extensively used as the narrative environment for a number of systems and prototypes, with different approaches. In this brief survey, we shall try to closely reproduce the main points conveyed by the referenced articles.

One of the earliest examples of an automated story writing system is the *Automatic Novel Writer* [Klein], programmed in FORTRAN. As output it produced murder mysteries of relatively considerable length (up to 2100 words), considering that it was one of the first automated story writing systems. It was also a good representative of the work done at that time on computer-based natural language generation. Indeed, while aiming at story composition, the focus was on linguistics and the construction of proper sentences.

Another early example is the laserdisc game *Murder, Anyone?*[4], which is played by two teams. In an opening video segment, the character Derrick Reardon is murdered. The object of the game is to guess the murderer, the motive and the method. After the introductory video, the first team selects either a one or a two. No significance is attached to the choice: players select a number randomly. There are a total of four such decision points. This

[4] *Murder, Anyone?* (1982). Cincinnati: Vidmax.

creates sixteen different plots, allowing the game to be played multiple times. The sixteen plots are not represented by sixteen different video presentations, though. The designers used the technique of voice-over narration to conserve space on the disc. Players are instructed to listen to audio track one if they have chosen story one, or audio track two for story two. Thus, the same video footage is used to tell two different stories. One audio track is voiceover while the other is sound from the actual scene: then they switch. Murky flashback shots are used whenever it is necessary for both audio tracks to be independent of the video [Bruckman].

Tea for Three [Weyhrauch] is a whodunit (inspired by Infocom's adventure game Deadline [Deadline]) where the user plays the role of a police detective who has to figure out (by seeking physical clues and talking to characters) if an apparent suicide was genuine, or if it was murder (in this case, also disclosing who killed the victim, how, and why). It uses an architecture called *Moe*, designed to, with the use of adversary search, decide how and when to guide the user's experience. The interactive drama is broken down into abstract pieces called user moves, and the system is able to assess any complete sequence of user moves by the evaluation function for its dramatic quality.

A location-based pervasive game prototype was developed by [Gustafsson], to be used during a car trip, including telephone and walkie-talkie interaction to unfold a crime story with supernatural twists. The user plays the role of a detective (teamed up with a partner) who tries to uncover an organized crime gang by investigating a series of crimes that seem to be related. The player's actions affect the moods and relationships between player and in-game characters, resulting in a dark adventure that may include the appearance of occult forces.

Another prototype system developed to explore location-based interactive stories is *Who Killed Hanne Holmgaard?*, presented by [Paay]. The story is a historical murder mystery set during World War II and situated in the city of Aalborg, Denmark. The system was designed for two participants to work cooperatively to unravel the mystery while on the move, each of them playing a different character in the story, and they must work together to solve the crime. The system is location-based, responding to the user's current location providing the episodes of the storyline related to that location. The interactive story, accessed using two networked PDAs, introduces the two participants to the characters they are playing, the other fictional characters in the story, episodic plot lines that interweave both fiction and fact, clues and logical puzzles that lead them through the story and through the city.

U-DIRECTOR [Mott] has a different purpose: to create a director agent able to orchestrate the events in a storyworld to improve the user's experience, coping with the uncertainty about the user's intentions and the absence of a complete theory of narrative, operating in real-time. *U-DIRECTOR* dynamically models narrative objectives, storyworld state, and user state with a dynamic decision network that continually selects storyworld actions to maximize narrative utility on an ongoing basis. This architecture has been implemented in a narrative planner for *Crystal Island*, a narrative-centered learning game in which users play the role of a medical detective solving a science mystery. Further research on the *Crystal Island* interactive storyworld has adopted (and tested) some other approaches, such as the use of supervised machine learning to recognize players' affective states [Rowe-1] or the use of dynamic Bayesian networks to model their knowledge [Rowe-2].

The non-linear interactive storytelling game engine *NOLIST* [Bangsø] also utilizes Bayesian networks to determine the culprit of a murder mystery. In this case, the Bayesian network dynamically changes in response to actions and observations made by the user, so that the engine will create a dynamic storyline attuned to player actions and choices. It utilizes the user's moves and logical inference to determine details of the story (which is not entirely preset), including the identity of the murderer. For example, if the user finds a body and a gun lying beside the body, then the probability that the victim was shot with the gun increases. *NOLIST* recreates the past as a reaction to player interaction; neither the plot nor the culprit are known by the game engine in the beginning but are determined in the course of the game.

Another example of the use of Bayesian networks to help creating a mystery narrative is the murder mystery game proposed in [Arinbjarnar-1]. At each game run, a new narrative plot and set of characters are generated. The initial plot is created with the *Dynamic Plot Generating Engine* (*DPGE*) [Arinbjarnar-2], which creates new mystery plots on demand using a Bayesian network and Proppian functions [Propp]. This use of Bayesian networks to form a murder mystery plot resembles that of *NOLIST*, but *NOLIST* does not fix the mystery plot from the start, but rather develops it continuously through game play. *DPGE*, on the other hand, fixes the initial plot at the start and then expects characters in the game to use it as background for their future actions.

Fabulator [Barros-1; Barros-2] is a prototype interactive storytelling system based on the "riddle" master plot [Tobias], which comprises stories in which there is a mystery that must be solved, under the typical guise of whodunits. This prototype uses a story-world called *Ugh's Story 2*, telling the story of cavemen whose worshiped statue was stolen. The user plays the role of a detective caveman, whose mission is to investigate the case and discover who committed the theft. This work uses a tension arcs model that assumes that the tension rises when the player acquires more knowledge leading towards the truth. The system also uses non-player characters (NPCs), who can help the player character or not, to dynamically adjust the level of difficulty to the desired level.

6. Concluding remarks

Though plot-composition methods based on plan-generation have the distinctive advantage of enforcing consistency within the genre specified, the specification itself is a worksome task, and when processed by the planning algorithm may lead to inefficient execution or even to failure to terminate, unless some care is taken to restrict the application of some clauses. In our example, for instance, we found convenient to indicate which characters could be the agents of certain operations.

Nevertheless we intend to amplify the scope of criminal action and of criminal investigation, especially in an effort to incorporate more subtle detective skills such as the analysis of the psychological profile of the suspects, on the basis of our preliminary study involving a chivalry tale domain [Barbosa]. We also propose to further develop our logic programming prototypes, possibly by trying additional capabilities such as those surveyed in the previous section.

Place and time factors should receive a closer attention. In the examples shown here, the current place of each character was merely indicated by city, whereas a more narrowly identified location and even the internal architecture of a specific building may often

provide decisive clues. Equally crucial may be the date, duration and temporal sequence of the events.

The events themselves should sometimes be narrated at the level of more detailed operations, a capability offered by the **PlotBoard** tool that we have exploited before [Ciarlini-2], but not yet in the context of detective stories. Murder events can take an amazing number of specialized forms: shooting, stabbing, poisoning, strangling, etc., etc. And each murder action can be decomposed into basic actions; shooting, for instance, comprises various basic acts starting with the obtention of a fire weapon and ending, perhaps, with the occultation of the weapon in a convenient secret chamber.

Finally, expanding the library of story-patterns, taking care to borrow from the work of detectives with different styles, seems to be a promising way of producing new plots by adapting and combining interesting ways of machinating – and solving – memorable criminal cases.

References

[Arinbjarnar-1] M. Arinbjarnar. *Rational Dialog in Interactive Games*. MSc Thesis, School of Computer Science – Reykjavík University, 2007.

[Arinbjarnar-2] M. Arinbjarnar. "Dynamic plot generation engine". *Proc. of the Workshop on Integrating Technologies for Interactive Stories*, 2008.

[Bangsø] O. Bangsø, O.G. Jensen, F.V. Jensen, P.B. Andersen, T. Kocka, T. "Non-Linear Interactive Storytelling Using Object-Oriented Bayesian Networks". *Proc. of the International Conference on Computer Games: Artificial intelligence,Design and Education*, 2004.

[Barbosa] S.D.J. Barbosa, A.L. Furtado, M.A. Casanova. "A Decision-making Process for Digital Storytelling". *Proc. of the Brazilian Symposium on Games and Digital Entertainment*, 2010.

[Barros-1] L.M. Barros, S.R. Musse, S. R. "Introducing narrative principles into planning-based interactive storytelling". *Proc. of the ACM SIGCHI International Conference on Advances in Computer Entertainment Technology*, 2005.

[Barros-2] L.M. Barros, S.R. Musse. "Towards consistency in interactive storytelling: tension arcs and dead-ends". *Computers in Entertainment*, vol. 6, no. 3, article 43, 2008.

[Batini] C. Batini, S. Ceri, S. Navathe. *Conceptual Design - an Entity-Relationship Approach*. Redwood: Benjamin Cummings, 1992.

[Booth] W. Booth. *A Rhetoric of Tropes*. Chicago: U. of Chicago Press, 1974.

[Bruckman] A. Bruckman. *The Combinatorics of Storytelling: Mystery Train Interactive*. (Interactive Cinema Group internal paper). MIT Media Lab, 1990.

[Burke] K. Burke. *A Grammar of Motives*. Berkeley: University of California Press, 1969.

[Chandler] D. Chandler. *Semiotics: the Basics*. London: Routledge, 2007.

[Chesterton] G.K. Chesterton. *The Secret of Father Brown*. West Valley City: The Editorium, 2006.

[Ciarlini-1] A.E.M. Ciarlini, A.L. Furtado. "Constructing libraries of typical plans". *Proc. 13th Conference on Advanced Information Systems Engineering*, 2001.

[Ciarlini-2] A.E.M. Ciarlini, S.D.J. Barbosa, M.A. Casanova, A.L. Furtado. "Event relations in plan-based plot composition". *Computers in Entertainment*, 7, 4, 2009.

[Ciarlini-3] A.E.M. Ciarlini, M.A. Casanova, A.L. Furtado, P.A.S. Veloso. "Modeling interactive storytelling genres as application domains". *Journal of Intelligent Information Systems*, v. 35, n. 3, 2010.

[Chandler] D. Chandler. *Semiotics: the Basics*. London: Routledge, 2002.

[Christie-1] A. Christie. *Curtain*. New York: Pocket Books, 1976.

[Christie-2] A. Christie. *Cards on the Table*. New York: HarperCollins, 2007.

[Christie-3] A. Christie. *Hallowe'en Party*. New York: HarperCollins, 2011.

[Culler] J. Culler. *The Pursuit of Signs: Semiotics, Literature, Deconstruction*. London: Routledge, 1981.

[Deadline] Deadline. Infocom, 1982, cf.: http://en.wikipedia.org/wiki/Deadline_(video_game)

[Deemter] K.V. Deemter, E. Krahmer, M. Theune. "Real versus Template-Based Natural Language Generation: A False Opposition?". *Computational. Linguistics*, vol. 31, n. 1, 2005.

[Doyle] A. Conan Doyle. *The Complete Sherlock Holmes*. New York: Doubleday, 1986.

[Fikes] R.E. Fikes, N.J. Nilsson. "STRIPS: a new approch to the application of theorem proving to problem solving". *Artificial Intelligence*, 2 (3-4).

[FIPA] FIPA Communicative Act Library Specification, 2002, at: http://www.fipa.org/specs/fipa00037/SC00037J.html

[Grice] H.P. Grice. "Logic and conversation". In P. Cole, J.L. Morgan (eds.), *Syntax and Semantics, Speech Acts*, vol. 3, New York: Academic Press, 1975.

[Gustafsson] A. Gustafsson, J. Bichard, L. Brunnberg, O. Juhlin, M. Combetto. "Believable environments: generating interactive storytelling in vast location-based pervasive games". *Proc. of the ACM SIGCHI international conference on Advances in computer entertainment technology*, 2006.

[Kenner] H. Kenner. *Paradox in Chesterton*. New York: Sheed & Ward, 1947.

[Klein] S. Klein, J.F. Aeschlimann, D.F. Balsiger, S.L. Converse, C. Court, M. Forster, R. Lao, J.D. Oakley, J. Smith. *Automatic Novel Writing: A Status Report*. Technical Report 186, Computer Sciences Department, University of Wisconsin, 1973.

[Kolodner] Kolodner, J.L. *Case-based Reasoning*. San Mateo: Morgan Kaufmann Publishers, 1993.

[Lakoff] G. Lakoff, M. Johnson. *Metaphors We Live By*. Chicago: University of Chicago Press, 2003.

[Mott] B. Mott, J. Lester. "U-Director: A Decision-theoretic narrative planning architecture for storytelling environments". *Pro. of the 5th International Conference on Autonomous Agents and Multiagent Systems*, 2006.

[Paay] J. Paay, J. Kjeldskov, A. Christensen, A. Ibsen, D. Jensen, G. Nielsen, R. Vutborg. "Location-based storytelling in the urban environment". *Proc. OZCHI: Designing for Habitus and Habitat*, 2008.

[Peirce] C.S. Peirce. *Writings of Charles S. Peirce: a Chronological Edition*. N. Houser (ed.). Bloomington: Indiana University Press, 1998.

[Poe] E.A. Poe. *Complete Stories and Poems of Edgar Alan Poe*. New York: Doubleday, 1984.

[Propp] V. Propp. *Morphology of the Folktale*. L. Scott (trans.). Austin: University of Texas Press, 1968.

[Ramus] P. Ramus. *Rhetoricae Distinctiones in Quintilianum*. J.J. Murphy (ed.), C. Newlands (trans.). Carbondale: Southern Illinois University, 2010.

[Rao] A.S. Rao, M.P. Georgeff. "Modeling rational agents within a BDI-architecture". *Proc. of the International Conference on Principles of Knowledge Representation and Reasoning*, 1991.

[Rowe-1] J. Rowe, J. Lester, J. "Modeling user knowledge with dynamic Bayesian networks in interactive narrative environments". *Proc. 6th Annu. AI Interact. Digital Entertain. Conerence.*, 2010.

[Rowe-2] J. Rowe, B. Mott, S. McQuiggan, J. Robison, S. Lee, J. Lester, J. "Crystal Island: A Narrative-Centered Learning Environment for Eighth Grade Microbiology". *Proc. of the AIED'09 Workshop on Intelligent Educational Games*, 2009.

[Sadek] M.D. Sadek. "Logical task modelling for man-machine dialogue". *Proc. Eighth National Conference on Artificial Intelligence*, 1990.

[Saussure] F. Saussure. *Cours de Linguistique Générale*. C. Bally, A. Sechehaye, A. Riedlinger (eds.). Paris: Payot, 1995.

[Schank] R.C. Schank, R.P. Abelson. *Scripts, Plans Goals, and Understanding*. New York: Psychology Press, 1977.

[Searle] J.R. Searle. Expression and Meaning. Cambridge: Cambridge University Press, 1979.

[Silva] F.A.G. da Silva, A.L. Furtado, A.E.M. Ciarlini, C.T. Pozzer, B. Feijó, E.S. de Lima. "Information-gathering events in story plots". *Proc. 11th International Conference on Entertainment Computing*, 2012.

[Tobias] R.B. Tobias, R. B. *20 Master Plots: And How to Build Them*. Cincinnati: Writer's Digest Books, 1993.

[Todorov] T. Todorov. *The Poetics of Prose*. Cornell: Cornell University Press, 1977.

[Vico] G. Vico. *The New Science*. T.G. Bergin, M.H. Finch (trans.). Ithaca: Cornell University Press, 1968.

[Weyhrauch] P. Weyhrauch. *Guiding Interactive Drama*, Ph.D. Dissertation, Tech report CMU-CS-97-109, Carnegie Mellon University, 1997.

[White] H. White. *Metahistory: the Historical Imagination in Nineteenth-Century Europe*. Baltimore: John Hopkins University Press, 1973.

[Winston] M.E. Winston, R. Chaffin., D. Herrmann.. "A taxonomy of part-whole relations". *Cognitive Science*, 11, 4, 1987.

Appendix

```
/* ================================ */
/* COMMUNICATIVE EVENTS PACKAGE */
/*==================================*/

% preparatory commands

:- set_prolog_flag(verbose,silent).
:- style_check([-singleton,-discontiguous]).
:- set_prolog_flag(toplevel_print_options,[max_depth(50)]).

:- dynamic asked/2, told/2, agreed/2, agreed_op/2, sensed/2,
     watched/2, questioned/2, related/2,
     inferred/2, supposed/2,
     state_rep_ini/1, ini/0 , believes/2 .
:- dynamic requested/2, complied/2, refused/2 .
:- dynamic remembered/2.
:- dynamic owns/2, carries_object/2.
:- dynamic exposed/2.
:- dynamic current_place/2.
:- dynamic obs/2, recognized/2, tried/2, exposed/2, will_ask/3.
:- dynamic it_m/1 .


% including and preparing for the planning algorithm

:- op(900,fy,not).
:- op(650,yfx,=>).
:- op(500,fx,/).
:- dynamic '/'(/1 ).
:- dynamic log/1 .
log(start).
log :-
  once(narrate),nl.
:- include(warbeta_info).
added(X,Y) :- /added(X,Y).
deleted(X,Y) :- /deleted(X,Y).
added(pre_state(O,S),O).
/(added(believes(A,F),O)) :- agent(A,O), added(F,O).
/(deleted(believes(A,F),O)) :- agent(A,O), deleted(F,O).


/* static schema - general properties required by the package */

entity(person,name).
attribute(person,gender).
relationship(current_place,[person,Place]) :-
  taken_as_place(Place).
attribute(person,believes).
attribute(person,told).
attribute(person,agreed).
attribute(person,agreed_op).
attribute(person,sensed).
attribute(person,watched).
attribute(person,related).
attribute(person,inferred).
attribute(person,supposed).
attribute(person,asked).
attribute(person,questioned).
attribute(person,requested).
attribute(person,complied).
```

```
attribute(person,refused).
attribute(person,obs).
attribute(person,recognized).
attribute(person,remembered).
attribute(person,exposed).
attribute(person,recognized).
attribute(person,tried).
relationship(trusts,[person,person]).
relationship(obeys,[person,person]).

perceptible(X) :- entity(E,_),X =.. [E,_].
perceptible(gender(X,Y)).
perceptible(current_place(X,Y)).
perceptible(trusts(X,Y)).
perceptible(obeys(X,Y)).


/* dynamic schema - operations of the package */

% communication operations

operation(tell(A,B,F)).
added(told(A,[B,F]),tell(A,B,F)).
precond(tell(A,B,F),P) :-
  not var(B),
  will_tell(A,B,F,P1),
  appc((current_place(A,L),/current_place(B,L)),P1,P).
precond(tell(A,B,F),P) :-
  var(B),
  will_tell(A,F,P).
template(tell(A,B,F),[A,' tells ',B,': "- '|Ft]) :-
  not var(B),
  ((fact(F); F = (not Fn), fact(Fn)), !,
   def_template(F,A,B,Ft1),
   append(Ft1,['"'],Ft);
   Ft = [F,'"']).
template(tell(A,B,F),[A,' says: "- '|Ft]) :-
  var(B),
  (fact(F),
   def_template(F,A,none,Ft1),
   append(Ft1,['"'],Ft);
   not fact(F),Ft = [F,'"']).

operation(ask(A,B,F)).
added(asked(A,[B,F]),ask(A,B,F)).
precond(ask(A,B,F),P) :-
  not var(B),
  will_ask(A,B,F,P1),
  appc((current_place(A,L),/current_place(B,L)),P1,P).
precond(ask(A,B,F),P) :-
  var(B),
  will_ask(A,F,P).
template(ask(A,B,F),[A,' asks ',B,': "- '|Ft]) :-
  not var(B),
  def_template(F,int,A,B,Ft1),
  append(Ft1,['?"'],Ft).
template(ask(A,B,F),[A,' asks: "- '|Ft]) :-
  var(B),
  def_template(F,int,A,none,Ft1),
  append(Ft1,['?"'],Ft).

operation(agree(A,B,F)).
added(believes(A,F),agree(A,B,F)).
```

```
added(agreed(A,[B,F]),agree(A,B,F)).
precond(agree(A,B,F),(fact(F),told(B,[A,F]),trusts(A,B))).
precond(agree(A,B,F),(not              var(F),F              =              (not
  Fn),fact(Fn),told(B,[A,F]),trusts(A,B))).
template(agree(A,B,F),[A,' agrees with ',B]).

operation(agree_event(A,B,O)).
added(believes(A,F),agree_event(A,B,O)) :- added(F,O).
added(believes(A,not F),agree_event(A,B,O)) :- deleted(F,O).
added(agreed_op(A,[B,O]),agree_event(A,B,O)).
precond(agree_event(A,B,O),(knows_effs(A,Ops),on(O,Ops),trusts(A,B),related(B,[A,
  O]))).
template(agree_event(A,B,O),[A,' agrees with ',B,' about the event']).


% perception operations

operation(sense(W,F)).
added(believes(W,F),sense(W,F)).
added(sensed(W,F),sense(W,F)).
precond(sense(W,F),(once(perceptible(F)),P)) :-
  sense_rule(W,F,P1),
  (not (F = (not current_place(_,_))),
   appc((/F,once(property(F,Pr,I)),current_place(W,L),/current_place(I,L)),P1,P);
   F = (not current_place(I,L)),
   appc((/F,current_place(W,L)),P1,P)).
template(sense(A,F),[A,' senses that '|Ft]) :-
  def_template(F,A,nil,Ft).

operation(sense(W,F1,F2)).
added(believes(W,F2),sense(W,F1,F2)).
added(sensed(W,F2),sense(W,F1,F2)).
precond(sense(W,F1,F2),(once(perceptible(F1)),P)) :-
  sense_rule(W,F1,F2,P1),
  appc((once(property(F1,Pr,I)),current_place(W,L),/current_place(I,L)),P1,P).
template(sense(W,F,F),[W,' senses that '|Ft]) :-
  def_template(F,W,nil,Ft).
template(sense(W,F1,F2),[W,' wrongly senses that '|Ft]) :-
  def_template(F2,W,nil,Fta),
  def_template(F1,W,nil,Ftb),
  (F1 = (not _),!,Ft = Fta;
   append(Fta,[' -- in fact '|Ftb],Ft)).

operation(watch(W,O)).
added(watched(W,O),watch(W,O)).
added(believes(W,F),watch(W,O)) :- main_eff(W,O,F).
precond(watch(W,O),(person(W),not
  agent(W,O),pre_state(O,S),T,once(holds(current_place(W,L),S)))) :-
  watch_rule(W,O,R,L),
  (R = true, T = true; not (R = true), T = holds(R,S)).
template(watch(W,F),[W,' watches the event: '''|Ft]) :-
  template(F,Ft1),
  append(Ft1,['''],Ft).

operation(ask_event(A,B,O)).
added(questioned(A,[B,O]),ask_event(A,B,O)).
precond(ask_event(A,B,O),P) :-
  not var(B),
  will_question(A,B,O,P1),
  appc((current_place(A,L),/current_place(B,L)),P1,P).
precond(ask_event(A,B,O),P) :-
  var(B),
  will_question(A,O,P).
```

```
template(ask_event(A,B,O),[A,' questions ',B,' about the event: '''|Ot]) :-
  not var(B),
  template(O,Ot1),
  append(Ot1,['''']',Ot).
template(ask_event(A,B,O),[A,' questions about the event: '''|Ot]) :-
  var(B),
  template(O,Ot1),
  append(Ot1,['''']',Ot).

operation(tell_event(A,B,O)).
added(related(A,[B,O]),tell_event(A,B,O)).
precond(tell_event(A,B,O),(pre_state(O,S),R2)) :-
  agent(A,O),!,
  relate_rule(A,B,O,R1),
  R2 = (agent(A,O),R1).
precond(tell_event(A,B,O),(pre_state(O,S),R2)) :-
  relate_rule(A,B,O,R1),
  R2 = (watched(A,O),R1).
template(tell_event(A,B,O),[A,' relates to ',B,' the event: '''|Ot]) :-
  template(O,Ot1),
  append(Ot1,['''']',Ot).


% reasoning operations

operation(infer(A,F)).
/added(believes(A,F),infer(A,F)).
added(inferred(A,F),infer(A,F)).
precond(infer(A,F),P) :- ded(A,F,P).
template(infer(A,F),[A,' assumes that '|Ft]) :-
  def_template(F,A,nil,Ft).

ded(A,F,Fc) :-
  inf_rule(A,P=>F),
  conj_list(P,L),
  ded1(A,L,Lc),
  conj_list(Fc,Lc).

ded1(A,[],[]).
ded1(A,[/X|R],T) :- !,
  ded1(A,[X|R],T).
ded1(A,[X|R],[believes(A,X)|S]) :-
  (property(X);
   (X = (not Xn)),property(Xn)),!,
  ded1(A,R,S).
ded1(A,[X|R],[X|S]) :-
  ded1(A,R,S).

operation(suppose(A,F)).
added(supposed(A,F),suppose(A,F)).
precond(suppose(A,F),C) :- property(F),abd(A,F,C).
template(suppose(A,F),[A,' supposes that '|Ft]) :-
  def_template(F,A,nil,Ft).

abd(A,Fr,Fc) :-
  inf_rule(A,P=>F),
  on_conj(Fr,P),
  (believes(A,F),!,Fc = true;
   Fc = /believes(A,F);
   Fc = sensed(A,F);
   Fc = told(_,[A,F])).
```

```
% directive and commissive operations

operation(request(A,B,F)).
added(requested(A,[B,F]),request(A,B,F)).
precond(request(A,B,F),P) :-
  not var(B),
  will_request(A,B,F,P1),
  appc((current_place(A,L),/current_place(B,L),not (A == B)),P1,P).
precond(request(A,B,F),P) :-
  var(B),
  will_request(A,_,F,P).
template(request(A,B,F),[A,R,B, ': "- '|Ft]) :-
  not var(B),
  (obeys(B,A), R = ' orders ';
   not obeys(B,A), R = ' requests '),
  template_req(F,Ft1),
  append(Ft1,['"'],Ft).
template(request(A,B,F),[A,' requests: "- '|Ft]) :-
  var(B),
  template_req(F,Ft1),
  append(Ft1,['"'],Ft).

operation(comply(A,B,F)).
added(complied(A,[B,F]),comply(A,B,F)).
precond(comply(A,B,F),P) :-
  will_comply(A,B,F,P1),
  appc((requested(B,[A,F]),/current_place(A,L),/current_place(B,L)),P1,P).
template(comply(A,B,F),[A,' complies with ',B,'''s request: "- '|Ft]) :-
  template_comp(F,Ft1),
  append(Ft1,['"'],Ft).
template(comply(A,B,F),[A,' complies with ',B,'''s request']) :-
  not template_comp(F,_).

operation(refuse(A,B,F)).
added(refused(A,[B,F]),refuse(A,B,F)).
precond(refuse(A,B,F),P) :-
  person(A), person(B), not obeys(A,B),
  will_refuse(A,B,F,P1),
  appc((requested(B,[A,F]),/current_place(A,L),/current_place(B,L)),P1,P).
template(refuse(A,B,F),[A,' refuses ',B,'''s request: "- '|Ft]) :-
  template_ref(F,Ft1),
  append(Ft1,['"'],Ft).
template(refuse(A,B,F),[A,' refuses ',B,'''s request']) :-
  not template_ref(F,_).


% library consulting operations

operation(collect(C,O)).
added(obs(C,O),collect(C,O)).
precond(collect(C,O),
  (pre_state(Op,S),
   once(findall(E,(op_plan(tell_event(X,C,E),S=>Op);
                   op_plan(watch(C,E),S=>Op)),O1)),
   reverse(O1,O))).
template(collect(C,O),[C,' collects observations']).

operation(recognize(C,C/Ag/Cr_type/Go/P/Q)).
added(recognized(C,C/Ag/Cr_type/Go/P/Q),recognize(C,C/Ag/Cr_type/Go/P/Q)).
precond(recognize(C,C/Ag/Cr_type/Go/P/Q),
      (obs(C,Obs),
       lib(L),
       recognize_lib(Obs,C/Ag/Cr_type/Go/P/Q,L))).
```

```
template(recognize(C,C/Ag/Cr_type/Go/P/Q),[C,' recognizes a plan']).
   recognize_lib(Obs,C/A/Cr_type/G/Pl/Q,Lib) :-
  Obs = [], !,
  member(C/Def/Cr_type/G/Pl/Q,Lib),
  Def == default.

recognize_lib(Obs,C/A/Cr_type/G/Pl/Q,Lib) :-
  member(C/A/Cr_type/G/Pl/Q,Lib),
  recognize(Obs,Pl).

recognize(Obs,Pl) :-
  ex1p(Pl,L),
  recog(Obs,L,Rec),
  chk_patt(Rec,Obs).

recog([],_,[]).
recog([X1|R],[X2|L],[X2|T]) :-
  copy(X2,X3),
  X1 = X3,
  recog(R,L,T).
recog([X|R],[_|S],T) :-
  recog([X|R],S,T).

operation(try(C,Q)).
  added(tried(C,Q),try(C,Q)).
  precond(try(C,Q),Q).
  template(try(C,Q),['']).


% go operation

operation(go(C,L1,L2)).
deleted(current_place(C,L1),go(C,L1,L2)).
added(current_place(C,L2),go(C,L1,L2)).
precond(go(C,L1,L2),
  (/(not dead(C,true)),city(L1),city(L2))).
template(go(C,L1,L2),[C,' goes from ',L1,' to ',L2]).
template_req(go(C,L1,L2),['Go to ',L2]).


/* facilities to handle facts */

% listing all property-denoting facts holding at the current state

facts :-
  nl,
  forall((property(X),X),describe(X)), nl.

property(not F,P,C) :-
  property(F,P,C).

property(F,P,C) :-
  fact(F),
  F =.. [P,C],
  entity(P,_).
property(F,P,C) :-
  fact(F),
  F =.. [P,C,_],
  attribute(_,P),
  not member(P,
   [believes,told,asked,questioned,agreed,agreed_op,
    sensed,watched,related,inferred,supposed,
  complied,refused,requested,exposed,remembered]).
```

```
property(F,P,C) :-
  fact(F),
  (F =.. [P,C,_]; F =.. [P,_,C]),
  relationship(P,_),
  not P = trusts,
  not P = obeys.

property(F) :-
  fact(F),
  one(property(F,_,_)).

% listing the information-package facts holding at the current state

info_facts :-
  nl,
  forall(
   (fact(X),
  not property(X,_,_),
  X,
  clause(X,true)),
   (template(X,T),
    xclist(T,L),
  write('    '),write(L),nl)),
   nl.


/* template-handling facilities */

def_template(F,C1,C2,T) :-
  (F = (not Ft), !, M = neg;
   Ft = F, M = pos),
  def_template(Ft,M,C1,C2,T).

def_template(F,M,C1,C2,T) :-
  not var(C1),
  F =.. [P,Cf,Cx],
  Cx == C1, !,
  (not (C2 == nil),F1 =.. [P,Cf,me];
   C2 == nil,
      (gender(C1,male),F1 =.. [P,Cf,him];
    gender(C1,female),F1 =.. [P,Cf,her])),
  def_template(F1,M,C1,C2,T).

def_template(F,M,C1,C2,T) :-
  not var(C2),
  F =.. [P,Cf,Cx],
  Cx == C2, !,
  F1 =.. [P,Cf,you],
  def_template(F1,M,C1,C2,T).

def_template(F,M,C1,C2,T) :-
  F =.. [P,Cf,V],!,
  (var(Cf), not (M = int), !, C3 = someone; true),
  (C2 = nil,!,
    (C1 == Cf,!,
     gender(C1,G),
      (G = male,!, C3 = he;
    G = female, C3 = she);
   C3 = Cf);
   C3 = Cf),
  Ft =.. [P,C3,V],
  f_template(Ft,Ts),
  (M = pos, !,
```

```
    (C1 = C3, !, Mt = i_pos;
   C2 = C3, !, Mt = y_pos;
   Mt = pos);
   M = neg, !,
    (C1 = C3, !, Mt = i_neg;
   C2 = C3, !, Mt = y_neg;
   Mt = neg);
   M = int, ground(F), !,
    (C1 = C3, !, Mt = i_int;
   C2 = C3, !, Mt = y_int;
   Mt = int);
   M = int, not ground(F), !,
    (var(C3), !, Mt = int_who;
     C1 = C3, !, Mt = i_int_v;
   C2 = C3, !, Mt = y_int_v;
   Mt = int_v)),
  member(Mt:T,Ts).

def_template(F,M,C1,C2,T) :-
  F =.. [P,Cf],
  (var(Cf), not (M = int), !, C3 = someone; true),
  (C2 = nil,!,
    (C1 == Cf,!,
     gender(C1,G),
     (G = male,!, C3 = he;
    G = female, C3 = she);
   C3 = Cf);
   C3 = Cf),
  Ft =.. [P,C3],
  f_template(Ft,Ts),
  (M = pos, !,
    (C1 = C3, !, Mt = i_pos;
   C2 = C3, !, Mt = y_pos;
   Mt = pos);
   M = neg, !,
    (C1 = C3, !, Mt = i_neg;
   C2 = C3, !, Mt = y_neg;
   Mt = neg);
   M = int, ground(F), !,
    (C1 = C3, !, Mt = i_int;
   C2 = C3, !, Mt = y_int;
   Mt = int);
   M = int, not ground(F), !,
     Mt = int_v),
  member(Mt:T,Ts).

f_template(watched(X,E),
  [pos: [X, ' watched the event: '''|Et],
   i_pos: ['I watched the event: '''|Et],
   y_pos: ['You watched the event: '''|Et],
   neg: [X, ' did not watch the event: '''|Et],
   i_neg: ['I did not watch the event: '''|Et],
   y_neg: ['You did not watch the event: '''|Et],
   int: ['Did ',X,' watch the event: '''|Et],
   i_int: ['Did I watch the event: '''|Et],
   y_int: ['Did you watch the event: '''|Et],
   int_v: ['Who did watch the event: '''|Et]]) :-
     template(E,Et1),
   varnames(Et1),
   append(Et1,['''''],Et).

f_template(related(X,[Y,E]),
  [pos: [X, ' related to ',Y,' the event: '''|Et],
```

```
   neg: [X, ' did not relate to ',Y,' the event: '''|Et]]) :-
     template(E,Et1),
   varnames(Et1),
   append(Et1,[''''],Et).

f_template(requested(X,[Y,E]),
  [pos: [X, ' requested from ',Y,' the event: '''|Et]]) :-
     template(E,Et1),
   varnames(Et1),
   append(Et1,[''''],Et).

f_template(told(X,[Y,E]),
  [pos: [X, ' told ',Y,': '''|Et]]) :-
     f_template(E,[pos:Et1|_]),
   varnames(Et1),
   append(Et1,[''''],Et).

f_template(person(X),
  [pos: [X,' is a person'],
   i_pos: ['I am a person'],
   y_pos: ['You are a person'],
   neg: [X,' is not a person'],
   i_neg: ['I am not a person'],
   y_neg: ['You are not a person'],
   int: ['Is ',X,' a person'],
   i_int: ['Am I a person'],
   y_int: ['Are you a person'],
   int_v: ['Who is a person']]).

f_template(current_place(X,Y),
  [pos: [X,' is now in ',Y],
   i_pos: ['I am now in ',Y],
   y_pos: ['You are now in ',Y],
   neg: [X,' is not in ',Y,' now'],
   i_neg: ['I am not in ',Y,' now'],
   y_neg: ['You are not in ',Y,' now'],
   int: ['Is ',X,' in ',Y,' now'],
   i_int: ['Am I in ',Y],
   y_int: ['Are you in ',Y],
   int_v: ['Where is ',X,' now' ],
   i_int_v: ['Where am I'],
   y_int_v: ['Where are you'],
   int_who: ['Who is now in ',Y]]).

template(told(A,[B,F]),T) :-
  template(tell(A,B,F),T1),
  replace(' tells ',' told ',T1,T).
template(told(A,[B, not F]),T) :-
  template(tell(A,B,not F),T1),nl,write(T1),nl,nl,
  replace(' tells ',' told ',T1,T).
template(asked(A,[B,F]),T) :-
  template(ask(A,B,F),T1),
  replace(' asks ',' asked ',T1,T).
template(sensed(A,F),T) :-
  template(sense(A,F),T1),
  replace(' senses that ',' sensed that ',T1,T).
template(agreed(A,[B,F]),T) :-
  template(agree(A,[B,F]),T1),
  replace(' agrees with ',' agreed with ',T1,T).
template(watched(A,E),T) :-
  template(watch(A,E),T1),
  replace(' watches the event: \'',' watched the event: \'',T1,T).
template(related(A,[B,E]),T) :-
```

```
  template(tell_event(A,B,E),T1),
  replace(' relates the event: \'',' related the event: \'',T1,T).
template(inferred(A,F),T) :-
  template(infer(A,F),T1),
  replace(' assumes that ',' assumed that ',T1,T).
template(exposed(A,F),T) :-
  template(expose(A,F),T1),
  replace('says','said',T1,T).
template(supposed(A,F),T) :-
  template(suppose(A,F),T1),
  replace(' supposes that ',' supposed that ',T1,T).
template(believes(A,F),[A,' believes that '|Ft]) :-
  def_template(F,A,nil,Ft).
template(trusts(A,B),[A,' trusts ',B]).
template(obeys(A,B),[A,' obeys ',B]).


/* general utilities */

evs_from_facts(E,F) :-
  xsetof(Ei,Fi^(on_conj(Fi,F),added(Fi,Ei)),E).

replace(_,_,[],[]) :- !.
replace(X,Y,X,Y) :-
  atomic(X), !.
replace(X,Y,X,Y) :-
  var(X), !.
replace(X,Y,Z,Z) :-
  atomic(Z), !,
  not (X = Z), !.
replace(X,Y,[Z|L],[Z1|L1]) :- !,
  replace(X,Y,Z,Z1),
  replace(X,Y,L,L1).
replace(X,Y,Z1,Z2) :-
  Z1 =.. [F|L],
  replace(X,Y,F,F1),
  replace(X,Y,L,L1),
  Z2 =.. [F1|L1].

conc(A,B,C) :-
  name(A,L1),
  name(B,L2),
  append(L1,L2,L3),
  name(C,L3).

describes(A) :-
  is_conj(A),
  forall(on_conj(B, A), describes(B)).
describes(A) :-
  not is_conj(A),
  (ground(A);
   not ground(A),
       once(A)),
  (f_template(A, [pos:B|_]);
   template(A, B)),
   xclist(B, C),
   write('    '),
   write(C),
   nl.

chk_patt(P,L) :-
  listvar(P,Lv),
  count(Lv,N),
```

```
  P = L,
  xsetof(X,member(X,Lv),Lvc),
  count(Lvc,N).

chk_watch(start) :- !.
chk_watch(start=>X) :- !.
chk_watch(P1=>O=>watch(C,O)) :- !,
  chk_watch(P1).
chk_watch(P1=>watch(C1,O)=>watch(C2,O)) :- !,
  chk_watch(P1=>watch(C1,O)).
chk_watch(P=>X=>watch(C,Y)) :- !, fail.
chk_watch(P=>O) :-
  chk_watch(P).

plansx(O,P) :- plans(O,P), chk_watch(P).

iterate(N,X) :-
  it_i,
  X,
  it(N), !,
  retract(it_m(_)).

it_i :-
  (it_m(X), retract(it_m(X));
   true),
  assert(it_m(1)).

it(N) :-
  it_m(I),
  (I = N,!;
   J is I+1,
   retract(it_m(I)),
   assert(it_m(J)), !,
   fail).

clear :-
  findall(H/N,(fact(F),F=..[H|_],current_predicate(H/N)),S),
  forall(member(H/N,S),abolish(H/N)),
  retractall(log(_)),
  retractall(pre_state(_,_)).

reconsult(P) :-
  clear, consult(P).
```

```
/*================================*/
/* DETECTIVE STORIES DOMAIN       */
/*================================*/

:- dynamic city/1,hair_colour/2,daltonic/2,
   dead/2,killed/2,attacked/2,stolen/2,
   loves/2,owns/2,carries_object/2,motive/2.

/* static schema */
/* also contains additional properties for the 'person' entity */

entity(city,city_name).
entity(object,object_name).
attribute(person,dead).
attribute(person,hair_colour).
attribute(person,daltonic).
attribute(person,motive).
attribute(person,stolen).
relationship(loves,[person,person]).
relationship(owns,[person,object]).
relationship(carries_object,[person,object]).
relationship(killed,[person,person]).
relationship(attacked,[person,person]).

perceptible(dead(X,Y)).
perceptible(hair_colour(X,Y)).
perceptible(daltonic(X,Y)).
perceptible(loves(X,Y)).
perceptible(owns(X,Y)).
perceptible(carries_object(X,Y)).
perceptible(killed(X,Y)).
perceptible(attacked(X,Y)).

taken_as_place(city).


/* dynamic schema - domain operations */

operation(kill(X,Y,M)).
/added(dead(Y,true),kill(X,Y,M)).
added(killed(X,Y),kill(X,Y,M)).
added(motive(X,[kill(X,Y),M]),kill(X,Y,M)).
precond(kill(X,Y,M),P) :-
  motivation(X,[kill(X,Y,M),S1]),
  prep_mot(X,S1,S),
  appc((current_place(X,L),/current_place(Y,L),/(not dead(Y,true))),S,P).
template(kill(X,Y,M),[X,' kills ',Y]) :- not (X = Y).
template(kill(X,X,M),[X,' commits suicide']).
template(kill(X,Y),[X,' kills ',Y]) :- not var(X), not var(Y), not (X = Y).
template(kill(X,Y),[X,' commited suicide']) :- not var(X), not var(Y), X = Y.
template(kill(X,Y),['someone killed ',Y]) :- var(X), not var(Y).
template(kill(X,Y),[X,' killed someone']) :- not var(X), var(Y).
template_req(kill(X,Y,M),[' Kill ',Y,'!']) :- obeys(X,_), !.
template_comp(kill(X,Y,M),['At your behest, I will kill ',Y]) :- obeys(X,_), !.
template_req(kill(X,Y,M),['I ask you to kill ',Y]).
template_comp(kill(X,Y,M),['As you demanded, I will kill ',Y]).

prep_mot(X,S1,S) :-
  conj_list(S1,Ls),
  ded1(X,Ls,Lc),
  conj_list(S,Lc).
```

```
operation(attack(X,Y)).
added(attacked(X,Y),attack(X,Y)).
precond(attack(X,Y),P) :-
  will_attack(X,Y,S),
  appc((current_place(X,L),/current_place(Y,L),/(not dead(Y,true))),S,P).
template(attack(X,Y),[X,' attacks ',Y]) :- not (X = Y).
template(attack(X,X),[X,' inflicts self punishsment']).
template(attack(X,Y),[X,' kills ',Y]) :- not var(X), not var(Y), not (X = Y).
template(attack(X,Y),[X,'  inflicted  self  punishsment'])  :-  not  var(X),  not
  var(Y), X = Y.
template(attack(X,Y),['someone attacked ',Y]) :- var(X), not var(Y).
template(attack(X,Y),[X,' attacked someone']) :- not var(X), var(Y).
template_req(attack(X,Y),[' Attack ',Y,'!']).

operation(steal(X,O,Y)).
deleted(carries_object(Y,O),steal(X,O,Y)).
added(carries_object(X,O),steal(X,O,Y)).
added(stolen(X,[Y,O]),steal(X,O,Y)).
precond(steal(X,O,Y),
  (person(X),person(Y),not (X = Y),object(O),
   current_place(X,L),/((current_place(Y,L),carries_object(Y,O))))).
template(steal(X,O,Y),[X,' steals ',O,' from ',Y]).
template_req(steal(X,O,Y),[' You should steal the ',O,' that ',Y,' possesses']).

operation(expose(D,S,V,M,E)).
added(exposed(D,[S,V,M,E]),expose(D,S,V,M,E)).
precond(expose(D,S,V,M,E),true).
template(expose(D,S,V,jealousy,E),
  [D,' says: "- The suspect is ',S,' and the motive is jealousy"']).
template(expose(D,S,V,greed,E),
  [D,' says: "- The suspect is ',S,' and the motive is greed"']).
template(expose(D,S,V,request,E),
  [D,' says: "- The suspect is ',S,', with ',E,' as the instigator"']).
template(expose(D,S,V,'self-defense',E),
  [D,' says: "- The suspect is ',S,', but the motive is just self-defense"']).
template(expose(D,S,V,vengeance,E),
  [D,' says: "- The suspect is ',S,' and the motive is vengeance, given that ',V,
  ' had first killed ',E,'"']).
template(expose(D,S,S,lovesickness,E),
  [D,' says: "- ',S,' commited suicide, moved by lovesickness"']).
template(expose(D,S,V,lib,E),T) :-
  append([D,' says: "- '],E,T1),
  append(T1,['"'],T).


/* behavioural schema */

/* situation-objective rules */

sit_obj('Patrick',
 (loves('Patrick',Y),not   dead(Y,true),not   loves(Y,'Patrick'),loves(Y,Z),   not
  ('Patrick' = Z)),
 (motive('Patrick',[kill('Patrick',Y),jealousy]))).

sit_obj('Jane',
  (owns(Y,O),not dead(Y,true),carries_object(Y,O)),

  (sensed('Jane',owns(Y,O)),carries_object('Jane',O),motive('Jane',[kill('Jane',Y
  ),greed]))).

sit_obj('Jane',
  (hair_colour(V,red),not dead(V,true),loves(H,V),loves(V,H)),
  (told('Jane',[V,not loves(H,V)]),
```

```
   watched('Cogsworth',tell('Jane',V,not loves(H,V))),
   agreed(V,['Jane',not loves(H,V)]))).

sit_obj('Marian',
  (not                dead('Marian',true),loves('Marian',H),believes('Marian',not
  loves(H,'Marian'))),
  (loves(M,'Marian'),not (M = H),told('Marian',[M,'Let us meet someday!']))).

sit_obj('Marian',
  (loves('Marian',H),believes('Marian',not loves(H,'Marian'))),
  (killed('Marian','Marian'),
   watched('Cogsworth',kill('Marian','Marian',_)),
   believes('Cogsworth',killed('Marian','Marian')))).

sit_obj('Cogsworth',
  (believes('Cogsworth',killed(V,V))),
  (related('Cogsworth',['Bertillon',tell(S,V,not loves(H,V))]),
   agreed_op('Bertillon',['Cogsworth',tell(S,V,not loves(H,V))]),
   related('Cogsworth',['Bertillon',kill(V,V,M)]),
   agreed_op('Bertillon',['Cogsworth',kill(V,V,M)]))).

sit_obj('Bertillon',
  (believes('Bertillon',killed(V,V))),
  (inferred('Bertillon',motive(V,[kill(S,V),M])),
   exposed('Bertillon',[S,S,M,nil]))).

sit_obj('Bertillon',
  (believes('Bertillon',killed(V,V)),exposed('Bertillon',[S,S,M,nil])),
  (obs('Bertillon',Obs),
   recognized('Bertillon','Bertillon'/Ag/Cr_type/Goals/Pl_lib/Q),
   tried('Bertillon',Q),
   exposed('Bertillon',[S,V,lib,Cr_type]))).


/* conditioners */

% rules: will_tell

will_tell(X,Y,F,not fact_p_n(F)).
will_tell(X,F,not fact_p_n(F)).

fact_p_n(F) :- fact(F).
fact_p_n(F) :- not var(F), F = (not Fn), property(Fn).

will_tell('Cogsworth','Bertillon',F,(believes('Cogsworth',F))).
will_tell('Cogsworth','Bertillon',F,(copy(F,Fc),asked('Bertillon',['Cogsworth',Fc
  ]),sensed('Cogsworth',F))).
will_tell('Cogsworth','Robin',hair_colour(M,C),believes('Cogsworth',hair_colour(M
  ,C))).
will_tell('Jane',X,F,(perceptible(F), not (F = hair_colour(_,_)),/(not F))).
will_tell('Jane',X,not F,(perceptible(F), not (F = hair_colour(_,_)),/F)).
will_tell('Jane',X,hair_colour('Marian',blond),true).
will_tell('Robin','Marian',hair_colour('Marian',C),believes('Robin',hair_colour('
  Marian',C))).

% rules: will_ask

will_ask('Bertillon',Y,F,true) :- person(Y).
will_ask(X,Y,Fc,(copy(Fc,F),supposed(X,F),/(not sensed(X,F)))) :- not var(Fc).

% rules: regular sense_rules

sense_rule(W,F,true) :-
```

```
  not (daltonic(W,true), (F = hair_colour(_,_); F = (not hair_colour(_,_)))).

% rules: distorted sense_rules

sense_rule(W,hair_colour(P,C1),hair_colour(P,C2),(hair_colour(P,C1),map_dalt(C1,C
  2))) :-
  daltonic(W,true).
sense_rule(W,not hair_colour(P,C1),not hair_colour(P,C2),
           (hair_colour(P,C1x),map_dalt(C1x,C2x),not            (C2x            =
  C2),map_dalt1(C2,C1,C1x))) :-
  daltonic(W,true).

map_dalt(C1,C2) :-
  (C1 = red, !, C2 = green;
   C1 = green, !, C2 = red;
   C2 = C1).

map_dalt1(C1,C2,C3) :-
  map_dalt(C1,Cx),
  (C1 = Cx, !;
   not (C1 = C3), !;
   C2 = Cx).

% rules: watch_rules

watch_rule('Cogsworth',kill(X,Y,M),current_place(Y,L),L).
watch_rule(C,kill(X,Y,M),current_place(Y,L),L) :- loves(C,Y).
watch_rule('Cogsworth',attack(X,Y),current_place(Y,L),L).
watch_rule('Cogsworth',tell(X,Y,F),current_place(Y,L),L).
watch_rule(C,request(A,B,M),current_place(B,L),L).
watch_rule(C,steal(A,O,B),current_place(A,L),L).
watch_rule(C,steal(A,O,C),current_place(A,L),L).
watch_rule(C,go(A,L1,L2),true,L1).
watch_rule(C,go(A,L1,L2),true,L2).

% rules: will_question

will_question('Bertillon',Y,O,true) :- operation(O),Y == 'Cogsworth'.
will_question('Bertillon',O,true) :- operation(O).


% rules: relate_rules and agent clause

relate_rule('Cogsworth','Bertillon',E,true).
relate_rule(C1,C2,kill(C1,V,M),requested(C2,[C1,kill(C1,V,request)]))           :-
  obeys(C1,C2).

agent(A,O) :-
  not var(A), not var(O),
  on(O,[kill(A,_,_),steal(A,_,_),go(A,_,_),request(A,_,_)]).

% rules: motivated clauses

motivation(A,[kill(A,B,greed),(owns(B,O),not (A = B),carries_object(A,O))]) :-
  person(A), not on(A,['Marian','Cogsworth','Bertillon']).
motivation(A,[kill(A,B,jealousy),(loves(A,B),loves(B,C),gender(A,M),gender(C,M),
  not (A = B), not (A = C))]) :-
  person(A), not on(A,['Jane','Cogsworth','Bertillon']).
motivation(A,[kill(A,B,request),(obeys(A,C),not               (B               =
  C),complied(A,[C,kill(A,B,request)]))]) :-
  A = 'Jane'.
motivation(A,[kill(A,B,vengeance),(loves(A,C),not   (A   =   B),   not   (A   =   C),
  killed(B,C), not (B = C))]) :-
```

```
  person(A), not on(A,['Jane','Bertillon']).

motivation(A,[kill(A,B,'self-defense'),(attacked(B,A), not (A = B))]) :-
  A = 'Marian'.
motivation(A,[kill(A,A,lovesickness),(loves(A,B),not loves(B,A),not (A = B))]) :-
  person(A),not on(A,['Jane','Cogsworth','Bertillon']).

% rules: will_attack

will_attack('Jane',C,sensed('Jane',hair_colour(C,red))).
will_attack(A,B,P) :-
  motivation(A,[kill(A,B,M),P]),not (M = 'self-defense').

% rules: inf_rules

inf_rule(A,killed(X,B) => dead(B,true)) :- A = 'Bertillon'.

inf_rule(A,(killed(X,Y),F) => motive(X,[kill(X,Y),M])) :-
  motivation(X,[kill(X,Y,M),F]), not (X == Y), A = 'Bertillon'.
inf_rule(A,killed(X,X) => motive(X,[kill(X,X),lovesickness])).
inf_rule(A,(daltonic(B,true),told(B,[A,hair_colour(P,C1)]),map_dalt(C1,C2),    not
  (C1 == C2)) =>
  hair_colour(P,C2)).
inf_rule(C,(watched(C,go(A,L1,L2)) => (not (current_place(A,L1))))).
inf_rule(C,(watched(C,go(A,L1,L2)) => (current_place(A,L2)))).

% rules: will_request

will_request(C1,C2,kill(C2,C3,request),(obeys(C1,C2),M)) :-
  C1 == 'Jane', C2 == 'Patrick', C3 == 'Marian', will_attack(C1,C3,M).
will_request(C1,C2,kill(C2,V,request),(obeys(C2,C1),P))                          :-
  motivation(C1,[kill(C1,V,M),P]).
will_request(C1,C2,go(C2,L1,L2),obeys(C2,C1)).
will_request(C1,C2,steal(C2,O,C3),obeys(C2,C1)).
will_request('Marian','Cogsworth',go('Cogsworth',L1,L2),true).

% rules: will_comply

will_comply(X,Y,F,true) :- obeys(X,Y).
will_comply(C1,C2,kill(C1,C3,request),(obeys(C2,C1),owns(C3,O),complied(C2,[C1,st
  eal(C2,O,C3)]))) :-
  C2 == 'Jane', C1 == 'Patrick', C3 == 'Marian'.
will_comply('Cogsworth','Marian',go('Cogsworth',L1,L2),true).

% rules: will_refuse

will_refuse('Cogsworth',X,F,(not (X = 'Marian'))).


/* LIBRARY OF PLOT-PATTERNS */

lib([
 (U/A/[A,' deceived ',B,', possibly with criminal intent']/
 (loves(B,C),told(A,[B,not                              loves(C,B)]),believes(B,not
  loves(C,B)),told(A,[B,loves(C,D)]),believes(B,loves(C,D)),
  killed(B,C),killed(B,B))/
 (start=>tell(A,B,not                                    loves(C,B))=>agree(B,A,not
  loves(C,B))=>kill(B,C,jealousy)=>kill(B,B,M))/
 (/told(A,[B,not
  loves(C,B)]),trusts(U,X),asked(U,[X,loves(C,B)]),told(X,[U,loves(C,B)]),agreed(
  U,[X,loves(C,B)]))),
```

```
 (U/A/[A,' may have used ',B,'''s theft as cover-up, but the suspect is ',A,'
  possibly moved by jealousy']/
 (obeys(B,A),person(D),not    D    =    A,    not    D    =    B,current_place(D,L),
  current_place(B,L),
  requested(A,[B,steal(B, O, V)]),watched(D,request(A,B,steal(B, O, V))),
  stolen(B, [V,O]),killed(A,V))/
 (start=>request(A, B, steal(B, O, V))=>steal(B, O, V)=>kill(A, V, jealousy))/
 (sensed(U,loves(A,V)),questioned(U,[D,request(A,B,steal(B,                    O,
  V))]),related(D,[U,request(A,B,steal(B, O, V))]))),

 (U/default/['The culprit is the butler!']/
 (true)/
 (start)/
 (true))
  ]).


/* templates for facts specific of the domain */

f_template(dead(X,Y),
  [pos: [X,' is dead'],
   i_pos: ['I am dead'],
   y_pos: ['You are dead'],
   neg: [X,' is not dead'],
   i_neg: ['I am not dead',Y],
   y_neg: ['You are not dead',Y],
   int: ['Is ',X,' dead'],
   i_int: ['Am I dead'],
   y_int: ['Are you dead'],
   int_v: ['Is ',X,'dead or not'],
   i_int_v: ['Am I dead or not'],
   y_int_v: ['Are you dead or not'],
   int_who: ['Who is dead']]).

f_template(killed(X,Y),
  [pos: [X,' committed suicide'],
   neg: [X,' did not commit suicide']]) :-
     not var(X), not var(Y), X = Y, !.

f_template(attacked(X,Y),
  [pos: [X,' inflicted self punishment'],
   neg: [X,' did not inflict self punishment']]) :-
     not var(X), not var(Y), X = Y, !.

f_template(killed(X,Y),
  [pos: [X,' killed ',Y],
   i_pos: ['I killed ',Y],
   y_pos: ['You killed ',Y],
   neg: [X,' did not kill ',Y],
   i_neg: ['I did not kill ',Y],
   y_neg: ['You did not kill ',Y],
   int: ['Has ',X,' killed ',Y],
   i_int: ['Have I killed ',Y],
   y_int: ['Have you killed ',Y],
   int_v: ['Who was killed by ',X],
   i_int_v: ['Whom did I kill'],
   y_int_v: ['Whom did you kill'],
   int_who: ['Who killed ',Y]]).

f_template(attacked(X,Y),
  [pos: [X,' attacked ',Y],
   i_pos: ['I attacked ',Y],
   y_pos: ['You attacked ',Y],
```

```
   neg: [X,' did not attack ',Y],
   i_neg: ['I did not attack ',Y],
   y_neg: ['You did not attack ',Y],
   int: ['Has ',X,' attacked ',Y],
   i_int: ['Have I attacked ',Y],
   y_int: ['Have you attacked ',Y],
   int_v: ['Who was attacked by ',X],
   i_int_v: ['Whom did I attack'],
   y_int_v: ['Whom did you attack'],
   int_who: ['Who attacked ',Y]]).

f_template(city(X),
  [pos: [X,' is a city'],
   neg: [X,' is not a city'],
   int: ['Is ',X,' a city'],
   int_v: ['What city is there']]).

f_template(object(X),
  [pos: [X,' is an object'],
   neg: [X,' is not a object'],
   int: ['Is ',X,' a object'],
   int_v: ['What object is there']]).

f_template(gender(X,Y) ,
  [pos: [X,' is ',Y],
   i_pos: ['I am ',Y],
   y_pos: ['You are ',Y],
   neg: ['You are not ',Y],
   i_neg: ['I am not ',Y],
   y_neg: ['You are not ',Y],
   int: ['Is ',X,' ',Y],
   i_int: ['Am I ',Y],
   y_int: ['Are you ',Y],
   int_v: ['Is ',X,' male or female'],
   i_int_v: ['What am I, male or female'],
   y_int_v: ['What are you, male or female'],
   int_who: ['Who is ',Y]]).

f_template(hair_colour(X,Y),
  [pos: [X,' has ',Y,' hair'],
   i_pos: ['My hair is ',Y],
   y_pos: ['Your hair is ',Y],
   neg: [X,' has not ',Y,' hair'],
   i_neg: ['My hair is not ',Y],
   y_neg: ['Your hair is not ',Y],
   int: ['Has ',X,' ',Y,' hair'],
   i_int: ['Have I ',Y,' hair'],
   y_int: ['Is your hair ',Y],
   int_v: ['What is the colour of the hair of ',X],
   i_int_v: ['What is the colour of my hair'],
   y_int_v: ['What is the colour of your hair'],
   int_who: ['Who has ',Y,' hair']]).

f_template(daltonic(X,Y),
  [pos: [X,' is daltonic'],
   i_pos: ['I am daltonic'],
   y_pos: ['You are daltonic'],
   neg: [X,' is not daltonic'],
   i_neg: ['I am not daltonic',Y],
   y_neg: ['You are not daltonic',Y],
   int: ['Is ',X,' daltonic'],
   i_int: ['Am I daltonic'],
   y_int: ['Are you daltonic'],
```

```
   int_v: ['Is ',X,'daltonic or not'],
   i_int_v: ['Am I daltonic or not'],
   y_int_v: ['Are you daltonic or not'],
   int_who: ['Who is daltonic']]).

f_template(home(X,Y),
  [pos: [X,' lives in ',Y],
   i_pos: ['I live in ',Y],
   y_pos: ['You live in ',Y],
   neg: [X,' does not live in ',Y],
   i_neg: ['I do not live in ',Y],
   y_neg: ['You do not live in ',Y],
   int: ['Does ',X,' live in ',Y],
   i_int: ['Do I live in ',Y],
   y_int: ['Do you live in ',Y],
   int_v: ['Where does ',X,' live'],
   i_int_v: ['Where do I live'],
   y_int_v: ['Where do you live'],
   int_who: ['Who lives in ',Y]]).

f_template(loves(X,Y),
  [pos: [X,' loves ',Y],
   i_pos: ['I love ',Y],
   y_pos: ['You love ',Y],
   neg: [X,' does not love ',Y,' now'],
   i_neg: ['I do not love ',Y,' now'],
   y_neg: ['You do not love ',Y,' now'],
   int: ['Is ',X,' in love with ',Y],
   i_int: ['Am I in love with ',Y],
   y_int: ['Are you in love with ',Y],
   int_v: ['Whom does ',X,' love' ],
   i_int_v: ['Whom do I love'],
   y_int_v: ['Whom do you love'],
   int_who: ['Who is in love with ',Y]]).

f_template(obeys(X,Y),[pos: [X,' obeys ',Y]]).

f_template(owns(X,Y),
  [pos: [X,' has a ',Y],
   i_pos: ['This ',Y,' is mine'],
   y_pos: ['This ',Y,' is yours'],
   neg: ['This ',Y,' is not yours'],
   i_neg: ['I do not own this ',Y],
   y_neg: ['You do not own this ',Y],
   int: ['Does ',X,' own this ',Y],
   i_int: ['Do I own this ',Y],
   y_int: ['Do you own this ',Y],
   int_v: ['What does ',X,' own' ],
   i_int_v: ['What do I own'],
   y_int_v: ['What do you own'],
   int_who: ['Who owns this ',Y]]).

f_template(carries_object(X,Y),
  [pos: [X,' carries a ',Y],
   i_pos: ['I am carrying a ',Y],
   y_pos: ['You are carrying a ',Y],
   neg: [X,' does not carry a ',Y],
   i_neg: ['I am not carrying a ',Y],
   y_neg: ['You are not carrying a ',Y],
   int: ['Does ',X,' carry a ',Y],
   i_int: ['Do I carry a ',Y],
   y_int: ['Do you carry a ',Y],
   int_v: ['What does ',X,' carry' ],
```

```
   i_int_v: ['What do I carry'],
   y_int_v: ['What do you carry'],
   int_who: ['Who carries this ',Y]]).

f_template(motive(X,EM),
  [pos: [X,', in the event ''',Tc,''', was motivated by ',M],
   neg: [X,', in the event ''',Tc,''', was not motivated by ',M],
   int: ['Was ',X,' motivated by ',M,' in the event ''',Tc,''''],
   int_v: ['What motivated ',X,' in the event ''',Tc,''''],
   int_who: ['Who is motivated by ',M,' in the event ''',Tc,''''']]) :-
    not var(EM),
    EM = [E,M],
    template(E,T),
  xclist(T,Tc).

f_template(stolen(X,[Y,O]),
  [pos: [X,' stole ',A,O,' from ',Y],
   i_pos: ['I stole ',A,O,' from ',Y],
   y_pos: ['You stole ',A,O,' from ',Y],
   int: ['Did ',X,' steal ',A,O,' from ',Y],
   neg: [X,' did not steal ',A,O,' from ',Y],
   i_pos: ['I did not steal ',A,O,' from ',Y],
   y_pos: ['You did not steal ',O,' from ',Y],
   int: ['Did ',X,' steal ',A,O,' from ',Y],
   i_int: ['Did I steal ',A,O,' from ',Y],
   y_int: ['Did you steal ',A,O,' from ',Y],
   int_v: ['From whom did ',X,' steal ',A,O],
   i_int_v: ['From whom did I steal ',A,O],
   y_int_v: ['From whom did you steal ',A,O],
   int_who: ['Who has stolen ',A,O,' from ',Y]]) :-
      (not var(O),A = 'a ';
       var(O), O = 'something (what was it?)', A = '').


/* INITIAL STATE  */

person('Cogsworth').
person('Marian').
person('Bertillon').
person('Jane').
person('Patrick').
person('Robin').
city('London').
city('Manchester').
object('jewel').
gender('Marian','female').
gender('Jane','female').
gender('Cogsworth','male').
gender('Patrick','male').
gender('Robin','male').
gender('Bertillon','male').
current_place('Cogsworth','London').
current_place('Bertillon','London').
current_place('Marian','London').
current_place('Robin','London').
current_place('Patrick','London').
current_place('Jane','Manchester').
daltonic('Robin',true).
hair_colour('Marian',red).
hair_colour('Jane',blond).
owns('Marian','jewel').
carries_object('Marian','jewel').
loves('Patrick','Marian').
```

```
loves('Marian','Robin').
loves('Robin','Marian').

trusts('Bertillon','Cogsworth').
trusts('Marian','Cogsworth').
trusts('Robin','Cogsworth').
trusts('Marian','Jane').

knows_effs('Bertillon',[kill(X,Y,M),steal(X,Y,T),
                        request(X,Y,E),attack(X,Y),
             go(X,L1,L2),tell(X,Y,F)]).

main_eff(C,kill(X,Y,M), killed(X,Y)).
main_eff(C,steal(X,T,Y), stolen(X,[Y,T])).
main_eff(C,attack(X,Y), attacked(X,Y)).

obeys('Jane','Patrick').

believes('Cogsworth',F) :- property(F), F.
believes('Cogsworth',F) :- not var(F), F = (not Fn), property(Fn), F.
believes(X,F) :- person(X), not (X == 'Cogsworth'), fact(F), property(F,_,X),F.
believes('Bertillon',gender(X,Y)) :- believes('Cogsworth',gender(X,Y)).
believes('Patrick',gender(X,Y)) :- believes('Cogsworth',gender(X,Y)), not (X =
  'Patrick').


/* BERTILLON'S SEVEN EARLY CASES */

test(I,J) :-
  nl,
  forall((between(I,J,N),xconc(ex,N,Exi)),
    (nl,write('====== '),write('Example '),write(N),write(' ======'),nl,Exi)).

test(N) :- test(N,N).

ex1 :-
  plans((motive('Patrick',[kill('Patrick','Marian'),jealousy]),
         watched('Cogsworth',kill('Patrick','Marian',M)),
         related('Cogsworth',['Bertillon',kill('Patrick','Marian',M)]),
    agreed_op('Bertillon',['Cogsworth',kill('Patrick','Marian',M)]),
         agreed('Bertillon',['Cogsworth',loves('Patrick','Marian')]),
    agreed('Bertillon',['Cogsworth',loves('Marian','Robin')]),
         inferred('Bertillon',motive(S,[kill(S,'Marian'),M])),
    exposed('Bertillon',[S,'Marian',jealousy,nil])
        ),Plan),narrate(Plan),nl,nl, !.

ex2 :-
  plans((sensed('Patrick',owns('Marian',jewel)),
         stolen('Patrick',['Marian',jewel]),
         motive('Patrick',[kill('Patrick','Marian'),greed]),
         watched('Cogsworth',kill('Patrick','Marian',M)),
    sensed('Cogsworth',carries_object('Patrick',jewel)),
         related('Cogsworth',['Bertillon',kill('Patrick','Marian',M)]),
    agreed_op('Bertillon',['Cogsworth',kill('Patrick','Marian',M)]),
         agreed('Bertillon',['Cogsworth',owns('Marian',jewel)]),
    believes('Bertillon',motive('Patrick',[kill('Patrick','Marian'),greed])),
         supposed('Bertillon',carries_object('Patrick',jewel)),
    asked('Bertillon',['Cogsworth',carries_object('Patrick',jewel)]),
    agreed('Bertillon',['Cogsworth',carries_object('Patrick',jewel)]),
         inferred('Bertillon',motive('Patrick',[kill(S,'Marian'),M])),
    exposed('Bertillon',[S,'Marian',greed,nil])
        ),Plan),narrate(Plan),nl,nl, !.
```

```
ex3 :-
  plans((current_place('Marian','Manchester'),
         current_place('Cogsworth','Manchester'),
    current_place('Bertillon','Manchester'),
         complied('Jane',['Patrick',kill('Jane','Marian',request)]),
    watched('Cogsworth',request('Patrick','Jane',kill('Jane','Marian',request))),
         killed('Jane','Marian'),
         watched('Cogsworth',kill('Jane','Marian',M)),
    related('Jane',['Patrick',kill('Jane','Marian',M)]),
         related('Cogsworth',['Bertillon',kill('Jane','Marian',M)]),
    agreed_op('Bertillon',['Cogsworth',kill('Jane','Marian',M)]),

  related('Cogsworth',['Bertillon',request('Patrick','Jane',kill('Jane','Marian',
  M))]),

  agreed_op('Bertillon',['Cogsworth',request('Patrick','Jane',kill('Jane','Marian
  ',M))]),
         inferred('Bertillon',motive('Jane',[kill(S,'Marian'),M])),
    believes('Bertillon',requested(I,['Jane',kill('Jane','Marian',request)])),
    exposed('Bertillon',[S,'Marian',request,I])
       ),Plan),narrate(Plan),nl,nl, !.

ex4 :-
  plans((attacked('Jane','Marian'),
         watched('Cogsworth',attack('Jane','Marian')),
    motive('Marian',[kill('Marian','Jane'),'self-defense']),
    watched('Cogsworth',kill('Marian','Jane',M)),
         related('Cogsworth',['Bertillon',attack('Jane','Marian')]),
    agreed_op('Bertillon',['Cogsworth',attack('Jane','Marian')]),
    related('Cogsworth',['Bertillon',kill('Marian','Jane',M)]),
    agreed_op('Bertillon',['Cogsworth',kill('Marian','Jane',M)]),
         inferred('Bertillon',motive('Marian',[kill(S,'Jane'),M])),
    exposed('Bertillon',[S,'Jane','self-defense',nil])
       ),Plan),narrate(Plan),nl,nl, !.

ex5 :-
  plans((killed('Patrick','Marian'),
         watched('Cogsworth',kill('Patrick','Marian',_)),
    watched('Robin',kill('Patrick','Marian',_)),
         killed('Robin','Patrick'),
         related('Cogsworth',['Bertillon',kill('Robin','Patrick',M)]),
    agreed_op('Bertillon',['Cogsworth',kill('Robin','Patrick',M)]),
         inferred('Bertillon',motive('Robin',[kill(S,'Patrick'),M])),
    believes('Bertillon',killed('Patrick',V)),
    exposed('Bertillon',[S,'Patrick',vengeance,V])
       ),Plan),narrate(Plan),nl,nl, !.

ex6 :-
  plans((sensed('Patrick',loves('Marian',L)),
         current_place('Jane','London'),
       complied('Patrick',['Jane',kill('Patrick','Marian',request)]),
    watched('Cogsworth',request('Patrick','Jane',steal('Jane',jewel,'Marian'))),
         stolen('Jane',['Marian',jewel]),
    watched('Cogsworth',steal('Jane',jewel,'Marian')),
       motive('Patrick',[kill('Patrick','Marian'),jealousy]),
    sensed('Bertillon',dead('Marian',true)),
    supposed('Bertillon',killed(X,'Marian')),
    related('Cogsworth',['Bertillon',steal('Jane',jewel,'Marian')]),
    obs('Bertillon',Obs),
    recognized('Bertillon','Bertillon'/Ag/Cr_type/Goals/Pl_lib/Q),
    tried('Bertillon',Q),
    exposed('Bertillon',['Patrick','Marian',lib,Cr_type])
       ),P),
```

```
  narrate(P),nl,nl,!.

ex7 :-
  plans((told('Jane',['Marian',not loves('Robin','Marian')]),
         watched('Cogsworth',tell('Jane','Marian',not loves('Robin','Marian'))),
         killed('Marian','Marian'),
    watched('Cogsworth',kill('Marian','Marian',_)),
         related('Cogsworth',['Bertillon',tell('Jane','Marian',not F)]),
    agreed_op('Bertillon',['Cogsworth',tell('Jane','Marian',not F)]),
    related('Cogsworth',['Bertillon',kill('Marian','Marian',M)]),
    agreed_op('Bertillon',['Cogsworth',kill('Marian','Marian',M)]),
    inferred('Bertillon',motive('Marian',[kill(S,'Marian'),M])),
    exposed('Bertillon',[S,S,M,nil]),
    obs('Bertillon',Obs),
    recognized('Bertillon','Bertillon'/Ag/Cr_type/Goals/Pl_lib/Q),
    tried('Bertillon',Q),
    exposed('Bertillon',['Jane','Marian',lib,Cr_type])
   ),Plan),narrate(Plan),nl,nl, !.
```