Serious computing with tense

Nauze, F; van Lambalgen, M.

Publication date
2004

Published in
Computational Semantics ; 3

Serious computing with tense *

Fabrice Nauze and Michiel van Lambalgen
Department of Philosophy, University of Amsterdam
Nieuwe Doelenstraat 15, 1012 CP Amsterdam
F.D.Nauze@uva.nl, M.vanLambalgen@uva.nl

1. Introduction

In this paper we present a novel approach to the formal semantics of the French tense system. More precisely, we give a synopsis of the computational theory of meaning developed in [13], [12] and the forthcoming book [14], and apply it to two French tenses, Passé Simple and Imparfait. Much work has been done on French tenses within the framework of DRT or extensions thereof such as SDRT. The latter uses so-called rhetorical relations such as elaboration, to explain the peculiar ways in which events described by sentences in Passé Simple form can be ordered in time. It is claimed here that a much more insightful description can be obtained by taking a computational point of view, in which the meaning of an expression corresponds to an algorithm which computes its denotation in a given context. 'Algorithm' is taken very seriously here – it is the precise form of the algorithm (constraint logic programming) that is important. A cognitive justification for this particular choice is provided in [14]; here we can only hope to convince the reader by examples of the algorithm in action.

2. Data

In this section we provide some data pertinent to Passé Simple and the Imparfait. We begin with a discussion of the Passé Simple, and continue with examples of the interplay between Imparfait and Passé Simple.

* A condensed form of this paper appeared as Chapter 10, 'Tense in French: Passé Simple and Imparfait', in van Lambalgen and Hamm [14].

2.1. Examples involving the Passé Simple

We will start our discussion with a typical example of a narrative discourse with the PS where the events described are in temporal succession:

(1) Pierre se leva, monta dans sa chambre, ferma la porte et alluma la radio. (4×PS)

What can be said about the role of the PS in this example? Obviously, the PS conveys the information that all events are located in the past. More interestingly, it is often claimed that these events are to be viewed as punctual in the sense that there are no other events which could partition them. The internal constitution of the events is not important; this means that the PS views events as perfective. The PS imposes a view of the events 'from the outside' and from a distance. This is then claimed to explain why multiple uses of the PS implies a succession of the events described. As the events are seen as punctual, irreducible and viewed from the outside, it is then natural to expect that two events in the PS are not simultaneous, and so that one is happening before the other. Then the obvious choice is to place first things first (unless explicitly stated otherwise). Hence in (1), the getting up of Pierre precedes his going up in his room, etc... This is why the PS is often considered to imply narrative succession.

Let us try to describe the above in a more formal manner. The most evident effect of the PS is to place the eventuality in the past of the speech time (this is what is known as "pure" tense information). We have now two options to account for the succession effect. We may assume, as in early versions of DRT, that the PS introduces a new reference point placed after an old one (this would amount to a direct representation of the "succession effect" of the PS). Alternatively, we may posit that the PS represents the eventuality as perfective and located in the past, and derive the succession effect from this, whenever it is appropriate.

We will choose the latter option, as it seems to be a better representation of the core meaning of the PS, succession being in our view only a (albeit quite frequent) side-effect. In fact, a good counter-example to the unconditional validity of the succession effect of the PS was given by Kamp and Rohrer, here slightly changed to

(2) L'été de cette année-là vit plusieurs changements dans la vie de nos héros. François épousa Adèle, Jean partit pour le Brésil et Paul s'acheta une maison à la campagne. (4×PS)

The first sentence introduces an event which gets divided in the following sentence (this phenomenon is known as the rhetorical relation of elaboration; it can also be viewed as a change of granularity in the description of events). How this first event is divided cannot be determined from those PS sentences alone. In a way the first sentence 'asks' for an enumeration afterwards, and so the next verb phrases enumerate the list of changes in the life of the 'heroes', but in the absence of adverbs or ordering conjunctions (like *puis*) we cannot give the precise temporal relationship between those events. Hence we have here two phenomena: the first sentence gets divided by others (in a way this could be seen as contradicting the perfectivity of the PS), and furthermore the following PS sentences do not impose a natural ordering on the events described by them. One of the causes of this lack of ordering is that the VPs have different subjects: *François*, *Jean* and *Paul*. We can reformulate example (2) by removing one of the subjects as in

(3) L'été de cette année-là vit plusieurs changements dans la vie de nos héros. François épousa Adèle et partit pour le Brésil, Paul s'acheta une maison à la campagne. (4×PS)

In sentence (3) we have now a succession of two events *François marrying Adèle* and then leaving to *Brazil*. However we still cannot derive any ordering of those two VPs with the third. We should also note that the inverse temporal order seems to be called for in the following example of Gosselin [4, p.117]

(4) Pierre brisa le vase. Il le laissa tomber. (PS ×2)

Even without the use of an explanative conjunction like *car*, it seems we can derive the explanation reading, and this for two reasons: first, the achievement of the first sentence is irreversible in the way that the object of the sentence is changed for good after the achievement (*briser*), second, the anaphoric pronoun *le* in the second sentence refers to the the vase, not to the broken vase which is the result of the first sentence, hence we expect that the second sentence applies to the not-yet-broken vase. We can further notice that the first sentence presupposes an action on the part of the subject Pierre on the vase (directly or indirectly), which causes the breaking. Furthermore, the subjects of the two sentences agree, and the pronoun of the second sentence refers to the object of the first sentence; and obviously to drop something is a plausible cause of breaking this same thing. It then seems natural that the second sentence actually describes the action

that leads to the breaking of the vase.[1] It should also be noticed that the fact that the two sentences in (4) are separated by a period is of major importance. If it is not the case, as in

(5) Pierre brisa le vase et le laissa tomber. (2×PS)

there is no ambiguity about the ordering of the two events described by the sentences: the breaking happens before the falling. Furthermore, even when the sentences are separated by a period we may get the ordering expressed by (5). If we add a further sentence, as in

(6) a. Pierre brisa le vase avec un marteau. Il le laissa tomber et s'en alla.
b. Pierre brisa le vase avec un marteau. Il le laissa tomber. Il s'en alla sans le regarder.

the narrative seems to force the events to be ordered corresponding to the sentences.

Let us now change the examples (1), (2) and (4) somewhat, to determine when and why narration occurs or on the contrary breaks down. In example (1) we have a simple succession of events affecting one subject, in (2) we have several events affecting different subjects and occurring in a certain period of time but not explicitly ordered with respect to each other, and finally in (4) we have two events affecting one subject and one object in inverse temporal order. Now consider the following variations.

(7) a. Pierre monta dans sa chambre et ferma la porte. (2×PS)
b. Pierre ferma la porte et monta dans sa chambre. (2×PS)
c. Pierre ferma la porte et Jean monta dans sa chambre. (2×PS)
d. # Pierre monta dans sa chambre, ferma la porte, alluma la radio et se leva. (4×PS)
e. Cet été-là, François épousa Adèle, Jean partit pour le Brésil et Paul s'acheta une maison à la campagne. (3×PS)
f. Cet été-là, François épousa Adèle et partit pour le Brésil et Paul s'acheta une maison à la campagne. (3×PS)

Examples (7-a) and (7-b) describe a succession of two events accomplished by a single subject: monter dans sa chambre (go upstairs in his room) and fermer la porte (close the door). In example (7-a) Pierre goes first in his room and then closes the door whereas in (7-b) he first closes the door and then goes in his room. As those eventualities are

[1] We have provided such an extensive discussion of example (4) because there appears to be a general agreement in the literature on the impossibility of the PS to give an inverse temporal reading.

seen as perfective (this is the aspectual effect of the PS), are ascribed to one subject (this is a syntactic property of the sentence) and can hardly be done simultaneously (this is part of the semantics of those eventualities), the only possibility is that those two events are consecutive. However, the claim that the PS *implies* succession must be revised. All we get is that in a discourse in which the PS describes eventualities which have few semantic connections (note that going upstairs doesn't presuppose closing the door and vice-versa) and in which there is a unique subject, the order of the events is isomorphic to the utterance order. What is heard (or read) first, happens first.

Here are some more examples to show that the two factors identified, semantic connections and uniqueness of subject, indeed influence the reading of a piece of discourse. The importance of uniqueness of subject can be seen in examples (7-c), (7-e) and (7-f). The only difference between (7-b) and (7-c) is that in the latter the second VP has a different subject than the first. The correct reading of this sentence is probably that of a succession but the possibility of simultaneity is not excluded, as in (7-b). This sentence can describe the simultaneous actions of two subjects but would be inadequate to described the inverse order.

Examples (7-e) (a simplified version of (2)) and (7-f) differ in that François is now the subject of two events. Furthermore those two events are successive but still in no particular relation to the third event. In (7-e) all subjects differ and we have no special ordering between the events.

Sentence (7-d) isn't correct because Pierre going into his room and closing the door presupposes (semantically) that he remains standing.[2] Hence to determine the temporal relation of a new PS VP with respect to a given sequence of PS VPs, all having the same subject, the meaning of the new VP must be compared with the possible lexical (semantic) information conveyed by the preceding VPs.

The last example we will give involves aspectual information. The reader may have noticed that the VPS in the preceding examples are either accomplishments or achievements. The PS can also be used with states or activities, however.

(8) Il fut président. (PS)

[2] This is not the case for the VP 'switch the radio on'. Therefore the following sentence is correct.

(i) Pierre alluma la radio et se leva. (2×PS)

In this example we obtain an inchoative reading. This is made clear by giving the proper English translation: *He became president* (and not *He was president*). The stative VP is coerced by the PS into its initiating event.[3]

2.2. Examples involving the Imparfait

It is illustrative to begin this section by citing several comments on this tense from the literature. De Swart says in [2, p. 57],

> sentences in the Imparfait are traditionally taken to describe background information that does not move the story forward.

It follows Kamp's view which is motivated by the study of the tenses in narrative context and where the fact that the Imp doesn't move the narration forward is directly opposed to the fact that the PS does. Gosselin, in [4, p. 199], doesn't put the emphasis on moving the story line forward, but notices that

> the Imp refers to a moment in the past during which the process is going on, without precision about the situation of the beginning and the end of the process.[4]

Sten in [10] focusses on its use as "present in the past":

> L'imparfait sert à indiquer une action qui serait du présent pour un observateur du passé,...", (the Imp serves to indicate an action which would be present for an observator in the past).

Finally, all authors stress the anaphoric nature of this tense, in the sense that it cannot be used by itself but only with reference to another sentence or with temporal adverbials.[5] We may summarize these positions by saying that the Imparfait is an anaphoric, imperfective past tense. We will now introduce some examples of the use of the Imparfait, however the reader should notice that those examples partially represent the possibilities of the Imparfait. In particular we won't give examples of the so-called narrative Imparfait or habitual and iterative readings.

---

[3] Notice that the combination PS + stative VP does not logically imply an inchoative reading.

(i) Il fut président de 1981 à 1995. (PS)

Here, we do not obtain an inchoative reading but just a perfective eventuality.

[4] p. 199, [4]: "L'imparfait renvoie donc typiquement à un moment du passé pendant lequel le procès se déroule, sans préciser la situation temporelle du début et de la fin du procès. Ce temps apparaît non autonome (anaphorique) et situe le procès comme simultané par rapport à d'autres procès du contexte, et comme se déroulant en un même lieu.

[5] See for instance [6, p. 35].

The anaphoric and imperfective nature of the Imp can be seen in the following example

(9) a. # Il faisait chaud. (Imp)
b. Il faisait chaud. Jean ôta sa veste. (Imp, PS)

That sentence (9-a) is not felicitous is explained in Kamp's theory by the fact that there is no previous "reference point" to anchor the sentence and that an Imp sentence such as (9-a) does not introduce its own reference point.[6]In sentence (9-b), the Imp sentence is "attached" to the reference point introduced by the PS sentence and the imperfective aspect of the Imp is due to the fact that the PS event happens while the Imp eventuality holds. It is however not a general rule for the Imp, that the Imp eventuality contains its reference point, as is shown by the following examples.

(10) Jean appuya sur l'interrupteur. La lumière l'éblouissait. (PS, Imp)

(11) Jean attrapa une contravention. Il roulait trop vite. (PS, Imp)

The Imp sentence in (10) is viewed as a consequence of the PS sentence; clearly the light cannot blind Jean before he switched it on. De Swart, in [2, p. 59-61], maintains that the reference point for the Imp sentence is not the PS sentence, but rather its consequent state (the light is switched on). Then we would have simultaneity between the Imp sentence and its reference point. On de Swart's approach, the decision whether the Imp overlaps with the PS reference point or with its consequent state is made on the basis of *rhetorical relations* between the sentences; this theory is what is known as SDRT. De Swart calls this rhetorical relation *temporal implication* and she provides an analogous explanation for (11), introducing the rhetorical relation of *temporal presupposition*. In example (11) the Imp sentence is understood as being the cause of getting a ticket hence even though the Imp sentence is placed after the PS sentence the activity *driving too fast* takes place before getting a ticket.

We believe that in this area explanations of much greater generality are possible than those provided by SDRT. Below we present a fully

[6] Notice that the reference point does not have to be introduced by a PS sentence; it can also be a temporal adverbial, or even the subject of the sentence, as in the following examples

(i) a. Mercredi, il pleuvait. Jeudi, il faisait soleil. (Imp, Imp)
b. Le grand-père de Marie était noir. (Imp)

computational semantics for French tenses, built upon the computational mechanism of (constraint) logic programming. In this setup, rhetorical relations will turn out to be derived constructs, abstracting certain features of the computations.

3. Computational semantics for tense

Tense is concerned with the grammatical localization of events in time. However, this does not mean that the ontology required for tense is restricted to on the one hand a time line, such as the reals, and on the other hand a set of discrete, structureless entities called events. The ontology must be very much richer, as has been put forcefully by Mark Steedman [9, p. 932]

> The semantics of tense and aspect is profoundly shaped by concerns with goals, actions and consequences ... temporality in the narrow sense of the term is merely one facet of this system among many.

The system of English future tenses provides a good illustration for this point of view: present tense in its future use, the futurate progressive, and the auxiliaries *will* and *be going to* all express different relations between goals, plans, and the actions comprising those plans. The case of the English future tense will be explained in detail in the forthcoming book [14]; here we concentrate on the French past tenses. We claim that also in this case a rich ontology involving goals, plans and actions is necessary to capture the data. As our basic representational format we choose a formalism, the *event calculus*, that was developed in robotics for the purpose of autonomous planning in robots. The book [14] contains a lengthy defense of the idea that human understanding of time is conditioned by human planning procedures. Indirectly one may then also expect that the grammatical correlate of that understanding is conditioned by planning. The next step is then, to try to model the semantics of tense by means of a planning formalism.

3.1. A calculus of events

By definition, planning means setting a goal and computing a sequence of actions which provably suffice to attain that goal. It involves reasoning about events, both actions of the agent and events in the environment, and about properties of the agent and the environment, which may undergo change as a consequence of those events. A simple example is that of an agent who wants a light L to burn from time t_0 until t_1, and knows that there is a switch S serving L. The obvious plan is then to turn S at t_0, and to leave S alone until t_1. Even this

simple plan hides a number of problems. We required that a plan should be provably correct. On a classical reading, that would mean that the plan is sure to achieve the goal in every model of the premisses, here the description of the situation and its causal relationships. Among these models, there will be some containing non-intended events, such as light turning off spontaneously (i.e. without an accompanying turn of the switch), or a gremlin turning off the switch between t_0 and t_1. In fact it is impossible to enumerate all the things that may go wrong. No planning formalism is therefore likely to give 'provably correct plans' in the sense of classical logic. The most one can hope for is a plan that works to the best of one's knowledge.

The event calculus[7] is a formalism for planning that addresses some of these concerns. It axiomatizes the idea that all change must be due to a cause–spontaneous changes do not occur. It thus embodies one sense of the *common sense principle of inertia*: a property persists unless it is caused to change by an event. That is, if an action a does not affect a property F, then if F is true before doing a, it will be true after. Of course, the crucial issue in this intuitive idea concerns the notion of 'affect'. This refers to a kind of causal web which specifies the influences of actions on properties.The other difficulty with planning identified above, the possibility of unexpected events, can be treated either in the axiomatic system, or equivalently in the logic underlying the system. The solution of this difficulty is essentially to restrict the class of models of the axiomatic system to those models which are in a sense minimal: only those events happen which are required to happen by the axioms, and similarly only those causal influences obtain which are forced by the axioms. Our treatment of the event calculus will correspond to the division just outlined: we first discuss its formalization of causality, and then move on to introduce the class of its minimal models.

Formally, the event calculus requires a many-sorted first order logic with sorts for the following:

1. individual objects, such as humans, chairs, tables, ...
2. real numbers, to represent time and variable quantities
3. time-dependent properties, such as states and activities
4. variable quantities, such as position, degree of sadness, state of completion of a painting, ...

[7] The version used in this paper is modelled on that developed by Murray Shanahan, which in turn was based on work by Kowalski and Sergot. A good reference is the book [8].

5. event types, whose instantiations (i.e. tokens) mark the beginning and end of time-dependent properties.

A few comments on this list are in order. The predicates of the event calculus will be seen to have an explicit parameter for time. We have chosen to represent time by the real numbers, actually by the structure $(\mathbb{R}, <; +, \times, 0, 1)$. It is explained at length in [14] why this choice is justified as well as harmless, so we do not dwell on this topic here.

It may furthermore strike the reader that *properties* are reckoned to belong to the ontology of the event calculus, on a par with individual objects and time points. Usually properties correspond to predicates, hence objects of a different type than that of entities. But in the event calculus a property is an object which may itself fill an argument slot in a predicate. There are several reasons for this, one having to do with the notion of 'cause'. Consider one of the most complex classes of verbs, the *accomplishments*, of which examples are 'draw a circle', 'write a letter', 'cross the street'. Eventualities representing such verbs have an elaborate internal structure. On the one hand there is an activity taking place (draw, write, cross), on the other hand an 'object' is being 'constructed': the circle, the letter, or the path across the street. Dowty (in [3]) analyzes the progressivized accomplishment

(12) Mary is drawing a circle

as

(13) CAUSE[Mary draws something, a circle comes into existence].

That is, the sentence is decomposed into an *activity* ('Mary draws something') and a *partial, changing, object* ('circle'); it is furthermore asserted that the activity is the *cause* of the change. For Dowty, causality is a relation between propositions, and accordingly he tries, not entirely successfully, to give an account of causality in terms of possible world semantics. By contrast, the event calculus gives an analysis of causality which has its roots in physics, as a relation between events.

The event calculus actually formalizes two notions of cause, and their relation. The first notion of cause is concerned with instantaneous change, as when two balls collide. We are thus concerned with an event (type) *collision*, which for simplicity is assumed to occur instantaneously. An event *type* together with a time at which it occurs (or happens) will be referred to as an event *token*. We furthermore need time-dependent properties such as, for example, 'ball b has momentum m'. In the case at hand, the property 'ball 1 has momentum m and ball 2 has momentum 0' will be true until the time of collision t, after which 'ball 2 has momentum m and ball 1 has momentum 0' is true. Such

time-dependent properties are called *fluents*[8]. A fluent is a function which may contain variables for individuals and reals, and which is interpreted in a model as a set of time points. We now want to be able to say that fluents are initiated and terminated by events, and that a fluent was true at the beginning of time. If f is a variable over fluents, e a variable over events, and t a variable over time points, we may write the required predicates as

1. Initially(f)
2. Happens(e,t)
3. Initiates(e,f,t)
4. Terminates(e,f,t)

If events happen instantaneously, these predicates are to be interpreted in such a way, that if $\mathit{Happens}(e,t) \wedge \mathit{Initiates}(e,f,t)$, then f will begin to hold after (but not at) t; if $\mathit{Happens}(e,t) \wedge \mathit{Terminates}(e,f,t)$, then f will still hold at t.

The second notion of causality is more like change due to a force which exerts its influence continuously. The paradigmatic example here is the acceleration of an object due to the gravitational field, but other examples abound: pushing a cart, filling a bucket, drinking a glass of wine, writing a letter, ... As the reader can see from this list, continuous change is important in providing a semantics for accomplishments.

Continuous change requires its own special predicates, namely

5. Trajectory(f_1,t,f_2,d)
6. Releases(e,f,t)

In the *Trajectory* predicate, one should think of f_1 as a force, and of f_2 as a variable quantity which my change under the influence of the force. The predicate then expresses that if f_1 holds from t until $t+d$, then at $t+d$, f_2 holds. In applications, f_2 will have a real number as argument, and will be of the form $f(g(t+d))$ for some continuous function g.

The predicate *Releases* is necessary to reconcile the two notions of cause with each other. Cause as instantaneous change leads to one form of inertia: after the occurrence of the event marking the change, properties will not change value until the occurrence of the next event. This however conflicts with the intended notion of continuous change, where variable quantities may change their values without concomitant occurrences of events. The solution is to exempt, by means of

[8] The name is appropriated from Newton's treatise on the calculus, where all variables are assumed to depend implicitly on time.

the special predicate *Releases*, those properties which we want to vary continuously, from the inertia of the first form of causation.

The axioms will be seen to have the form: if there are no 'f-relevant' events between t_1 and t_2, then the truth value of f at t_1 is the same as that at t_2. We introduce two special predicates to formalize the notion of 'f-relevant' events. The first predicate expresses that there is a terminating or releasing event between t_1 and t_2; the second predicate expresses that there is an initiating or releasing event between t_1 and t_2.

7. Clipped(t_1,f,t_2)

Lastly, we need the 'truth predicate'

8. HoldsAt(f,t).

The intuitive meaning of $HoldsAt(f, t)$ is that the fluent f is true at time t. In the usual setup of the event calculus, such defining axioms for this truth predicate are lacking, and this can easily lead to contradictions. However, due to lack of space we cannot furnish the relevant truth theory and we ask the reader to simply assume that $HoldsAt(f, t)$ can indeed be forced to have the meaning 'the fluent f is true at time t'.

For language processing we need a lexicon, which can be thought of as associating a theory to each lexical item. We will show below that for the purposes of discussing tense it is very useful to formulate these theories in the language of the event calculus. In fact we claim that there is a profitable analogy between linguistics and robotics here. In order to derive predictions, e.g. on when a robot will reach its destination, one needs a theory describing the robot's situation, conveniently divided in axioms, holding for every situation, and a scenario, laying down properties of a particular situation; this latter theory corresponds to the lexicon. We first study the axioms.

3.2. The axiom system EC

The axioms of the event calculus given below are modified from [8], the difference being due to the fact that we prefer a logic programming approach, whereas Shanahan uses a technique for obtaining minimal models called circumscription. In the following, all variables are assumed to be universally quantified.

AXIOM 1. *Initially(f)* $\wedge$ $\neg$*Clipped(0, f, t)* $\rightarrow$ *HoldsAt(f, t)*

AXIOM 2. *Happens(e, t)* $\wedge$ *Initiates(e, f, t)* $\wedge$ *t*$<$*t'* $\wedge$ $\neg$*Clipped(t, f, t')* $\rightarrow$ *HoldsAt(f, t')*

AXIOM 3. *Happens(e, t)* $\wedge$ *Initiates(e,* f_1*, t)* $\wedge$ *t<t'* $\wedge$ *t'=t+d* $\wedge$ *Trajectory(*f_1*, t,* f_2*, d)* $\wedge$ $\neg$*Clipped(t,* f_1*, t')* $\rightarrow$ *HoldsAt(*f_2*, t')*

AXIOM 4. *Happens(e, s)* $\wedge$ *t<s<t'* $\wedge$ *(Terminates(e, f, s)* $\vee$ *Releases(e, f, s))* $\rightarrow$ *Clipped(t, f, t')*

The set of axioms of the event calculus will be abbreviated by EC. We add some explanatory comments on the axioms. The meaning of these axioms can be seen most clearly in the case of axiom 2. Suppose a fluent f is initiated at time $t_1 > 0$, and that no 'f-relevant' event occurs between t_1 and t_2. Here, 'f-relevant' is rendered formally by the predicate *Clipped*, whose meaning is given by axiom 4. Axiom 2 then says that f also holds at time t_2. The role of the *Releases* predicate is important here, because it provides the bridge between the two notions of causality. Axiom 2 really embodies the principle of inertia as it relates to the first notion of causality, instantaneous change: in the absence of relevant events, no changes occur. However, continuous change occurs due to a force, not an event, and hence absence of relevant events does not always entail absence of change. The *Releases* predicate then provides the required loophole.

The first axiom says that if a fluent holds at time 0 and no event has terminated or released it before time $t > 0$, it still holds at t. Axiom 3 is best explained by means of the example of filling a bucket with water. So let f_1 be instantiated by *filling*, and f_2 by *height*(x). If *filling* has been going on uninterruptedly from t until t', then for a certain x, *height(x)* will be true at t', the particular x being determined by the law of the process as exemplified by the *Trajectory*–predicate.

3.3. A model for EC

In the absence of further statements constraining the interpretation of the primitive predicates, a simple model for EC is obtained by taking the extensions of *Happens* and *Initially* to be empty. If we then set $\neg HoldsAt(f, t)$ for all f, t we obtain a model. However, this model is not very informative, and to facilitate the reader's comprehension of the axioms, we will sketch an intuitively appealing class of models of EC. It can in fact be proven that all models of EC of interest in this context are of the form to be presented. We have to specify the sorts of fluents and event types in such a way that EC holds automatically. We interpret fluents as sets of intervals of the form $[0, b]$ or $(a, b]$, where a is the instant at which an initiating event occurs, and b is the instant

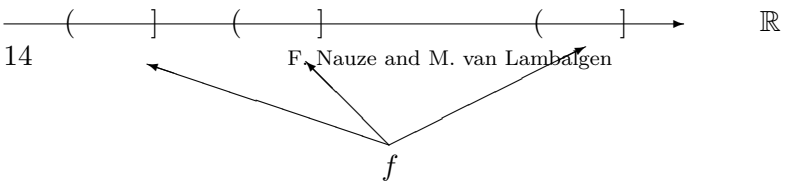

Figure 1. Structure of fluents

where 'the next' terminating event occurs[9]. Talk about 'the next' seems justified due to the inertia inherent in fluents. A typical fluent therefore looks as in figure 1.

For the purpose of constructing models, we think of event (types) as derivative of fluents, in the sense that each event either initiates or terminates a fluent, and that fluents are initiated or terminated by events only. The *instants* are taken to be nonnegative reals. Each fluent f is a finite set of disjoint halfopen intervals $(a, b]$, with the possible addition of an interval $[0, c]$. Event types e are of the form $e = e_f^+$ or $e = e_f^-$ where $e_f^+ := \{(f, r) \mid \exists s((r, s] \in f)\}$ and $e_f^- := \{(f, s) \mid \exists r((r, s] \in f)\}$.

This then yields the following interpretations for the distinguished predicates.

1. $\mathit{HoldsAt} := \{(f, t) \mid \exists I \in f(t \in I)\}$
2. $\mathit{Initially} := \{f \mid \exists s > 0[0, s] \in f)\}$
3. $\mathit{Happens} := \{(e, t) \mid \exists f((e = e_f^+ \vee e = e_f^-) \wedge (f, t) \in e)\}$
4. $\mathit{Initiates} := \{(e, f, t) \mid e = e_f^+ \wedge (f, t) \in e\}$
5. $\mathit{Terminates} := \{(e, f, t) \mid e = e_f^- \wedge (f, t) \in e\}$
6. $\mathit{Releases} := \emptyset$
7. $\mathit{Clipped} := \{(t_1, f, t_2) \mid \exists t(t_1 < t < t_2 \wedge (f, t) \in e_f^-)\}$

PROPOSITION 1. *EC is true under the above interpretation.*

The model captures an important intuition, namely that fluents can be represented by intervals, that is, very simple sets, as a consequence of 'the common sense law of inertia'. The reader is advised to have this interpretation in mind when interpreting statements in the formal language.

[9] Note that a fluent does not hold at the instant it is initiated, but does hold at the moment it is terminated. We need intervals $[0, b]$ to account for *Initially* statements. We allow b to be ∞.

3.4. Scenarios

The above axioms provide a general theory of causality. We also need 'micro-theories' which state the specific causal relationships holding in a given situation, and which list the events that have occurred in that situation. We claim that an important part of the lexicon can also be represented in this causal format. For example, in the case of 'draw a circle' the situation contains (at least) an activity ('draw') and a changing partial object (the circle in its various stages of completion); the micro-theory should specify how the activity 'draw' is causally related to the amount of circle constructed. This is done by means of two definitions, of *state* and *scenario*.

DEFINITION 1. *A* state $S(t)$ *at time* t *is a conjunction of*

1. *literals of the form* $(\neg)HoldsAt(f,t)$*, for* t *fixed and possibly different* f*.*
2. *(in)equalities between fluent terms, between event terms and between constants for individuals.*
3. *formulas in the language of the structure* $(\mathbb{R}, <; +, \times, 0, 1)$

DEFINITION 2. *A* scenario *is a conjunction of statements of the form*

1. $Initially(f)$,
2. $\forall t(S(t) \rightarrow Initiates(e,f,t))$,
3. $\forall t(S(t) \rightarrow Terminates(e,f,t))$,
4. $\forall t(S(t) \rightarrow Releases(e,f,t))$,
5. $\forall t(S(t) \rightarrow Happens(e,t))$,
6. $S(f_1,f_2,t,d) \rightarrow Trajectory(f_1,t,f_2,d)$.

where $S(t)$ *(more generally* $S(f_1,f_2,t,d)$*) is a state in the sense of definition 1.*

These formulas may contain additional constants for objects, reals or time points, and can be prefixed by universal quantifiers over reals (including time points) and objects. Formulas of type 6 are said to define a *dynamics*.

One final remark before we consider an example. The definition of 'state' refers only to *fluents* being true or false; it is not allowed to include conjuncts using *Happens*. This may seem strange for a formalism

concerned with causality; after all, the archetypical form of causality is one where $Happens(e_1, t)$ implies $Happens(e_2, s)$ for some s slightly later than t. There is a formal reason for our choice: allowing *Happens* in states increases the danger of nonterminating computations. In practice the restriction can be liberalized: one must only take care that no loops are introduced. In one example below we shall make use of the more liberal form; the reader can easily check that this causes no harm.

3.5. Minimal models

We now turn to an important feature in which the proposed computational semantics differs from, say, DRT. DRSs are always taken to be substructures of the 'real' world (or a world of fiction, as the case may be). By contrast, the models that we consider are 'closed worlds' in the sense that events which are not forced to occur by the scenario, are assumed not to occur. Later additions to the scenario may overturn this assumption, so that incremental processing of a discourse does not lead to a 'nice' chain of DRSs ordered by the substructure relation. Instead, we obtain a nonmonotonic progression. In [12] it is shown how the peculiar meaning of the progressive form in English can be explained in this way, and, also, that the ubiquitous phenomenon of coercion is a natural consequence of this computational model. This form of nonmonotonicity is easiest explained in terms of planning; we will latter return to its linguistic relevance.

Consider a problem with planning referred to above: it is impossible to construct a plan which is provably correct in the sense that it works whatever is true in the real world. We can only hope for plans which are correct with respect to the eventualities that are envisaged now, barring unforeseen circumstances. Formally, this means that we must restrict the class of models of event calculus and scenario to models which are *minimal* in the sense that the occurrences of events and their causal influences are restricted to what is required by the scenario and *EC*. Thus if a scenario contains only the following statements involving *Happens*

- Happens(switch-on, 5)
- Happens(switch-off, 10)

a non-minimal model of this scenario would be one in which the following events are interpolated between times 5 and 10:

- Happens(switch-off, 8)
- Happens(switch-on, 9)

Similarly, if the scenario contains only the following statements involving *Initially*, *Initiates* and *Terminates*

- ¬HoldsAt(light-on, t) → Initiates(switch-on, light-on, t)
- Terminates(switch-off, light-off, t)

a non-minimal model of this scenario could contain the additional statements concerning causal influences

- Initially(light-on)
- HoldsAt(light-on, t) → Terminates(switch-on, light-on, t)

While this intuition about minimality is fairly straightforward, its implementation is less so. Somewhat surprisingly, there exist essentially different ways of defining 'minimal model'. We favour the definition implicit in logic programming, because of its computational nature. It will be shown that in the cases of interest to us, there exists in fact a unique minimal model, which defines the denotations of all expressions occurring in the scenario. Moreover, there exists a computable procedure for obtaining the minimal model, so that the denotations are in fact computable.

3.6. Logic programming with constraints and negation as failure

This section introduces the computational machinery that we will use. Examples of actual computations will be given below. We have not worried about details of implementation, such as providing selection rules, but apart from this the computational mechanism is fully explicit.

Our favoured formalism, constraint logic programming, is in general concerned with the interplay of two languages. In our case these will be the languages $\mathcal{L} = \{0, 1, +, \cdot, <\}$, and the language $\mathcal{K}$ consisting of the primitive predicates of the event calculus. The latter will also be called *programmed* predicate symbols, because we will write logic programs defining the primitive predicates.

Not considering negation for the moment[10], clauses in a constraint logic program based on $\mathcal{L}$ and $\mathcal{K}$ are generally of the following form

$$B_1, \ldots, B_n, c \rightarrow A,$$

where the $B_1, \ldots, B_n, A$ are primitive predicates and c is a constraint. Constraints may occur only in the bodies of clauses. Likewise, a query has the logical form

$$B_1 \wedge \ldots \wedge B_m \wedge c \rightarrow \bot.$$

[10] I.e. restricting attention to so called *definite* constraint logic programs.

We shall use the notation

$$?c, B_1, \ldots, B_m$$

for queries, always with the convention that c denotes the constraint, and that the remaining formulas come from $\mathcal{K}$. The words 'query' and 'goal' will be used interchangeably.

The aim of a constraint computation is to express a programmed predicate symbol entirely in terms of constraints, or at least to find an assignment to the variables in the programmed predicate which satisfies a given constraint. Thus, unlike the case of ordinary logic programming, the last node of a successful branch in a derivation tree contains a constraint instead of the empty clause. To make this precise, we have to spell out the notion of derivation step and derivation tree.

One difference between standard logic programming and constraint logic programming is its treatment of substitution. In the former, the unification algorithm, applied to two atoms, determines which terms have to be substituted for the variables occurring in the atoms, in order for the atoms to become identical. In constraint logic programming the treatment is different: when the unification algorithm has determined that a term t should be substituted for a given variable x, one adds a *constraint* $t = x$ but no substitution is effected. If A, B are atoms, we let $\{A = B\}$ denote the set of equations between terms which unify A and B if A and B are unifiable; otherwise $\{A = B\}$ is set to $\bot$. The constraints are then simply accumulated in the course of the derivation. There are some clear notational advantages to this approach, which avoids nested, possibly unreadable terms[11]. The main advantage is conceptual, however, since it allows a more symmetric treatment of positive and negative information.

The main derivation rule is *resolution*, which can be formalized as follows. Suppose $?c, B_1, \ldots, B_i, \ldots, B_m$ is a goal, and $D_1, \ldots, D_k, c' \rightarrow A$ a program clause. A new goal

$$?c'', B_1, \ldots, D_1, \ldots, D_k, \ldots, B_m$$

can be derived from these two clauses if the constraint c'', defined as c'' $= (c \wedge \{B_i = A\} \wedge c')$ is satisfiable in $\mathcal{A}$. That is, if A can be unified with B_i, one can replace B_i by $D_1, \ldots, D_k$ if in addition the given constraint c is narrowed down to contain also the unifying substitution and the

[11] A consequence of this approach may seem that the constraint language $\mathcal{L}$ has to be extended, since constraints in the wider sense may now also involve objects, (parametrized) events and (parametrized) fluents. However, all such syntactic objects can be coded into $\mathcal{L}$.

constraint c'. Given this inference rule, the concepts of derivation tree and branch in a derivation tree have straightforward definitions[12].

DEFINITION 3. *A branch in a derivation tree is* successful *if it is finite and ends in a query of the form* $?c$*, where* c *is a satisfiable constraint; note that the query is not allowed to contain an atom. A branch in a derivation tree is* finitely failed *if it ends in a query* $?c, B_1 \ldots B_m$ *such that either* c *is not satisfiable, or no program clause is applicable to the* B_i*. Otherwise the branch is called* infinite*.*

Intuitively, this definition applied to the situation of interest means the following. Suppose we start from a query $?HoldsAt(f,t)$ and find a successful branch ending in $?c$. This should mean that for all t, if $c(t)$ is true, then so is $HoldsAt(f,t)$. Likewise, if a branch finitely fails and ends in $?c, B_1 \ldots B_m$, we should have, for all t satisfying $c(t)$, $\neg HoldsAt(f,t)$.

For our purposes, definite constraint logic programs are not yet expressive enough, due to the occurrence of $\neg$ in the bodies of the axioms of the event calculus[13].

DEFINITION 4. *A* complex body *is a conjunction of literals, i.e. atoms and negated atoms, and constraints. A* normal program *is a formula* $\psi \rightarrow A$ *of* $CLP(\mathcal{T})$ *such that* ψ *is a complex body and* A *is an atom.*

The form of negation most congenial to constraint logic programming is *constructive* negation ([11]). In the customary negation as failure paradigm, negative queries differ from positive queries: the latter yield computed answer substitutions, the former only the answers 'true' or 'false'. Constructive negation tries to make the situation more symmetrical by also providing computed answer substitutions for negative queries. Applied to constraint logic programming, this means that both positive and negative queries can start successful computations ending in constraints. The full operational definition of constructive negation is somewhat involved (see [11]), but we will provide a simplified version (disregarding the possibility of infinite derivations) modeled on negation as failure, which suffices for our purposes.

The operational meaning of constructive negation may be given as follows. Suppose we are given the goal $?\mathcal{L}_1, \ldots \mathcal{L}_i, \ldots, \mathcal{L}_n, c$.Here, the ψ_i are literals and c is a set of constraints Consider a $\mathcal{L}_i$ of the form

[12] As in the case of standard logic programming, one also needs the concept of a *selection rule*, which determines which atom should be chosen at a particular stage in a derivation. The interested reader may consult [11]; we need not dwell on this topic here.

[13] The following definition is deliberately simplified from the one given in [11].

$\mathcal{L}_i = \neg B$, which has been selected for processing. Start a subderivation with goal $?B$. Assume this derivation tree is finite. Collect the constraints $c_1, \ldots, c_l$ occurring on the successful branches of the tree (the finitely failing branches can be disregarded). The children of the goal $?\mathcal{L}_1, \ldots, \mathcal{L}_i, \ldots, \mathcal{L}_{n,c}$ are now of the form

$$?c \wedge \neg c_i, \mathcal{L}_1, \ldots, \mathcal{L}_{i-1}, \ldots, \mathcal{L}_{i+1}, \ldots \mathcal{L}_n$$

for all i such that $c \wedge \neg c_i$ is satisfiable. There may be no such i, in which case the goal has no children. The subderivation may itself feature negative goals, so that an abstract definition of a derivation tree allowing constructive negation involves a recursion. We will not provide a definition, but the reader may check that the derivations used in the linguistic applications below all conform to the above characterization. A global concept of success for a derivation tree is given by the following definition.

DEFINITION 5. *A query $?c, G$ is* totally successful *if its derivation tree includes successful branches ending in constraints $c \wedge c_1$, ..., $c \wedge c_n$ such that $\mathcal{A} \models \forall \overline{x}(c \rightarrow c_1 \vee \ldots \vee c_n$.*

Intuitively, a query $?c, G$ is totally successful if all instances of the query also succeed; this is much stronger than saying that the query is satisfiable, as one would in standard logic programming.

3.7. Semantics

As in the case of negation as failure, the fundamental technical tool in describing the *semantics* of the above procedure is the *completion* of a program:

DEFINITION 6. *Let $\mathcal{P}$ be a normal program, consisting of clauses*

$$\overline{B}^1 \wedge c_1 \rightarrow p^1(\bar{t}^1), \ldots, \overline{B}^n \wedge c_n \rightarrow p^n(\bar{t}^n),$$

where the p^i are atoms and the B^i are complex bodies. The completion of $\mathcal{P}$, denoted by comp($\mathcal{P}$), is computed by the following recipe:

1. *choose a predicate p that occurs in the head of a clause of $\mathcal{P}$*
2. *choose a sequence of new variables $\overline{x}$ of length the arity of p*
3. *replace in the i-th clause of $\mathcal{P}$ all occurrences of a term in $\bar{t}_i$ by a corresponding variable in $\overline{x}$ and add the conjunct $\overline{x} = \bar{t}_i$ to the body; we thus obtain $\overline{B}^i \wedge c_i \wedge \overline{x} = \bar{t}_i \rightarrow p^i(\overline{x})$*

4. *for each i, let $\overline{z}_i$ be the set of free variables in $\overline{B}^i \wedge c_i \wedge \overline{x} = \overline{t}_i$ not in $\overline{x}$*

5. *given p, let $n_1, \ldots, n_k$ enumerate the clauses in which p occurs as head*

6. *define Def(p) to be the formula*

$$\forall \overline{x}(p(\overline{x}) \leftrightarrow \exists \overline{z}_{n_1}(\overline{B}^{n_1} \wedge c_{n_1} \wedge \overline{x} = \overline{t}_{n_1}) \vee \ldots \vee \exists \overline{z}_{n_k}(\overline{B}^{n_k} \wedge c_{n_k} \wedge \overline{x} = \overline{t}_{n_k}).$$

7. *comp($\mathcal{P}$) is then obtained as the formula $\bigwedge_p Def(p)$, where the conjunction ranges over predicates p occurring in the head of a clause of $\mathcal{P}$.*

The soundness of the operational definition of constructive negation is expressed by

THEOREM 1. *Let $\mathcal{P}$ be a normal program on the constraint structure $\mathcal{A}$, and let $\mathcal{T}$ be the axiomatization of $\mathcal{A}$.*

1. *If the query $?c, G$ is totally successful, then $\mathcal{T} + \mathcal{P} \models \forall \overline{x}(c \rightarrow G$.*

2. *If the query $?c, G$ is finitely failed, then $\mathcal{T} + \mathcal{P} \models \neg \exists \overline{x}(c \wedge G)$.*

There is also a corresponding completeness result (for which see [14]), but this need not detain us here. It is however of some importance to note the following consequence of the computational procedure just outlined. The formulation of the result is somewhat sloppy (see [14] for a rigorous version), but it suffices to capture the main idea.

THEOREM 2. *Let $\mathcal{P}$ be a normal program on the constraint structure $\mathcal{A}$, and let $\mathcal{T}$ be the axiomatization of $\mathcal{A}$. Then $\mathcal{T} + comp(\mathcal{P})$ has a unique model which is of the form given in section 3.3.*

4. Where do the fluents come from?

We now have to connect the preceding material with natural language. The basic idea of the approach is that meaning is computational, and that computations are performed in the language of the event calculus. Here is an example of this kind of computation involving the English progressive, taken from [5] and [14]. Consider the sentences

(14) a. John was crossing the street.
b. John crossed the street.

The second sentence implies that John arrived at the other side, but the first one does not. Nevertheless, the VP 'cross the street' is telic in that it comes with a canonical culminating event. The so-called 'imperfective paradox' generated by sentences (14-a) and (14-b) is: on the one hand the canonical culminating event essentially belongs to the meaning of 'cross the street', on the other hand the actual occurrence of that event can be denied with impunity. The solution given in [14] is that an accomplishment such as 'cross the street' actually corresponds semantically to a *plan* for reaching the other side of the street. This plan will be brought to successful completion in a minimal model of the plan, but not necessarily in extensions of the minimal model. The plan corresponds to a scenario in the language of the event calculus, and the minimal model of the plan can be computed using the procedure outlined in the previous section. What we have not yet explained is how to associate a plan to a lexical expression.

The first step in setting up a corresponding plan consists of the transformation of lexical material into fluents and events. Here is a brief sketch of how this can be done – a more detailed exposition can be found in [5] and [14]. We assume that verbs correspond to predicates which have a time parameter. Thus, in the intransitive case, 'walk' corresponds to the predicate $walk(x,t)$. Given this predicate, one can form two kinds of abstraction over it, corresponding to perfect and imperfect aspect, respectively:

1. $\exists t\, walk(x,t)$
2. $\{t \mid walk(x,t)\}$

Since time is interpreted on the reals, which contain the integers, such expressions may be assigned Gödel numbers as 'codes', with the consequence that these codes may themselves figure as arguments in predicates[14]. Codes of expressions of the first kind will play the role of event types in the event calculus, whereas those of the second kind will function as fluents. Indeed, for fixed x, $\{t \mid walk(x,t)\}$ can be viewed as a function from times to truth values, i.e. as a time-dependent property, and this is precisely how fluents are characterized.

We shall employ the following notation in using events and fluents derived from natural language expressions: event types will be denoted by $e(\overline{x})$, and fluents by $f[\overline{x}]$, where in both cases e and f are replaced by suitable natural language expressions. For example, we will meet the expressions *leave-to(Jean,Brasil)* (an event type), and *is-president*(x) (a fluent).

[14] In AI, this process is known as *reification*.

The second step in setting up a plan corresponding to a lexical expression consists in writing down a scenario in the language of the event calculus, which captures the (causal aspects of) the meaning of that expression. This second step is best explained by means of examples, of which many will be given below.

We have now completed our introduction to the computational machinery. In the next section we exploit a feature of this machinery to give an account of reference time, together with utterance time and event time the main pillar of the semantics of tense.

5. Reference time as integrity constraint

Reichenbach's great insight into tense was his identification of the importance of the reference time, on a par with event time and utterance time. The reference time is a marker for the time, context or situation that we are talking about. R must be known by the participants in order for the temporal discourse to make sense. Reichenbach noticed that the reference time can be different from the event time, as for instance in the present perfect

(15) I have caught a flu.

Here the infection-event lies in the past, but the reference time is identical to the utterance time: the sentence is meant to have present relevance, e.g. as an explanation for my being bad–tempered. We now have to investigate how the reference time is to be formulated in our framework. This is not at all easy, as the following example will make clear.

Suppose we try to model the English present perfect, in a very simple situation, where there is an event type e (say a viral infection) which initiates a consequent state f e.g. having a flu); there are no further events or fluents. The scenario therefore contains only the statement

(16) Initiates(e, f, t).

Suppose the utterance time is denoted by a constant *now*, to be interpreted on the reals; this constant belongs to the constraint language. The present relevance of the present perfect then suggests that the contribution of this tense to the scenario is the addition of formula (17-b), so that the scenario becomes

(17) a. Initiates(e, f, t)
b. HoldsAt(f, now).

We would like to derive from (17), using the axioms of the event calculus, that for some $t < now$, $Happens(e,t)$. Naively one might reason as follows: completing the axioms of the event calculus plus the scenario gives us

$$Happens(e,t) \wedge Initiates(e,f,t) \wedge t < t' \wedge \neg Clipped(t,f,t') \leftrightarrow HoldsAt(f,t'),$$

so that the desired result follows after applying the given (17-b). Although this argument embodies an important intuition, it cannot be pushed through as stated, since the completion at issue is actually

$$[Happens(e,t) \wedge Initiates(e,f,t) \wedge t<t' \wedge \neg\ Clipped(t,f,t')] \vee [now=t'] \leftrightarrow HoldsAt(f,t'),$$

from which nothing can be derived. Thus the contribution of the reference time cannot simply be an addition to the scenario.

To clarify the contribution of the reference time, we need a small excursion in database theory, taking an example from Kowalski [7, p. 232].

An *integrity constraint* in a database expresses obligations and prohibitions that the states of the database must satisfy if they fulfill a certain condition. A simple example is a database of family relationships, which should satisfy constraints such as 'everybody has a genetic father' (obligation) and 'no one is both father and mother' (prohibition). The operational meaning of the integrity constraints is that each time the database is updated, it must be checked whether the constraints still hold.

Kowalski [7, p. 232] advocated the use of integrity constraints in the slightly different context of reactive agents, who have to perform appropriate actions given certain sensory input. Here the integrity constraint is used to request an update (e.g. stating that an action has been performed), rather than to regiment the updates as in the examples above. For instance, the 'obligation' to carry an umbrella when it is raining, may be formalized by the integrity constraint

$$HoldsAt(rain,t) \ \rightarrow \ HoldsAt(carry\_umbrella, t+\epsilon). \qquad (1)$$

The intended meaning is that if it rains at t, then the agent should carry an umbrella soon after. The crucial point here is the meaning of the $\rightarrow$. The formula 1 cannot be an ordinary program clause, for in that case the addition of $HoldsAt(rain,t)$ would trigger the consequence $HoldsAt(carry\_umbrella,t)$ which may well be false, and in any case does not express an obligation. Below we will therefore replace $\rightarrow$ by the expression IF... THEN, whose meaning is defined operationally.

A good way to think of an integrity constraint expressing an obligation is to view the consequent as a constraint that the database

must satisfy if the antecedent holds. Similarly, an integrity constraint expressing a prohibition can be taken to mean: if the antecedent holds, then the database should not satisfy the consequent. In other words, in the first case the integrity constraint imposes an obligation on the ordinary sentences in the database to establish that the consequent should hold. This entails in general that the database has to be updated with a true statement about the world; which statement that is, has to be found out by abduction. To return to our example, there will be an action *take_umbrella*, whose meaning is given by the database clause

$$Initiates(take\_umbrella, carry\_umbrella, t).$$

Suppose the database is updated with $HoldsAt(rain, now)$, i.e. the antecedent of the integrity constraint 1. The integrity constraint then requires us to set up a derivation starting from the query

$$?HoldsAt(carry\_umbrella, now + \epsilon).$$

Applying the event calculus we can reduce this query to

$$?Happens(take\_umbrella, now), \neg Clipped(now, carry\_umbrella, now{+}\epsilon).$$

We now have to update the database in such a way that the query succeeds. This can be achieved if we *only* add the clause

$$Happens(take\_umbrella, now),$$

and no other occurrences of events. For in this case the query

$$?Happens(take\_umbrella, now), \neg Clipped(now, carry\_umbrella, now{+}\epsilon)$$

reduces to

$$?\neg Clipped(now, carry\_umbrella, now + \epsilon),$$

and this query can be shown to succeed by applying negation as failure to

$$?Clipped(now, carry\_umbrella, now + \epsilon).$$

Indeed, the latter query fails because of the way we updated the database. Of course, it is assumed that a statement such as $Happens(take\_umbrella, now)$ only gets added when in fact the action has been performed.

As this example makes clear, an integrity constraint requires us to update a database in a particular way. A derivation is started with the consequent of the integrity constraint as the top query. Then resolution with clauses from the database is applied for as long as possible. The

derivation will in general end with a query that cannot be further resolved. If this were an ordinary derivation we would then apply negation as failure to the top query. In the case of an integrity constraint we use the unresolved bottom query instead to suggest an addition to the database which will make the top query succeed after all. The procedure chosen has the effect of making a minimal update of the database to ensure success of the top query: the computation exploits as much of the database as is possible, and only plugs in facts when absolutely necessary.

These considerations lead us to the following definition of integrity constraint as it applies in our context.

DEFINITION 7. *Let $R, R', R'' \ldots$ be a finite set of constants each denoting a reference time; these constants belong to the constraint language. An* integrity constraint *is a formula of the form*

(†) *IF φ THEN $\psi(R, R', R'' \ldots)$,*

where φ and ψ are formulas of the event calculus.

The operational meaning of (†) *is that if the scenario satisfies φ, the goal $?\psi(R, R', R'' \ldots)$ must succeed, or fail finitely. To determine whether the scenario satisfies φ, one has to investigate whether the goal $?\varphi$ succeeds. Hence if the integrity constraint expresses an obligation it may be represented by the demand that $?\varphi, \psi(R, R', R'' \ldots)$ succeeds.*

The case that the goal must succeed expresses an obligation; the case where it must fail finitely expresses a prohibition. A typical application is where $\psi(t, t', t'' \ldots) = \mathit{HoldsAt}(f, t) \wedge t \leq \mathit{now}$.In this case we require that the goal?$\mathit{HoldsAt}(f, R) \wedge R \leq \mathit{now}$ succeeds or fails finitely by either of the following strategies.

1. One may update the scenario with true *Happens*, *Initiates* and $\neg$*Clipped* formulas, using the axioms of the event calculus; this is the strategy of choice if f is an activity fluent. It has the effect of making the temporal denotation of f extended in time.

2. If the first strategy fails, the scenario may also be updated with true *HoldsAt* or *Initially* formulas, or (in)equalities in R in the language of the reals. This is a possible strategy if f represents a state, since states do not have to be caused by events. For example, they may be a particular instance of a parametrized state, which evolves continuously via a dynamics.

Another typical application of integrity constraints in our context is where the goal which must succeed or fail finitely is of the form $?\mathit{Happens}(e, R) \wedge R \leq \mathit{now}$. Again there are two possible strategies for handling the goal:

1. if $\mathit{Happens}(e,t)$ occurs in the head of a clause with nontrivial body $\theta(e,t)$, proceed by resolving the query $?\theta(e,R)$;

2. otherwise, replace $\mathit{Happens}(e,R)$ by a set of true (in)equalities in R.

It is also possible to have an integrity constraint without a condition, i.e. where φ is a tautology. An entry in my diary like 'appointment in Utrecht, Friday at 9.00' expresses an unconditional obligation to satisfy *HoldsAt(be-in-Utrecht,Friday at* 9.00), and presented with this integrity constraint, my internal database comes up with a plan to satisfy the constraint. We will usually refer to an unconditional integrity constraint by means of the query that must succeed, or fail finitely.

We now illustrate the linguistic relevance of the preceding definition by the example of the English perfect: for the present perfect the integrity constraint is that the query

(18) ?HoldsAt(f,R), R=now,

must succeed, whereas for the pluperfect success is required for the query

(19) ?HoldsAt(f,R), $R <$now.

In both cases the logic programming mechanism starts a computation from the given query by applying the axioms of the event calculus. For instance, applying axiom 2 means that the database searches for an event e such that $\mathit{Initiates}(e,f,t)$, $\mathit{Happens}(e,t_0)$ and $\neg\mathit{Clipped}(t_0,f,t)$. If this query does not succeed, the database may ask the outside world for input. For instance, in the above the scenario consists of the formula $\mathit{Initiates}(e,f,t)$ only. The database then asks for input of a true formula $\mathit{Happens}(e,t_0) \wedge t_0 \leq R$. This is the computational meaning of the perfect. It is also the computational meaning of the progressive, where the fluent f represents an activity and e an initiating event. If the database would be unable to find a formula $\mathit{Initiates}(e,f,t)$, it could also ask the world for input of a true formula of this type. Alternatively, it could forego the search for an f-triggering event and ask for a set of (in)equalities in R or (using axiom 1) a true *Initially* formula instead. These strategies may occur if f represents a stative verb, for in that case f need not be triggered by an action or event.

The upshot of the preceding discussion is that a reference time is characterized by a set of fluents which must hold at that time. This stipulation captures the idea that the role of a reference time is to

fix the situation or context that we are talking about. In general such situations are only partially determined. Suppose the reference time is characterized by fluents $f_1, \ldots, f_n$, i.e. by the integrity constraint

$$?HoldsAt(f_1, R), \ldots, HoldsAt(f_n, R).$$

If we want to stipulate that another fluent f, say the result fluent involved in the perfect, holds at R, the only way to do this is to enlarge the integrity constraint to

$$?HoldsAt(f_1, R), \ldots, HoldsAt(f_n, R), HoldsAt(f, R),$$

i.e f must occur in a subgoal in the integrity constraint. In general we shall mention only the immediately relevant part of the integrity constraint, in this case $?HoldsAt(f, R)$, and leave out the contextually given part[15].

Before we move to a discussion of the Passé Simple and Imparfait, let us take stock. The principal function of the scenario is to contribute lexical information, which is general and does not talk about specific times. The addition of temporal information is required to construct a sentence out of lexical material. Contrary to first impressions, the reference time cannot be added as a fact to a scenario, as we have seen in the case of the perfect. The reference time is an integrity constraint formulated in terms of *fluents*, which typically puts constraints upon possible temporal locations of *event types*.

We have to exercise some care here: the traditional phrase 'event time' obscures the fact that there are at least two different kinds of events, what we have termed here fluents and event types. Localizing stative verbs or VPs in the English progressive, involves anchoring a fluent, whereas the English simple tenses locate event types. This is related to an important issue: the role of *Aktionsart* in defining tense. Some authors, such as Comrie [1], prefer to define tense abstracting from aspectual features, the idea being that tense talks only about localization in time; an example will be given below. We try to go a different route, and allow for the possibility that tense works out differently for different *Aktionsarten*. Some indications of what this means in practice are given below.

[15] In the presence of integrity constraints, the model whose existence is posited in theorem 2 depends of course on the constraints used the make the goal in the integrity constraint succeed or fail.

6. Formalizing the Passé Simple and Imparfait

We will now formalize the examples of section 2 using the event calculus formalism.

6.1. Passé Simple: scenarios and integrity constraints

Guided by the previous analysis we propose that the effect of the Passé Simple is to introduce an integrity constraint of the form ?*HoldsAt*(f, R), *Happens*(e, R), $R <$ *now*, where e is the event type derived from the VP which occurs in the PS, and the fluent f represents the context in which the PS is interpreted. If the context is empty, for example if the sentence considered is the first sentence of the discourse, we leave out the *HoldsAt* clause. One may observe immediately that this stipulation accounts for two features of the PS: it presents the eventuality as perfective, and it places the eventuality in the past of the speech time.

We also need a meaning postulate for the conjunction *et*, which is not a simple Boolean conjunction. The following stipulation seems to capture what we need: if the PS occurs in the form 'S*et* PS-VP', then the fluent f occurring in the integrity constraint for the PS refer to the state which results from the event described by S (and not from material that was processed earlier). We view the construction 'S_1, S_2 *et* S_3' as an iterated form of *et*, that is, as '(S_1 *et* S_2) *et* S_3'. Sentences conjoined by *et* are thus bound together more tightly then sentences conjoined by a period.

6.1.1. *Succession (non)effects*

Recall that we have argued in section 2.1 for the succession effect in PS narratives as a side-effect of the semantics of the PS. Consider the sentences

(20) a. Pierre monta dans sa chambre et ferma la porte. (2×PS)
b. Pierre ferma la porte et monta dans sa chambre. (2×PS)

If our approach is to be correct, the implied succession in sentences (20-a) and (20-b) should derive from the ordering of the sentences, the identity of the subject in the conjuncts related by *et*, and the perfectivity of the PS. We propose the following derivation of this effect. The scenario for sentence (20-a) looks as follows

1. a) Initiates(go-upstairs(x), upstairs[x], t)
   b) ?Happens(go-upstairs(Pierre), R), $R <$ now *succeeds*
2. a) Initiates(close(x, y), closed[y], t)

b) ?HoldsAt(upstairs[Pierre], R'), Happens(close(Pierre,door), R'), $R' <$ now *succeeds*

The formulas collected in 1. represent the information in the scenario induced by the first VP[16]. The first formula states lexical information, and the integrity constraint gives the contribution of the PS. For completeness we should have added a *HoldsAt* clause as well, whose fluent has information about the context; but this clause would be irrelevant to the computation. The minimal model for this scenario looks as follows. Until reference time R, the fluent *upstairs[Pierre]* does not hold (by negation as failure), at R the event *go-upstairs(Pierre)* happens and initiates the fluent *upstairs[Pierre]*.

The next pair of formulas introduces the semantic contribution of the second conjunct of (20-a). The lexical information introduced in 2a is straightforward. The choice of the integrity constraint 2b requires some explanation. The *Happens* clause of the integrity constraint represents the effect of the PS (perfective event in the past of the speech time), and the *HoldsAt* clause represents the context in which the PS is interpreted. Since the two clauses in (20-a) are linked by *et*, the fluent in this *HoldsAt* clause must refer to the state resulting from 1. We show that this choice accounts for the default succession effect of a sequence of PS sentences ascribed to a single subject. In the minimal model we have $R < R'$, as is shown by the following derivation. The first few steps in the argument look like this: The top node of this derivation

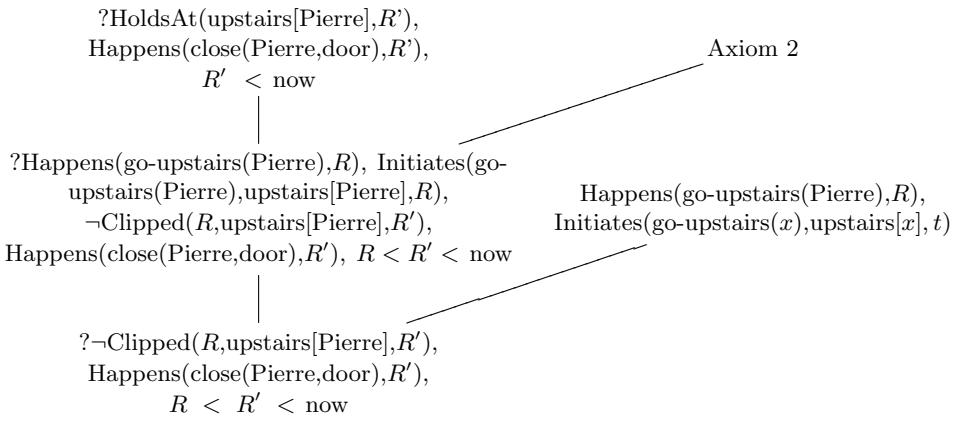

Figure 2. Effect of the second integrity constraint in (20-a).

contains the integrity constraint, i.e. a goal which is assumed to succeed. The derivation shows that the top goal can only succeed if the goal ?¬Clipped(R,upstairs[Pierre],R'), Happens(close(Pierre,door),R'),$R <$

[16] Recall that *go-upstairs*(x) denotes an event type and *upstairs*[x] a fluent.

$R' <now$ also succeeds. A simple negation as failure argument shows that the subgoal ?¬Clipped(R,upstairs[Pierre],R') succeeds. This leaves us with the goal ?Happens(close(Pierre,door),R'),$R < R' <now$, which means that any R' satisfying Happens(close(Pierre,door),R') must also satisfy $R < R' <now$.

Having explained the main idea, we shall usually leave the last few steps, including the proof that the ?¬Clipped subgoal succeeds, to the reader.

Analogously, the scenario for sentence (20-b) looks as follows

1. a) Initiates(close(x, y), closed[y], t)
   b) ?Happens(close(Pierre,door), R), $R <$ now *succeeds*
2. a) Initiates(go-upstairs(x), upstairs[x], t)
   b) ?HoldsAt(closed[door], R'), Happens(go-upstairs(Pierre), R'), $R' <$ now *succeeds*

A derivation analogous to figure 2 shows that in the minimal model we must have $R < R'$. In both this case and the previous, the derivation does not branch, corresponding to the fact that sentences (20-a) and (20-b) have a single reading.

So far so good, but we also have to check whether the proposed integrity constraint does not overgenerate, that is, we have to look at examples where the succession does not hold. Let us first look at example (7-d), here adapted to

(21) # Pierre monta dans sa chambre et se leva. (2×PS)

As we remarked above, sentence (21) is not felicitous because the information conveyed by the second VP contradicts the lexical presupposition of the first. That is, you go upstairs walking, hence you need to be standing up; and if you are already standing up you cannot perform the action of getting up. The scenario for this case has the following form

1. a) Initiates(go-upstairs(x), upstairs[x], t)
   b) ?Happens(go-upstairs(Pierre), R), $R <$ now *succeeds*
2. a) HoldsAt(sitting-down[x], t) → Initiates(get-up(x), upright[x], t)
   b) ?HoldsAt(upstairs[Pierre], R'), Happens(get-up(Pierre), R'), $R' <$ now *succeeds*

Using only the material in 1. we obtain a minimal model where the fluent *upstairs[Pierre]* is initiated at time R (hence does not hold before R). Viewed superficially, the material in 2. enforces that the event *get-up(Pierre)* happens at time R' with $R' < now$. However, the integrity constraint in 2(b) is actually inconsistent with 2(a), as we can see when we try to compute the query *?HoldsAt(upright[Pierre],t)*,$R' < t$.

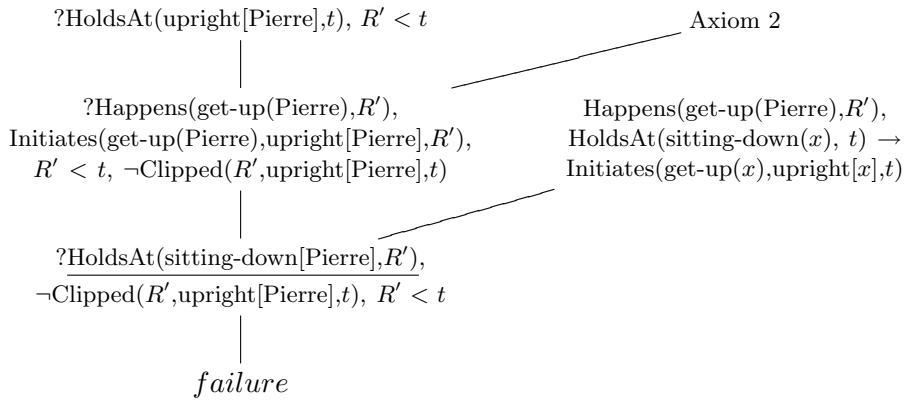

Figure 3. Conditions on the fluent *upright[Pierre]* in (21).

The problem is that, in the course of the derivation, the query *?HoldsAt(upright[Pierre],t)* is transformed into *?HoldsAt(sitting-down [Pierre],R′)* which cannot lead to successful termination, as we do not have any information in the scenario pertaining to an event initiating this fluent (see figure 3). This means that also for t later than R', *?HoldsAt(upright[Pierre],t)* is false. As a consequence, the goal 2(b) cannot succeed.

The following example presents a case where the order of the events can actually be the inverse of the order of the sentences describing them.

(22) Pierre brisa le vase. Il le laissa tomber. (2×PS)

When introducing this example in section 2.1 we noted that one may get the standard ordering back upon enlarging the discourse:

(23) a. Pierre brisa le vase avec un marteau. Il le laissa tomber et s'en alla.
b. Pierre brisa le vase avec un marteau. Il le laissa tomber. Il s'en alla sans le regarder.

Thus we must be able to explain the inversion of example (22) by a construction which is flexible enough to also accomodate examples (23-a) and (23-b).

It is important to mention at this stage that lexical expressions do not come with unique scenarios. A clause in a scenario can be seen as an activated part of semantic memory[17]; which part is activated depends on all kinds of circumstantial factors. For this example we assume that the scenario for 'break' contains an open-ended set of clauses specifying possible causes of the breaking. We choose a simplified formulation here; e.g. 1b below could be derived in more elaborate scenario detailing the relationship between 'drop', 'fall' and impact on the ground. The simplified formulation is better suited, however, to illustrate the main points of the argument. Accordingly, we will take the scenario to be of the following form, where we omit the *HoldsAt* components of the integrity constraints because they play no role in the derivation.

1. a) Initiates(break(x, y), broken$[y], t)$
   b) Happens(drop(x, y),$t - \epsilon) \rightarrow$ Happens(break$(x, y), t)$
   c) Happens(smash(x, y),$t) \rightarrow$ Happens(break$(x, y), t)$
   $\vdots$
   d) ?Happens(break(Pierre,vase), R), $R <$ now *succeeds*
2. a) ?Happens(drop(il,le), R'), $R' <$ now *succeeds*[18]

A successful computation starting from the query *?Happens(break(Pierre,vase), R), R< now* is given in figure 4.

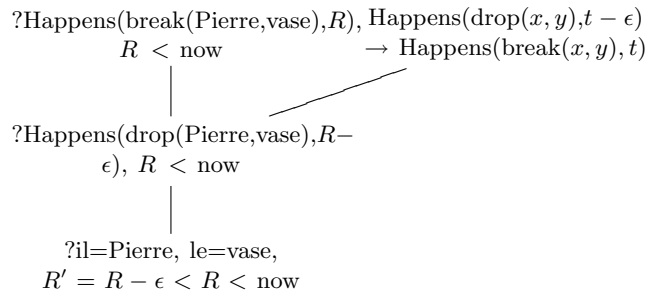

Figure 4. The effect of the two integrity constraints in example (22).

This computation explains the reversed order. Notice however that if we would bind the two sentences with an *et*, as in (5), the integrity

[17] There is actually a close connection between logic programming with negation as failure and the spreading activation networks beloved of psycholinguists.

[18] The anaphors 'il' and 'le' are really variables to be unified with concrete objects; we keep the words as handy mnemonics.

constraint for the second sentence would be

$$?Happens(drop(il, le), R'), HoldsAt(broken[vase], R'), R' < \ now$$

. By negation as failure, *Initially(broken[vase])* is false, hence there must have been an event initiating the fluent *broken[vase]*. If 'il' is unified with Pierre and 'le' with the vase, this is impossible, because dropping the vase would have to take place before R'. If the fluent *broken[vase]* goes proxy for the broken vase (as an object)[19], it is possible to unify 'le' with *broken[vase]*, and get a coherent interpretation again. The examples (23-a) and (23-b) can be treated in the same manner.

Finally, we come to an example where the events described have no natural order.

(24) Cet été-là, François épousa Adèle, Jean partit pour le Brésil et Paul s'acheta une maison à la campagne. (3×PS)

The fact that we cannot order the enumerated events in sentence (24) is mainly due to the different subjects of the VPs. The temporal adverbial (Cet été-là) only places the events in a certain period of time, without implying anything about their order.

A scenario might look as follows:

1. a) Initiates(begin, this-summer, t)
   b) Terminates(end, this-summer, t)
2. a) Initiates(marry(x, y), married[x, y],t)
   b) ?HoldsAt(this-summer, R_1), Happens(marry(Francois, Adèle), R_1), $R_1 <$ now *succeeds*
3. a) Initiates(leave-for(x, y), be-in[x, y], t)
   b) ?HoldsAt(this-summer, R_2), Happens(leave-for(Jean,Brasil), R_2), $R_2 <$ now *succeeds*
4. a) Initiates(buy(x, y), have[x, y], t)
   b) ?HoldsAt(this-summer, R_3), Happens(buy(Paul,countryhouse), R_3), $R_3 <$ now *succeeds*

What we obtain from the integrity constraints, by means of a derivation like the ones given above, is that there are times R_0 and R_4 such that *Happens(begin, R_0)*, *Happens(end, R_4)*, $R_0 < \{R_1, R_2, R_3\}$ and $\{R_1, R_2, R_3\} \leq R_4$. However, the order of R_1, R_2 and R_3 cannot be determined.

[19] This trick is explained in [14].

6.1.2. *Inchoative use of the PS*
Consider again the example

(25) Mitterand fut président. (PS)

We have to derive formally that the PS applied to the stative expression 'be president', picks out the initiating event. Interestingly, when we are only given the fluent 'be president', there is no explicitly given event which warrants the application of the PS. Applying the PS means that a form of coercion is going on, in which the fluent is somehow transformed into an event. The proper of way of doing this involves so-called hierarchical planning, for which we refer to [14], since it is too involved to explain here; we give a simplified treatment, based on the idea that the PS 'searches' the scenario to find the event which saturate the event-argument in the integrity constraint. Since presidents are usually elected, the scenario for 'be president' will contain a statement such as 1a. This statement contains a reference to the event 'elect', which may thus figure in an integrity constraint. We thus get

1. a) Initiates(elect(x), president[x], t)
   b) ?Happens(elect(M.), R), $R <$ now *succeeds*

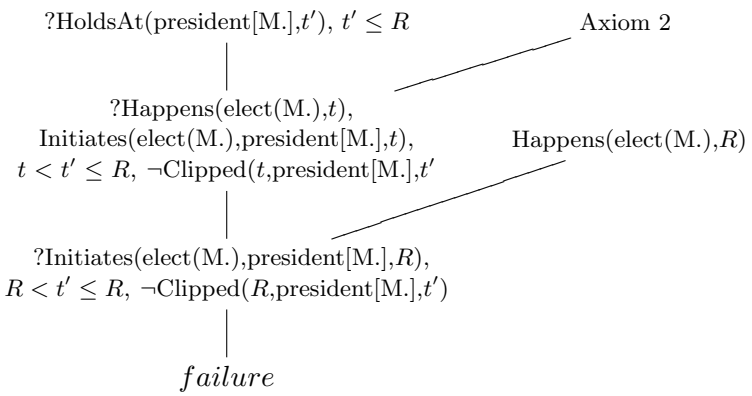

Figure 5. Fluent *president[M.]* before R.

As can be seen from figure 5, the fluent *president[M.]* does not hold before R. A similar derivation shows that it must hold after R.

6.2. Imparfait: scenarios and integrity constraints

The integrity constraint associated to the Imparfait must be very different from that associated to the Passé Simple, for example because an Imp sentence is not felicitous in isolation, unlike a PS sentence. An

Imp sentence must be anchored by means of PS in the discourse. We therefore propose the following.

An Imp VP with an adjacent PS VP introduces an integrity constraint of the form

$$?Happens(e, R),\ HoldsAt(f_1, R'), ...,\ HoldsAt(f_n, R'),\ R < now,\ R' < now$$

where e is some PS event of the discourse context (this sentence can precede or come after the Imp sentence), and f_1,..., f_n are the relevant fluents describing the Imp verb phrase.

The most relevant part of the integrity constraint for the Imp is the *HoldsAt(*f, R'*)* part. This part is what distinguishes the PS and the Imp: the PS introduces an integrity constraint of the form *Happens(*e, R*)*, possibly together with some other fluents that hold at R, while the integrity constraint associated to the Imp introduces a number of *HoldsAt(*f, R'*)* statements that are combined with the *Happens(*e, R*)* statement of a PS VP in the discourse.

6.2.1. *Imparfait as background*
Consider the discourse

(26) Il faisait chaud. Jean ôta sa veste. (Imp, PS)

The scenario for these sentences must contain a fluent *warm*, and an event and a fluent for the achievement 'take off one's sweater'. For the latter we choose the event *take-off*, which terminates the fluent *wearing*; equivalently, we could have *take-off* initiate *not-wearing*. The integrity constraint anchors the fluent *warm*; note again that anchoring is only possible given a PS VP.

1. a) Terminates(take-off(x, y), wearing[x, y],t)
   b) ?HoldsAt(warm, R), HoldsAt(wearing[Jean,vest],R), Happens(take-off(Jean,vest),R), $R <$ now *succeeds*

The derivation in figure 6 shows that ' Il faisait chaud' really functions as a background.

The final query can succeed only if *warm* is true from the start. The next derivation (figure 7) shows the fate of the fluent *wearing[Jean,vest]*. Hence the fluent *warm* is true at all times, while the fluent *wearing[Jean,vest]* holds until R and is terminated at this time.

6.2.2. *Imparfait for a resultant state*
(27) Jean appuya sur l'interrupteur. La lumière l'éblouissait. (PS, Imp)

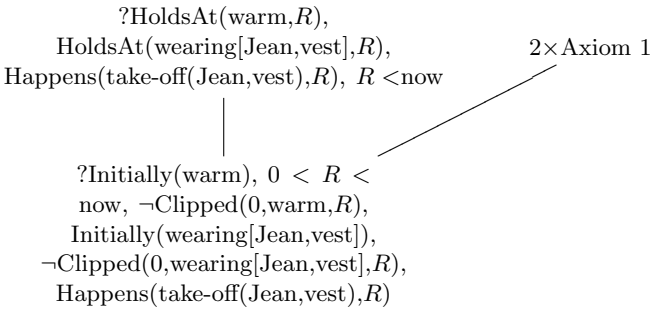

Figure 6. Integrity constraint in example (26).

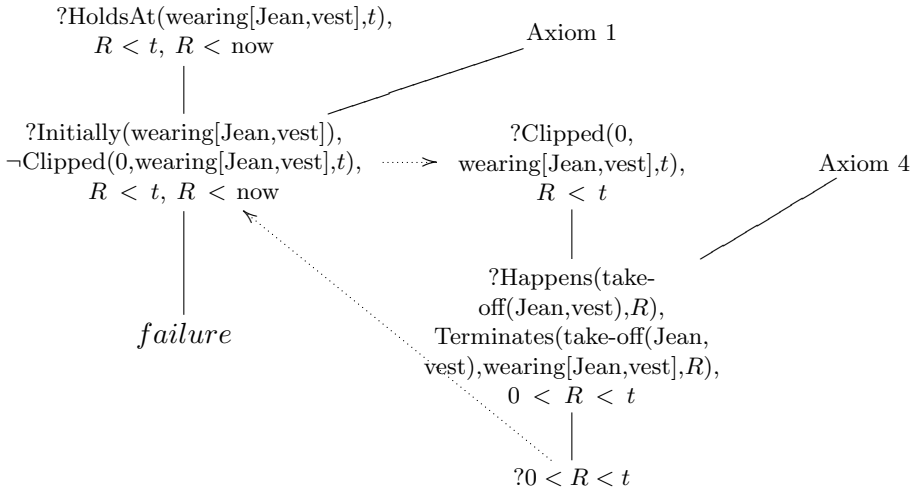

Figure 7. Fluent *wearing[Jean,vest]* in example (26) for $t > R$.

This is an example where there is no overlap between the two eventualities, pushing a button and being blinded. The desired effect is obtained only when the scenario gives some information about the causal relation between the light being on and being blinded; this is the purpose of part 2 of the scenario.

1. a) Initiates(push(x,on), light-on, t)
   b) Terminates(push(x,off), light-on, t)
   c) ?Happens(push(Jean,y), R), R <now *succeeds*
2. a) Releases(push(x,on), blinded[x], t)
   b) Trajectory(light-on, t, blinded[x], d)
   c) ?Happens(push(Jean,y), R), HoldsAt(light-on, R'), HoldsAt(blinded[Jean],R'), R <now, R' <now *succeeds*

Figure 8 shows the derivation starting from the integrity constraint 2c. The substitution leading to success is indicated. The last query in the derivation can be made to succeed because the scenario makes no mention of a push-off event, and we therefore obtain the conclusion $R < R' < now$.

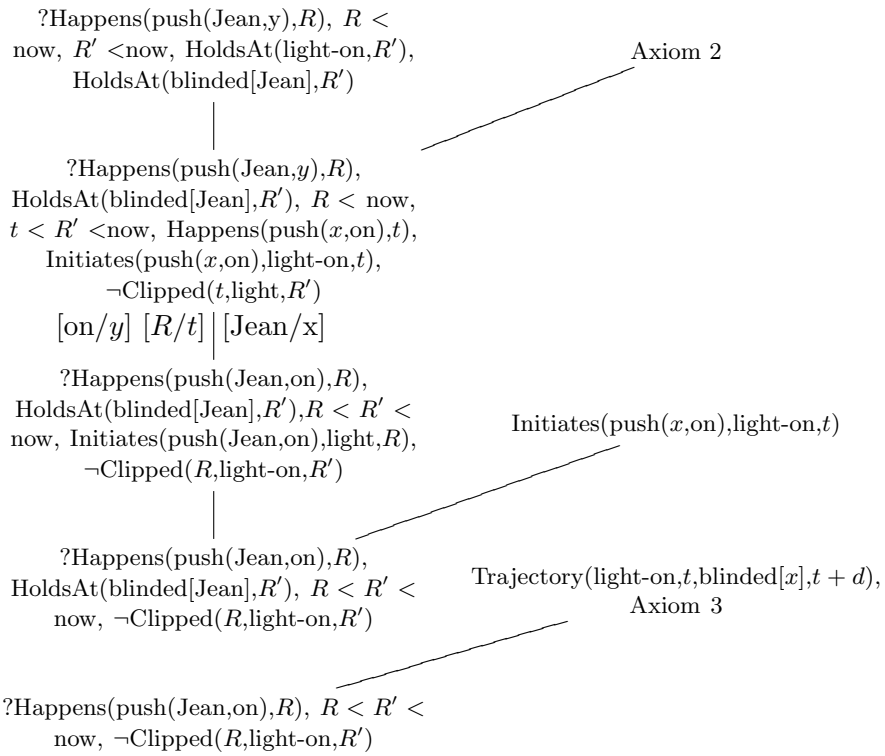

Figure 8. Integrity constraint in example (27).

6.2.3. *Imparfait in an explanatory context*
In the following discourse, the second sentence has the function of explaining the event described in the first sentence. The eventuality described in the second sentence should therefore be placed in its entirety before the event described in the first sentence.

(28) Jean attrapa une contravention. Il roulait trop vite. (PS, Imp)

The scenario for this situation may look as follows. The first two statements have been included for convenience only; it would make no difference if we pushed the beginning of the scene further in the past and introduced an event initiating driving.

1. a) Initially(driving[Jean])

b) Initially(speed[s])

c) Initiates(get(x,ticket), have[x,ticket],t)

d) ?Happens(get(Jean,ticket),R), $R <$now *succeeds*

2. a) Terminates(get(x,ticket), driving[x],t)

b) Terminates(get(x,ticket), speed[s],t)

c) Initiates(get(x,ticket), speed[0],t)

d) ?Happens(get(Jean,ticket),R), HoldsAt(speed[s],R'), HoldsAt(driving[Jean],R'), $s >$ limit, $R <$ now, $R' <$ now *succeeds*

In the first step we start from the query in 2d, and we expand the derivation tree according to the different possibilites for the relation of R and R', and then we recombine to get the possibilities $R' \leq R$ and $R < R'$. These possibilities are considered in the figures 10 and 11, respectively.

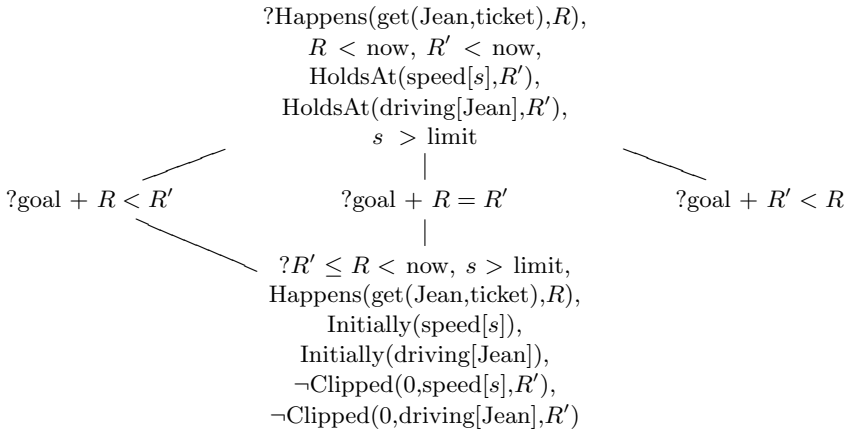

Figure 9. Integrity constraint in example (28).

The reader should notice that, for the sake of readability, in figures 10 and 11 we process the *Clipped* formulas in the same tree and delete them from the goal. The proper treatment of the integrity constraint would be, as described in section 5, to first update the database with the *Happens* statement and then begin a new tree for the *Clipped* statement and check it for failure.

Derivation 10 terminates successfully with the constraint $R' \leq R <$ *now*, because of part 1 of the scenario.

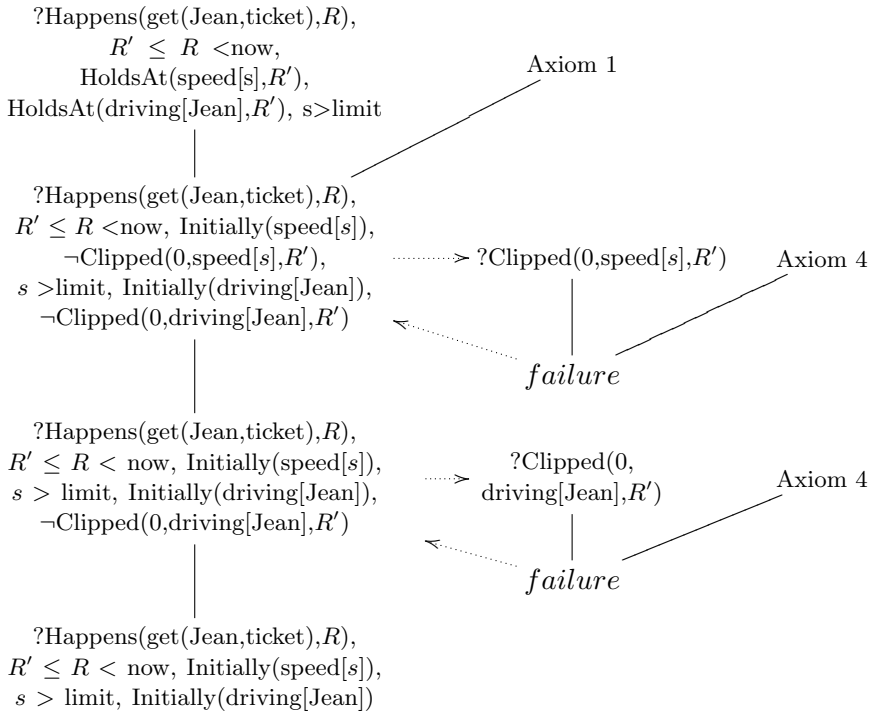

Figure 10. Integrity constraint in example (28) with $R' \leq R$.

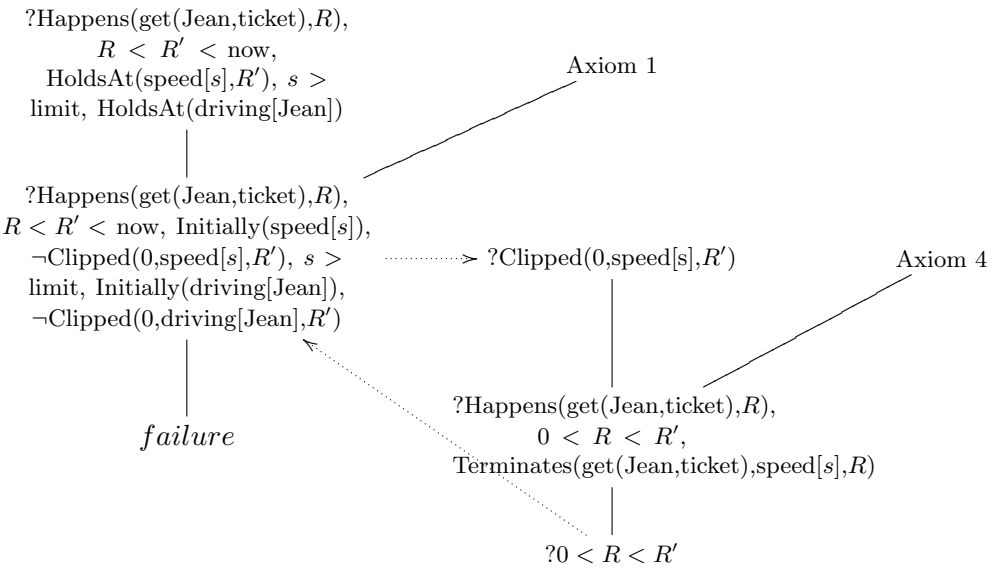

Figure 11. Integrity constraint in example (28) with $R < R'$.

Now consider derivation 11 for the other possibility, $R < R'$. This derivation ends in failure, because the subderivation for *Clipped(0, driving[Jean],R′)* will end in success, given that getting a ticket at $R < R'$ ends in terminating the driving at that point.

7. Coda

In conclusion we can do no better than quote from the eloquent 'Apology and guide to the reader' of Kamp and Rohrer [6]

> ... the mechanisms which natural language employ to refer to time cannot be properly understood by analyzing the properties of single sentences. Thus the methodology of modern generative grammar, which takes the single sentence as the basic unit of study is not, we believe, suited to this particular domain. Rather, a proper analysis of temporal reference must
> (a) make explicit its *anaphoric* aspects – the systematic ways in which such devices of temporal reference as tenses and temporal adverbs rely for their interpretation on temporal elements contained in the antecedent discourse –
> and
> (b) discover the temporal organization of those conceptual structures which extended discourses produce in the human recipients who are able to interpret them.

This is precisely what we have attempted to do here.

References

1. B. Comrie. *Tense*. Cambridge University Press, Cambridge, 1985.
2. H. de Swart and F. Corblin (eds). *Handbook of French Semantics*. CSLI Publications, February 2002. NWO-CNRS PICS project.
3. D. Dowty. *Word meaning and Montague grammar*. Reidel, Dordrecht, 1979.
4. L. Gosselin. *Sémantique de la temporalié en français*. Champs Linguistiques. Editions Duculot, 1996.
5. F. Hamm and M. van Lambalgen. Event calculus, nominalisation and the progressive. Research report, ILLC, Amsterdam, December 2000. 76 pp. To appear in *Linguistics and Philosophy*. Available at http://www.semanticsarchive.net.
6. H. Kamp and C. Rohrer. unpublished progress-report for research on tenses and temporal adverbs of French, 200?
7. R.A. Kowalski. Using meta-logic to reconcile reactive with rational agents. In *Meta-logics and logic programming*, pages 227–242. MIT Press, 1995.
8. M.P. Shanahan. *Solving the frame problem*. The M.I.T. Press, Cambridge MA, 1997.
9. M. Steedman. Temporality. In J. van Benthem and A. ter Meulen, editors, *Handbook of Logic and Language*, chapter 16, pages 895–938. Elsevier Science, 1997.

10. H. Sten. *Les temps du verbe fini (indicatif) en français moderne*. Historik-filologiske Meddelelser. Det Kongelige Danske Videnskabernes Selskab, 1952.
11. P.J. Stuckey. Negation and constraint logic programming. *Information and Computation*, 118:12–33, 1995.
12. M. van Lambalgen and F. Hamm. Intensionality and coercion. In R. Kahle, editor, *Intensionality*. ASL Lecture Notes in Logic, A.K. Peters, 2003.
13. M. van Lambalgen and F. Hamm. Moschovakis' notion of meaning as applied to linguistics. In M. Baaz and J. Krajicek, editors, *Logic Colloquium '01*. ASL Lecture Notes in Logic, A.K. Peters, 2003.
14. M. van Lambalgen and F. Hamm. *The proper treatment of events*. To appear with Blackwell Publishing, Oxford and Boston, 2004. Until publication, manuscript available at http://staff.science.uva.nl/˜michiell.