[In: Uwe Reyle (ed.)(2000) Presuppositions and Underspecification in the Computation of Temporal and other Relations in Discourse. SFB 340, Report 164. 149-200.]

Computing Discourse Relations on a Linguistic Basis

Kai-Uwe Carstensen

Abstract

This paper describes the implementation of some of the theoretical work in B9/A12 (cf. [6],[5],[4], [3]) that was developped in the final year of the projects. It can be clearly divided into two phases, which are documented in two parts of this paper, correspondingly. They can be roughly distinguished by the way discourse relations were computed by the implemented system.

The first phase was dominated by the view that possible discourse relations (cf. [5]) between sentences in a coherent text (called "very short stories", cf. [6]) can be derived from a set of relevant linguistic features and can then act as hypotheses to be proven with respect to other information available in that text and/or gained by additional inferences. Computing discourse relations thus meant creating a space of possibilities and then reducing that space to an adequate extent ("top-down approach").

The second phase was characterized by the view that an underspecified representation of a text already codes this space of possible discourse relations (represented by pertinent alternatives in UDRSes introduced by linguistic information). According to this view it is the impact of both the presuppositional and assertional information introduced by textual elements –together with the temporal constraints of extant temporal information– that leads to direct but implicit pruning of the space ("bottom-up approach", cf. [4]). Computing discourse relations here meant having all possibilities somehow available from the beginning and spending the main effort in disambiguation (cf. [3]).

Common to both phases is the maxim to (only) take linguistic aspects as far as possible into account and to ban world knowledge from all considerations (hence the title of the chapter). The computation of discourse relations is therefore restricted to a subset of all possible relations that can be derived from a set of relevant linguistic features (essentially, tense, aspect, and aktionsart features) and does not include discourse relations related to lexical items (e.g., sentence conjunctions like *although*).

From an implementation point of view the focus in phase 1 was on developping a workbench for the correct prediction of possible discourse relations, given some linguistic information of the text. The focus in phase 2 was on implementing a system based on UDRSes that handles underspecified text interpretation. Both foci somehow complement each other. As focussing always implies restrictions, however, we would have needed much more time to fully integrate the results of both phases into a general, working system.

Part 1: "Top-down approach": A Workbench for the explicit computation of Discourse Relations based on Linguistic Features

1 Introduction

A general aim of computational text interpretation is the ability to process sequences of sentences that are assumed to be coherent ("very short stories", cf. [6]) and to assign the correct (possible) discourse relations between text parts. The specific aim of this project was to compute possible discourse relations implied by the texts on the basis of a restricted set of linguistic features (e.g., tense, aspect, aktionsart) and to represent them within the paradigm of underspecified semantics. In the first project phase, this goal was restricted to building a workbench for handling discourse rules, i.e. a set of mappings from linguistic features to discourse relations, and to using the rules produced by/with the workbench for the representation of a complex discourse.

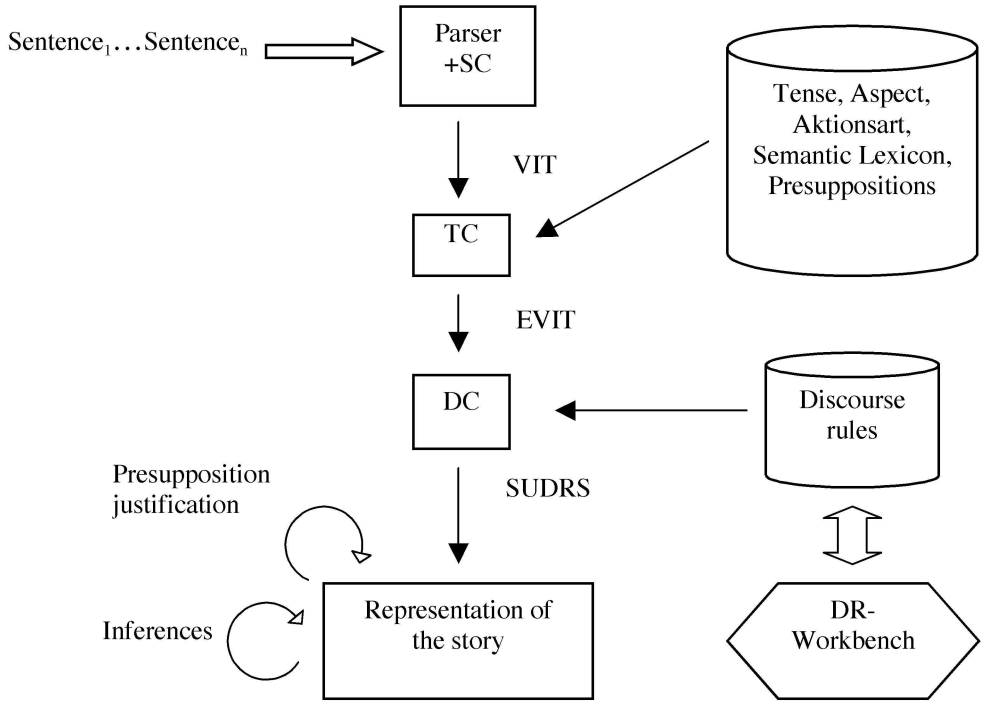

Figure 1

Figure 1 gives an overview of the system architecture we had in mind at the beginning. An LFG-parser developed at the IMS extended by a semantics construction component by Michael Schiehlen produces VITs (Verbmobil Interface Terms, cf. [1]) as output. The flat semantics coded by VITs are transformed into Extended VITs

(EVITs) by a Temporal Construction (TC) component that adds further information gained from finer grained temporal analysis and lexical semantic representations. A Discourse Construction (DC) component applies discourse rules to the EVITS. Its output are Segmented UDRSes (SUDRSes) which constitute the representation of the story (available for further inferences). However, as theoretical work on the underspecified representation for text interpretation was still in progress, we simplified this picture in phase 1 and concentrated on the workbench aspect. The resulting system whose architecture is shown in Figure 2 does without, e.g., parsing and underspecification (we come back to that in Part II). It focusses on finding the set of discourse rules such that every coherent story made up of a subset of a (systematically constructed) set of test sentences will be automatically given all and only its adequate "story readings" (i.e., sequences of discourse relations).

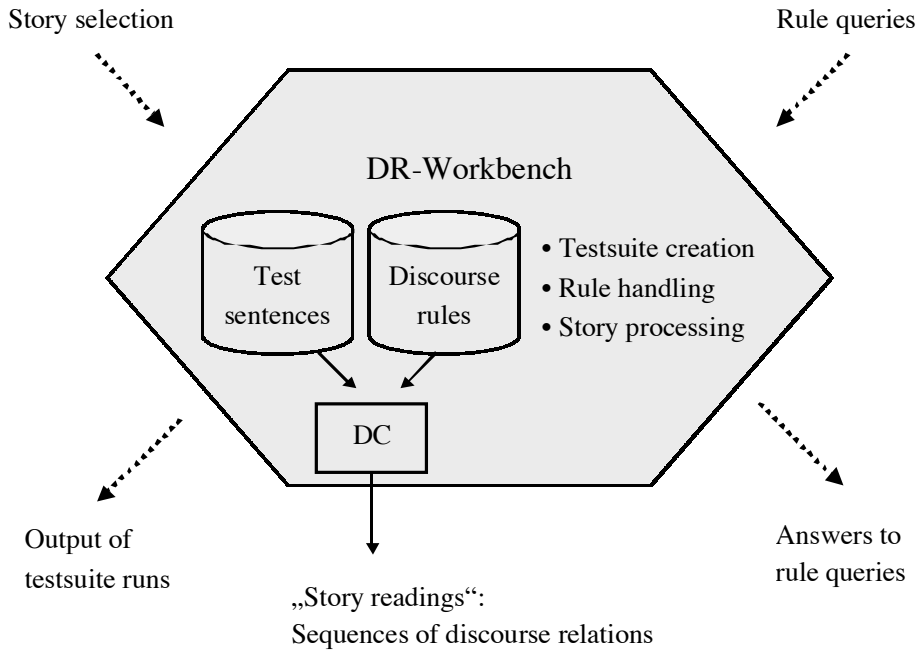

Figure 2

The questions that are supposed to be answered are

- "Which discourse relations are to be assumed (given our somewhat restricted interest in temporal phenomena)?",
- "Which parameters (linguistic features) of the discourse rules are to be assumed?", and
- "Which discourse relations are implied by certain parameter constellations?".

They are answered by running a testsuite (i.e., applying the discourse rules to the test sentences in a systematic way), by querying various aspects of the rules, or by processing user selected stories (i.e., subsets of the test sentences).

2 Discourse relations

Three types of *rhetorical relations* figure as the most prominent discourse relations: Continuation(e1, e2), Elaboration(e1, e2) and Explanation(e1, e2). Each of them implies a specific temporal relation between the arguments. The first implies temporal precedence, the second inclusion, and the third inverse precedence. We are interested in the more fine grained set of *coherence relations*, however. The relations that are currently used in the workbench (cf. [5]) are:

- 'ZERO'
- 'CAUSE/EFFECT'
- 'EFFECT/CAUSE'
- 'EFFECT/ADDCAUSE'
- 'CAUSE/RESPONSE'
- 'RESPONSE/CAUSE'
- 'FIG/BACKGROUND'
- 'BACKGROUND/FIG'
- 'FIG/FIG-BACKGROUND'
- 'MEANS/GOAL'
- 'GOAL/MEANS'
- 'GOAL/GOAL'
- 'MEANS/MEANS-GOAL'
- 'INTERFERED/INTERFERING'

An important aspect of coherence relations is the temporal relation corresponding to them. The relationship between discourse and temporal relation is represented by the following Prolog relation:

```
temporalRelationCorrespondingToDR('CAUSE/EFFECT'(A,B),contains(A,B)):-
   testsort(A,state),!.
temporalRelationCorrespondingToDR('CAUSE/EFFECT'(A,B),precedes(A,B)).
temporalRelationCorrespondingToDR('EFFECT/CAUSE'(A,B),precedes(B,A)):-
   testsort(B,event).
temporalRelationCorrespondingToDR('EFFECT/CAUSE'(A,B),contains(B,A)):-
   testsort(B,state).
temporalRelationCorrespondingToDR('CAUSE/RESPONSE'(A,B),contains(A,B)):-
   testsort(A,state),!.
temporalRelationCorrespondingToDR('CAUSE/RESPONSE'(A,B),precedes(A,B)).
temporalRelationCorrespondingToDR('RESPONSE/CAUSE'(A,B),precedes(B,A)):-
   testsort(B,event).
temporalRelationCorrespondingToDR('RESPONSE/CAUSE'(A,B),contains(B,A)):-
   testsort(B,state).
temporalRelationCorrespondingToDR('FIG/FIGBACKGROUND'(A,B),anytemprel(A,B)).
temporalRelationCorrespondingToDR('GOAL/MEANS'(A,B),contains(A,B)).
```

```
temporalRelationCorrespondingToDR('MEANS/GOAL'(A,B),precedes(A,B)).
temporalRelationCorrespondingToDR('GOAL/GOAL'(A,B),precedes(A,B)).
temporalRelationCorrespondingToDR('INTERFERED/INTERFERING'(A,B),
                                  overlaps_(A,B)).
temporalRelationCorrespondingToDR('BACKGROUND/FIG'(A,B),contains(A,B)):-
   testsort(A,state),testsort(B,event).
temporalRelationCorrespondingToDR('FIG/BACKGROUND'(A,B),contains(B,A)).
temporalRelationCorrespondingToDR('MEANS/MEANS-GOAL'(A,B),precedes(A,B)).
temporalRelationCorrespondingToDR('ZERO'(A,B),anytemprel(A,B)).
temporalRelationCorrespondingToDR('ENABLING/ENABLED'(A,B),contains(B,A)):-
   testsort(A,event),testsort(B,state).
temporalRelationCorrespondingToDR('EFFECT/ADDCAUSE'(A,B),precedes(B,A)):-
   testsort(A,event),testsort(B,event).

testsort(s(_),state):-!.
testsort(s_res(_),state):-!.
testsort(ev(_),state):-!.
testsort(e(_),event):-!.
testsort(ev(_),event):-!.
```

In the paradigm of SUDRT, discourse relations are assumed to hold between *nodes*, i.e., abstract referents for eventualities of sentences. Discourse relations are listed as two-argument predicates, where the arguments are either the nodes A or B, or eventuality functions of those nodes (remember that there might be more than one relevant eventuality per node, like the event and the resultative state of a telic sentence, and there might be a relation between a complex of nodes and another node).

Every SUDRS features both a set of *anchor points* (backward docking nodes) and a set of *open attachment points* (forward docking nodes) as defining elements (cf. [6]). More specifically, discourse relations always relate an attachment point with an anchor point. There is a restriction on this, however: Each discourse relation does not only verify the existence of two nodes to be related but may –according to its role in discourse– even modify the set of attachment points. This is accounted for by explicitly listing the operations associated with a discourse relation. The operations are

addatt(NODE):	add NODE to the set of attachment points
removeatt(NODE):	remove NODE from the set of attachment points
clearatt:	empty the set of attachment points
addgroupatt(LISTofNODES):	create a complex node from LISTofNODES and add it to the set of attachment points

The impact of a discourse relation on the set of attachment nodes is coded by the following Prolog relation:

```
impactOfRelationOnNodes('CAUSE/EFFECT'(A,B),[addatt(A),addatt(B)]):-
   testsort(A,event), testsort(B,event),!.
impactOfRelationOnNodes('CAUSE/EFFECT'(_A,B),[clearatt,addatt(B)]):-
   testsort(B,state).
```

```
impactOfRelationOnNodes('CAUSE/EFFECT'(_A,B),[clearatt,addatt(B)]):-
   testsort(B,event).
impactOfRelationOnNodes('EFFECT/CAUSE'(A,_B),[clearatt,addatt(A)]).
impactOfRelationOnNodes('CAUSE/RESPONSE'(A,B),[addatt(A),addatt(B)]):-
   testsort(A,event), testsort(B,event),!.
impactOfRelationOnNodes('CAUSE/RESPONSE'(_A,B),[clearatt,addatt(B)]):-
   testsort(B,state).
impactOfRelationOnNodes('CAUSE/RESPONSE'(_A,B),[clearatt,addatt(B)]):-
   testsort(B,event).
impactOfRelationOnNodes('RESPONSE/CAUSE'(A,_B),[clearatt,addatt(A)]).
impactOfRelationOnNodes('FIG/FIG-BACKGROUND'(A,B),
                        [removeatt(A),addatt(B),addgroupatt([A,B])]).
impactOfRelationOnNodes('GOAL/GOAL'(A,B),
                        [removeatt(A),addatt(B),addgroupatt([A,B])]).
impactOfRelationOnNodes('GOAL/MEANS'(A,B),[clearatt,addatt(A),addatt(B)]).
impactOfRelationOnNodes('MEANS/GOAL'(_A,B),[clearatt,addatt(B)]).
impactOfRelationOnNodes('INTERFERED/INTERFERING'(_A,B),
                        [clearatt,addatt(B)]).
impactOfRelationOnNodes('BACKGROUND/FIG'(_A,B),[clearatt,addatt(B)]).
impactOfRelationOnNodes('FIG/BACKGROUND'(A,_B),[clearatt,addatt(A)]).
impactOfRelationOnNodes('MEANS/MEANS-GOAL'(A,B),[removeatt(A),addatt(B)]).
impactOfRelationOnNodes('ZERO'(A,B),[clearatt,addgroupatt([A,B])]).
impactOfRelationOnNodes('ENABLING/ENABLED'(_A,B),[clearatt,addatt(B)]).
impactOfRelationOnNodes('EFFECT/ADDCAUSE'(A,B),[clearatt,addatt(A)]):-
   testsort(B,state).
impactOfRelationOnNodes('EFFECT/ADDCAUSE'(A,B),
                        [clearatt,addgroupatt([A,B])]):-
   testsort(B,event).
```

For the processing of stories it is furthermore necessary to know whether a story may end, must end, or must not end with node B of a certain discourse relation. This information about the closedness of a story is noted by the values ‚poss', ‚closed', and ‚nil' of a corresponding feature coded by the following Prolog relation.

```
closednessOfStory('CAUSE/EFFECT'(_A,_B),poss).
closednessOfStory('EFFECT/CAUSE'(_A,_B),poss).
closednessOfStory('CAUSE/RESPONSE'(_A,_B),poss).
closednessOfStory('RESPONSE/CAUSE'(_A,_B),poss).
closednessOfStory('FIG/FIG-BACKGROUND'(_A,_B),poss).
closednessOfStory('GOAL/MEANS'(_A,_B),poss).
closednessOfStory('GOAL/GOAL'(_A,_B),poss).
closednessOfStory('MEANS/GOAL'(_A,_B),closed).
closednessOfStory('INTERFERED/INTERFERING'(_A,_B),poss).
closednessOfStory('BACKGROUND/FIG'(_A,_B),poss).
closednessOfStory('FIG/BACKGROUND'(_A,_B),poss).
closednessOfStory('MEANS/MEANS-GOAL'(_A,_B),nil).
closednessOfStory('ZERO'(_A,_B),nil).
```

```
closednessOfStory('ENABLING/ENABLED'(_A,_B),closed).
closednessOfStory('EFFECT/ADDCAUSE'(_A,_B),closed).
```

3 Mapping Linguistic features to discourse relations: Discourse rules

Discourse rules determine with which discourse relations two nodes A and B (i.e., either single sentences or groups of sentences, characterized by some relevant features) can be combined. They are elements of $\mathcal{P}$(F) x $\mathcal{P}$(F) x $\mathcal{P}$(DR), where F is a set of features and DR is a set of discourse relations. Discourse rules are coded as lists of lists. The features —implicitly referring to the corresponding node A or B —are the following:

t(X):	tense(s)= X ∈ {past,pres,fut }
perf:	aspect(s)=perfective
state(X):	eventuality(s)=X_state
	with X ∈ {lexical,progressive,result,habitual,generic}
event:	eventuality(s)=event
telic:	telic(s)
action(X):	s denotes an action and the agent is X
instantaneous:	the event of s is instantaneous (i.e., is an achievement sentence)
non-X:	X does not hold

For example, the following rule

```
[ [t(past),non-perf,state(lexical)],
  [t(past),non-perf,state(lexical)],
  ['ZERO'(s(A),s(B))] ].
```

reads: "A possible discourse relation [in fact, the only possible one] between two non-perfective sentences in the past both denoting a lexical state is the ZERO relation between the states". If there are more than one relations listed, then their order reflects the "ranking"of applicability of the relations given the feature description. In the following, a representative set of discourse rules is listed (a prettyprint of the consulted discourse rules).

```
Rule 1:
   [t(past),non-perf,state(lexical)]
   [t(past),non-perf,state(lexical)]
-->[ZERO(s(_143),s(_141))]
Rule 2:
   [t(past),non-perf,event,telic,non-action]
   [t(past),non-perf,event,telic,non-action]
```

```
-->[CAUSE/EFFECT(e(_844),e(_842)),FIG/FIG-BACKGROUND(e(_844),e(_842)),
    ZERO(e(_844),e(_842))]
Rule 3:
   [t(past),non-perf,event,telic,action(_1708)]
   [t(past),non-perf,event,telic,action(_1708)]
-->[MEANS/GOAL(e(_1680),e(_1678)),GOAL/MEANS(e(_1680),e(_1678)),
    MEANS/MEANS-GOAL(e(_1680),e(_1678)),
    FIG/FIG-BACKGROUND(e(_1680),e(_1678))]
Rule 4:
   [t(past),non-perf,event,telic,non-instantaneous,non-action]
   [t(past),non-perf,event,telic,action(_2499)]
-->[CAUSE/RESPONSE(e(_2490),e(_2488)),
    INTERFERED/INTERFERING(e(_2490),e(_2488)),
    FIG/FIG-BACKGROUND(e(_2490),e(_2488)),ZERO(e(_2490),e(_2488))]
Rule 5:
   [t(past),non-perf,event,telic,instantaneous,non-action]
   [t(past),non-perf,event,telic,action(_3368)]
-->[CAUSE/RESPONSE(e(_3359),e(_3357)),
    FIG/FIG-BACKGROUND(e(_3359),e(_3357)),ZERO(e(_3359),e(_3357))]
Rule 6:
   [t(past),non-perf,event,telic,non-action]
   [t(past),non-perf,event,non-telic,action(_4172)]
-->[CAUSE/RESPONSE(e(_4163),e(_4161)),
    FIG/FIG-BACKGROUND(e(_4163),e(_4161)),ZERO(e(_4163),e(_4161))]
Rule 7:
   [t(past),non-perf,event,telic,non-instantaneous,action(_5044)]
   [t(past),non-perf,event,telic,action(_5025),not(_5044=_5025)]
-->[CAUSE/RESPONSE(e(_5009),e(_5007)),
    INTERFERED/INTERFERING(e(_5009),e(_5007)),
    FIG/FIG-BACKGROUND(e(_5009),e(_5007)),ZERO(e(_5009),e(_5007))]
Rule 8:
   [t(past),non-perf,event,telic,instantaneous,action(_5937)]
   [t(past),non-perf,event,telic,action(_5918),not(_5937=_5918)]
-->[CAUSE/RESPONSE(e(_5902),e(_5900)),
    FIG/FIG-BACKGROUND(e(_5902),e(_5900)),ZERO(e(_5902),e(_5900))]
Rule 9:
   [t(past),non-perf,event,telic,non-instantaneous,action(_6759)]
   [t(past),non-perf,event,telic,non-action]
-->[CAUSE/EFFECT(e(_6730),e(_6728)),
    INTERFERED/INTERFERING(e(_6730),e(_6728)),
    FIG/FIG-BACKGROUND(e(_6730),e(_6728))]
Rule 10:
   [t(past),non-perf,state(lexical)]
   [t(past),non-perf,event,telic,action(_7571)]
-->[CAUSE/RESPONSE(s(_7562),e(_7560)),
```

```
    BACKGROUND/FIG(s(_7562),e(_7560))]
Rule 11:
   [t(past),non-perf,state(lexical)]
   [t(past),non-perf,event,action(_8299)]
-->[BACKGROUND/FIG(s(_8290),e(_8288)),CAUSE/RESPONSE(s(_8290),e(_8288))]
Rule 12:
   [t(past),non-perf,state(lexical)]
   [t(past),non-perf,event,telic,non-action]
-->[CAUSE/EFFECT(s(_9014),e(_9012)),BACKGROUND/FIG(s(_9014),e(_9012))]
Rule 13:
   [t(past),non-perf,event,telic,non-action]
   [t(past),non-perf,state(lexical)]
-->[CAUSE/EFFECT(e(_9778),s(_9776)),EFFECT/CAUSE(e(_9778),s(_9776)),
    FIG/BACKGROUND(e(_9778),s(_9776))]
Rule 14:
   [t(past),non-perf,event,telic,non-action]
   [t(past),non-perf,event,non-telic,non-action]
-->[CAUSE/EFFECT(e(_10570),e(_10568)),
    FIG/BACKGROUND(e(_10570),e(_10568)),
    FIG/FIG-BACKGROUND(e(_10570),e(_10568))]
Rule 15:
   [t(past),non-perf,event,non-telic,non-action]
   [t(past),non-perf,event,telic,non-action]
-->[CAUSE/EFFECT(e(_11425),e(_11423)),
    BACKGROUND/FIG(e(_11425),e(_11423))]
Rule 16:
   [t(past),non-perf,event,non-telic,non-action]
   [t(past),non-perf,event,non-telic,non-action]
-->[CAUSE/EFFECT(e(_12252),e(_12250)),
    FIG/FIG-BACKGROUND(e(_12252),e(_12250))]
Rule 17:
   [t(past),non-perf,event,non-telic,action(_13145)]
   [t(past),non-perf,event,telic,non-action]
-->[CAUSE/EFFECT(e(_13116),e(_13114)),
    BACKGROUND/FIG(e(_13116),e(_13114))]
Rule 18:
   [t(past),non-perf,event,telic,action(_13948)]
   [t(past),non-perf,event,non-telic,non-action]
-->[CAUSE/EFFECT(e(_13916),e(_13914)),
    INTERFERED/INTERFERING(e(_13916),e(_13914))]
Rule 19:
   [t(past),non-perf,event,non-telic,action(_14763)]
   [t(past),non-perf,event,non-telic,action(_14741),
    not(_14763=_14741)]
-->[CAUSE/RESPONSE(e(_14725),e(_14723)),
```

```
    FIG/FIG-BACKGROUND(e(_14725),e(_14723)),ZERO(e(_14725),e(_14723))]
Rule 20:
   [t(past),non-perf,event,non-telic,action(_15654)]
   [t(past),non-perf,event,non-telic,action(_15654)]
-->[GOAL/MEANS(e(_15623),e(_15621)),
    MEANS/MEANS-GOAL(e(_15623),e(_15621)),
    GOAL/GOAL(e(_15623),e(_15621))]
Rule 21:
   [t(pres),perf,event,telic,non-action]
   [t(pres),perf,event,telic,non-action]
-->[CAUSE/EFFECT(e(_16479),s_res(_16477)),
    EFFECT/CAUSE(e(_16479),e(_16477)),
    FIG-FIG/BACKGROUND(e(_16479),e(_16477)),
    ZERO(s_res(_16479),s_res(_16477))]
Rule 22:
   [t(pres),perf,event,telic,action(_17308)]
   [t(pres),perf,event,telic,action(_17308)]
-->[BACKGROUND(n,s_res(_17283)),BACKGROUND(n,s_res(_17283)),
    CAUSE/RESPONSE(e(_17269),s_res(_17283)),
    RESPONSE/CAUSE(e(_17269),e(_17283)),
    FIG-FIG/BACKGROUND(e(_17269),e(_17283)),
    ZERO(s_res(_17269),s_res(_17283))]
Rule 23:
   [t(past),event,telic,non-action]
   [t(past),perf,event]
-->[BACKGROUND(t,s_res(_18048)),EFFECT/CAUSE(e(_18041),e(_18048)),
    FIG/BACKGROUND(e(_18041),s_res(_18048))]
Rule 24:
   [t(past),perf,event,telic,non-action]
   [t(past),perf,event,non-telic,action(_18735)]
-->[CAUSE/RESPONSE(e(_18726),e(_18724))]
Rule 25:
   [t(past),event,telic,action(_19456)]
   [t(past),perf,event,telic,action(_19456)]
-->[BACKGROUND(t,s_res(_19431)),MEANS/MEANS-GOAL(e(_19424),e(_19431)),
    FIG/BACKGROUND(e(_19424),s_res(_19431))]
Rule 26:
   [t(past),perf,event,non-action]
   [t(past),event,non-telic,action(_20118)]
-->[CAUSE/RESPONSE(s_res(_20109),e(_20107)),
    BACKGROUND/FIG(s_res(_20109),e(_20107))]
Rule 27:
   [t(past),perf,event,telic,non-action]
   [t(past),perf,event,telic,instantaneous,non-action]
-->[CAUSE/EFFECT(e(_20836),e(_20834)),
```

```
    EFFECT/CAUSE(e(_20836),e(_20834)),BACKGROUND(e(_20836),e(_20834))]
Rule 28:
   [t(past),perf,event,telic,action(_21632)]
   [t(past),perf,event,telic,non-instantaneous,action(_21611)]
-->[GOAL/MEANS(e(_21602),e(_21600)),MEANS/GOAL(e(_21602),e(_21600)),
    MEANS/MEANS-GOAL(e(_21602),e(_21600)),
    FIG/FIG-BACKGROUND(e(_21602),e(_21600))]
Rule 29:
   [t(past),perf,event,telic,non-action]
   [t(past),non-perf,event,telic,instantaneous,non-action]
-->[CAUSE/EFFECT(e(_22370),e(_22368)),EFFECT/CAUSE(e(_22370),e(_22368)),
    FIG-FIG/BACKGROUND(e(_22370),e(_22368))]
Rule 30:
   [t(past),non-perf,event,non-telic,non-action]
   [t(past),perf,event,telic]
-->[EFFECT/CAUSE(e(_23146),e(_23144))]
Rule 31:
   [t(past),complex]
   [t(past),state(lexical)]
-->[CAUSE/EFFECT(complex(_23858),s(_23856))]
Rule 32:
   [t(past),complex]
   [t(past),event,non-action]
-->[CAUSE/EFFECT(complex(_24449),e(_24447))]
Rule 33:
   [t(past),complex]
   [t(past),event,action(_25080)]
-->[CAUSE/RESPONSE(complex(_25071),e(_25069))]
```

4 Test sentences

For test sentences to be available for the DR-workbench, they must be manually entered into a corresponding file. The format is ‚[Char,String,List].‘ where Char is an token identifier distinguishing between sentences with the same feature description type, String is the test sentence, and List is an element of $\mathcal{P}$(F) describing String. The following are sample entries:

```
[a,'Es war kalt',[t(past),state(lexical)]].
[b,'Der Mann war wuetend',[t(past),state(lexical)]].
[c,'Die ganze Bevoelkerung war vor dem Rathaus versammelt',
    [t(past),state(lexical)]].
[a,'Es ist kalt gewesen',[t(pres),perf,state(lexical)]].
[b,'Der Mann ist wuetend gewesen',[t(pres),perf,state(lexical)]].
```

Note that, different from the specification of the feature descriptions in the discourse rules, features missing in List are interpreted as negatively valued.

5 The workbench

By looking at the discourse rules as defined by the linguist it becomes immediately obvious that it is difficult to keep the overview with respect to the correctness and completeness of the rule set. This gave rise to the idea of implementing a workbench that offers tools for improving the quality according to these criteria. The currently implemented procedures concern

- listings (of test sentences and discourse rules)
- search for relations (in which rules does a certain relation appear?)
- qualitative search (which rules match a certain feature pattern?)
- similarity (which rules are similar to a certain one?)
- information about rules (possible subsumption relation to other rules)
- generalization of rules (can the rules in which a certain relation appears be generalized?).

By consulting the test suite, the test sentences and discourse rules are compiled and made available for the workbench. Both are numbered such that they can be uniquely addressed for rule queries and story processing.

Listings. The command `drlist` lists all discourse rules. The command tslist enumerates all test sentences by adding the internal unique numerical identifier:

```
[1,a,Es war kalt,[t(past),state(lexical)]]
[2,b,Der Mann war wtend,[t(past),state(lexical)]]
[3,c,Die ganze Bevoelkerung war vor dem Rathaus versammelt,
     [t(past),state(lexical)]]
[4,a,Es ist kalt gewesen,[t(pres),perf,state(lexical)]]
[5,b,Der Mann ist wuetend gewesen,[t(pres),perf,state(lexical)]]
[6,a,Es war kalt gewesen,[t(past),perf,state(lexical)]]
[7,b,Der Mann war wuetend gewesen,[t(past),perf,state(lexical)]]
[8,a,Das Ding machte Geraeusche,[t(past),event]]
[9,b,Der Mann laechelte,[t(past),event]]
[10,a,Das Ding hat Geraeusche gemacht,[t(pres),perf,event]]
[11,b,Der Mann hat gelaechelt,[t(pres),perf,event]]
[12,a,Das Ding hatte Geraeusche gemacht,[t(past),perf,event]]
[13,b,Der Mann hatte gelaechelt,[t(past),perf,event]]
[14,a,Maria rannte,[t(past),event,action(Maria)]]
[15,b,Peter dachte angestrengt nach,[t(past),event,action(Peter)]]
[16,a,Paul laechelte freundlich,[t(past),event,action(Paul)]]
[17,b,Paul dachte angestrengt nach,[t(past),event,action(Paul)]]
[18,c,Die Maenner schrien,[t(past),event,action(Die Mnner)]]
[19,d,Die Frauen weinten,[t(past),event,action(Die Frauen)]]
[20,e,Die Soldaten schossen in die Luft,[t(past),event,action(Die Soldaten)]]
[21,a,Peter hat freundlich gelaechelt,[t(pres),perf,event,action(Peter)]]
```

```
[22,b,Peter hat angestrengt nachgedacht,[t(pres),perf,event,action(Peter)]]
[23,a,Paul hat freundlich gelaechelt,[t(pres),perf,event,action(Paul)]]
[24,b,Paul hat angestrengt nachgedacht,[t(pres),perf,event,action(Paul)]]
[25,a,Peter hatte freundlich gelaechelt,[t(past),perf,event,action(Peter)]]
[26,b,Peter hatte angestrengt nachgedacht,[t(past),perf,event,action(Peter)]]
[27,a,Paul hatte freundlich gelaechelt,[t(past),perf,event,action(Paul)]]
[28,b,Paul hatte angestrengt nachgedacht,[t(past),perf,event,action(Paul)]]
[29,a,Die Sonne ging unter,[t(past),event,telic]]
[30,b,Der Mann wurde wuetend,[t(past),event,telic]]
[31,c,Die Whisky-Flasche rollte vom Tisch herunter,[t(past),event,telic]]
[32,d,Adrenalin scho in seinen Krper,[t(past),event,telic]]
[33,a,Es ist kalt geworden,[t(pres),perf,event,telic]]
[34,b,Der Mann ist wuetend geworden,[t(pres),perf,event,telic]]
[35,a,Es war kalt geworden,[t(past),perf,event,telic]]
[36,b,Der Mann war wuetend geworden,[t(past),perf,event,telic]]
[37,a,Maria erreichte die Bushaltestelle,
      [t(past),event,instantaneous,telic,action(Maria)]]
[38,b,Peter lief los,[t(past),event,instantaneous,telic,action(Peter)]]
[39,a,Maria ging zur Bushaltestelle,[t(past),event,telic,action(Maria)]]
[40,b,Der Mann fing die Flasche auf,[t(past),event,telic,action(Mann)]]
[41,c,Verrgert rief er nach einer neuen Flasche,
      [t(past),event,telic,action(_110)]]
[42,a,Peter ist losgelaufen,
      [t(pres),perf,event,instantaneous,telic,action(Peter)]]
[43,b,Peter hat sich gesetzt,[t(pres),perf,event,telic,action(Peter)]]
[44,a,Paul ist losgelaufen,
      [t(pres),perf,event,instantaneous,telic,action(Paul)]]
[45,b,Paul hat sich gesetzt,[t(pres),perf,event,telic,action(Paul)]]
[46,a,Peter war losgelaufen,
      [t(past),perf,event,instantaneous,telic,action(Peter)]]
[47,b,Peter hatte sich gesetzt,[t(past),perf,event,telic,action(Peter)]]
[48,a,Paul war losgelaufen,
      [t(past),perf,event,instantaneous,telic,action(Paul)]]
[49,b,Paul hatte sich gesetzt,[t(past),perf,event,telic,action(Paul)]]
[50,a,Der Bus ist gekommen,[t(pres),perf,event,telic]]
[51,b,Die Bahn ist weggefahren,[t(pres),perf,event,telic]]
[52,a,Der Bus war gekommen,[t(past),perf,event,telic]]
[53,b,Die Bahn war weggefahren,[t(past),perf,event,telic]]
[54,a,Es fing an zu regnen,[t(past),event,telic,instantaneous]]
[55,b,Die Bahn fuhr ab,[t(past),event,telic,instantaneous]]
[56,c,Die Fensterscheibe zerbarst,[t(past),event,telic,instantaneous]]
[57,a,Es hat angefangen zu regnen,[t(pres),perf,event,telic,instantaneous]]
[58,b,Die Bahn ist abgefahren,[t(pres),perf,event,telic,instantaneous]]
[59,a,Es hatte angefangen zu regnen,[t(past),perf,event,telic,instantaneous]]
[60,b,Die Bahn war abgefahren,[t(past),perf,event,telic,instantaneous]]
```

Search for relations. The procedure `occursWhere(DR-SUBSTRING)` collects all discourse rules that contain discourse relations matching `DR-SUBSTRING`:

```
| ?- occursWhere('Interf').
```

```
Rules found: [4,7,9,18]
Want to view them? (Return)

Rule 4:
   [t(past),non-perf,event,telic,non-instantaneous,non-action]
   [t(past),non-perf,event,telic,action(_1223)]
   [CAUSE/RESPONSE(e(_1214),e(_1212)),INTERFERED/INTERFERING(e(_1214),
    e(_1212), FIG/FIG-BACKGROUND(e(_1214),e(_1212)),ZERO(e(_1214),e(_1212))]
Rule 7:
   [t(past),non-perf,event,telic,non-instantaneous,action(_2103)]
   [t(past),non-perf,event,telic,action(_2084),not(_2103=_2084)]
   [CAUSE/RESPONSE(e(_2068),e(_2066)),INTERFERED/INTERFERING(e(_2068),
    e(_2066), FIG/FIG-BACKGROUND(e(_2068),e(_2066)),ZERO(e(_2068),e(_2066))]
    .
    .
    .
```

Qualitative Search. The procedure `drmatch/[1,2]` collects all discourse rules that contain discourse relations matching its feature description arguments (in the single-argument version, only matches with the FD of node A are performed):

```
| ?- drmatch([non-perf,instantaneous]).

Rule 5:
   [t(past),non-perf,event,telic,instantaneous,non-action]
   [t(past),non-perf,event,telic,action(_334)]
   [CAUSE/RESPONSE(e(_325),e(_323)),FIG/FIG-BACKGROUND(e(_325),e(_323)),
    ZERO(e(_325),e(_323))]
Rule 8:
   [t(past),non-perf,event,telic,instantaneous,action(_366)]
   [t(past),non-perf,event,telic,action(_347),not(_366=_347)]
   [CAUSE/RESPONSE(e(_331),e(_329)),FIG/FIG-BACKGROUND(e(_331),e(_329)),
    ZERO(e(_331),e(_329))]

| ?- drmatch([state(lexical)],[event,action(_)]).

Rule 10:
   [t(past),non-perf,state(lexical)]
   [t(past),non-perf,event,telic,action(_353)]
   [CAUSE/RESPONSE(s(_344),e(_342)),BACKGROUND/FIG(s(_344),e(_342))]
Rule 11:
   [t(past),non-perf,state(lexical)]
   [t(past),non-perf,event,action(_333)]
   [BACKGROUND/FIG(s(_324),e(_322))]
```

Similarity. It might be interesting to know which rules are similar to a certain rule and to get the results in increasing distance. This functionality is provided by the procedure `nearRules(RuleID)` which is based on the computation of distance between rules at compile time. It presents the results in qualitative steps:

```
| ?- nearRules(18).

*********
Rule 18:
   [t(past),non-perf,event,telic,action(_223)]
   [t(past),non-perf,event,non-telic,non-action]
   [CAUSE/EFFECT(e(_191),e(_189)),INTERFERED/INTERFERING(e(_191),e(_189))]
*********
Rule distance value: 2:

Rule 14:
   [t(past),non-perf,event,telic,non-action]
   [t(past),non-perf,event,non-telic,non-action]
   [CAUSE/EFFECT(e(_1427),e(_1425)),FIG/BACKGROUND(e(_1427),e(_1425)),
    FIG/FIG-BACKGROUND(e(_1427),e(_1425))]

More distant rules? (Return)
|:

Rule distance value: 3:

Rule 9:
   [t(past),non-perf,event,telic,non-instantaneous,action(_1482)]
   [t(past),non-perf,event,telic,non-action]
   [CAUSE/EFFECT(e(_1453),e(_1451)),INTERFERED/INTERFERING(e(_1453),
    e(_1451)), FIG/FIG-BACKGROUND(e(_1453),e(_1451))]

More distant rules? (Return)
|:
```

Information about rules. The procedure `drinfo(RuleID)` prints the rule identified by `RuleID` (the numerical identifier) and in addition shows subsumption information wrt. other rules, which is provided by the computation of subsumption relations between the first feature descriptions of rules at compile time:

```
| ?- drinfo(23).

*********
[23,[t(past),event,telic,non-action],
    [t(past),perf,event],
    [BACKGROUND(t,s_res(_182)),EFFECT/CAUSE(e(_175),e(_182)),
     FIG/BACKGROUND(e(_175),s_res(_182))]]

subsumes:
-[24,[t(past),perf,event,telic,non-action],
     [t(past),perf,event,non-telic,action(_160)],
     [CAUSE/RESPONSE(e(_151),e(_149))]]
```

Automatic Generalization of rules. Sometimes it might be interesting to get automatic proposals for generalizations of a set of rules in which a certain discourse relation

appears by looking at the generalizations of increasingly smaller subsets of this set. This functionality is provided by the procedure `msg(DR)`:

```
| ?- msg('EFFECT/CAUSE').

Rules found:  [13,21,23,27,29,30]
Considering rules [13,21,23,27,29,30]
Considering rules [21,23,27,29,30]

Generalized rule: [event,non-action] & [event] -> !!!
Want more proposals? (Return)

Considering rules [13,23,27,29,30]

Generalized rule: [event,t(past),non-action] & [t(past)] -> !!!
Want more proposals? (Return)

Considering rules [13,21,27,29,30]
Considering rules [13,21,23,29,30]
Considering rules [13,21,23,27,30]
Considering rules [13,21,23,27,29]
Considering rules [23,27,29,30]

Generalized rule: [event,t(past),non-action] & [event,t(past)] -> !!!
Want more proposals? (Return)

Considering rules [21,27,29,30]

Generalized rule: [event,non-action] & [event,telic] -> !!!
Want more proposals? (Return)
```

5.1 Running the testsuite

In order to get a quick overview where the holes and/or bugs of the discourse rules might be, an automatic application of the discourse rules on the test sentences may be performed. To do this, the corresponding procedure iteratively tries to apply the rules to each test sentence pair with different token identifier and outputs the results in the following form (all possible rule applications are listed):

```
**********
Es war kalt. (1. Satz)

1 ---- Der Mann war wuetend.
       ZERO(s(_886),s(_884)) (Rule 1)
2 ---- Die ganze Bevoelkerung war vor dem Rathaus versammelt.
       ZERO(s(_886),s(_884)) (Rule 1)
3 ---- Der Mann war wuetend gewesen.
4 ---- Der Mann laechelte.
5 ---- Der Mann hatte gelaechelt.
```

```
6 ---- Peter dachte angestrengt nach.
       BACKGROUND/FIG(s(_897),e(_895)) (Rule 11)
7 ---- Paul dachte angestrengt nach.
       BACKGROUND/FIG(s(_897),e(_895)) (Rule 11)
8 ---- Die Maenner schrien.
       BACKGROUND/FIG(s(_897),e(_895)) (Rule 11)
9 ---- Die Frauen weinten.
       BACKGROUND/FIG(s(_897),e(_895)) (Rule 11)
10 --- Die Soldaten schossen in die Luft.
       BACKGROUND/FIG(s(_897),e(_895)) (Rule 11)
11 --- Peter hatte angestrengt nachgedacht.
12 --- Paul hatte angestrengt nachgedacht.
13 --- Der Mann wurde wuetend.
       CAUSE/EFFECT(s(_895),e(_893)) (Rule 12)
       BACKGROUND/FIG(s(_895),e(_893)) (Rule 12)
14 --- Die Whisky-Flasche rollte vom Tisch herunter.
       CAUSE/EFFECT(s(_895),e(_893)) (Rule 12)
       BACKGROUND/FIG(s(_895),e(_893)) (Rule 12)
15 --- Adrenalin scho in seinen Krper.
       CAUSE/EFFECT(s(_895),e(_893)) (Rule 12)
       BACKGROUND/FIG(s(_895),e(_893)) (Rule 12)
16 --- Der Mann war wuetend geworden.
17 --- Peter lief los.
       BACKGROUND/FIG(s(_901),e(_899)) (Rule 10)
       BACKGROUND/FIG(s(_901),e(_899)) (Rule 11)
18 --- Der Mann fing die Flasche auf.
       BACKGROUND/FIG(s(_899),e(_897)) (Rule 10)
       BACKGROUND/FIG(s(_899),e(_897)) (Rule 11)
19 --- Verrgert rief er nach einer neuen Flasche.
       BACKGROUND/FIG(s(_899),e(_897)) (Rule 10)
       BACKGROUND/FIG(s(_899),e(_897)) (Rule 11)
20 --- Peter hatte sich gesetzt.
21 --- Paul hatte sich gesetzt.
22 --- Die Bahn war weggefahren.
23 --- Die Bahn fuhr ab.
       CAUSE/EFFECT(s(_897),e(_895)) (Rule 12)
       BACKGROUND/FIG(s(_897),e(_895)) (Rule 12)
24 --- Die Fensterscheibe zerbarst.
       CAUSE/EFFECT(s(_897),e(_895)) (Rule 12)
       BACKGROUND/FIG(s(_897),e(_895)) (Rule 12)
25 --- Die Bahn war abgefahren.
.
.
.
```

5.2 Processing stories

Yet another method of verifying completeness and correctness of the discourse rules is to process meaningful sequences of test sentences and see whether the results agree

with theoretical assumptions. This requires a more elaborated environment which implements SUDRSes with anchor and attachment points and in which the impact of discourse relations on attachment points as well as the constraints on the closedness of a story are respected (cf. [6]).

The main predicate, `stories(ListOfSentences)`, provides this functionality. Its basis is a representation of a discourse as a set of SUDRSes, where each SUDRS is a six-argument Prolog term

```
sudrs(AnchorNodes,Relations,AttachmentNodes,FeaturesOfSentence,
      TemporalFeatures,Closedness).
```

The main program structure for stories is the following:

```
stories([],D):-
    !,
    outputStories(D).
stories([N|R],D):-
    processNewSentence(N,Sudrs,D),
    determineRelationToDiscourse(Sudrs,D),!,
    stories(R,D).
```

Recursively, each of the specified sentences is processed by retrieving the features of a test sentence N, building new nodes for each of the involved eventualities, creating a new Sudrs and filling its arguments with the pertinent information, and then determining the relation of this Sudrs with the previous discourse.

Due to the possibly destructive operations on the set of attachment points, an Sudrs cannot underspecify the possible developments of the discourse. Therefore, each Sudrs can only code a single discourse history, accumulating the nodes and relations on that line. A discourse is then a tree with Sudrses as nodes, whose leafs are either expanded or deleted, depending on the possibility to anchor a node of the following sentence. The output of the stories is simply a traversal of this tree from root to leaves printing out the discourse relation on the way. Here is an example of story processing:

```
| ?- stories([29,30,15]).

Die Sonne ging unter. Der Mann wurde wuetend. Peter dachte angestrengt nach.

***
Possible story 1:

ZERO(e(n1),e(n2))
CAUSE/RESPONSE(complex([e(n1),e(n2)]),e(n3))

***
Possible story 2:

FIG/FIG-BACKGROUND(e(n1),e(n2))
```

```
FIG/FIG-BACKGROUND(e(n2),e(n3))

***
Possible story 3:

FIG/FIG-BACKGROUND(e(n1),e(n2))
CAUSE/RESPONSE(e(n2),e(n3))

***
Possible story 4:

FIG/FIG-BACKGROUND(e(n1),e(n2))
CAUSE/RESPONSE(complex([e(n1),e(n2)]),e(n3))

***
Possible story 5:

CAUSE/EFFECT(e(n1),e(n2))
FIG/FIG-BACKGROUND(e(n2),e(n3))

***
Possible story 6:

CAUSE/EFFECT(e(n1),e(n2))
FIG/FIG-BACKGROUND(e(n1),e(n3))

***
Possible story 7:

CAUSE/EFFECT(e(n1),e(n2))
CAUSE/RESPONSE(e(n2),e(n3))

***
Possible story 8:

CAUSE/EFFECT(e(n1),e(n2))
CAUSE/RESPONSE(e(n1),e(n3))
```

After having found an adequate set of discourse rules, the method for story processing presented here could have been easily integrated into the system architecture depicted in Figure 1. By simply changing from test sentences to EVITs in processNewSentence, the results could have been used as hypotheses to be proven by the system (be they underspecified or not). We did not continue this line of research, however, but switched to the bottom-up view in order to investigate what exactly is involved in disambiguating ambiguous underspecified text representations.

Part II: "Bottom-up approach": Implementing text interpretation with underspecification and computing discourse relations implicitly

6 Introduction

We discontinued the approach presented in Part I because it turned out that, at that time, there was enough theoretical knowledge at hand to try to implement an integrated, bottom-up approach to text interpretation during the last six months of the project. With theoretical work still going on, this proved to be an interesting enterprise.

Although text interpretation effectively requires a full natural language processing system, we were interested in a restricted set of phenomena only. We therefore made some shortcuts with respect to the generality of the treatment of other phenomena that will be marked accordingly. As we could use the LFG parser developed at the IMS and the semantic construction module created by Michael Schiehlen, things turned out to be quite manageable.

In addition to that, we cut back on the generality of treating discourse relations: Our primary goal became the implementation of the "Busbeispiel", the classic example of discussions in the project and topic of the meticulous theoretical analysis in [3].

7 Overview

The blueprint for the system dealing with temporal underspecification in discourse (cf. [3]) presented here is shown in Figure 3 (which is only superficially similar to the one shown in Figure 1). Parsed input sentences are extended by a flat compositional semantics resulting in VITs. These are input to the temporal construction (TC) component that enriches them with finer grained (temporal and lexical) semantic information (cf. [7]). The TC component passes instances of an interface format ("Interface Format Structures", IFSes) to the discourse construction (DC) component, which adds Internal UDRSes (IUDRSes) to an underspecified representation of the text (i.e, of the "very short story"). It is the central issue of our approach that, although ambiguous and underspecified representations of the single sentences accumulate, ambiguity and underspecification of the whole text is systematically reduced by various factors during discourse processing, among them presupposition justification, the resolution of temporal constraints, and the application of interpretation principles.

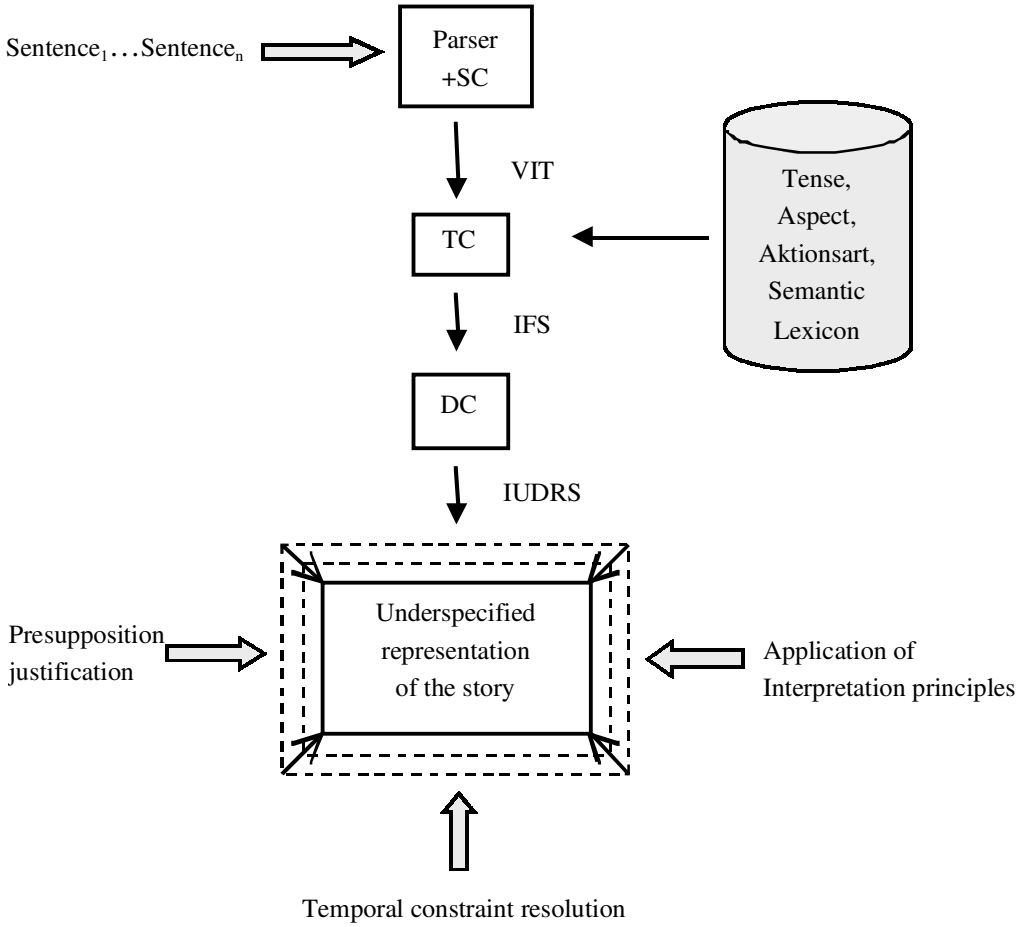

Figure 3

8 Decisions made for a system implementing underspecified text interpretation

There were a lot of general decisions regarding algorithms and data structures we had to come to at the beginning. The first concerned the question of how to represent *discourse* information as opposed to information contained in single sentences. Note that VITs were designed to code sentence information and that this meant encapsulation of data one would have rather had easily available for discourse procedures. We therefore decided to use pure Prolog for coding discourse information and to compile the information contained in the VITs into corresponding Prolog predicates ("Internal

UDRSes", IUDRSes).

Another decision concerned the question of when and how to expand or transform the VIT in the course of integrating temporal and decompositional lexical semantic information. We soon dismissed the possibility of changing the VITs with and within the VIT formalism as this proved to be too inflexible for our purposes. Instead we developed an own rule-based mechanism for building enriched UDRSes out of VITs (as described in Saric, this volume). At the same time we specified a format for interface structures (IFS) between TC and DC because these components were developped separately and in parallel. Note in passing that we did not deal with aspects of *segmented* discourse (that is, with SUDRSes) in phase 2.

Within the paradigm of underspecification, it is nothing but common to stay monotonic as far as possible. Although there is a distinction between *secure* and *ambiguous* information (cf. [3]), even the latter (by making alternatives explicit) can be treated declaratively. We decided that all indefeasible information was to be implemented by way of assertion to the Prolog database. What is straightforward in the case of lexical ambiguity, however, is less clear in the case of, say, choices of antecedents for anaphora. Although one could perhaps collect alternative antecedents, represent the resulting ambiguity declaratively and so postpone choice until a later point of time (as is done in [8]) we opted for treating these cases defeasibly and for implementing this with Prolog backtracking.

As to presupposition and anaphora, we noted their similarity (in the tradition of van der Sandt, cf. [9]) and implemented them with the same mechanism. Both require finding adequate antecedents in discourse. They are dissimilar, however, in that only presuppositions can be justified by inferences. By distinguishing *features* and *conditions* in the implementation, we could capture this distinction quite naturally: Only proofs of conditions can give rise to the triggering of inferences, but only presuppositions can have conditions. Besides that, we chose to implement only a simplified version of handling presuppositions and anaphora, and disregarded, e.g., sophisticated aspects of the choice between different accommodation sites, of bridging, and of centering theory.

Concerning the representation and processing of ambiguities, one has to ensure that, after having made a choice wrt. a certain ambiguity, only the selected disjunct will be available for further processing. We therefore chose to use a context mechanism which names each alternative (with a context variable), stores the relation of content and its licensing context, and performs a bookkeeping of current context variables that only admits licensed content during story processing.

With respect to theorem proving we took the pragmatic approach of implementing the proofs we needed instead of using or implementing a general theorem prover.

9 Discourse construction

9.1 Interface format structures (IFSes)

We used interface format structures as a bridge between TC and DC because the rule mechanism for transforming VITs and the discourse processing mechanism operate

on different data structures (and were developped in parallel, as mentioned before). IFSes represent a sentence's (or node's) UDRS as known from the literature but in addition contain presuppositional, ambiguity and feature information.

All DRSes are marked (with "P-Marker") as to whether they represent assertional ("a") or presuppositional (p(Label)) information. In the latter case, the Label points to the DRS the presuppositional DRS is attached to.

Unlike the coding of a DRS's universe in VITs, discourse referents introduced by a DRS are not only explicitly listed, but there is also an additional slot for their (linguistic) features. Features are different from conditions in that they are string-valued.

On the level of IFSes, ambiguity information is represented as exclusive disjunction of DRSes corresponding to the ambiguity operator '$\dot{\vee}$'of Reyle/Roßdeutscher ([3]). Each disjunction is coded as a list of lists of equations between label variables and labels.

The following Backus-Naur form describes IFSes:

```
IFS             ::=  "node(" NodeId "," UDRS ")"
NodeId          ::=  Identifier for Nodes
UDRS            ::=  "udrs(" DRSList  "," CONSTRAINTS "," AMBIGUITIES ")"
DRSList         ::=  List of LabelledDRS
LabelledDRS     ::=  Label ":" DRS
Label           ::=  Identifier for DRSes
DRS             ::=  "drs(" P-Marker "," UNIVERSE "," CONDITIONS ")"
P-Marker        ::=  "p(" Label ")" | "a"
UNIVERSE        ::=   "[" DRefs "," FEATURES "]"
DRefs           ::=  List of DRefId
DRefId          ::=  Identifier for discourse referents
FEATURES        ::=  List of FEATURE
FEATURE         ::=  FeatureName "(" FARG "," FVAL ")"
FeatureName     ::=  {sort, gend, num, ...}
FARG            ::=  DRefId
FVAL            ::=  String
CONDITIONS      ::=  List of CONDITION
CONDITION       ::=  ConditionName "(" CONDARGS")"
ConditionName   ::=  {move, precedes, ...}
CONDARGS        ::=  CONDARG "," CONDARGS
CONDARGS        ::=  CONDARG
CONDARG         ::=  DRefId | Label
CONSTRAINTS     ::=  List of CONSTRAINT
CONSTRAINT      ::=  ConstraintName1 "(" CONSTARG ")" |
                     ConstraintName2 "(" CONSTARG "," CONSTARG ")"
ConstraintName1 ::=  {toplabel, bottomlabel}
ConstraintName2 ::=  {leq, eq, less}
CONSTARG        ::=  Label | LabelVar
LabelVar        ::=  Variable ranging over Label
AMBIGUITIES     ::=  List of XOR
XOR             ::=  "xor(" LLVA ")"
LLVA            ::=  List of LVA
```

```
LVA             ::=  List of VAREQUATION
VAREQUATION     ::=  LabelVar "=" Label
```

The corresponding abstract pattern of an IFS is this:

```
node(N,
     udrs([                    %DRSes
           Label:drs(P-MARKER,   %Marker for assertion/presupposition
                     [           %Universe
                      [...],       %Discourse referents
                      [...]        %Features
                     ],
                     [...]       %Conditions
                    )
                        .
                        .
                        .
          ],
          [...],               %Constraints
          [...]                %Ambiguity information
         )
    ).
```

Within an IFS, identifiers for discourse referents and labels can be named arbitraryly as they will be renamed during the construction of IUDRSes. The following shows the IFS for the sentence "Maria geht zum Bahnhof. (Maria walks to the railway station.)".[1]

```
node(1,
     udrs([l1:drs(a,
                  [[],
                   []],
                  [decl(l012)]),
           l2:drs(a,
                  [[],
                   []],
                  []),
           l3:drs(p(l4),
                  [[bh],
                   [sort(bh,location),gend(bh,masc), num(bh,sg)]],
                  [bahnhof(bh)]),
           l4:drs(a,
                  [[],
                   []],
                  []),
```

[1]Observe that we make heavy use of additional, reificational arguments of the predicates of DRS conditions (i.e., even of the temporal ones). At least from an implementational point of view, this is important, as it allows us to easily access conditions of a specific type by checking the sorts of these arguments. See the subsection on proving temporal theorems below for more on this topic (and for a detailed characterization of the temporal relations used).

```
l5:drs(p(l6),
       [[m],
        [sort(m,person),gend(m,fem)]],
       []),
l6:drs(a,
       [[],
        []],
       []),
l7:drs(a,
       [[],
        [named(m,"maria")]],
       []),
l8:drs(p(l10),
       [[s1pr],
        [sort(s1pr,state)]],
       ['NEG'(l89)]),
l9:drs(a,
       [[],
        []],
       ['AT'(s1pr,m,bh)]),
l10:drs(a,
        [[s,e1,t1,t2,c1,n,tr1,tr2,tr3,tr4,tr5,tr6],
         [sort(s,state),sort(e1,event), sort(t1,tt),
          sort(t2,tf), sort(c1,cut),sort(n,now),
          sort(tr1,temprel), sort(tr2,temprel),
          sort(tr3,temprel), sort(tr4,temprel),
          sort(tr5,temprel), sort(tr6,temprel), mode(s,"gehen")]],
        ['PROG'(s,e1,l1011),move(s),arg1(s,m), zu_prep(e1,bh),
         endsNotBeforeEndOf(tr1,s1pr,s),
         startsBeforeStartOf(tr2,s1pr,s),contains(tr3,t2,t1),
         contains(tr4,t1,s),precedes(tr5,t1,n),precedes(tr6,t2,n)]),
l11:drs(a,
        [[],
         []],
        ['BEC'(e1,s1,l1112)]),
l12:drs(a,
        [[],
         []],
        ['AT'(s1,m,bh)]),
l14:drs(a,
        [[tr7],
         [sort(tr7,temprel)]],
        [contains(tr7,s,c1)]),
l15:drs(a,
        [[e,tr8,tr9,tr10],
         [sort(e,event), sort(tr8,temprel), sort(tr9,temprel),
          sort(tr10,temprel)]],
        ['BEC'(e,s1,l1516), prog(e,s),endsWithEndOf(tr8,s1pr,e),
         contains(tr9,t1,e), precedes(tr10,e,c1)]),
```

```
        l16:drs(a,
                [[],
                 []],
                ['AT'(s1,m,bh)])
        ],
        [toplabel(l1), bottomlabel(l10), leq(l4,l2), leq(l6,l2),leq(l7,l6),
         leq(l10,l4), leq(l2,l012), leq(l10,l6), leq(l9,l89), leq(l11,l1011),
         leq(l12,l1112),leq(l16,l1516), less(l012,l1), less(l89,l8),
         less(l1011,l10), less(l1112,l11), less(l1516,l15),leq(L,l10)],
        [xor([[L=l14],[L=l15]])]
       )
).
```

9.2 The internal representation of UDRSes

IFSes are the input to the DC component which transforms these molecular structures into atomic pieces of information that are merged with those of the previous discourse and thus are made available for general procedures of discourse processing. IUDRSes are therefore distributed data structures implemented as dynamic Prolog predicates. Ease of access to information determined that the predicate names may not be specific. Here is how specific types of information are coded:

- Conditions:
  `cond(DRefId, CondLABEL, LABEL, Predicate, Contexts)`
  where
  arg1 is the referential, or otherwise external, variable of Predicate,
  arg2 is the label of the condition (a relict of VIT processing)
  arg3 is the group label (the label of the DRS containing the condition)
  arg4 is the DRS condition, e.g., move(e, x)
  arg5 is the set of contexts set for this condition

- Anchoring of discourse referents:
  `d_in(DRefId, LABEL, Contexts)`

- Features:
  `feat(FARG, FeatureName, FVAL)`

- Constraints (label relations):
  `lr(LABEL, ConstraintName2 , LABEL, Contexts)`
  `toplabel(LABEL)`
  `bottomlabel(LABEL)`

9.3 The discourse construction (DC) component

Discourse construction consists of adding IUDRSes as non-defeasible information to the representation of previous discourse. The global structure of this procedure of compiling IFSes (implemented by `addToDiscourse/2`) is as follows:

```
addToDiscourse(udrs(DRSes,Constraints,AmbInfo),PresupList):-
    compileAmbiguities(AmbInfo,[]-Bind1),
    generateGroupLabel(DRSes,Constraints,Bind1-Bind2),
    compileConstraints(Constraints,Bind2),
    collectPresuppositions(DRSes,Bind2,[]-PresupList),
    member(toplabel(L),Constraints),!,
    changeConstraints(Constraints,[]-CConstraints),
    compileDRSes([[L,lk0,[]]],DRSes-DRSOut,CConstraints,
                 Bind2-Bindings,[]-ContextsOut)
```

`addToDiscourse` takes the UDRS of an IFS as input, produces the IUDRS and returns the list of collected presuppositions to be justified. During that process binding information associating constants of IFSes with their corresponding new constants in IUDRSes is accumulated (by 'Input-Output' arguments). Bindings are lists with equations 'IFSconstant= IUDRSconstant' as elements.

Ambiguities are compiled as follows: Each ambiguity is given a name `A` and each of its disjuncts a name `D`. Then variable `V`, `A`, `D` and label `L` are associated by adding '`V:A:D:L`' to the bindings. Besides that, value information about `V` is accumulated and stored in the bindings as '`V=[A1:D1:L1, A2:D2:L2, ...]`' equations.

After compiling the ambiguity information, internal labels for DRSes ("group labels") are introduced, both on the basis of the DRSes and Constraints in the IFSes. Then the constraints are compiled (into `lr/4` predicates) and the presuppositions collected (according to the markers in the DRSes). Finally, the DRSes are compiled into IUDRS predicates.

Starting with toplabel `L` as local top and `lk0` as global top, `compileDRSes/6` acts recursively traversing the UDRS-graph top down (and sometimes back up again, in order to collect the superordinate context restrictions). On the basis of the label relations in CConstraints, it generates daughter structures '`[Label, MotherLabel, LocalContexts]`' which leads to the compilation of single DRSes including the compilation of their universe and conditions:

```
compileDRS(L:drs(_,U,Conds),Din-Dout,C,InB-OutB,LocalContexts):-
    identifyVar(L,DrsID,InB),!,
    compileUniverse(U,DrsID,InB-InB1,LocalContexts),
    compileConds(Conds,L:DrsID,Din-Dout,C,InB1-OutB,LocalContexts).

compileUniverse([DRs,Features],DrsID,InB-OutB,LocalContexts):-
    compileDrefs(DRs,DrsID,InB-Bindings,LocalContexts),
    compileFeatures(Features,Bindings-OutB).
```

In `compileDrefs/4`, `compileFeatures/2` and `compileConds/6`, the IUDRS predicates `d_in/3`, `feat/3` and `cond/5` are asserted, respectively, and their Contexts-slots are filled, accordingly, by their superordinate local contexts collected top down. Only if DRSOut is the empty list, compilation of the IFSes was successful.

10 Discourse processing

10.1 The general picture

The upper implementational structure of discourse processing is as follows:

```
dp:-                              % for: discourse processing
    retractpreds,                 % clearing the IUDRS database
    iUDRSAlreadyAddedretract,     % of previous stories
    processDiscourse(1,[],[]).

processDiscourse(N,BindingsIn,ContextsIn):-
    node(N,_),!,
    processSentence(N,BindingsIn-BindingsOut,ContextsIn-Contexts1),
    N1 is N+1,
    processDiscourse(N1,BindingsOut,Contexts1).
processDiscourse(_,_,_).
```

Sentences to be processed are stored as IFSes (note that this would have to be changed after the TC and DC components were put together). The identifiers of nodes are integers coding the position of the sentence in discourse. Given that there is a sentence node available, discourse processing means successive processing of sentences, which means adding the IUDRS of the nth IFS, resolving the presuppositions and then applying discourse principles to the underspecified representation of the text so far (consisting of more than one sentence).

```
processSentence(N,BindingsIn-BindingsOut,ContextsIn-ContextsOut):-
    addIUDRS(N,PresupList),
    resolvePresuppositions(PresupList,BindingsIn-B1,
                           ContextsIn-Contexts1),
    (N > 1 -> applicationOfDiscoursePrinciples(B1-BindingsOut,
                                               Contexts1-ContextsOut);
              B1 = BindingsOut,Contexts1=ContextsOut).
```

There are two types of information that are globally carried along with and handed through from one processing step to the next by corresponding ('Input-Output') lists: *Binding* information which on this level of processing means binding of presuppositions/anaphora to one of their antecedents; *context* information containing the choices made with respect to ambiguities. This clearly marks the status of binding and context information as defeasible and allows for backtracking. While this would normally lead to adding the already processed nondefeasible information anew, this is prevented with `addIUDRS/2`, which implements a "skipping over"these processing steps.

```
addIUDRS(N,PresupList):-
    iUDRSAlreadyAdded(N,PresupList),!.
addIUDRS(N,PresupList):-
    node(N,UDRS),
    addToDiscourse(UDRS,PresupList),
    asserta(iUDRSAlreadyAdded(N,PresupList)),!.
```

10.2 Handling presuppositions

Resolving a presupposition/anaphor is effectively a matter of theorem proving, at least in its widest sense. It must be shown that a given piece of information can be derived by some mechanism from another piece of information available (the "discourse representation axioms") given that the relation of *accessibility* between those pieces is respected. If this is possible wrt. the given discourse information, the presupposition can be *bound* to the antecedent found; if it is not possible, the presupposition has to be *accommodated*, that is, added to the discourse information. These cases are implemented here. We will not deal with the possibility of proving a presupposition by way of mediating inferences from background knowledge ("bridging").

Presuppositions are represented in IUDRSes as normal DRSes which are associated with their triggering sites by the label relation `presupOf`. The basic mechanism needed for handling presuppositions is therefore one to prove a DRS. As it is possible that the antecedent information is distributed over more than one DRS, accessibility can be required to hold between the presuppositional DRS in situ and the local maximum of the found antecedent DRSes. In the case of binding, this amounts to the following main program structure:

```
bindPresupposition(DrsId,Bin-Bout,ConIn-ConOut):-
    inference:bb_put(prooferror,noInfo),
    proveDRS(DrsId,1,[]-Antecedents,Bin-Bout,ConIn-Con1),
    localMaxOfL(Max,Antecedents,Con1-Con2),
    isAccessibleFor(Bout,Max,DrsId,Con2-ConOut).
```

If we were to prove an arbitrary DRS of a UDRS, we would have to retrieve its polarity value first. With presuppositional DRSes this is not necessary. The second argument of `proveDRS` therefore indicates and sets positive polarity. Before that, information about a possible prooferror is set to "noinfo".

The following shows the difference of proving presuppositional and anaphoric DRSes, namely, the existence and non-existence of conditions to be proved, respectively.

```
proveDRS(Drs,Pol,AntIn-AntOut,Bin-Bout,ConIn-ConOut):-
    conditionsOfDRS(Drs,Conds),!,
    drefsOfDRS(Drs,Drefs),
    addContextsOfDrefs(Drefs,ConIn-Con1),
    proveConds(Conds,Pol,AntIn-AntOut,Bin-Bout,Con1-ConOut).

proveDRS(Drs,_Pol,P-NewP,Bin-Bout,ConIn-ConOut):-
    drefsOfDRS(Drs,Drefs),
    addContextsOfDrefs(Drefs,ConIn-Con1),
    collectFeatures(Drefs,Bin-Bout,Features,FeaturesVar),
    callAllBT(FeaturesVar,[],Insts),
    \+ Features = Insts,
    drefsOfFeatures(Insts,DrefsNew),
    addContextsOfDrefs(DrefsNew,Con1-ConOut),
    drsesOfDrefs(DrefsNew,Bout,DRSes),
    append(DRSes,P,NewP).
```

The main part of the mechanism for the justification of a presupposition obviously is proving all of its conditions. The general case is shown in the following:

```
proveConds([C|Conds],Pol,AntIn-AntOut,Bin-Bout,ConIn-ConOut):-
    drefsToVariables([C],[]-[P],Bin-B1),
    provePredicate(C,P,Pol,AntIn-Ant1,B1-B2,ConIn-Con1),
    checkFeatures(C,B2-_,Con1-Con11),
    proveConds(Conds,Pol,Ant1-AntOut,B2-Bout,Con11-ConOut).
```

With this clause, it becomes soon apparent that "proving"means "matching"here for the most part. A condition `C` is turned from ground to nonground into `P` by systematically replacing discourse referents with variables and storing their relation in the bindings. After finding a proof/match and before continuing recursively with `proveConds/5`, it is checked whether the features of the involved discourse referents of `C` are compatible with `P`, given the new Bindings and Contexts.

`provePredicate/6` implements the core of this mechanism. In the standard case, `callPred/4` finds an instantiation of `P` (`CONDOUT`) different from `C`, checks the contexts and retrieves the group label `GL`. Then the polarity of the antecedent DRS is computed (`Pol1`) and checked for consistence. In case of failure `checkConsistence` sets `prooferror` to "inconsistence(C)".

```
provePredicate(OLD,CONDIN,Pol,AntIn-[GL|AntIn],B-B,ConIn-ConOut):-
    callPred(CONDIN,CONDOUT,GL,ConIn-Con1),
    \+ CONDOUT=OLD,
    polarityOfLabel(GL,Pol1,Con1-ConOut),
    checkConsistence(Old,Pol,Pol1),
    bb_put(bindings,B).
```

That proving a predicate with this mechanism may also involve inferences is realized by the next two clauses of `provePredicate/6`. Sentences like *If a man has a daughter then he loves his child* require the presupposition triggered by "the child"to be justified by *a daughter*, which goes beyond simple matching. To handle cases like this, the predicate of the IUDRS condition (`KprimePred`) must be literally exchanged by another one (`KPred`) given that there is an implicative relation between them in the right direction (cf. [2] for the definition of the '`>>`' relations) relative to the input polarity.

```
provePredicate(Old,cond(EV,L,GL1,KprimePred,V),1,AntIn-[GL|AntIn],
               B-B,ConIn-ConOut):-
    '>>'(KPred,KprimePred),
    callPred(cond(EV,L,GL1,KPred,V),_CONDOUT,GL,ConIn-Con1),
    polarityOfLabel(GL,Pol1,Con1-ConOut),
    checkConsistence(Old,1,Pol1).
```

Given negative polarity (that is, for sentences like *If a girl doesn't like fish, then she doesn't like sushi*), the direction of implication is reversed:

```
provePredicate(Old,cond(EV,L,GL1,KprimePred,V),0,AntIn-[GL|AntIn],
               B-B,ConIn-ConOut):-
    '>>'(KprimePred,KPred),
    callPred(cond(EV,L,GL1,KPred,V),_CONDOUT,GL,ConIn-Con1),
    polarityOfLabel(GL,Pol1,Con1-ConOut),
    checkConsistence(Old,0,Pol1).
provePredicate(_,_,_,_,_,_):-
    bb_get(prooferror,N),N=noInfo,bb_put(prooferror,noMatch),
    !,fail.
```

With regard to accommodating presuppositions, we have not come to a final conclusion how to represent it in the IUDRS. What is clear, however, is that one needs to show that the presupposition cannot be refuted, which is the prerequisite for adding the presuppositional material to the discourse representation.

```
accommodatePresupposition(X,B-B,C-C):-
    \+ presuppositionIsRefuted(X,C-_),
    ...
```

10.3 Proving temporal theorems with a temporal constraint solver

During the application of the interpretation principles it is sometimes necessary to check whether a certain temporal relation between two eventualities holds or not. We decided to use a temporal constraint solver for the computation of the consistency of a set of temporal relations and to model logical queries on top of this solver.

Let *TR* be a set of temporal relations, and let temprel(tr1,ev1,ev2) be a specific temporal relation. Let tcs(*TR*)=1 signify success of applying the temporal constraint solver on *TR* representing *consistence*, and let tcs(*TR*)=0 signify the corresponding failure representing *inconsistence*.

temprel(tr1,ev1,ev2) will be called *compatible* with *TR* iff tcs(*TR* ∪ {temprel(tr1, ev1,ev2)})=1. Then temprel(tr1,ev1,ev2) can be called *derivable* from *TR* iff temprel(tr1,ev1,ev2) is compatible with *TR* and its converse is not compatible with *TR*. Thus, temprel(tr1,ev1,ev2) can be called *not derivable* from *TR* if either it is not compatible with *TR*, or both temprel(tr1,ev1,ev2) and its converse are compatible with *TR*. By explicitly specifying the relevant conditions, this can be implemented as follows (`Cond` is the condition to be derived):

```
isNotDerivable(Cond,ConverseCond,SecTempConds,UnsecTempConds,
               ConIn-ConOut):-
    isTemporallyCompatibleWith(Cond,SecTempConds,
                               UnsecTempConds,ConIn-Con1),
    !,
    isTemporallyCompatibleWith(ConverseCond,SecTempConds,
                               UnsecTempConds,Con1-ConOut).
isNotDerivable(_,_,_,_,Con-Con).
```

Our temporal constraint solver uses Allen-like temporal interval relations defined by relations between the interval endpoints. Using endpoint relations makes it easy to

define temporal relations that would present representational difficulties on the logical level. For example, the relation *precedes(tr1,end(ev1), end(ev2))* can be defined as `endsBeforeEndOf(tr1,ev1,ev2)` and *neg(precedes(tr1,ev1,ev2))* can be defined as `doesnotprecede(tr1,ev1,ev2)`.

We use the constraint logic programming over finite domains (`clpfd`) library of Sicstus Prolog to define such temporal relations. Here are some definitions of endpoint-based temporal relations:

```
doesnotprecede(XS,XE,YS,YE):-
    XS #< XE,
    YS #< YE,
    XE #>= YS.
endsNotBeforeEndOf(XS,XE,YS,YE):-
    XS #< XE,
    YS #< YE,
    XE #>= YE.
startsBeforeStartOf(XS,XE,YS,YE):-
    XS #< XE,
    YS #< YE,
    XS #< YS.
startsBeforeEndOf(XS,XE,YS,YE):-
    XS #< XE,
    YS #< YE,
    XS #< YE.
endsBeforeStartOf(XS,XE,YS,YE):-
    XS #< XE,
    YS #< YE,
    XE #< YS.
endsBeforeEndOf(XS,XE,YS,YE):-
    XS #< XE,
    YS #< YE,
    XE #< YE.
endsWithStartOf(XS,XE,YS,YE):-
   XS #< XE,
    YS #< YE,
    XE #= YS.
endsWithEndOf(XS,XE,YS,YE):-
    XS #< XE,
    YS #< YE,
    XE #= YE.
precedes(XS,XE,YS,YE):-
    XS #< XE,
    XE  #< YS,
    YS #< YE.
meets(XS,XE,YS,YE):-
    XS #< XE,
    XE  #= YS,
    YS #< YE.
abuts(XS,XE,YS,YE):-meets(XS,XE,YS,YE).
```

```
contains(XS,XE,YS,YE):- %contains improper
    XS #< XE,
    YS #< YE,
    XS #=< YS,
    XE #>= YE.
overlaps_(XS,XE,YS,YE):-
    XS  #< YS,
    YS #< XE,
    XE #< YE.
```

There temporal conditions on the level of IUDRSes and the temporal relations used by the constraint solver are quite different, however. While the former take discourse referents as arguments, the arguments of the latter are integers of a finite domain. In order to use the solver, the gap between different data structures must be bridged first. Most importantly, it has to be secured that the context mechanism goes well together with the constraint solver. On the one hand, only those temporal conditions licensed by the current contexts are allowed as input to the solver. On the other hand, the solver itself must be allowed to select between ambiguity options. Using the solver therefore requires the following steps:

Collection of relevant temporal conditions. This is done by finding all IUDRS conditions whose referential variable (remember, the first argument of `cond/5`) is of sort `temprel`, and then, by filtering those conditions into secure temporal conditions (those who necessarily conform to the current contexts) and insecure temporal conditions (those who conform but introduce new contexts), and removing those who do not conform to the current contexts.

```
collectRelevantTempConds(SecTempConds,UnsecTempConds,Contexts):-
    findall(cond(A,B,C,TR,Con),(feat(A,sort,temprel),
                                cond(A,B,C,TR,Con)),AllTempConds),
    filterConditions(AllTempConds,Contexts,[]-SecTempConds,
                     []-UnsecTempConds).
```

Checking for compatibility of a temporal condition. The basis for logical proofs is testing for compatibility of a certain IUDRS temporal condition. Thus the solver has to be called with this condition to be checked, the secure and insecure relations, and the information about the current contexts. Checking for compatibility then means finding a consistent solution at least with respect to the secure conditions (actually, the current implementation tries to find maximal consistent sets of conditions including insecure ones). This, in turn, means first checking whether the contexts of each insecure condition are compatible. If so, the temporal constraint solver is called with `callTCS/1`.

```
isTemporallyCompatibleWith(TCToBeChecked,SecTempConds,UnsecTempConds,
                           ConIn-ConOut):-
    consistentSolution(UnsecTempConds,[]-UnusedUnsec,
                       [TCToBeChecked|SecTempConds],Solution,
                       ConIn-ConOut),
```

```
    (Solution = [] ->
     format('~n~nNo (further) solution found!',[]),!,fail
     ;
     true).


consistentSolution([],U-U,L,L,C-C).
consistentSolution([X|R],U1-U2,L,NL,ConIn-ConOut):-
    X=cond(_,_,_,_,Con),
    checkContexts00(Con,ConIn-Con1),
    consistentSolution([X|L]),
    consistentSolution(R,U1-U2,[X|L],NL,Con1-ConOut).
consistentSolution([X|R],U1-U2,L,NL,ConIn-ConOut):-
    consistentSolution(R,[X|U1]-U2,L,NL,ConIn-ConOut).
```

Calling the temporal constraint solver (TCS). The first task of finding a consistent solution for a set of conditions (`consistentSolution/1`) is to associate and replace each eventuality constant with two variables. After that, the set of variables is collected from the bindings. With the list of these variables and the list of conditions as constraints, the TCS is called.

```
consistentSolution(ConstraintList):-
    buildVarConstraints(ConstraintList,[],[],
                        VarConstraintList,VarBindings),
    collectVariablesfromVarList(VarBindings,Vars),
    callTCS(Vars,VarConstraintList),!.
```

Calling the TCS involves first setting the domain for the list of variables, which is a simple function of the length of the list. After that, the constraints are applied, which means calling Prolog with the temporal relation deprived of its referential variable (here we have the four arguments of the temporal relations defined above!).

```
callTCS(VarList,VarConstraintList):-
    length(VarList,VL),
    N is (VL * 2),
    domain(VarList,0,N),
    applyConstraints(VarConstraintList),!.


applyConstraints([],_).
applyConstraints([cond(_,_,_,TemporalRelation,_)|ConstraintList]):-
    TemporalRelation=..[R,_|Args],
    Pred=..[R|Args],!,
    Pred,
    applyConstraints(ConstraintList).
```

10.4 Application of the interpretation principles

With the mechanisms for handling ambiguities, presupposition justification and the resolution of temporal relationships, the foundation is prepared for the application of interpretation principles for the implicit computation of discourse relations as described in ([3]). Although we will refrain from reporting all the loose ends that remained due to the shortness of development time, we will at least sketch the realization of JoINC (Justification of $\mathcal{K}$-Incompatibility) using the example (the "Busbeispiel") of the case study: "Maria ging zur Bushaltestelle. Der Bus kam. Sie rannte (zur Bushaltestelle)."

What we want to show is how the INTERFERED/INTERFERENCE relation between the progressive state of going to the bus stop (S1) and the progressive state of running (to the bus stop) (S2) is excluded and, instead, a SEQUENCE relation between S1 and S2 is implicitly assumed by justifying their incompatibility with an intervening CAUSAL eventuality EV (which implies a CAUSE/RESPONSE between EV and S2 as well as an INTERFERED/INTERFERENCE relation between S1 and EV). The general implementational structure (note the different arities of `'JoINC'`) is therefore

```
applicationOfDiscoursePrinciples(Bin-Bout,ConIn-ConOut):-
    'JoINC'(Bin-Bout,ConIn-ConOut).

'JoINC'(Bin-Bout,ConIn-ConOut):-
    getBottomlabel(BL),
    format('~n~nChecking compatibility of ~w...',[BL]),
    'JoINC'(BL,Bin-Bout,ConIn-ConOut),!.

'JoINC'(BL,Bin-Bout,ConIn-ConOut):-
    findMatchingState(BL,Bin-Bout,ConIn-ConOut),
    format('~nOk, no inconsistency for ~w found
            by direct matching',[BL]),!.
```

First, the bottom label with the main eventuality of the corresponding sentence is retrieved and with it, `'JoINC'/3` is performed. It tries to find a matching state for S1 by:

- collecting the relevant discourse referents, i.e., the arguments of the predicates, of S1 (that's where the referential argument of the IUDRS conditions comes in handy),
- collecting their DRSes and proving/matching them with the previous discourse information
- collecting and proving the arguments' features and, finally,
- proving S1's DRS with the actual bindings:

```
findMatchingState(Drs,ConIn-ConOut,Bin-Bout):-
    bb_put(prooferror,noInfo),
```

```
    d_in(S,Drs),
    feat(S,sort,state),
    bb_put(matchstate,S),
    getArgsOfState(S,Args),
    getDRSesOfDRs(Args,DRSes),
    !,
    proveDRSes(DRSes,[]-_P1,Bin-B1,ConIn-Con1),
    collectFeatures(Args,B1-B2,Features,FeaturesVar),
    callAllBT(FeaturesVar,[],Insts),
    \+ Features = Insts,
    !,
    bb_put(bindings,[]),
    proveDRSes([Drs],[]-_P2,B2-Bout,Con1-ConOut),!.
findMatchingState(_Drs,C-C,B-B).
```

This should guarantee that it is exactly the match for Maria's special movement that either succeeds or fails. For the case of failure, the relevant information is stored with blackboard primitives. This is relevant for the second clause of `'JoINC'/3` , which imlements the main part of that principle:

```
'JoINC'(BL,Bin-Bout,Contexts-ConOut):-
    inference:bb_get(prooferror,PE),
    (PE=inconsistence(Cond) ->
     (getIncompatibleStates(S1,S2),
      d_in(S2,_,Con),
      checkContexts00(Con,Contexts-Con0),
      collectRelevantTempConds(SecTempConds,UnsecTempConds,Con0), !,
      (isNotDerivable(cond(1,2,3,precedes(1,S1,S2),[]),
                      cond(5,6,7,precedes(5,S2,S1),[]),
                      SecTempConds,UnsecTempConds,Con0-Con1),
       isNotDerivable(cond(1,2,3,precedes(1,S2,S1),[]),
                      cond(5,6,7,precedes(5,S1,S2),[]),
                      SecTempConds,UnsecTempConds,Con0-Con1),
       format('~n~n~w led to inconsistence of states ~w and ~w !',
              [Cond,S1,S2]),!,
       findInterveningEventuality(S2,S1,EV,SecTempConds,
                                  UnsecTempConds, Con1-ConOut),
       iFSCompilation:myGensym(dref,DRefID),
       iFSCompilation:drsassert(Bin,ConOut,d_in(DRefID,BL)),
       iFSCompilation:myGensym(ml,CondID),
       iFSCompilation:drsassert(Bin,ConOut,cond(DRefID,CondID,BL,
                               precedes(DRefID,S2,S1),ConOut)),
       format('~nAdded temporal precedence of ~w and ~w to
               discourse',[S2,S1])
       ;
       format('~n~n~w led to incompatibility of states ~w and ~w,
               but there is no inconsistency',[Cond,S1,S2])
       )
      )
```

```
  ;
format('~nOk, no inconsistency for ~w found
         by direct matching',[BL]),Contexts=ConOut
),
Bin=Bout.
```

In case of an inconsistency found (in this case, because of the mismatch of Maria's running and walking), the states S1 and S2 are retrieved, S2's contexts are checked, and the relevant temporal conditions are collected (see above). Only if neither `precedes(S1,S2)` nor `precedes(S2,S1)` are derivable we have shown INC ($\mathcal{K}$-Incompatibility). We then have to find an intervening eventuality. If this succeeds, temporal precedence of S2 and S1 (corresponding to a SEQUENCE discourse relation) can be asserted, as well as the relation of non-accidental dependency (which is not done here).

11 Final remarks

We have tried to give an idea of how some of the theoretical considerations concerning presuppositions and underspecification in the computation of temporal and other relations in discourse can be implemented.

In the following appendix I, the reader will find the IFSes input to the DC component (and to subsequent discourse processing). Note that the IFSes were not constructed automatically and may therefore contain both theoretical insufficiencies as well as simple errors. See [7] for a graphical presentation of the Busbeispiel's UDRS representation. Appendix II shows some inline outputs of the program in order to show its processing the Busbeispiel. Unfortunately, we were faced with major fundamental problems of UDRS presentation when trying to adapt an existing visualization tool to the IUDRSes as developed here. Appendix III shows the internal representation of the UDRSes (IUDRSes) of the Busbeispiel in the current state of the program.

References

[1] Michael Dorna. The adt package for the verbmobil interface term. Technical report, Verbmobil Report 104, Institut für maschinelle Sprachverarbeitung, Universität Stuttgart, 1996.

[2] Uwe Reyle. Co-indexing labelled drss to represent and reason with ambiguities. In Kees van Deemter Stanley Peters, editor, *Semantic Ambiguity and Underspecification*. CSLI Publications, Stanford, 1996.

[3] Uwe Reyle and Antje Rossdeutscher. Constraint based, bottom up discourse interpretation. In Uwe Reyle, editor, *Presuppositions and Underspecification in the Computation of Temporal and other Relations in Discourse*. Arbeitsberichte des Sonderforschungsbereichs 340, Stuttgart/Tübingen, Nr.164, 2000.

[4] Uwe Reyle and Antje Rossdeutscher. Justifying presuppositions to resolve temporal underspecification. In Uwe Reyle, editor, *Presuppositions and Underspecification in the Computation of Temporal and other Relations in Discourse*. Arbeitsberichte des Sonderforschungsbereichs 340, Stuttgart/Tübingen, Nr.164, 2000.

[5] Antje Rossdeutscher. Discourse relations. In Uwe Reyle, editor, *Presuppositions and Underspecification in the Computation of Temporal and other Relations in Discourse*. Arbeitsberichte des Sonderforschungsbereichs 340, Stuttgart/Tübingen, Nr.164, 2000.

[6] Antje Rossdeutscher and Uwe Reyle. Very short stories. In Uwe Reyle, editor, *Presuppositions and Underspecification in the Computation of Temporal and other Relations in Discourse*. Arbeitsberichte des Sonderforschungsbereichs 340, Stuttgart/Tübingen, Nr.164, 2000.

[7] Jasmin Saric. Kompositionalität regelbasiert. die implementierung des lexikons und der tempuskonstruktion. In Uwe Reyle, editor, *Presuppositions and Underspecification in the Computation of Temporal and other Relations in Discourse*. Arbeitsberichte des Sonderforschungsbereichs 340, Stuttgart/Tübingen, Nr.164, 2000.

[8] Michael Schiehlen. Semantic construction from parse forests. In *Proceedings of 16th International Conference on Computational Linguistics (COLING 96)*, Copenhagen, Denmark, 1996.

[9] Rob van der Sandt. Presupposition projection as anaphora resolution. *Journal of Semantics*, 9(4):333–378, 1992.

Appendix I: The IFSes for the Busbeispiel (constructed non-automatically)

```
node(1,
     udrs([l1:drs(a,
                  [[],
                   []],
                  [decl(l012)]),
           l2:drs(a,
                  [[],
                   []],
                  []),
           l3:drs(p(l4),
                  [[bh],
                   [sort(bh,location),gend(bh,fem),num(bh,sg)]],
                  [bushaltestelle(bh)]),
           l4:drs(a,
                   [[],
                    []],
                  []),
           l5:drs(p(l6),
                  [[m],
                   [sort(m,person),gend(m,fem)]],
                  []),
           l6:drs(a,
                  [[],
                   []],
                  []),
           l7:drs(a,
                  [[],
                   [named(m,"maria")]],
                  []),
           l8:drs(p(l10),
                  [[s1pr],
                   [sort(s1pr,state)]],
                  ['NEG'(l89)]),
           l9:drs(a,
                  [[],
                   []],
                  ['AT'(s1pr,m,bh)]),
           l10:drs(a,
                   [[s,e1,t1,t2,c1,n,tr1,tr2,tr3,tr4,tr5,tr6],
                    [sort(s,state),sort(e1,event),
                     sort(t1,tt),sort(t2,tf), sort(c1,cut),
                     sort(n,now), sort(tr1,temprel),
                     sort(tr2,temprel), sort(tr3,temprel),
                     sort(tr4,temprel), sort(tr5,temprel),
                     sort(tr6,temprel), mode(s,"gehen")] ],
                   ['PROG'(s,e1,l1011),move(s),arg1(s,m),zu_prep(e1,bh),
```

```
                    endsNotBeforeEndOf(tr1,s1pr,s),
                    startsBeforeStartOf(tr2,s1pr,s),
                    contains(tr3,t2,t1),contains(tr4,t1,s),
                    precedes(tr5,t1,n),precedes(tr6,t2,n)]),
          l11:drs(a,
                  [[],
                   []],
                  ['BEC'(e1,s1,l1112)]),
          l12:drs(a,
                  [[],
                   []],
                  ['AT'(s1,m,bh)]),
          l14:drs(a,
                  [[tr7],
                   [sort(tr7,temprel)]],
                  [contains(tr7,s,c1)]),
          l15:drs(a,
                  [[e,tr8,tr9,tr10],
                   [sort(e,event),sort(tr8,temprel),sort(tr9,temprel),
    sort(tr10,temprel)]],
                  ['BEC'(e,s1,l1516),prog(e,s),endsWithEndOf(tr8,s1pr,e),
                   contains(tr9,t1,e),precedes(tr10,e,c1)]),
          l16:drs(a,
                  [[],
                   []],
                  ['AT'(s1,m,bh)])
         ],
         [toplabel(l1),bottomlabel(l10),leq(l4,l2),leq(l6,l2),leq(l7,l6),
          leq(l10,l4),leq(l2,l012),leq(l10,l6),leq(l9,l89),leq(l11,l1011),
          leq(l12,l1112),leq(l16,l1516),less(l012,l1),less(l89,l8),
          less(l1011,l10),less(l1112,l11),less(l1516,l15),leq(L,l10)],
         [xor([[L=l14],[L=l15]])]
    )).

node(2,
     udrs([l1:drs(a,
                  [[],
                   []],
                  [decl(l012)]),

           l2:drs(a,
                  [[],
                   []],
                  []),
           l5:drs(p(l6),
                  [[b],
                   [sort(b,vehicle),gend(b,masc)]],
                  []),
           l6:drs(a,
```

```
       [[],
        []],
       []),
l8:drs(p(l10),
       [[s2pr1,s2pr2,y,l,tr1],
        [sort(s2pr1,state),sort(s2pr2,state),
         sort(y,person), sort(l,location),
         sort(tr1,temprel)]],
       ['NEG'(l89),kommenATT(s2pr2,y,l),contains(tr1,s2pr2,t1)]),
l9:drs(a,
       [[],
        []],
       ['AT'(s2pr1,b,l)]),
l10:drs(a,
        [[s,e1,t1,t2,c1,n,tr2,tr3,tr4,tr5,tr6,tr7],
         [sort(s,state),sort(e1,event),
          sort(t1,tt),sort(t2,tf), sort(c1,cut),
          sort(n,now), sort(tr2,temprel),
          sort(tr3,temprel), sort(tr4,temprel),
          sort(tr5,temprel), sort(tr6,temprel),
          sort(tr7,temprel)]],
        ['PROG'(s,e1,l1011), move(s),arg1(s,m),
         zu_prep(e1,bh),
         endsNotBeforeEndOf(tr2,s2pr1,s),
         startsBeforeStartOf(tr3,s2pr1,s),
         contains(tr4t2,t1), contains(tr5,t1,s),
         precedes(tr6,t1,n), precedes(tr7,t2,n)]),
 l11:drs(a,
         [[],
          []],
         ['BEC'(e1,s1,l1112)]),
 l12:drs(a,
         [[],
          []],
         ['AT'(s1,b,l)]),

 l14:drs(a,
         [[tr8],
          [sort(tr8,temprel)]],
         [contains(tr8s,c1)]),
 l15:drs(a,
         [[e,tr9,tr10,tr11],
          [sort(e,event), sort(tr9,temprel),
           sort(tr10,temprel), sort(tr11,temprel)]],
         ['BEC'(e,s1,l1516),prog(e,s),
          endsWithEndOf(tr9,s2pr1,e),
          contains(tr10,t1,e), precedes(tr11,e,c1)]),
 l16:drs(a,
         [[],
```

```
                    []],
                   ['AT'(s1,b,l)])
          ],
          [toplabel(l1),bottomlabel(l10),leq(l6,l2),leq(l2,l012),
           leq(l10,l6),leq(l9,l89),leq(l11,l1011),leq(l12,l1112),
           leq(l16,l1516),less(l012,l1),less(l89,l8),less(l1011,l10),
           less(l1112,l11),less(l1516,l15),leq(L,l10)],
          [xor([[L=l14],[L=l15]])]
       )).

node(3,
     udrs([l1:drs(a,
                  [[],
                   []],
                  [decl(l012)]),
           l2:drs(a,
                  [[],
                   []],
                  []),
           l3:drs(p(l4),
                  [[bh],
                   [sort(bh,location),gend(bh,fem),num(bh,sg)]],
                  [bushaltestelle(bh)]),
           l4:drs(a,
                  [[],
                   []],
                  []),
           l5:drs(p(l6),
                  [[m],
                   [sort(m,person),gend(m,fem)]],
                  []),
           l6:drs(a,
                  [[],
                   []],
                  []),
           l7:drs(a,
                  [[],
                   []],
                  []),
           l8:drs(p(l10),
                  [[s3pr],
                   [sort(s3pr,state)]],
                  ['NEG'(l89)]),
           l9:drs(a,
                  [[],
                   []],
                  ['AT'(s3pr,m,bh)]),
           l10:drs(a,
                   [[s,e1,t1,t2,c1,n,tr1,tr2,tr3,tr4,tr5,tr6],
```

```
                [sort(s,state),sort(e1,event),
                 sort(t1,tt),sort(t2,tf), sort(c1,cut),
                 sort(n,now), sort(tr1,temprel),
                 sort(tr2,temprel), sort(tr3,temprel),
                 sort(tr4,temprel), sort(tr5,temprel),
                 sort(tr6,temprel), mode(s,"rennen")] ],
                ['PROG'(s,e1,l1011),
                 move(s),arg1(s,m), zu_prep(e1,bh),
                 endsNotBeforeEndOf(tr1,s1pr,s),
                 startsBeforeStartOf(tr2,s1pr,s),
                 contains(tr3,t2,t1), contains(tr4,t1,s),
                 precedes(tr5,t1,n), precedes(tr6,t2,n)]),
     l11:drs(a,
             [[],
              []],
             ['BEC'(e1,s1,l1112)]),
     l12:drs(a,
             [[],
              []],
             ['AT'(s1,m,bh)]),

     l14:drs(a,
             [[tr7],
              [sort(tr7,temprel)]],
             [contains(tr7,s,c1)]),
     l15:drs(a,
             [[e,tr8,tr9,tr10,tr11],
              [sort(e,event), sort(tr8,temprel),
               sort(tr9,temprel), sort(tr10,temprel),
               sort(tr11,temprel)]],
             ['BEC'(e,s1,l1516), prog(e,s),
              endsWithEndOf(tr8,s1pr,e),
              contains(tr9,t1,e), precedes(tr10,e,c1),
              endsWithEndOf(tr11,s,e)]),
     l16:drs(a,
             [[],
              []],
             ['AT'(s1,m,bh)])
    ],
    [toplabel(l1),bottomlabel(l10),leq(l4,l2),leq(l6,l2),leq(l7,l6),
     leq(l10,l4),leq(l2,l012),leq(l10,l6),leq(l9,l89),leq(l11,l1011),
     leq(l12,l1112),leq(l16,l1516),less(l012,l1),less(l89,l8),
     less(l1011,l10),less(l1112,l11),less(l1516,l15),leq(L,l10)],
    [xor([[L=l14],[L=l15]])]
)).
```

Appendix II: Program Output

```
| ?- dp.

Bindings: [l20=l42,tr10=i21,tr9=i20,tr8=i19,e=i18,tr7=i17,l17=l29,
           s1pr=i16,bh=i15,m=i14,l19=l25,s1=i13,l18=l23,tr6=i12,
           tr5=i11,tr4=i10,tr3=i9,tr2=i8,tr1=i7,n=i6,c1=i5,t2=i4,
           t1=i3,e1=i2,s=i1,l1516=l20,l1112=l19,l1011=l18,l89=l17,
           l012=l16,l16=l15,l12=l14,l11=l13,l10=l12,l9=l11,l8=l10,
           l7=l9,l6=l8,l5=l7,l4=l6,l3=l5,l2=l4,l1=l3,_102:a1:d2:l2,
           l15=l2,_102:a1:d1:l1,l14=l1]

...trying to resolve l10 ([NEG(l29)])...

Current contexts (bindPresup): []

l10 gets accommodated to l12

...trying to resolve the anaphor in l7...

Current contexts (bindPresup): []

l7 gets accommodated to l8

...trying to resolve l5 ([bushaltestelle(i15)])...

Current contexts (bindPresup): []

l5 gets accommodated to l6

Current contexts: []

Bindings: [l64=l88,tr11=i49,tr10=i48,tr9=i47,e=i46,tr8s=i45,tr8=i44,tr4t2=i43,
           bh=i42,m=i41,l61=l73,tr1=i40,y=i39,s2pr2=i38,s2pr1=i37,l=i36,b=i35,
           l63=l70,s1=i34,l62=l68,tr7=i33,tr6=i32,tr5=i31,tr4=i30,tr3=i29,
           tr2=i28,n=i27,c1=i26,t2=i25,t1=i24,e1=i23,s=i22,l60=l66,l1516=l64,
           l1112=l63,l1011=l62,l89=l61,l012=l60,l16=l59,l12=l58,l11=l57,
           l10=l56,l9=l55,l8=l54,l6=l53,l5=l52,l2=l51,l1=l50,_164:a2:d4:l49,
           l15=l49,_164:a2:d3:l48,l14=l48]

...trying to resolve l54
([NEG(l73),contains(i40,i38,i24),kommenATT(i38,i39,i36)])...

Current contexts (bindPresup): []

l54 gets accommodated to l56

...trying to resolve the anaphor in l52...

Current contexts (bindPresup): []
```

```
l52 gets accommodated to l53

Checking compatibility of l56...
Args of situation i22:[i41]
Ok, no inconsistency for l56 found by direct matching

Current contexts: []

Bindings: [l113=l136,tr11=i72,tr10=i71,tr9=i70,tr8=i69,e=i68,tr7=i67,s1pr=i66,
           l110=l123,s3pr=i65,bh=i64,m=i63,l112=l119,s1=i62,l111=l117,tr6=i61,
           tr5=i60,tr4=i59,tr3=i58,tr2=i57,tr1=i56,n=i55,c1=i54,t2=i53,t1=i52,
           e1=i51,s=i50,l109=l115,l1516=l113,l1112=l112,l1011=l111,l89=l110,
           l012=l109,l16=l108,l12=l107,l11=l106,l10=l105,l9=l104,l8=l103,l7=l102,
           l6=l101,l5=l100,l4=l99,l3=l98,l2=l97,l1=l96,_226:a3:d6:l95,l15=l95,
           _226:a3:d5:l94,l14=l94]

...trying to resolve l103 ([NEG(l123)])...

Current contexts (bindPresup): []

l103 gets accommodated to l105

...trying to resolve the anaphor in l100...

Current contexts (bindPresup): []

Current contexts(after bind): []

l100 gets bound to l7
Equations: [i63=i14]

...trying to resolve l98 ([bushaltestelle(i64)])...

Current contexts (bindPresup): []

Current contexts(after bind): []

l98 gets bound to l5
Equations: [l98=l5,l121=l27,i64=i15,i63=i14]

Checking compatibility of l105...
Args of situation i50:[i63]

feat(i1,mode,[114,101,110,110,101,110])
led to inconsistence of states i50 and i1 !

***intervening eventuality: i51
```

```
Added temporal precedence of i1 and i50 to discourse

Current contexts: [a3:d5,a2:d4,a1:d1]
```

Appendix III: IUDRSes of the Busbeispiel

```
cond(i73, l143, l105, precedes(i73,i1,i50), [a3:d5,a2:d4,a1:d1]).
cond(i72, l142, l105, endsWithEndOf(i72,i50,i68), [a3:d6]).
cond(i71, l141, l105, precedes(i71,i68,i54), [a3:d6]).
cond(i70, l140, l105, contains(i70,i52,i68), [a3:d6]).
cond(i69, l139, l105, endsWithEndOf(i69,i66,i68), [a3:d6]).
cond(i68, l138, l105, prog(i68,i50), [a3:d6]).
cond(i62, l137, l108, 'AT'(i62,i63,i64), [a3:d6]).
cond(i68, l135, l105, 'BEC'(i68,i62,l136), [a3:d6]).
cond(i67, l134, l105, contains(i67,i50,i54), [a3:d5]).
cond(i61, l133, l105, precedes(i61,i53,i55), []).
cond(i60, l132, l105, precedes(i60,i52,i55), []).
cond(i59, l131, l105, contains(i59,i52,i50), []).
cond(i58, l130, l105, contains(i58,i53,i52), []).
cond(i57, l129, l105, startsBeforeStartOf(i57,i66,i50), []).
cond(i56, l128, l105, endsNotBeforeEndOf(i56,i66,i50), []).
cond(i51, l127, l105, zu_prep(i51,i64), []).
cond(i50, l126, l105, arg1(i50,i63), []).
cond(i50, l125, l105, move(i50), []).
cond(i65, l124, l104, 'AT'(i65,i63,i64), []).
cond(l123, l122, l103, 'NEG'(l123), []).
cond(i64, l121, l98, bushaltestelle(i64), []).
cond(i62, l120, l107, 'AT'(i62,i63,i64), []).
cond(i51, l118, l106, 'BEC'(i51,i62,l119), []).
cond(i50, l116, l105, 'PROG'(i50,i51,l117), []).
cond(l115, l114, l96, decl(l115), []).
cond(i49, l93, l56, precedes(i49,i46,i26), [a2:d4]).
cond(i48, l92, l56, contains(i48,i24,i46), [a2:d4]).
cond(i47, l91, l56, endsWithEndOf(i47,i37,i46), [a2:d4]).
cond(i46, l90, l56, prog(i46,i22), [a2:d4]).
cond(i34, l89, l59, 'AT'(i34,i35,i36), [a2:d4]).
cond(i46, l87, l56, 'BEC'(i46,i34,l88), [a2:d4]).
cond(i45, l86, l56, contains(i45,i26), [a2:d3]).
cond(i33, l85, l56, precedes(i33,i25,i27), []).
cond(i32, l84, l56, precedes(i32,i24,i27), []).
cond(i31, l83, l56, contains(i31,i24,i22), []).
cond(i43, l82, l56, contains(i43,i24), []).
cond(i29, l81, l56, startsBeforeStartOf(i29,i37,i22), []).
cond(i28, l80, l56, endsNotBeforeEndOf(i28,i37,i22), []).
cond(i23, l79, l56, zu_prep(i23,i42), []).
cond(i22, l78, l56, arg1(i22,i41), []).
cond(i22, l77, l56, move(i22), []).
cond(i40, l76, l54, contains(i40,i38,i24), []).
cond(i38, l75, l54, kommenATT(i38,i39,i36), []).
cond(i37, l74, l55, 'AT'(i37,i35,i36), []).
cond(l73, l72, l54, 'NEG'(l73), []).
cond(i34, l71, l58, 'AT'(i34,i35,i36), []).
cond(i23, l69, l57, 'BEC'(i23,i34,l70), []).
cond(i22, l67, l56, 'PROG'(i22,i23,l68), []).
```

```
cond(l66, l65, l50, decl(l66), []).
cond(i21, l47, l12, precedes(i21,i18,i5), [a1:d2]).
cond(i20, l46, l12, contains(i20,i3,i18), [a1:d2]).
cond(i19, l45, l12, endsWithEndOf(i19,i16,i18), [a1:d2]).
cond(i18, l44, l12, prog(i18,i1), [a1:d2]).
cond(i13, l43, l15, 'AT'(i13,i14,i15), [a1:d2]).
cond(i18, l41, l12, 'BEC'(i18,i13,l42), [a1:d2]).
cond(i17, l40, l12, contains(i17,i1,i5), [a1:d1]).
cond(i12, l39, l12, precedes(i12,i4,i6), []).
cond(i11, l38, l12, precedes(i11,i3,i6), []).
cond(i10, l37, l12, contains(i10,i3,i1), []).
cond(i9, l36, l12, contains(i9,i4,i3), []).
cond(i8, l35, l12, startsBeforeStartOf(i8,i16,i1), []).
cond(i7, l34, l12, endsNotBeforeEndOf(i7,i16,i1), []).
cond(i2, l33, l12, zu_prep(i2,i15), []).
cond(i1, l32, l12, arg1(i1,i14), []).
cond(i1, l31, l12, move(i1), []).
cond(i16, l30, l11, 'AT'(i16,i14,i15), []).
cond(l29, l28, l10, 'NEG'(l29), []).
cond(i15, l27, l5, bushaltestelle(i15), []).
cond(i13, l26, l14, 'AT'(i13,i14,i15), []).
cond(i2, l24, l13, 'BEC'(i2,i13,l25), []).
cond(i1, l22, l12, 'PROG'(i1,i2,l23), []).
cond(l15, l21, l3, decl(l15), []).

lr(l103, leq, l96, []).
lr(l100, presupOf, l101, []).
lr(l98, presupOf, l99, []).
lr(l94, leq, l105, [a3:d5]).
lr(l95, leq, l105, [a3:d6]).
lr(l113, less, l95, []).
lr(l112, less, l106, []).
lr(l111, less, l105, []).
lr(l110, less, l103, []).
lr(l109, less, l96, []).
lr(l108, leq, l113, []).
lr(l107, leq, l112, []).
lr(l106, leq, l111, []).
lr(l104, leq, l110, []).
lr(l105, leq, l101, []).
lr(l97, leq, l109, []).
lr(l105, leq, l99, []).
lr(l102, leq, l101, []).
lr(l101, leq, l97, []).
lr(l99, leq, l97, []).
lr(l96, leq, lk0, []).
lr(l96, follows, l50, []).
lr(l52, leq, l50, []).
lr(l54, leq, l50, []).
```

```
lr(l48, leq, l56, [a2:d3]).
lr(l49, leq, l56, [a2:d4]).
lr(l64, less, l49, []).
lr(l63, less, l57, []).
lr(l62, less, l56, []).
lr(l61, less, l54, []).
lr(l60, less, l50, []).
lr(l59, leq, l64, []).
lr(l58, leq, l63, []).
lr(l57, leq, l62, []).
lr(l55, leq, l61, []).
lr(l56, leq, l53, []).
lr(l51, leq, l60, []).
lr(l53, leq, l51, []).
lr(l50, leq, lk0, []).
lr(l50, follows, l3, []).
lr(l5, leq, l3, []).
lr(l7, leq, l3, []).
lr(l10, leq, l3, []).
lr(l1, leq, l12, [a1:d1]).
lr(l2, leq, l12, [a1:d2]).
lr(l20, less, l2, []).
lr(l19, less, l13, []).
lr(l18, less, l12, []).
lr(l17, less, l10, []).
lr(l16, less, l3, []).
lr(l15, leq, l20, []).
lr(l14, leq, l19, []).
lr(l13, leq, l18, []).
lr(l11, leq, l17, []).
lr(l12, leq, l8, []).
lr(l4, leq, l16, []).
lr(l12, leq, l6, []).
lr(l9, leq, l8, []).
lr(l8, leq, l4, []).
lr(l6, leq, l4, []).
lr(l3, leq, lk0, []).

toplabel(l96).
toplabel(l50).
toplabel(l3).

bottomlabel(l105).
bottomlabel(l56).
bottomlabel(l12).

feat(i72, sort, temprel).
feat(i71, sort, temprel).
feat(i70, sort, temprel).
```

```
feat(i69, sort, temprel).
feat(i68, sort, event).
feat(i67, sort, temprel).
feat(l110, pol, 0).
feat(i65, sort, state).
feat(i63, gend, fem).
feat(i63, sort, person).
feat(i64, num, sg).
feat(i64, gend, fem).
feat(i64, sort, location).
feat(i50, mode, [114,101,110,110,101,110]).
feat(i61, sort, temprel).
feat(i60, sort, temprel).
feat(i59, sort, temprel).
feat(i58, sort, temprel).
feat(i57, sort, temprel).
feat(i56, sort, temprel).
feat(i55, sort, now).
feat(i54, sort, cut).
feat(i53, sort, tf).
feat(i52, sort, tt).
feat(i51, sort, event).
feat(i50, sort, state).
feat(l100, presup, 1).
feat(l98, presup, 1).
feat(i49, sort, temprel).
feat(i48, sort, temprel).
feat(i47, sort, temprel).
feat(i46, sort, event).
feat(i44, sort, temprel).
feat(l61, pol, 0).
feat(i40, sort, temprel).
feat(i36, sort, location).
feat(i39, sort, person).
feat(i38, sort, state).
feat(i37, sort, state).
feat(i35, gend, masc).
feat(i35, sort, vehicle).
feat(i33, sort, temprel).
feat(i32, sort, temprel).
feat(i31, sort, temprel).
feat(i30, sort, temprel).
feat(i29, sort, temprel).
feat(i28, sort, temprel).
feat(i27, sort, now).
feat(i26, sort, cut).
feat(i25, sort, tf).
feat(i24, sort, tt).
feat(i23, sort, event).
```

```
feat(i22, sort, state).
feat(i14, named, [109,97,114,105,97]).
feat(i21, sort, temprel).
feat(i20, sort, temprel).
feat(i19, sort, temprel).
feat(i18, sort, event).
feat(i17, sort, temprel).
feat(l17, pol, 0).
feat(i16, sort, state).
feat(i14, gend, fem).
feat(i14, sort, person).
feat(i15, num, sg).
feat(i15, gend, fem).
feat(i15, sort, location).
feat(i1, mode, [103,101,104,101,110]).
feat(i12, sort, temprel).
feat(i11, sort, temprel).
feat(i10, sort, temprel).
feat(i9, sort, temprel).
feat(i8, sort, temprel).
feat(i7, sort, temprel).
feat(i6, sort, now).
feat(i5, sort, cut).
feat(i4, sort, tf).
feat(i3, sort, tt).
feat(i2, sort, event).
feat(i1, sort, state).

d_in(i73, l105, [a1:d1,a2:d4,a3:d5]).
d_in(i72, l105, [a3:d6]).
d_in(i71, l105, [a3:d6]).
d_in(i70, l105, [a3:d6]).
d_in(i69, l105, [a3:d6]).
d_in(i68, l105, [a3:d6]).
d_in(i67, l105, [a3:d5]).
d_in(i65, l103, []).
d_in(i63, l100, []).
d_in(i64, l98, []).
d_in(i61, l105, []).
d_in(i60, l105, []).
d_in(i59, l105, []).
d_in(i58, l105, []).
d_in(i57, l105, []).
d_in(i56, l105, []).
d_in(i55, l105, []).
d_in(i54, l105, []).
d_in(i53, l105, []).
d_in(i52, l105, []).
d_in(i51, l105, []).
```

```
d_in(i50, l105, []).
d_in(i49, l56, [a2:d4]).
d_in(i48, l56, [a2:d4]).
d_in(i47, l56, [a2:d4]).
d_in(i46, l56, [a2:d4]).
d_in(i44, l56, [a2:d3]).
d_in(i40, l54, []).
d_in(i36, l54, []).
d_in(i39, l54, []).
d_in(i38, l54, []).
d_in(i37, l54, []).
d_in(i35, l52, []).
d_in(i33, l56, []).
d_in(i32, l56, []).
d_in(i31, l56, []).
d_in(i30, l56, []).
d_in(i29, l56, []).
d_in(i28, l56, []).
d_in(i27, l56, []).
d_in(i26, l56, []).
d_in(i25, l56, []).
d_in(i24, l56, []).
d_in(i23, l56, []).
d_in(i22, l56, []).
d_in(i21, l12, [a1:d2]).
d_in(i20, l12, [a1:d2]).
d_in(i19, l12, [a1:d2]).
d_in(i18, l12, [a1:d2]).
d_in(i17, l12, [a1:d1]).
d_in(i16, l10, []).
d_in(i14, l7, []).
d_in(i15, l5, []).
d_in(i12, l12, []).
d_in(i11, l12, []).
d_in(i10, l12, []).
d_in(i9, l12, []).
d_in(i8, l12, []).
d_in(i7, l12, []).
d_in(i6, l12, []).
d_in(i5, l12, []).
d_in(i4, l12, []).
d_in(i3, l12, []).
d_in(i2, l12, []).
d_in(i1, l12, []).
```