Chapter 9

Semantics as a Cognitive Process II: Active Search for Cataphora Antecedents and the Semantics of Conditionals

In this chapter, we generalize our eager left-corner incremental interpreter to cover conditionals and conjunctions. We focus on the (dynamic) semantic contrast between conditionals and conjunctions because the interaction between these sentential operators and anaphora/cataphora provides a strong argument that semantic parsing needs to be incremental, eager and competence-based, as argued in Brasoveanu and Dotlačil (2015a).

An extreme, but clear way to state the main theoretical proposal made by this chapter is the contention that anaphora, and presupposition in general, are properly understood as *processing-level* phenomena that *guide and constrain memory retrieval processes associated with incremental interpretation*. That is, they guide and constrain the cognitive process of integration, or linking, of new and old semantic information.

Under the assumption that anaphora and presuppositions are components of memory retrieval processes associated with the real-time integration of semantic information, the intrusion of world knowledge in these processes, i.e., the “pragmatics of anaphora resolution and presupposition resolution,” comes in naturally: world knowledge is stored in declarative memory, so it is natural for memory retrieval processes to be modulated by it.

Thus, the (most probably oversimplifying) hypothesis is that anaphora and presupposition have semantic effects, but anaphora and presupposition are not exclusively, or even primarily, semantics. The proper way to analyze them is as a part of the *processing component* of a broad theory of natural language interpretation.

This proposal is very close in spirit to the DRT account of presupposition proposed in van der Sandt (1992), Kamp (2001a, b), among others. Kamp (2001b), with its extended argument for and extensive use of *preliminary representations*—that is, meaning representations that explicitly include unresolved presuppositions—is a particularly close idea.

To see the connection between our proposal and the theory of presuppositions proposed in Kamp (2001b), consider the problem associated with the use of preliminary representations raised at the end of Kamp (2001b):

“[S]emanticists of a model-theoretic persuasion may want to see a formal semantics of [...] preliminary representations. [...] [T]he possibility of such a semantics is limited. To define a syntax of preliminary representations [...] which characterizes them as the expressions of a given representation formalism (or as data structures of a certain form) is not too difficult. Moreover, for those preliminary representations [...] in which all presuppositions appear in the highest possible position, an intuitively plausible model-theoretic semantics can be stated without too much difficulty. But for representations with presuppositions in subordinate positions [...] I very much doubt that one is to be had.” (Kamp 2001b, 250–51)

The solution to this problem that we propose in this chapter is that we shouldn’t even ask for a semantics of preliminary representations. This is a category error: preliminary representations are central to natural language interpretation, but they are not semantic representations: they are processing-level representations that support incremental interpretation mechanisms, and have semantic effects because of this.

The chapter is structured as follows. In Sect. 9.1, we discuss why the interaction of conditionals and cataphora provide a strong empirical argument for incremental interpretation *at the semantic level*. We also describe two experiments investigating the interaction between conditionals and pronominal cataphora on one hand (Sect. 9.1.1), and the interaction between conditionals and cataphoric presuppositions on the other hand (Sect. 9.1.2).

Section 9.2 sets up the general theoretical scene for the remainder of the chapter by arguing that mechanistic processing goals should be one of the core explanatory goals for formal semantics, and that a cognitive-architectural approach to natural language meaning and interpretation is one way to pursue that goal.

Section 9.3 moves on to the specifics of an ACT-R processing model for conditionals with a sentence-final *if*-clause that explicitly models their incremental interpretation. We show that the model qualitatively captures the interaction between conditionals and pronominal cataphora in a simple example.

Section 9.4 expands this model to capture the interaction between conditionals and cataphoric presuppositions for the items of the study reported in Sect. 9.1.2. We embed the resulting model in a Bayesian model, and show that it can quantitatively fit the experimental data fairly well. This section shows that the Bayes+ACT-R+DRT framework we introduced enables us to specify in a fully explicit way different competence-performance hypotheses about conditionals, and that experimental evidence from real-time experiments can be used to quantitatively compare alternative theories of conditionals and cataphora.

Finally, Sect. 9.5 concludes with a brief summary and an outline of directions for future research.

9.1 Two Experiments Studying the Interaction Between Conditionals and Cataphora

Brasoveanu and Dotlačil (2015a) investigate whether meaning representations commonly used in formal semantics are built up incrementally and predictively when language is used in real time, similar to the incremental and predictive construction of syntactic representations (Steedman 2001; Lewis and Vasishth 2005; Lau 2009; Hale 2011 among many others).

The main empirical challenge when studying the incremental processing of *semantic* representations is identifying phenomena that can tease apart the syntactic and semantic components of the interpretation process. The pervasive aspects of meaning composition that are syntax-based cannot provide an unambiguous window into the nature of semantic representation building: the incremental and predictive nature of real-time compositional interpretation could be primarily or exclusively due to our processing strategies for building syntactic representations.

There is a significant amount of work in psycholinguistics on incremental interpretation (Hagoort et al. 2004; Pickering et al. 2006 among many others), but this research usually focuses on the processing of lexical semantic and syntactic representations, and the incremental integration of world knowledge into the language interpretation process. The processing of logical representations of the kind formal semanticists are interested in is much less studied.

Similarly, there is a significant amount of work on incremental interpretation in natural language processing/understanding (Poesio 1994; Bos et al. 2004; Bos 2005; Hough et al. 2015 among many others), but this research usually discusses it from a formal and implementation perspective, and focuses much less on the cognitive aspects of processing semantic representations (the research in Steedman 2001 and related work is a notable exception).

Brasoveanu and Dotlačil (2015a) report two studies that argue for the incremental nature of processing formal semantic representations, as distinct from the syntactic representations they supervene on. The crucial evidence is provided by the interaction of anaphora and presupposition resolution on one hand, and conjunctions versus conditionals with a sentence-final antecedent on the other. Consider the contrast between AND and IF in the example below, where the presupposition trigger *again* is cataphoric:

- (1) Tina will have coffee with Alex again AND/IF she had coffee with him at the local café.

Assume the construction of semantic representations is *incremental*, i.e., the interpreter processes *if* as soon as it is encountered. Furthermore, assume incremental interpretation is *predictive*, i.e., once *if* is read, the interpreter expects the upcoming *if*-clause to provide (some of) the interpretation context for the previously processed matrix clause. Then we expect to see a facilitation/speed-up in the second clause *she had coffee with him* after *if* is read, compared to when the same clause follows *and*. As we will see, this is what the experimental results show.

Specifically, we expect the second clause in (1) to be more difficult after *and* than after *if* because *and* signals that a potential antecedent for the *again* presupposition is unlikely to come after this point. Dynamic conjunction is interpreted sequentially: the second conjunct is interpreted relative to the context provided by the first conjunct, and not vice-versa. Consequently, an unresolved presupposition in the first conjunct cannot find an antecedent in the second conjunct.

In contrast, *if* leaves open the possibility that a suitable resolution for the *again* presupposition is forthcoming since the first clause (the matrix) is interpreted relative to the context provided by the second clause (the *if*-clause). This possibility allows interpreters to make better predictions about the content of the clause coming after *if*, which should ease its processing.

Crucially, our expectations—which arise from the interaction between the presupposition trigger *again* and the operators *and* versus *if*—are semantically driven. Nothing in the syntax of conjunction versus *if*-adjunction could make a successful presupposition resolution more or less likely.

9.1.1 Experiment 1: Anaphora Versus Cataphora in Conjunctions Versus Conditionals

Elbourne (2009, 1) defines donkey cataphora as “a configuration in which a pronoun precedes and depends for its interpretation on an indefinite that does not c-command it.” Some cataphora examples with conditionals are provided below, both with sentence-initial, (2)–(6), and with sentence-final *if*-clauses, (7).

- (2) If it is overcooked, a hamburger doesn't taste good. (Chierchia 1995, 129)
- (3) If she finds it spectacular, a photographer takes many pictures of a landscape. (Chierchia 1995, 130)
- (4) If it enters his territory, a pirate usually attacks a ship. (Chierchia 1995, 130)
- (5) If it spots a mouse, a cat attacks it. (Chierchia 1995, 130)
- (6) If a foreigner asks him for directions, a person from Milan replies to him with courtesy. (Chierchia 1995, 130)
- (7) John won't eat it if a hamburger is overcooked. (Elbourne 2009, 3)

Certain configurations are not acceptable (Elbourne 2009, 2), e.g., (8c) below, due to Principle C violations. Antecedents are marked with a superscript, and the corresponding anaphors/cataphors are marked with a subscript.

- (8) a. Johnⁱ is upset if he_i sees a donkey.
 b. If Johnⁱ sees a donkey, he_i is upset.
 c. *He_i is upset if Johnⁱ sees a donkey.

The contrast between (8b) and (8c), as well as the fact that Principle C is not violated if cataphoric pronouns appear in object position (see (7) above and (9) below), provide evidence that a sentence-final *if*-clause is adjoined lower than the matrix-clause subject, but higher than the object. For concreteness, let's say that a sentence-final *if*-clause is VP-adjoined.

- (9) Bill visits her_i if Mary_i is sick.

In contrast, a sentence-initial *if*-clause is adjoined higher than the matrix-clause subject. For concreteness, let's say it is CP-adjoined.

Other arguments for these two syntactic structures are provided by VP ellipsis, as in (10), and VP topicalization, as in (11); see Bhatt and Pancheva (2006) for more discussion.

- (10) I will leave if you do, and John will [~~leave if you do~~] too / do so too.

- (11) I told Peter to take the dog out if it rains, and [~~take the dog out if it rains~~] he will. (Iatridou 1991, 12)

Based on these observations, Brasoveanu and Dotlačil (2015a) conclude that there is no 'ordinary' syntax-mediated binding from a c-commanding position for direct object (DO) donkey cataphora in conditionals with sentence-final *if* clauses. That is, donkey cataphora from the DO position of the matrix clause is a 'true' example of donkey cataphora that can be used to test the incrementality and predictiveness of semantic parsing.

Brasoveanu and Dotlačil's Experiment 1 tested donkey anaphora and cataphora in a 2 × 2 design, exemplified in (12):

- (12) Brasoveanu and Dotlačil's Experiment 1: AND/IF × DO ANAPHORA/CATAPHORA
- a. An electrician examined a radio for several minutes AND his helpful colleague held it that whole time. AND & ANAPHORA
 - b. An electrician examined a radio for several minutes IF his helpful colleague held it that whole time. IF & ANAPHORA
 - c. An electrician examined it for several minutes AND his helpful colleague held the radio that whole time. AND & CATAPHORA
 - d. An electrician examined it for several minutes IF his helpful colleague held the radio that whole time. IF & CATAPHORA

Kazanina et al. (2007) used an on-line reading methodology (self-paced reading) to show that a cataphoric pronoun triggers an active search for an antecedent in the following material, and that this search takes into account structural constraints (Principle C) from an early stage.¹ Kazanina et al. (2007) take the temporal priority of syntactic information as evidence for the incremental and predictive nature of

¹That is, cataphoric dependencies are processed with a syntactically constrained search mechanism similar to the mechanism used for processing long-distance *wh*-dependencies (Stowe 1986; Traxler and Pickering 1996; Wagers et al. 2009 among others).

syntactic constraints. The question investigated in this experiment can therefore be further specified as: is this active search mechanism also semantically constrained?

The methodology used in Brasoveanu and Dotlačil's Experiment 1 was also self-paced reading (Just et al. 1982) with a non-cumulative moving window. The regular anaphora cases provide the baseline conditions. Assuming a deep enough incremental and predictive interpretation, the second clause in these conditions is expected to be more difficult after *if* than after *and* because of extra cognitive load coming from two sources.

The first source of difficulty is the semantics of conditionals versus conjunctions. For conditionals, we generate a hypothetical intermediate interpretation context satisfying the antecedent, and we evaluate the consequent relative to this hypothetical context. That is, we need to maintain both the actual, global interpretation context and the intermediate, antecedent-satisfying context, to complete the interpretation of conditionals. There is no similar cognitive load for conjunctions.

The second source of difficulty is specific to conditionals with a sentence-final *if*-clause. When such conditionals are incrementally interpreted, the matrix clause needs to be semantically reanalyzed. The matrix clause is initially interpreted relative to the global context, just like a top-level assertion is. However, when *if* is reached, the matrix clause has to be reinterpreted relative to the intermediate, antecedent-satisfying context: the comprehender realizes that the matrix clause is not a top-level assertion, but is the consequent of a conditional instead. There is no such difficulty for conjunctions: the first conjunct is simply interpreted relative to the global context, and the second conjunct is interpreted relative to the context that is the result of the update contributed by the first-conjunct.

For the cataphora (non-baseline cases), we expect a cognitive load reversal. The conjunction *and* signals that no suitable antecedent for the cataphor is forthcoming since the second clause is interpreted relative to the context provided/updated by the first clause. In contrast, *if* triggers the semantic reanalysis of the matrix clause and leaves open the possibility that a suitable antecedent for the cataphor is forthcoming since the matrix clause is interpreted relative to the context provided/updated by the second clause. This expectation (and the fact that it is confirmed) should speed up the processing. So we expect to see a speed-up in the IF & CATAPHORA cases, i.e., a negative IF $\times$ CATAPHORA interaction.

These predictions were only partially confirmed in Brasoveanu and Dotlačil's first experiment: baseline IF was indeed harder (statistically significant) but the IF $\times$ CATAPHORA interaction, while negative, did not reach significance.

The regions of interest (ROIs) were primarily (i) the post-connective ROIs *his helpful colleague*, and secondarily (ii) the post-resolution ROIs *that whole*; see (13) below.

The mean log reading times (log RTs) for the 5 ROIs are plotted in Fig. 9.1 (plots created with R and ggplot2; R Core Team 2014; Wickham 2009).

- (13) An electrician examined a radio/it for several minutes and/if
his helpful colleague held it/the radio that whole time.

Fig. 9.1 Experiment 1: mean log RTs for the five regions of interest (ROIs)

Brasoveanu and Dotlačil analyzed the data using linear mixed-effects models. The response was the log-transformed readings times (log RTs) for the 3 ROIs immediately following the sentential operator *and/if* (RTs are log-transformed to mitigate their characteristic right-skewness). Residualized log RTs (residualized for word length and word position, following Trueswell et al. 1994) were also analyzed, but the pattern of results did not change, so Brasoveanu and Dotlačil report the more easily intelligible models with raw log RT responses.

The predictors (fixed effects) were as follows:

- main effects of CONNECTIVE and ANA/CATAPHORA, and
- their interaction;
- the levels of the CONNECTIVE factor: AND (reference level) versus IF;
- the levels of the ANA/CATAPHORA factor: ANAPHORA (reference level) versus CATAPHORA.

Crossed random effects for subjects and items were included, and the models with maximal random effect structures that converged (Barr et al. 2013) were reported, usually subject and item random intercepts, and subject and item random slopes for at least one of the two main effects. The maximum likelihood estimates (MLEs) and associated standard errors (SEs) and *p*-values are provided in (14) and (15) below (we omit the intercepts). Significant and nearly significant effects ($p < 0.1$) are boldfaced.

(14)

	his			helpful			colleague		
	MLE	SE	p	MLE	SE	p	MLE	SE	p
IF	0.02	0.02	0.3	0.05	0.02	0.04	0.05	0.02	0.04
CATA	0.01	0.02	0.7	0.02	0.02	0.4	0.03	0.02	0.16
IF×CATA	−0.003	0.03	0.9	−0.04	0.03	0.15	−0.02	0.03	0.6

(15)

	that			whole		
	MLE	SE	p	MLE	SE	p
IF	0.03	0.02	0.20	0.03	0.02	0.24
CATA	0.04	0.02	0.07	0.03	0.02	0.18
IF×CATA	−0.03	0.03	0.26	−0.03	0.03	0.43

Table 14 shows several effects. First, baseline IF (i.e., IF & ANAPHORA) is more difficult than baseline AND (i.e., AND & ANAPHORA). This effect is compatible with the hypothesis that to interpret conditionals, we need to maintain both the actual, global interpretation context and the intermediate, antecedent-satisfying context. The effect is also compatible with the hypothesis that the matrix clause is reanalyzed in conditionals with a final *if*-clause: it is initially interpreted relative to the global context until *if* is reached, at which point it is reinterpreted relative to the intermediate, antecedent-satisfying context.

CATAPHORA seems to be more difficult than ANAPHORA for AND, but the effect never reaches significance (it is close to significant in the first ROI after cataphora is resolved). Maybe the AND & CATAPHORA condition is simply too hard, so readers stop trying to fully comprehend the sentence and speed up. If so, this will obscure the IF × CATAPHORA interaction: there is a negative interaction between IF and CATAPHORA in all ROIs, i.e., IF seems to facilitate CATAPHORA (as expected if semantic evaluation is incremental and predictive), but this effect is not significant.

The consistent negative interaction is promising, so Brasoveanu and Dotlačil (2015a) elicited it in a follow-up experiment with a hard presupposition trigger (*again*; Abusch 2010; Schwarz 2014 among others), which might have a larger effect. A third ‘(mis)match’ manipulation was also added to control for readers speeding up through conditions that are too hard.

9.1.2 Experiment 2: Cataphoric Presuppositions in Conjunctions Versus Conditionals

The second experiment in Brasoveanu and Dotlačil (2015a) had a 2 × 2 × 2 design, exemplified in (16) below. The match/mismatch manipulation was new, and consisted of verbs in the second clause that matched or didn’t match the corresponding verbs in the first clause.

- (16) Experiment 2: (MIS)MATCH × AND/IF × NOTHING/CATAPHORA
- a. Jeffrey will *argue* with Danielle AND he *argued* with her in the courtyard last night.
- MATCH & AND & NOTHING

- b. Jeffrey will *argue* with Danielle IF he *argued* with her in the courtyard last night. MATCH & IF & NOTHING
- c. Jeffrey will *argue* with Danielle again AND he *argued* with her in the courtyard last night. MATCH & AND & CATAPHORA
- d. Jeffrey will *argue* with Danielle again IF he *argued* with her in the courtyard last night. MATCH & IF & CATAPHORA
- e. Jeffrey will *argue* with Danielle AND he *played* with her in the courtyard last night. MISMATCH & AND & NOTHING
- f. Jeffrey will *argue* with Danielle IF he *played* with her in the courtyard last night. MISMATCH & IF & NOTHING
- g. Jeffrey will *argue* with Danielle again AND he *played* with her in the courtyard last night. MISMATCH & AND & CATAPHORA
- h. Jeffrey will *argue* with Danielle again IF he *played* with her in the courtyard last night. MISMATCH & IF & CATAPHORA

The method was similar to Experiment 1. Self-paced reading with a moving window was used, but each stimulus ended with an acceptability judgment on a 5-point Likert scale, from 1 (very bad) to 5 (very good). The acceptability judgment was elicited on a new screen after every item or filler. Every experimental item was followed by a comprehension question. Each of the 8 conditions was tested 4 times, for 32 items total; one item had a typo, and was discarded from all subsequent analyses. There were 70 fillers: monoclausal and multiclausal, conditionals, conjunctions, *when*-clauses, relative clauses, quantifiers, adverbs, etc.

Thirty-two native speakers of English participated (UCSC undergraduate students). They completed the experiment online for course (extra-)credit on a UCSC hosted installation of the IBEX platform. Each item was passed through all 8 conditions, and 8 lists were generated following a Latin square design. The participants were rotated through the 8 lists. Every participant responded to 102 stimuli (32 items + 70 fillers), the order of which was randomized for every participant. Any two items were separated by at least one filler.

There were fillers that were both acceptable (*Bob ate his burger and he rented something to watch, but he didn't say what*) and unacceptable (*Willem visited Paris because Sarah visited Amsterdam too*). All participants exhibited the expected difference in acceptability ratings between these two types of fillers.

There were 72 comprehension questions with correct/incorrect answers, 32 after experimental items. The accuracy for all participants was above 80%.

The results of this study are visually summarized in Fig. 9.2. The ROIs for Exp. 2 are the words following the verb in the second clause, i.e., the words immediately following the last experimental manipulation, which is (MIS)MATCH. Brasoveanu and Dotlačil examined only the 4 immediately post-verbal ROIs because the fifth word was the final one for some items, and the wrap-up effect associated with sentence-final words would contribute additional, possibly biasing noise.

Fig. 9.2 Experiment 2: mean log RTs for the four regions of interest (ROIs)

(17) Jeffrey will argue with Danielle $\emptyset$ /again and/if he argued/played
with her in the courtyard last night.

The data analysis was similar to the one conducted for Experiment 1: linear mixed-effects models with log RTs as the response variable, and main effects of CONNECTIVE and NOTHING/CATAPHORA, MATCH/MISMATCH and their 2-way and 3-way interactions as predictors (fixed effects). The levels of the 3 factors were as follows:

- for CONNECTIVE, AND (reference level) versus IF,
- for NOTHING/CATAPHORA: NOTHING (reference level) versus CATAPHORA, and
- for (MIS)MATCH: MATCH (reference level) versus MISMATCH.

The models also included crossed random effects for subjects and items, namely the maximal random effect structure that converged (usually subject and item random intercepts, and subject and item random slopes for at least two of the three main effects). The statistical modeling results are summarized in (18) below (once again, we omit the intercepts).

(18)	with			her			in			the		
	MLE	SE	p	MLE	SE	p	MLE	SE	p	MLE	SE	p
CATA	0.05	0.04	0.21	0.05	0.04	0.30	0.03	0.04	0.50	0.05	0.04	0.23
MISMATCH	0.05	0.04	0.25	0.04	0.04	0.33	0.06	0.05	0.19	0.07	0.05	0.15
IF	0.08	0.04	0.054	0.07	0.04	0.084	0.05	0.04	0.24	−0.003	0.04	0.96
CATA × MISMATCH	−0.11	0.06	0.056	−0.05	0.06	0.42	−0.03	0.06	0.59	−0.14	0.06	0.03
CATA × IF	−0.13	0.06	0.026	−0.11	0.06	0.077	−0.08	0.05	0.15	−0.04	0.06	0.54
MISMATCH × IF	−0.10	0.06	0.083	−0.06	0.06	0.30	−0.02	0.05	0.73	−0.02	0.06	0.76
CATA × MISMATCH × IF	0.20	0.08	0.015	0.10	0.08	0.22	0.06	0.08	0.42	0.11	0.09	0.19

Just as in Experiment 1, baseline IF (i.e., IF & NOTHING & MATCH) is more difficult than baseline AND (i.e., AND & NOTHING & MATCH). This is compatible with the hypothesis that conditionals are harder than conjunctions—because we need to maintain two evaluation contexts, and/or because the matrix clause is semantically reanalyzed when *if* is reached.²

There is a significant negative interaction of MISMATCH×IF (note: *again* is not present here), which basically cancels out the main effect of IF. That is, conditionals with non-identical VP meanings in the antecedent and consequent clauses are processed more easily than conditionals with identical VP meanings, about as easily as conjunctions with non-identical VP meanings in the two conjuncts. The difficulties tied to conditionals with identical VP meanings are probably caused by a violation of Maximize Presupposition (Heim 1991), which requires that a presupposed VP meaning should be marked as such by *again*. This penalizes conditionals with matching VP meanings, while conditionals with non-identical VP meanings are not affected. Furthermore, if participants interpret incrementally and predictively, Maximize Presupposition should not affect coordinations (specifically, the first conjunct in a coordination), which corresponds to our findings.

The Maximize Presupposition constraint provides a third possible reason for the cost of baseline IF relative to baseline AND aside from the suggestions discussed before, namely that conditionals are harder than conjunctions because we need to maintain two evaluation contexts, and/or because the matrix clause is semantically reanalyzed. They all might be at work here (distinguishing between them is left for a future occasion), but Maximize Presupposition might be particularly suitable as an explanation for the Experiment 2 results: it explains the cost of IF, but it also explains the negative interaction MISMATCH × IF, which is unexpected under the hypothesis that IF on its own is costly. Furthermore, in Experiment 2, the effect of IF is observed on *with* and *her*, which makes the explanation in terms of reanalysis unlikely given the lateness of the effect. In Experiment 1, the effect of IF was detectable on the second word after *if*, so the reanalysis explanation is more plausible for that experiment.

There are no main effects of CATAPHORA and MISMATCH, but their 2-way interaction is negative and significant (or close to significant) in two out of the four regions. Whenever (close to) significant, this interaction effectively cancels the main effects of both MISMATCH and CATAPHORA. That is, the AND & CATAPHORA & MISMATCH condition is about as difficult as the reference condition AND & NOTHING & MATCH, which suggests that participants stopped trying to properly interpret the difficult condition AND & CATAPHORA & MISMATCH and moved on/sped up.

²Since Experiment 2 included acceptability judgments, Brasoveanu and Dotlačil were able to check whether the effect, replicated from Experiment 1, is not due to the fact that the IF condition (*Jeffrey will argue with Danielle if he argued with her in the courtyard last night*) is less felicitous than the corresponding AND condition (*Jeffrey will argue with Danielle and he argued with her in the courtyard last night*). This is not so: the only statistically significant fixed effect was a *positive* main effect for IF, i.e., baseline IF is more acceptable than baseline AND, as estimated by a mixed-effects ordinal probit model (full fixed-effect structure, i.e., main effects + all interactions, & maximal random effect structure that converged).

There is a (close to) significant negative interaction of CATAPHORA $\times$ IF in the two regions immediately following the verb (note: we are discussing MATCHING conditions). In both regions, this 2-way interaction effectively cancels out the positive main effects of CATAPHORA and IF put together. This is exactly the configuration Brasoveanu and Dotlačil were looking for in Experiment 1, only it did not reach significance there. That is, IF facilitates the processing of CATAPHORA, even though IF and CATAPHORA on their own are more difficult. This supports the hypothesis that the construction of formal semantic representations is incremental and predictive.

Finally, the statistically significant and positive 3-way interaction CATAPHORA $\times$ IF $\times$ MISMATCH in the region immediately following the verb provides further empirical support for the hypothesis that the construction of formal semantic representations is incremental and predictive. The MISMATCH is surprising because the human interpreter expects to find a suitable antecedent for the *again* presupposition, and that expectation is not satisfied.

In sum, Experiments 1 and 2 in Brasoveanu and Dotlačil (2015a) provide coherent support for the incremental and predictive nature of the process of constructing meaning representations of the kind employed in formal semantics.

9.2 Mechanistic Processing Models as an Explanatory Goal for Semantics

The main questions at this point are the following. As formal semanticists, should we account for the incremental and predictive nature of the real-time semantic interpretation process? And if so, how?

It is important to remember that addressing these questions is firmly rooted in the tradition of dynamic semantics. Kamp (1981) begins like this:

“Two conceptions of meaning have dominated formal semantics of natural language. The first of these sees meaning principally as that which determines conditions of truth. [...] According to the second conception meaning is, first and foremost, that which a language user grasps when he understands the words he hears or reads. [...] these two conceptions [...] have remained largely separated for a considerable period of time. This separation has become an obstacle to the development of semantic theory [...] The theory presented here is an attempt to remove this obstacle. It combines a definition of truth with a systematic account of semantic representations.” (Kamp 1981, 189)

Thus, the implicit overarching goal for us as (cognitive) scientists studying natural language meaning and interpretation is to provide a formally explicit account of natural language interpretive *behavior*, i.e., a mathematically explicit, unified theory of semantic/pragmatic competence *and* performance.

To contextualize our position and outline some possible alternatives, let us consider the corresponding debate on the syntax side. Phillips and Lewis (2013, 14) identify two reasonable positions that working linguists more or less implicitly subscribe to in practice: (i) principled extensionalism, and (ii) strategic extensionalism.

Principled extensionalism takes a grammar/grammatical theory to be merely an abstract characterization of a function whose extension is all and only the well-formed sentences of a given language (plus their denotations, if the grammar incorporates a semantic component).³ The individual components of the grammatical theory have no independent status as mental objects or processes: they are components of an abstract function, not of a more concrete description of a mental system.

This kind of position cannot be tested using most empirical evidence aside from acceptability (or truth-value/entailment) judgments, since the position only aims to capture the ‘end products’ of the grammatical system and not the way these products are actually produced/comprehended.

The ‘principled’ part is that the extensionalist enterprise is understood as an end in itself, relevant even if lower-level characterizations of the human language system are provided (algorithmic/mechanistic, or implementation/neural level; Marr 1982). The linguist’s task is to characterize *what* the human language system computes and distinguish it from *how* speakers actually carry out that computation, which is the psycholinguist’s task.

The *strategic extensionalism* position takes the goal of formulating a grammatical theory to be a reasonable interim goal, but not an end in itself. The ultimate goal is to move beyond extensional description to a more detailed, mechanistic understanding of the human language system: describing an abstract function that identifies all the grammatical sentences of a language is just a first step in understanding how speakers actually comprehend/produce sentences in real time. We seek theories that capture *how* sentences are put together, and not just *what* their final form is. From this perspective, we *should* try to account for left-to-right structure building mechanisms, both at the syntactic and at the semantic level.

The strategic-extensionalism position is closely related to the cognitive-architecture based approach to research in cognitive science, which we have in fact followed throughout this book. As Anderson (2007, 7–8) puts it:

“A *cognitive architecture* is a specification of the structure of the brain at a level of abstraction that explains how it achieves the function of the mind [...] [i.e.,] human cognition in all of its complexity. [...] Th[is] type of architectural program [...] requires paying attention to three things: brain, mind (functional cognition), and the architectural abstractions that link them. [...] [A]pproaches that tried to get by with less [...] can be viewed as shortcuts to understanding.”

Anderson (2007) goes on to consider three such shortcuts. The first one is classical information-processing psychology that completely ignores the brain (at *any* level of abstraction), and that is basically the same as the principled-extensionalism position we characterized above. However, unlike much of formal semantics, cognitive psychology has realized by now that “cognition is not so abstract that our understanding of it can be totally divorced from our understanding of [the] physical reality [underlying it].” (Anderson 2007, 11)

³More precisely, assume some background alphabet Σ that consists of the lexicon/set of words (‘alphabet’ in the sense of formal language theory), and let Σ^* be the set of all finite strings over Σ . Σ^* is the domain of the function, and $\{0, 1\}$ its range, so that the function is a characteristic function of the set of grammatical strings.

This does not mean, of course, that the opposite position—eliminative connectionism—is not a shortcut also. “This approach ignores mental function as a constraint and just provides an abstract characterization of brain structure [...] [but these models] work only because we are able to imagine how [they] could serve a useful function in a larger system [...] [However, this] functionality is not achieved by a connectionist system.” (Anderson 2007, 11–14)

Finally, a third shortcut that has become recently fairly popular in formal semantics and pragmatics, is the rational-analysis approach to cognition. The basic insight behind this approach is that “a constraint on how the brain achieves the mind is that both the brain and the mind have to survive in the real world: rather than focus on architecture as the key abstraction, focus on adaptation to the environment [...] [T]he Bayesian statistical methodology that accompanies much of this research [...] comes to take the place of the cognitive architecture.” (Anderson 2007, 15–16)

Anderson’s own work on declarative memory is an early instantiation of this approach; see Anderson (1990); Anderson and Schooler (1991); Schooler and Anderson (1997). As we discussed in detail in Chap. 6, the ACT-R base activation equation encodes that “a memory for something diminishes in proportion to how likely people are to need that memory. [...] Human memory [...] mirror[s] statistical relationship[s] in the environment. [...] Thus, the argument goes, one does not need a description of how memory works, which is what an architecture gives; rather, one just needs to focus on how memory solves the problems it encounters.” (Anderson 2007, 17)

While possibly enlightening for individual cognitive components, this approach falls short of a complete theory of the human mind.

“[T]he human mind is not just the sum of core competences such as memory, or categorization, or reasoning. It is about how all these pieces and other pieces work together to produce cognition. All the pieces might be adapted to the regularities in the world, but understanding their individual adaptations does not address how they are put together. [...] What distinguishes humans is their ability to bring the pieces together, and this unique ability is just what adaptive analyses do not address, and just what a cognitive architecture is all about.” (Anderson 2007, 18)

In this book, we have consistently taken a cognitive-architectural approach to natural language meaning and interpretation. Our ultimate goal is to provide a framework in which we can build mechanistic processing models for natural language comprehension, with pieces that are independently needed for other higher-level cognitive processes.

It is in this context that we introduced and used Bayesian methods: we use them for theoretically-informed *data analysis*. More precisely, we use them as essential bridges that systematically connect independently-motivated semantics and processing theories and cognitive-architectural organization principles and constraints on one hand, and experimental data on the other hand.

9.3 Modeling the Interaction of Conditionals and Pronominal Cataphora

Assuming a cognitive-architecture based approach to semantics, like we have done throughout this book (or a strategic extensionalist position), the next question is: how should we account for the incremental and predictive nature of semantic interpretation? We will not settle this question here, but we will outline two distinct approaches and flesh out in detail one of them.

As far as we can tell, there is a spectrum of approaches to incrementality effects, and the two extremes on that spectrum are accounting for incrementality (i) in *the semantics* versus (ii) in *the processor*.

The first alternative is parallel to the proposal in Phillips (1996, 2003) on the syntax side. The main claim in Phillips (1996, 2003) is that syntactic structures are built left-to-right, not top-down/bottom-up, and the incremental left-to-right system is the *only* structure-building system that humans have ('the parser is the grammar').

A similar proposal on the semantics side is sketched in Brasoveanu and Dotlačil (2015a, b). The idea is to provide a recursive definition of truth and satisfaction for first-order predicate logic that is fully incremental, building on the incremental propositional logic system in Vermeulen (1994). The resulting system, dubbed Incremental Dynamic Predicate Logic (IDPL), builds incrementality into the heart of semantics.

The second alternative is parallel to the proposal in Hofmeister et al. (2013) on the syntax side, the main goal of which is to argue that "many of the findings from studies putatively supporting grammar-based interpretations of island phenomena have plausible, alternative interpretations rooted in specific, well-documented processing mechanisms" (Hofmeister et al. 2013, 44). The remainder of this chapter is dedicated to fleshing out this approach.

Our specific proposal on the processing side is to extend the eager left-corner parser for DRT we introduced in the previous chapter with conjunctions, conditionals and anaphora/cataphora, so that we can explicitly and fully model the two self-paced reading experiments discussed in Sect. 9.1 above.

In this section, we introduce the basic model that captures the qualitative pattern of interactions between cataphora and conjunctions versus conditionals in Experiment 1. In the next section, we introduce the model in its full complexity. The full model can syntactically and semantically parse the items in Experiment 2, which enables us to quantitatively fit it to the data from Experiment 2.

To model pronominal and presuppositional anaphora/cataphora, we add a new goal-like buffer `unresolved_discourse` to our ACT-R mind, which will store the unresolved DRs contributed by pronouns and the presuppositional trigger *again*. We set the encoding delay for this buffer, as well as the `imaginal` and `discourse_context` buffers we used in the previous chapter to 0:

```
(19) parser.set_goal(name="imaginal", delay=0)           1
      parser.set_goal(name="discourse_context", delay=0) 2
      parser.set_goal(name="unresolved_discourse", delay=0) 3
```

In principle, we might be able to model anaphora and cataphora without this additional `unresolved_discourse` buffer, but we decided to use it here for presentational clarity.⁴

9.3.1 *Chunk Types and the Lexical Information Stored in Declarative Memory*

The chunk types we will need are the same as the ones we used in the previous chapter, plus a new chunk type for predicates. They are listed in (20) below.

```
(20)  actr.chunktype("parsing_goal",                                1
      "task stack1 stack2 stack3 \                                2
      arg_stack1 arg_stack2 \                                    3
      right_edge_stack1 right_edge_stack2 \                      4
      right_edge_stack3 right_edge_stack4 \                      5
      parsed_word found discourse_status \                        6
      dref_peg event_peg drs_peg prev_drs_peg embedding_level \  7
      entity_cataphora event_cataphora if_conseq_pred")          8
      actr.chunktype("parse_state",                                9
      "node_cat daughter1 daughter2 daughter3 \                  10
      mother mother_of_mother lex_head")                        11
      actr.chunktype("word", "form cat pred1 pred2")             12
      actr.chunktype("pred", "constant_name arity")              13
      actr.chunktype("drs",                                       14
      "dref pred1 pred2 event_arg arg1 arg2 \                    15
      discourse_status drs embedding_level")                      16
```

Parsing goal chunks have the expected features:

- the current parsing task;
- the stack of syntactic goals driving the parsing process (`stack1 stack2 stack3`);
- a stack for arguments (`arg_stack1 arg_stack2`) that need to be passed across different semantic chunks, e.g., from the subject to the verbal predicate;
- the right-edge stack keeps track of possible points of attachment made available by the current, partially-built syntactic tree (`right_edge_stack1 right_edge_stack2 ...`);
 - we called this stack `right_frontier` in the previous chapter, but we renamed it here for brevity;
 - the right-edge stack we need in this chapter has additional positions because of the need to attach conjuncts and *if*-adjuncts;
- the `parsed_word` and `found` features are used in much the same way as in the previous chapters;
- the `discourse_status` feature will keep track of whether a DRS constructed at some point during the incremental interpretation process:

⁴We are of course aware that determining the exact number of modules and buffers needed for natural language interpretation, as well as their subsymbolic properties, is an empirical issue.

- contributes to the `at_issue` meaning, or
- is unresolved, e.g., it is the presupposition contributed by a pronoun or the adverb *again*, or
- is presupposed, i.e., it is the resolved presupposition contributed by a pronoun or the adverb *again*;
- just as in the previous chapter, we introduce new discourse referents (drefs) with fresh (previously unused) indices by keeping track of the current `dref_peg/index` in the goal buffer and updating it as soon as a dref with that `peg/index` is introduced;
 - but in this chapter, we have event drefs (needed for *again*) in addition to individual-level drefs, so we also keep track of the current `event_peg`;
 - in addition, we keep track of which sub-DRSs are part of the main DRS by associating them with current `drs_peg`; this is a flatter/simpler solution than the one we used in the previous chapter, where a main DRS had sub-DRSs stored in its slots;
 - DRS drefs are basically propositional drefs and are independently needed for conditionals, for example; we keep track of the previous DRS `peg` (`prev_drs_peg`) to be able to capture the semantic reanalysis triggered by sentence-final *if*-clauses;
- finally, the three features `entity_cataphora`, `event_cataphora` and `if_conseq_pred` are needed to account for the processing of cataphoric pronouns and *again*, and will be discussed in detail later in the chapter.

Chunks of `parse_state` type have the same structure as before, except for the addition of a `mother_of_mother` feature. This feature enables us to keep track of the partial syntactic structure constructed by the incremental comprehension process in a little more detail at the local level of an individual chunk.

The lexical entry of a word keeps track of:

- its written form (a proxy for its phonological representation),
- its syntactic category, and
- up to two predicates `pred1` and `pred2` that represent the meaning of that word.

We use these two predicate slots in various ways. For example, a proper name like *Danielle* contributes:

- a predicate `DANIELLE` (with a singleton set denotation) as its `pred1` value (this follows the analysis of proper names in Kamp and Reyle 1993), and
- the gender predicate `FEMALE` as its `pred2` value that can be leveraged to resolve a subsequent pronoun *she/her* anaphoric to the proper name.

The values of these two `pred1` and `pred2` features are chunks of type `pred`, which specify the name of the non-logical constant associated with the predicate (the feature `constant_name`) and the arity of the constant.

Finally, chunks of type `drs` have the same structure as the one used in the previous chapter, with the addition of several new features necessary to capture the interaction of cataphora and conjunctions/conditionals:

- the `dref` feature keeps track of the new propositional `dref` (if any) introduced by the DRS;
- `pred1` and `pred2` store the predicates that are part of the conditions contributed by the DRS;
- `event_arg` stores the event `dref` (if any) taken as an argument by `pred1` and/or `pred2`;
- `arg1` and `arg2` store the entity `drefs` that are the arguments of `pred1` and/or `pred2`;
- `discourse_status` keeps track of the discourse status of the DRS;
 - we only need three possible values for this feature in this chapter: `at_issue`, `unresolved` and `presupposed`;
- the `drs` feature keeps track of the DRS `peg` that the current DRS is associated with;
- finally, `embedding_level` keeps track of the embedding level of the DRS, the main function of which is to constrain pronoun and anaphora resolution, just as in Kamp and Reyle (1993);
 - discourse-initial main clauses are `embedding_level 0`;
 - conditional antecedents or the second conjunct in a conjunction are `embedding_level 1`;
 - conditional consequents or the third conjunct in a conjunction will be `embedding_level 2`;
 - pronouns and presuppositions, whether anaphoric or cataphoric, can only find antecedents at a higher embedding level (or at the same level in certain cases).

Let us look at some example lexical entries that will be stored in declarative memory (`dm`). The lexical entry of a proper name like *Danielle* is a chunk of the following form:

```
(21)  actr.chunkstring(string="""
        isa  word
        form Danielle
        cat  ProperN
        pred1 DANIELLE
        pred2 FEMALE
        """)
```

1
2
3
4
5
6
7

In (21), we use the `chunkstring` method to assemble the lexical entry for the proper name *Danielle* (of type `word`) from a Python3 string. The values of the `form` and `cat` features are as expected. The `pred1` and `pred2` values are themselves chunks of type `pred`. These predicate chunks are assumed to be already available in `dm` at the time we assemble the lexical entry for *Danielle*, and they are declared as follows:

```

(22)  actr.chunkstring(name="DANIELLE", string=""           1
      isa      pred                                         2
      constant_name _danielle_                               3
      arity      1                                         4
      """)                                                5

(23)  actr.chunkstring(name="FEMALE", string=""           1
      isa      pred                                         2
      constant_name _female_                               3
      arity      1                                         4
      """)                                                5

```

Just as in the previous chapter, the lexical entry for the determiner *a* does not contain any semantic information. The associated semantic representations and operations, namely, introducing a new dref, predicating the common noun and the verbal predicate of it etc., are all contributed by production rules stored in procedural memory.

A pronoun like *she* has a lexical entry of the following form:

```

(24)  dm.add(actr.chunkstring(string=""                   1
      isa word                                             2
      form she                                           3
      cat PRO                                             4
      pred1 EQUALS                                       5
      pred2 FEMALE                                       6
      """))                                              7

```

The gender of the pronoun, which the antecedent of the pronoun will have to satisfy, is stored as the `pred2` value. We follow the semantics for pronouns proposed in Kamp and Reyle (1993) and assume that pronouns introduce their own dref, but they need to equate it with the dref contributed by a suitable antecedent. This is the reason for making `EQUALS` the ‘main’ predicate contributed by the pronoun, i.e., the value of the `pred1` feature. The exact specification of the `EQUALS` predicate, which has an arity of 2 (as expected), is provided in (25) below.

```

(25)  actr.chunkstring(name="EQUALS", string=""           1
      isa      pred                                         2
      constant_name _equals_                               3
      arity      2                                         4
      """)                                                5

```

It would be natural to have a lexical entry for *again* that would be parallel to that of pronouns, except that it would relate event drefs instead of entity drefs, and it would contribute the predicate `PRECEDES` in (26) below instead of `EQUALS`.

```

(26)  actr.chunkstring(name="PRECEDES", string=""         1
      isa      pred                                         2
      constant_name _precedes_                             3
      arity      2                                         4
      """)                                                5

```

It turns out, however, that it is more convenient to give *again* a semantically empty lexical entry, shown in (27) below, and let suitably formulated production rules make the correct semantic contributions.

(27) `actr.chunkstring(string=""` 1
 `isa word` 2
 `form again` 3
 `cat Adv` 4
 `""`) 5

This superficial difference between pronouns and *again* is due to the fact that, syntactically, *again* is an adjunct that needs to retrieve the VP it adjoins to and reopen it for adjunction. In addition, semantically, *again* is ‘parasitic’ on the event dref contributed by the verbal predicate it adjoins to. This is in contrast to pronouns, which introduce their own entity dref.

However, what *again* and pronouns do have in common is that they both need to be resolved: they have to relate their dref, whether they introduce that dref or ‘inherit’ it from the VP, to another dref. A successful resolution requires the other dref to be available in, and retrieved from, declarative memory.

Just like proper names, common nouns introduce two predicates, one being the common noun itself and the other being the gender. For example, the lexical entry for *car* and its associated predicate chunks are as follows:

(28) a. `actr.chunkstring(string=""` 1
 `isa word` 2
 `form car` 3
 `cat N` 4
 `pred1 CAR` 5
 `pred2 NONHUMAN` 6
 `""`) 7

b. `actr.chunkstring(name="CAR", string=""` 1
 `isa pred` 2
 `constant_name _car_` 3
 `arity 1` 4
 `""`) 5

c. `actr.chunkstring(name="NONHUMAN", string=""` 1
 `isa pred` 2
 `constant_name _nonhuman_` 3
 `arity 1` 4
 `""`) 5

Intransitive and transitive verbs introduce a single predicate, with an arity that specifies that an event argument is required, plus 1 or 2 individual-level arguments. For example, *laughed* and *greeted* have the lexical entries in (29) and (30) below.

(29) a. `actr.chunkstring(string=""` 1
 `isa word` 2
 `form laugh` 3
 `cat Vi` 4
 `pred1 LAUGH` 5
 `""`) 6

b. `actr.chunkstring(name="LAUGH", string=""` 1
 `isa pred` 2
 `constant_name _laugh_` 3
 `arity event_plus_1` 4
 `""`) 5

(30) a. `actr.chunkstring(string=""` 1
 `isa word` 2
 `form greet` 3
 `cat Vt` 4
 `pred1 GREET` 5
 `""`) 6

```

b.      actr.chunkstring(name="GREET", string=""           1
        isa      pred                                     2
        constant_name _greet_                             3
        arity    event_plus_2                             4
        """)                                              5

```

The items used in Experiment 2 (see Sect. 9.1.2 above) also contain prepositional verbs like *argue/play with*. We analyze them as transitive verbs, but we assign a different syntactic category VtPP to them. This will enable us to formulate production rules that will syntactically and semantically integrate them with the subsequent preposition.

```

(31) a.      actr.chunkstring(string=""                   1
        isa    word                                     2
        form   play                                     3
        cat    VtPP                                     4
        pred1  PLAY                                     5
        """)                                           6

b.      actr.chunkstring(name="PLAY", string=""           1
        isa      pred                                     2
        constant_name _play_                             3
        arity    event_plus_2                             4
        """)                                              5

```

For simplicity, we assume that prepositions like *with* that are part of prepositional verbs do not take an event argument, as shown in (32) below. We make the same simplifying assumption about adjectives like *overcooked*, as shown in (33).

```

(32) a.      actr.chunkstring(string=""                   1
        isa    word                                     2
        form   with                                     3
        cat    P                                         4
        pred1  WITH                                     5
        """)                                           6

b.      actr.chunkstring(name="WITH", string=""           1
        isa      pred                                     2
        constant_name _with_                             3
        arity    2                                         4
        """)                                              5

(33) a.      dm.add(actr.chunkstring(string=""            1
        isa    word                                     2
        form   hungry                                    3
        cat    A                                         4
        pred1  HUNGRY                                    5
        """))                                           6

b.      actr.chunkstring(name="HUNGRY", string=""         1
        isa      pred                                     2
        constant_name _hungry_                           3
        arity    1                                         4
        """)                                              5

```

Finally, we take the lexical entry of the sentential operators *and* and *if* to be semantically empty. The associated semantic representations and operations will all be contributed by production rules.

```

(34) a.      actr.chunkstring(string=""                   1
        isa    word                                     2
        form   and                                       3
        cat    Conj                                      4
        """)                                              5

```

```

b.      actr.chunkstring(string=""
        isa word
        form if
        cat C
        """)

```

The full code for this part of the model is linked to at the end of the chapter in Appendix 9.6.1.

9.3.2 Rules to Advance Dref Peg Positions, Key Presses and Word-Related Rules

The entire set of production rules is linked to at the end of this chapter (Appendix 9.6.2). Here, we will highlight only the most crucial ones.

We have several families of production rules that advance the dref peg position for (i) entity/individual-level drefs, (ii) event drefs and (iii) DRS drefs. The idea of peg positions was introduced and justified in Chap. 8 (see Sect. 8.3) and the rules we use in this chapter are the same, except we generalize this idea to drefs for types other than entities/individuals, namely event drefs and DRS/propositional drefs.

Turning to word-related rules, the "encode word" rule in (35) below fires whenever the parser is not engaged in a set of tasks that should take priority relative to word encoding (lines 4–14). We can think of the "encode word" rule as an 'elsewhere' rule: if the parser is not engaged in a more pressing task, it should check whether there is a value in the visual buffer that can be encoded (lines 17–19 in (35)). If such a value is available, it is encoded in the goal buffer as the value of the `parsed_word` feature (line 24).

```

(35) parser.productionstring(name="encode word", string=""
    =g>
    isa parsing_goal
    task ~move_dref_peg
    task ~move_event_peg
    task ~move_event_peg_and_wait_for_retrieval
    task ~move_drs_peg
    task ~attempting_to_resolve_PRO
    task ~attempting_to_resolve_AGAIN
    task ~attempting_to_resolve_cataphoric_PRO
    task ~attempting_to_resolve_cataphoric_AGAIN
    task ~if_reanalysis
    task ~stop_resolution_attempt_PRO
    task ~stop_resolution_attempt_AGAIN
    found None
    parsed_word None
    =visual>
    isa _visual
    value =val
    ==>
    =g>
    isa parsing_goal
    task encoding_word
    parsed_word =val
    ~visual>
    ~retrieval>
    """, utility=-1)

```

The ‘elsewhere’ nature of the "encode word" rule is also reflected in the fact that we assign it a lower utility of -1 (line 27 in (35)) than the default, which is 0. In a more realistic system that learns rule utilities from data, this utility would be automatically inferred, and we would be able to greatly simplify the rule by removing the long list of negative conditions on lines 4–14: the fact that all the tasks listed on lines 4–14 need to take priority over word encoding should arise from the utilities of the relevant productions rules, rather than being hard-coded in this fashion. However, to make rule preferences transparent, we opted for hard-coding them in this model.

The "retrieve word" rule in (36) below requests lexical information about the word we just encoded from declarative memory.

```
(36) parser.productionstring(name="retrieve word", string="" 1
      =g> 2
      isa parsing_goal 3
      task encoding_word 4
      parsed_word =w 5
      ==> 6
      +retrieval > 7
      isa word 8
      form =w 9
      =g> 10
      isa parsing_goal 11
      task retrieving_word 12
      """) 13
```

Once the lexical information is retrieved and available in the `retrieval` buffer, we shift and project the word (37). This means that we build a unary-branching syntactic structure in the `imaginal` buffer, with the word as the daughter and its syntactic category as the mother (lines 15–18 in (37)). At the same time, we update the `found` slot in the goal buffer to the same syntactic category (line 14), so that other syntax/semantics processing steps necessary to integrate the retrieved word are triggered. Finally, we start a `key_press` task (line 13). The motor operations associated with this task will execute in parallel to the additional syntax/semantics processing steps associated with the retrieved word.

```
(37) parser.productionstring(name="shift and project word (not N)", string="" 1
      =g> 2
      isa parsing_goal 3
      task retrieving_word 4
      =retrieval> 5
      isa word 6
      form =w 7
      cat =c 8
      cat ~N 9
      ==> 10
      =g> 11
      isa parsing_goal 12
      task key_press 13
      found =c 14
      +imaginal> 15
      isa parse_state 16
      node_cat =c 17
      daughter1 =w 18
      """) 19
```

The shift-and-project rule in (37) applies to all words except nouns (N). The shift-and-project N rule is different only because the preceding Det has already created an NP syntactic structure in the `imaginal` buffer that the N needs to update before

creating its own unary-branching structure in the same buffer. We do not list the rule here; the complete set of rules is linked to in Appendix 9.6.2.

The `key_press` task consists of only one rule: the "press spacebar" rule in (38) below. This rule adds a command to press the spacebar to the `manual` buffer (lines 13–16). Once that is done, we revert to the default task of parsing (line 12).

```
(38) parser.productionstring(name="press spacebar", string=""           1
      =g>                                                                2
      isa      parsing_goal                                             3
      task     key_press                                                4
      ?manual>                                                                5
      state    free                                                     6
      ?retrieval>                                                            7
      state    free                                                     8
      ==>                                                                9
      =g>                                                                10
      isa      parsing_goal                                             11
      task     parsing                                                  12
      +manual>                                                                13
      isa      _manual                                                  14
      cmd      press_key                                                15
      key      'space'                                                  16
      """)                                                                17
```

Finally, when there are no more words to be read on the virtual screen, we end the syntax/semantics parsing process with the "finished: no visual input" rule in (39) below. This rule flushes all the goal/goal-like buffers.

```
(39) parser.productionstring(name="finished: no visual input", string="" 1
      =g>                                                                2
      isa      parsing_goal                                             3
      task     parsing                                                  4
      stack1   None                                                    5
      ?visual>                                                                6
      state    free                                                     7
      buffer   empty                                                    8
      ?manual>                                                                9
      state    free                                                     10
      buffer   empty                                                    11
      ==>                                                                12
      ~g>                                                                13
      ~imaginal>                                                            14
      ~discourse_context>                                                  15
      ~unresolved_discourse>                                              16
      """)                                                                17
                                                                18
```

9.3.3 Phrase Structure Rules

The project and project-and-complete phrase structure rules encode most of the syntactic parsing and all the semantic parsing work. Consequently, these rules tend to have relative large lists of actions to be executed. We will only discuss here some of the most important phrase structure rules. The remaining ones, linked to in Appendix 9.6.2, have the same kind of structure and should be straightforward to understand.

Consider first the "project: NP ==> Det N" rule in (40) below. This rule is triggered once a determiner is shifted and projected, so the `found` feature stores a `Det` value (line 10). Another important condition for this rule is that the top parsing goal is to parse an `S` (line 5). That is, this rule is triggered by determiners in subject position. Determiners in object position trigger a different rule ("project and complete: NP ==> Det N"; see Appendix 9.6.2), which is very similar except that the top parsing goal is an `NP` rather than an `S`.

```
(40) parser.productionstring(name="project: NP ==> Det N", string="") 1
    =g> 2
    isa parsing_goal 3
    task parsing 4
    stack1 S 5
    arg_stack1 =a1 6
    right_edge_stack1 =re1 7
    right_edge_stack2 =re2 8
    parsed_word =w 9
    found Det 10
    dref_peg =dref_peg 11
    drs_peg =drs_peg 12
    embedding_level =e1 13
    ~retrieval> 14
    isa word 15
    pred1 =p1 16
    pred2 =p2 17
    ==> 18
    =g> 19
    isa parsing_goal 20
    task move_dref_peg 21
    stack1 N 22
    stack2 NP 23
    stack3 S 24
    found None 25
    parsed_word None 26
    arg_stack1 =dref_peg 27
    arg_stack2 =a1 28
    +imaginal> 29
    isa parse_state 30
    node_cat NP 31
    daughter1 Det 32
    daughter2 N 33
    mother =re1 34
    mother_of_mother =re2 35
    +discourse_context> 36
    isa drs 37
    dref =dref_peg 38
    arg1 =dref_peg 39
    drs =drs_peg 40
    embedding_level =e1 41
    discourse_status at_issue 42
    ~retrieval> 43
    """) 44
```

The "project: NP ==> Det N" rule in (40) also conditions on the retrieval buffer making available the semantics of the word we just parsed, specifically, the two predicates `=p1` and `=p2` it contributes (lines 14–17). This is spurious in the present case because we have just parsed a determiner, which does not contribute any predicates, but we include it for uniformity with the phrase structure rules for proper names, nouns, verbs, pronouns etc.

The "project: NP ==> Det N" rule in (40) triggers three main parsing actions: (i) it updates the goal buffer, thereby setting the context for subsequent parsing rules (lines 19–28); (ii) it updates the `imaginal` buffer with the

expected NP node with two daughters (Det and N), and it attaches the NP to the top right-edge attachment point (the S node) (lines 29–35); (iii) finally, it updates the `discourse_context` buffer with a new DRS (lines 36–42).

The new DRS has a similar structure to the sub-DRSs contributed by indefinites that we discussed in the previous chapter (Chap. 8): we introduce a new individual-level `dref` (line 38) and we plug it in as the argument of the upcoming N (line 39).

However, in contrast to what we did in the previous chapter, we mark the current sub-DRS as being part of a larger DRS (i.e., part of a larger propositional update) by means of a `drs` slot which has the current DRS `dref = drs_peg` as its value (line 40). Thus, the fact that this sub-DRS is part of a larger main DRS is only implicit in this representation, unlike the more explicitly hierarchical structure we built in the previous chapter. We prefer this flatter structure for sub-DRSs and main DRSs in this chapter because it is simpler to use, and sufficient for our purposes.

However, because we are modeling pronoun and presupposition resolution, we need to keep track of two other semantic features: the embedding level of the current sub-DRS (line 41), which is crucial for the resolution process, and the discourse status of the current sub-DRS (line 42). The discourse status is `at_issue`, to be distinguished from either (i) the `unresolved` discourse status associated with unresolved pronouns/presuppositions, or (ii) the `presupposed` discourse status associated with resolved pronouns/presuppositions.

Given these `imaginal` and `discourse_context` buffer updates, we update the goal buffer in a variety of ways:

- we update the task to `move_dref_peg` (line 21 in (40)): we have just introduced a new `dref` indexed with the current `dref_peg`, so we need to move the `peg` to the next position in preparation for subsequent indefinites;
- we set up the stack of parsing goals in the expected way (lines 22–24): we have just parsed a Det in subject position, so we expect to parse an N next, after which we expect to finish parsing the subject NP, and once that is completed, we revert to the initial goal of parsing an S;
- we reset the `found` and `parsed_word` features to `None` (lines 25–26) to indicate that we are finished parsing the current word;
- we push the `dref` we just introduced on the argument stack (lines 27–28) so that it is available as an argument for the predicate(s) introduced by the upcoming VP.

Finally, we flush the retrieval buffer (line 43 in (40)), since we have no further use for the lexical information associated with the current word.

The syntactic and semantic parsing actions triggered by the project-NP rule set up the context for the "project and complete: N" rule in (41) below. If the syntactic category at the top of the goal stack is N (line 5) and we have just shifted and projected an N (line 8), the semantics of which is available in the retrieval buffer (lines 9–12), we take two main parsing actions. On one hand, we add the two predicates `=p1` and `=p2` lexically contributed by the N to the DRS in the `discourse_context` buffer, which was contributed by the preceding indefinite determiner (lines 23–26). On the other hand, we pop the N goal off the goal-buffer stack and reset the `found`

and `parsed_word` features back to `None`. To wrap up, we do some cognitive-state clean-up and flush the retrieval and imaginal buffers (lines 27–28).

```
(41) parser.productionstring(name="project and complete: N", string="" 1
    =g> 2
    isa parsing_goal 3
    task parsing 4
    stack1 N 5
    stack2 =s2 6
    stack3 =s3 7
    found N 8
    =retrieval> 9
    isa word 10
    pred1 =p1 11
    pred2 =p2 12
    ?discourse_context> 13
    buffer full 14
    ==> 15
    =g> 16
    isa parsing_goal 17
    stack1 =s2 18
    stack2 =s3 19
    stack3 None 20
    found None 21
    parsed_word None 22
    =discourse_context> 23
    isa drs 24
    pred1 =p1 25
    pred2 =p2 26
    ~retrieval> 27
    ~imaginal> 28
    """) 29
```

At this point, we have fully parsed the subject NP, so we need to pop that goal off the top of the goal-buffer stack. Moreover, we have found the left corner of the S, namely the subject NP, so we can also pop the S goal off the stack and replace it with the goal of finding the VP that will complete the S. The "project and complete: S ==> NP VP" in (42) below triggers all these parsing actions, and also builds the binary-branching [_SNP VP] structure in the imaginal buffer.

```
(42) parser.productionstring(name="project and complete: S ==> NP VP", 1
    string="" 2
    =g> 3
    isa parsing_goal 4
    task parsing 5
    stack1 NP 6
    stack2 S 7
    stack3 =s3 8
    right_edge_stack1 =re1 9
    right_edge_stack2 =re2 10
    right_edge_stack3 =re3 11
    right_edge_stack4 =re4 12
    ==> 13
    =g> 14
    isa parsing_goal 15
    stack1 VP 16
    stack2 =s3 17
    stack3 None 18
    right_edge_stack1 VP 19
    right_edge_stack2 =re1 20
    right_edge_stack3 =re2 21
    right_edge_stack4 =re3 22
    found None 23
    parsed_word None 24
    +imaginal> 25
    isa parse_state 26
```

```

node_cat      S      27
daughter1     NP     28
daughter2     VP     29
~retrieval>   30
~imaginal>    31
~discourse_context> 32
""")          33

```

At this point, various project-and-complete-VP rules can be triggered, depending on the syntactic category of the finite verb. We will only discuss here the simplest case, namely intransitive verbs, and then work through a simple example. The "project and complete: VP ==> Vi" rule is provided in (43) below. This rule is triggered if the top syntactic category in the goal-buffer stack is VP (line 5), and we have just shifted and projected an intransitive verb (line 6), the semantics of which is available in the retrieval buffer (lines 16–18).

```

(43) parser.productionstring(name="project and complete: VP ==> Vi", string="" 1
      =g> 2
      isa      parsing_goal 3
      task      parsing 4
      stack1     VP 5
      found      Vi 6
      parsed_word =w 7
      right_edge_stack1 VP 8
      right_edge_stack2 =re2 9
      right_edge_stack3 =re3 10
      right_edge_stack4 =re4 11
      arg_stack1 =a1 12
      drs_peg      =drs_peg 13
      event_peg     =ev_peg 14
      embedding_level =el 15
      =retrieval> 16
      isa      word 17
      pred1     =p1 18
      ==> 19
      =g> 20
      isa      parsing_goal 21
      task      move_event_peg 22
      stack1     None 23
      found      None 24
      parsed_word None 25
      right_edge_stack1 =re2 26
      right_edge_stack2 =re3 27
      right_edge_stack3 =re4 28
      right_edge_stack4 None 29
      +imaginal> 30
      isa      parse_state 31
      mother    =re2 32
      mother_of_mother =re3 33
      node_cat   VP 34
      daughter1  Vi 35
      lex_head   =w 36
      +discourse_context> 37
      isa      drs 38
      dref      =ev_peg 39
      event_arg =ev_peg 40
      arg1      =a1 41
      pred1     =p1 42
      drs       =drs_peg 43
      embedding_level =el 44
      discourse_status at_issue 45
      ~retrieval> 46
      ~imaginal> 47
      ~discourse_context> 48
      """) 49

```

If these pre-conditions are met, the "project and complete: VP ==> Vi" rule in (43) triggers three main parsing actions, just like the project-NP rule in (40) above, or most other rules associated with words contributing essential semantic information and operations. Just as before, the three main parsing actions involve: (i) the goal buffer (lines 20–29); (ii) the imaginal buffer, where new syntactic information is encoded (lines 30–36); (iii) the discourse_context buffer, where new semantic information is encoded (lines 37–45).

The imaginal buffer update (lines 30–36 in (43) simply builds a unary-branching structure [vpVi] and specifies its lexical head and its attachment point in the larger syntactic structure.

The discourse_context buffer update on lines 37–45 is more substantial:

- we introduce a new DRS and mark it as a sub-DRS of the current main DRS by setting the `drs` feature to the current `=drs_peg` (line 43), its `embedding_level` feature to the current embedding level `=el` (line 44), and its discourse status to `at_issue` (line 45);
- since this DRS is contributed by a verb, we introduce a new event dref and index it with the current event dref `peg =ev_peg` (line 39);
- the intransitive verb contributes a predicate `=p1` (line 42) that takes two arguments: an event argument that is set to the newly introduced event dref `=ev_peg` (line 40), and an entity argument that is set to the dref `=a1` previously introduced by the subject and stored at the top of the goal-buffer argument stack (line 41).

With the syntactic and semantic updates in place, the goal buffer can be updated accordingly (lines 20–29):

- since we just introduced a new event dref, we need to update the event dref `peg`, so the new task is set to `move_event_peg` (line 22);
- we have just completely finished parsing the intransitive verb, so we pop the VP category off the goal-buffer stack (line 23) and reset the `found` and `parsed_word` features to `None` (lines 24–25);
- we also pop the S node off the right-edge stack since this node is not available for future attachments anymore (lines 26–29).

We are now ready to parse a simple example. When the model reads the sentence in (44) below word by word (as in a self-paced reading task), it goes through a cognitive process whose temporal trace is provided in (45). To run the model on the sentence in (44) and obtain the temporal trace, uncomment the relevant line in the `run_parser.py` file (linked to in Appendix 9.6.3) and run the file in the terminal with the command:

- `python3 run_parser.py`

The full `pyactr` output is very detailed, so we edit it down significantly in (45) below to be able to focus on the main steps of the incremental interpretation process.

(44) A woman smiled.

(45) ****Environment: {1: {'text': 'A', 'position': (320, 180)}} 1
 (0.0255, 'PROCEDURAL', 'RULE FIRED: encode word') 2
 (0.038, 'PROCEDURAL', 'RULE FIRED: retrieve word') 3
 (0.1762, 'retrieval', 'RETRIEVED: word(cat=Det, form=A, pred1=, pred2=)') 4
 (0.1887, 'PROCEDURAL', 'RULE FIRED: shift and project word (not N)') 5
 (0.1887, 'imaginal', 'CREATED A CHUNK: parse_state(daughter1=A, 6
 node_cat=Det)') 7
 (0.2012, 'PROCEDURAL', 'RULE FIRED: press spacebar') 8
 (0.2137, 'PROCEDURAL', 'RULE FIRED: project: NP ==> Det N') 9
 (0.2137, 'imaginal', 'CREATED A CHUNK: parse_state(daughter1=Det, 10
 daughter2=N, daughter3=, mother=S, node_cat=NP)') 11
 (0.2137, 'discourse_context', 'CREATED A CHUNK: drs(arg1=x1, arg2=, 12
 discourse_status=at_issue, dref=x1, drs=d1, embedding_level=0, 13
 event_arg=, pred1=, pred2=)') 14
 (0.2262, 'PROCEDURAL', 'RULE FIRED: move entity/individual dref peg to x2') 15
 (0.3512, 'manual', 'KEY PRESSED: SPACE') 16
 ****Environment: {1: {'text': 'woman', 'position': (320, 180)}} 17
 (0.3712, 'PROCEDURAL', 'RULE FIRED: encode word') 18
 (0.3837, 'PROCEDURAL', 'RULE FIRED: retrieve word') 19
 (0.5302, 'retrieval', 'RETRIEVED: word(cat=N, form=woman, 20
 pred1=pred(arity=1, constant_name=_woman_), 21
 pred2=pred(arity=1, constant_name=_female_))') 22
 (0.5427, 'PROCEDURAL', 'RULE FIRED: shift and project N') 23
 (0.5427, 'imaginal', 'CREATED A CHUNK: parse_state(daughter1=woman, 24
 node_cat=N)') 25
 (0.5552, 'PROCEDURAL', 'RULE FIRED: press spacebar') 26
 (0.5677, 'PROCEDURAL', 'RULE FIRED: project and complete: N') 27
 (0.5677, 'discourse_context', 'MODIFIED') 28
 (0.5802, 'PROCEDURAL', 'RULE FIRED: project and complete: S ==> NP VP') 29
 (0.5802, 'imaginal', 'CREATED A CHUNK: parse_state(daughter1=NP, 30
 daughter2=VP, node_cat=S)') 31
 (0.7052, 'manual', 'KEY PRESSED: SPACE') 32
 ****Environment: {1: {'text': 'smiled', 'position': (320, 180)}} 33
 (0.7285, 'PROCEDURAL', 'RULE FIRED: encode word') 34
 (0.741, 'PROCEDURAL', 'RULE FIRED: retrieve word') 35
 (0.8899, 'retrieval', 'RETRIEVED: word(cat=Vi, form=smiled, 36
 pred1=pred(arity=event_plus_1, constant_name=_smile_))') 37
 (0.9024, 'PROCEDURAL', 'RULE FIRED: shift and project word (not N)') 38
 (0.9024, 'imaginal', 'CREATED A CHUNK: parse_state(daughter1=smiled, 39
 node_cat=Vi)') 40
 (0.9149, 'PROCEDURAL', 'RULE FIRED: press spacebar') 41
 (0.9274, 'PROCEDURAL', 'RULE FIRED: project and complete: VP ==> Vi') 42
 (0.9274, 'imaginal', 'CREATED A CHUNK: parse_state(daughter1=Vi, 43
 lex_head=smiled, mother=S, node_cat=VP)') 44
 (0.9274, 'discourse_context', 'CREATED A CHUNK: drs(arg1=x1, 45
 discourse_status=at_issue, dref=e1, drs=d1, 46
 embedding_level=0, event_arg=e1, 47
 pred1=pred(arity=event_plus_1, constant_name=_smile_))') 48
 (0.9399, 'PROCEDURAL', 'RULE FIRED: move event dref peg to e2') 49
 (1.2274, 'PROCEDURAL', 'RULE FIRED: finished: no visual input') 50
 51
 Parse states in declarative memory at the end of the simulation 52
 ordered by time of (re)activation: 53
 0.2137 parse_state(daughter1=A, node_cat=Det) 54
 0.5427 parse_state(daughter1=Det, daughter2=N, lex_head=woman, 55
 mother=S, node_cat=NP) 56
 0.5677 parse_state(daughter1=woman, node_cat=N) 57
 0.9024 parse_state(daughter1=NP, daughter2=VP, node_cat=S) 58
 0.9274 parse_state(daughter1=smiled, node_cat=Vi) 59
 1.2274 parse_state(daughter1=Vi, lex_head=smiled, mother=S, 60
 node_cat=VP) 61
 62
 DRSSs in declarative memory at the end of the simulation 63
 ordered by time of (re)activation: 64
 0.5802 drs(arg1=x1, discourse_status=at_issue, dref=x1, 65
 drs=d1, embedding_level=0, 66
 pred1=pred(arity=1, constant_name=_woman_), 67
 pred2=pred(arity=1, constant_name=_female_)) 68
 1.2274 drs(arg1=x1, discourse_status=at_issue, dref=e1, 69
 drs=d1, embedding_level=0, event_arg=e1, 70
 pred1=pred(arity=event_plus_1, constant_name=_smile_)) 71

Line 1 of the temporal trace in (45) indicates that the first word, namely the indefinite determiner *A*, is displayed on the virtual screen from which the model takes its visual input. This word is encoded 25.5 ms later (line 2), after which a retrieval request for its lexical information is placed (line 3, time: 38 ms after the start of the cognitive process). The lexical information is retrieved after ≈ 140 ms (line 4), then the word is shifted and projected (line 5), and as a result, our first syntactic structure [_{Det}A] is built in the imaginal buffer (lines 6–7). At this point, the cognitive process branches into two sub-processes running in parallel: (i) a motor sub-process that will culminate in pressing the spacebar to reveal the next work (line 8), and (ii) the continuation of the incremental parsing process triggered by the indefinite determiner *A* (lines 9–15).

The incremental parsing process continues with the project-NP rule (line 9), which results in the creation of an [_{NP}Det N] syntactic structure in the imaginal buffer (lines 10–11), and the creation of our first DRS in the `discourse_context` buffer (lines 12–14). Using the familiar DRT format, the DRS can be represented as follows:

(46) DRS contributed by the indefinite *A*:

x_1
PREDICATES- NOT- YET- SPECIFIED
[part of main DRS d_1]
[at-issue]
[embedding level: 0]

Given that we just introduced a new dref x_1 for entities/individuals, we have to move the entity dref *peg* to the next position x_2 (line 15 in (45)). After this, the incremental interpretation sub-process has nothing left to do, so we wait until the motor sub-process completes and the spacebar is pressed (line 16).

At that point, the next word *woman* is displayed on the virtual screen (line 17), and we go through the same cycle of encode-retrieve-project rules for the new word (lines 18–31). Specifically, we encode the word (line 18) and place a retrieval request for its lexical information (line 19). The lexical entry for the noun *woman* is retrieved at ≈ 530 ms after the start of the entire cognitive process (lines 20–22). Notably, the noun contributes two predicates: `_woman_` and `_female_`, both of arity 1. The explicit gender specification is useful for pronoun resolution.

The noun is then shifted and projected (line 23) and, as a result, the unary branching structure [_Nwoman] is created in the imaginal buffer (lines 24–25). At this point, the cognitive process branches again into a motor process that will culminate in pressing the spacebar (line 26) and the continuation of the incremental parsing process. Incremental parsing continues with the project-and-complete-N rule (line 27), which results in an update of the DRS contributed by the indefinite determiner *A* and stored in the `discourse_context` buffer (line 28). The DRS is updated with the two predicates contributed by the noun *woman*, and can be represented in the familiar DRT format as follows.

(47) DRS contributed by the indefinite *A* and updated by the noun *woman*:

x_1
WOMAN(x_1) FEMALE(x_1)
[part of main DRS d_1] [at-issue] [embedding level: 0]

The project-S rule is then fired (line 29) and the binary-branching structure [_SNP VP] is created in the imaginal buffer (lines 30–31). This completes the sequence of incremental interpretation steps triggered by the noun *woman*. The process then waits for the motor sub-process to complete and the spacebar to be pressed, which happens ≈ 120 ms later (line 32).

At this point, the final word *smiled* is displayed on the virtual screen (line 33). After it is encoded (line 34) and the request for its lexical entry is placed (line 35), we have access to its lexical information in the retrieval buffer (lines 36–37). The most notable aspect of this lexical entry is the arity *event_plus_1* of the predicate *_smile_* (line 37): this simply means that the predicate *_smile_* takes an event argument and, in addition, an entity argument. The intransitive verb is then shifted and projected (line 38), at which point the unary branching syntactic structure [_V*i*smiled] is built in the imaginal buffer (lines 39–40).

Once again, and for the final time, the cognitive process branches into a motor sub-process that will culminate with a spacebar press (line 41) and a sub-process that continues with the incremental parsing triggered by the intransitive verb *smiled*. The parsing process continues with the project-and-complete-VP rule (line 42), which simultaneously creates a syntactic structure [_{VP}*Vi*] in the imaginal buffer (lines 43–44) and a DRS in the *discourse_context* buffer (lines 45–48). This DRS can be represented in the usual DRT format as shown below.

(48) DRS contributed by the intransitive verb *smiled*:

e_1
SMILE(e_1, x_1)
[part of main DRS d_1] [at-issue] [embedding level: 0]

The parsing process is basically done, so once the event-dref peg is updated (line 49), the "finished" rule ends the entire cognitive process because of the lack of visual input (no more words on the virtual screen). For convenience, we list the syntactic structures built during the parsing process together with their time stamps on lines 54–61 in (45), and the two DRSs with their time stamps on lines 65–71.

The model includes a variety of other rules—for transitive verbs, prepositional verbs, NPs in object position, adjectives etc. They are all available in the file linked

to in Appendix 9.6.2. We encourage you to run the model on the variety of sentences in the `run_parser.py` file (Appendix 9.6.3) and examine the temporal-trace outputs to understand the time-course predictions of the syntax/semantics interpretation process implemented by this model.

9.3.4 Rules for Conjunctions and Anaphora Resolution

We are now ready to model basic bi-clausal examples, specifically, conjunctions. The project-and-complete rule for *and* is provided in (49) below. The rule starts a new sentence S, that is, the second conjunct, by placing the category S at the top of the goal-buffer stack (line 13), advancing the DRS dref peg (line 12: the task is updated to `move_drs_peg`), and setting the embedding level for the second conjunct to 1 (line 22).

Incrementing the embedding level ensures that drefs in the second conjunct are not available as antecedents for pronouns in the first conjunct. In general, we use embedding levels to model both the explicit and the implicit aspects of the discourse accessibility relation used in DRT. The fact that discourse referents in the second conjunct cannot serve as antecedents to pronouns in the first conjunct is not explicitly encoded in the accessibility relation defined in Kamp and Reyle (1993), but it is a by-product of the DRS construction algorithm that requires the DRS construction for the second conjunct to take place in the context of the DRS constructed based on the first conjunct. In our model, we use embedding level uniformly to constrain dref accessibility, both in conjunctions and conditionals.

Finally, the *and* rule creates a (non-headed) ternary-branching structure [ConjS Conj S] in the imaginal buffer (lines 23–31).

```
(49) parser.productionstring(name="project and complete: and", string="" 1
      =g> 2
      isa parsing_goal 3
      task parsing 4
      found Conj 5
      parsed_word and 6
      parsed_word =w 7
      embedding_level 0 8
      ==> 9
      =g> 10
      isa parsing_goal 11
      task move_drs_peg 12
      stack1 S 13
      arg_stack1 None 14
      arg_stack2 None 15
      found None 16
      parsed_word None 17
      right_edge_stack1 S 18
      right_edge_stack2 ConjS 19
      right_edge_stack3 None 20
      right_edge_stack4 None 21
      embedding_level 1 22
      +imaginal> 23
      isa parse_state 24
      daughter1 S 25
      daughter2 Conj 26
      daughter3 S 27
```

```

node_cat      ConjS      28
mother        None       29
mother_of_mother None     30
lex_head      =w         31
~discourse_context>      32
""")                  33

```

We need to introduce three more rules, related to pronoun resolution, before we can go through an example. The first rule is the "project: NP ==> PRO" rule in (50) below, which completes the syntax/semantics parsing of pronouns in subject position. Given that a PRO has just been found (line 11) and is available in the retrieval buffer (lines 15–18), we take the usual three types of parsing actions and update the goal, imaginal and discourse_context buffers. In addition, we also add an unresolved DRS to the unresolved_discourse buffer. Let's discuss them in turn.

```

(50) parser.productionstring(name="project: NP ==> PRO", string=""      1
    =g>                                                                    2
    isa      parsing_goal      3
    task     parsing           4
    stack1   S                 5
    stack2   =s2               6
    arg_stack1 =a1             7
    right_edge_stack1 =re1     8
    right_edge_stack2 =re2     9
    parsed_word =w            10
    found      PRO            11
    dref_peg    =dref_peg     12
    drs_peg     =drs_peg     13
    embedding_level =el      14
    =retrieval> 15
    isa      word            16
    pred1    =p1            17
    pred2    =p2            18
    ==>      19
    =g>      20
    isa      parsing_goal    21
    task     move_dref_peg   22
    stack1   NP             23
    stack2   S              24
    stack3   =s2            25
    arg_stack1 =dref_peg     26
    arg_stack2 =a1          27
    found     None          28
    +imaginal> 29
    isa      parse_state     30
    node_cat NP             31
    daughter1 PRO           32
    mother    =re1          33
    mother_of_mother =re2    34
    lex_head  =w            35
    +discourse_context>      36
    isa      drs             37
    dref     =dref_peg       38
    arg1     =dref_peg       39
    pred1    =p2            40
    drs      =drs_peg       41
    embedding_level =el      42
    discourse_status at_issue 43
    +unresolved_discourse> 44
    isa      drs             45
    arg1     =dref_peg       46
    arg2     UNKNOWN        47
    pred1    =p1            48
    pred2    =p2            49
    drs      =drs_peg       50

```

```

embedding_level    =e1                                51
discourse_status   unresolved                          52
~retrieval>        53
~imaginal>         54
~discourse_context> 55
""")                                                    56

```

The imaginal buffer update on lines 29–35 in (50) is straightforward: we create the expected unary branching [NP_{PRO}] structure.

The DRS created in the `discourse_context` buffer on lines 36–43 follows the analysis of pronouns in Kamp and Reyle (1993). We introduce a new dref (line 38) and predicate the pronoun gender of it (line 40).

The `unresolved_discourse` buffer stores a DRS that encodes the unresolved presupposition of the pronoun (lines 44–52):

- the main predicate =p1 contributed by the pronoun (line 48) is identity (EQUALS), relating the dref introduced by the pronoun (line 46) and an UNKNOWN second dref that needs to be retrieved;
- the UNKNOWN dref is the antecedent dref that needs to be found to complete the pronoun resolution;
- the antecedent dref needs to be accessible, which is why we keep track of the embedding level of the pronoun (line 51), and it also needs to satisfy the gender predicate =p2 contributed by the pronoun (line 49);
- finally, the entire DRS is marked as having an unresolved discourse status (line 52).

The goal buffer is updated in the expected way after parsing an NP in subject position (lines 20–28): the task is updated to `move_dref_peg` (line 22), the dref introduced by the pronoun is added to the top of the argument stack (line 26), and NP becomes the top goal on the goal-buffer stack (line 23) in preparation for the project-S rule.

After fully parsing a pronoun, the cognitive state satisfies the conditions for attempting to resolve it. There are three different rules for anaphoric pronoun resolution, depending on the embedding level of the pronoun. We only discuss here the rule for embedding level 1, provided in (51) below. The other rules, linked to in Appendix 9.6.2, are very similar. The rule in (51) is triggered only if the `unresolved_discourse` buffer contains an unresolved DRS (lines 15–21).

Pronoun resolution rules are fired when other, higher-ranked tasks have already been completed (lines 5–11 in 51). Once these higher-ranked tasks are completed, pronoun resolution has high priority (it has a utility of 5—line 33). As we mentioned before, ordering rule firing preferences in this fashion (multiple negative specifications for the task slot & manually setting the utility to a non-default, i.e., non-0, value) is not cognitively realistic, and does not scale up to larger systems of rules. The conditions for these rules, as well as their utility, should emerge as a result of a learning algorithm that leverages both production compilation (for new rule generation) and reinforcement learning (for utility ‘tuning’).

The rule triggers two actions. The main action is a retrieval request for a suitable antecedent for the pronoun (lines 26–32): we need to retrieve a DRS in which a new

dref was introduced (line 28) that was not an event dref (line 29: `event_arg None` ensures that only entities, not events, will be considered as possible antecedents). Furthermore, this DRS should not be indexed with the same DRS dref (line 30): this is a way to enforce Binding Principle B (Chomsky 1981). Finally, the antecedent should have an embedding level higher than 2 (line 31) and an `at_issue` discourse status (line 32).

The second action is updating the task so that the resolution attempt concludes with this retrieval request (line 25), and we do not keep attempting to retrieve an antecedent again and again.

```
(51) parser.productionstring(name="attempting to resolve pronoun;\n      pronoun at embedding level 1", string="" 1
      => 2
      isa parsing_goal 3
      task ~reading_word 4
      task ~move_dref_peg 5
      task ~move_event_peg 6
      task ~attempting_to_resolve_PRO 7
      task ~attempting_to_resolve_cataphoric_PRO 8
      task ~stop_resolution_attempt_PRO 9
      task ~if_reanalysis 10
      found None 11
      ?retrieval> 12
      state free 13
      =unresolved_discourse> 14
      isa drs 15
      arg2 UNKNOWN 16
      pred2 =p2 17
      drs =drs 18
      embedding_level 1 19
      discourse_status unresolved 20
      ==> 21
      => 22
      => 23
      isa parsing_goal 24
      task stop_resolution_attempt_PRO 25
      +retrieval> 26
      isa drs 27
      dref ~None 28
      event_arg None 29
      drs ~=drs 30
      embedding_level ~2 31
      discourse_status at_issue 32
      """, utility=5) 33
```

The retrieval request for an antecedent either succeeds or fails. If it succeeds, the rule in (52) below fires. The rule takes the antecedent available in the retrieval buffer (lines 6–10) and the unresolved presupposition in the `unresolved_discourse` buffer (lines 11–18) and merges information from them into a new DRS added to the `discourse_context` buffer.

This DRS encodes the resolved pronominal presupposition: it basically takes the unresolved presupposition from the `unresolved_discourse` buffer and specifies its second, UNKNOWN argument to be the same dref as the dref of the antecedent DRS available in the retrieval buffer (line 27). The discourse status of the resolved presupposition is marked as `presupposed` (line 32).

With the pronoun resolved, we can flush the `unresolved_discourse` and retrieval buffers (lines 33–34).

```

(52) parser.productionstring(name="resolution of PRO succeeded", string="" 1
    =g> 2
    isa parsing_goal 3
    task stop_resolution_attempt_PRO 4
    embedding_level =el 5
    =retrieval> 6
    isa drs 7
    dref ~None 8
    dref =dref 9
    pred2 =p2 10
    =unresolved_discourse> 11
    isa drs 12
    arg1 =a1 13
    arg2 UNKNOWN 14
    pred1 =p1 15
    pred2 =p2 16
    drs =drs 17
    discourse_status unresolved 18
    ==> 19
    =g> 20
    isa parsing_goal 21
    task parsing 22
    entity_cataphora None 23
    +discourse_context> 24
    isa drs 25
    arg1 =a1 26
    arg2 =dref 27
    pred1 =p1 28
    pred2 =p2 29
    drs =drs 30
    embedding_level =el 31
    discourse_status presupposed 32
    ~retrieval> 33
    ~unresolved_discourse> 34
    """) 35

```

If the retrieval request for an antecedent fails, we trigger the rule in (53) below. The rule checks that the unsuccessfully resolved presupposition targets an entity, not an event (line 12): we check that `arg2` is `UNKNOWN`. For events, we will see that the `arg1` slot will be marked as `UNKNOWN`. The retrieval of a suitable entity `dref` has failed, but we do not simply mark the pronoun as unresolved and move on: we assume that the pronoun is in fact cataphoric, so we set the `entity_cataphora` feature to `True` (line 18). This will ensure that when entity `drefs` will be subsequently introduced in discourse, a cataphoric search will be triggered to check if they could be suitable antecedents for the unresolved pronoun. Finally, the rule flushes the `unresolved_discourse` and retrieval buffers (lines 20–21).

```

(53) parser.productionstring(name="resolution of PRO failed: no antecedent", 1
    string="" 2
    =g> 3
    isa parsing_goal 4
    task stop_resolution_attempt_PRO 5
    ?retrieval> 6
    state error 7
    ?unresolved_discourse> 8
    buffer full 9
    =unresolved_discourse> 10
    isa drs 11
    arg2 UNKNOWN 12
    discourse_status unresolved 13
    ==> 14
    =g> 15
    isa parsing_goal 16
    task parsing 17
    entity_cataphora True 18

```

```

found          no_antecedent          19
~unresolved_discourse>                20
~retrieval>                             21
""" )                                   22

```

There is another way that pronoun resolution could fail. Recall that the retrieval request for an antecedent placed in (51) above does not constrain the gender of the antecedent, it only constrains its embedding level and DRS peg (in addition to requiring the *dref* slot to be non-empty and the *event_arg* slot to be empty). We could, therefore, retrieve an antecedent *dref* that is suitable with respect to all features, but that does not match in gender. This is very much like the process of retrieving foil sentences in the fan experiment and the corresponding fan model discussed in the previous chapter (Chap. 8).

The "resolution of PRO failed: antecedent with non-matching gender" rule in (54) below takes care of this case: if the retrieved DRS has a certain gender specification =p2 (line 10) and the unresolved pronoun presupposition has a different gender specification ~p2 (line 13), we declare the pronoun unresolved and assume that it is a cataphoric pronoun (line 19).

```

(54) parser.productionstring(name="resolution of PRO failed:\
antecedent with non-matching gender", string=""          1
      =g>                                                  2
      isa          parsing_goal                            3
      task         stop_resolution_attempt_PRO            4
      ?unresolved_discourse>                              5
      buffer       full                                    6
      =retrieval>                                         7
      isa          drs                                     8
      pred2        =p2                                    9
      =unresolved_discourse>                             10
      isa          drs                                    11
      pred2        ~p2                                    12
      discourse_status unresolved                         13
      ==>                                                  14
      =g>                                                  15
      isa          parsing_goal                            16
      task         parsing                                 17
      entity_cataphora True                               18
      found        no_antecedent                          19
      ~retrieval>                                         20
      ~unresolved_discourse>                             21
      "" )                                                22

```

To bring all these rules together, let us work through two examples of conjoined discourses, one in which the pronoun resolution succeeds and one in which the resolution fails because of a gender mismatch. The following subsection will discuss cataphora, so we will show an example of a 'no retrieved antecedent' resolution failure at that point.

Let us first simulate the example in (55) below. The temporal trace is provided in (56). We omit the parsing steps associated with the first conjunct since they are identical to the ones in (45) above, except for minor random noise in visual, motor and retrieval timings. To obtain the temporal trace, we uncomment the relevant sentence in the `run_parser.py` file and run the file with the command `python3 run_parser.py`, as we did before.

(55) A woman smiled and she left.

(56) (1.0599, 'manual', 'KEY PRESSED: SPACE') 1
 ****Environment: {1: {'text': 'and', 'position': (320, 180)}} 2
 (1.0787, 'PROCEDURAL', 'RULE FIRED: encode word') 3
 (1.0912, 'PROCEDURAL', 'RULE FIRED: retrieve word') 4
 (1.2415, 'retrieval', 'RETRIEVED: word(cat=Conj, form=and)') 5
 (1.2540, 'PROCEDURAL', 'RULE FIRED: shift and project word (not N)') 6
 (1.2540, 'imaginal', 'CREATED A CHUNK: parse_state(daughter1=and, 7
 node_cat=Conj)') 8
 (1.2665, 'PROCEDURAL', 'RULE FIRED: press spacebar') 9
 (1.279, 'PROCEDURAL', 'RULE FIRED: project and complete: and') 10
 (1.279, 'imaginal', 'CREATED A CHUNK: parse_state(daughter1=S, 11
 daughter2=Conj, daughter3=S, lex_head=and, mother=None, 12
 mother_of_mother=None, node_cat=ConjS)') 13
 (1.2915, 'PROCEDURAL', 'RULE FIRED: move DRS/propositional dref peg to d2') 14
 (1.4165, 'manual', 'KEY PRESSED: SPACE') 15
 ****Environment: {1: {'text': 'she', 'position': (320, 180)}} 16
 (1.4354, 'PROCEDURAL', 'RULE FIRED: encode word') 17
 (1.4479, 'PROCEDURAL', 'RULE FIRED: retrieve word') 18
 (1.5993, 'retrieval', 'RETRIEVED: word(cat=PRO, form=she, 19
 pred1=pred(arity=2, constant_name=_equals_), 20
 pred2=pred(arity=1, constant_name=_female_))') 21
 (1.6118, 'PROCEDURAL', 'RULE FIRED: shift and project word (not N)') 22
 (1.6118, 'imaginal', 'CREATED A CHUNK: parse_state(daughter1=she, 23
 node_cat=PRO)') 24
 (1.6243, 'PROCEDURAL', 'RULE FIRED: press spacebar') 25
 (1.6368, 'PROCEDURAL', 'RULE FIRED: project: NP ==> PRO') 26
 (1.6368, 'imaginal', 'CREATED A CHUNK: parse_state(daughter1=PRO, 27
 lex_head=she, mother=S, mother_of_mother=ConjS, 28
 node_cat=NP)') 29
 (1.6368, 'discourse_context', 'CREATED A CHUNK: drs(arg1=x2, 30
 discourse_status=at_issue, dref=x2, drs=d2, 31
 embedding_level=1, 32
 pred1=pred(arity=1, constant_name=_female_))') 33
 (1.6368, 'unresolved_discourse', 'CREATED A CHUNK: drs(arg1=x2, 34
 arg2=UNKNOWN, discourse_status=unresolved, drs=d2, 35
 embedding_level=1, 36
 pred1=pred(arity=2, constant_name=_equals_), 37
 pred2=pred(arity=1, constant_name=_female_))') 38
 (1.6493, 'PROCEDURAL', 'RULE FIRED: move entity/individual dref peg to x3') 39
 (1.6618, 'PROCEDURAL', 'RULE FIRED: attempting to resolve pronoun;\ 40
 pronoun at embedding level 1') 41
 (1.7743, 'manual', 'KEY PRESSED: SPACE') 42
 ****Environment: {1: {'text': 'left', 'position': (320, 180)}} 43
 (1.8005, 'retrieval', 'RETRIEVED: drs(arg1=x1, discourse_status=at_issue, 44
 dref=x1, drs=d1, embedding_level=0, 45
 pred1=pred(arity=1, constant_name=_woman_), 46
 pred2=pred(arity=1, constant_name=_female_))') 47
 (1.8130, 'PROCEDURAL', 'RULE FIRED: resolution of PRO succeeded') 48
 (1.8130, 'discourse_context', 'CREATED A CHUNK: drs(arg1=x2, arg2=x1, 49
 discourse_status=presupposed, drs=d2, embedding_level=1, 50
 pred1=pred(arity=2, constant_name=_equals_), 51
 pred2=pred(arity=1, constant_name=_female_))') 52
 (1.8255, 'PROCEDURAL', 'RULE FIRED: project and complete: S ==> NP VP') 53
 (1.8255, 'imaginal', 'CREATED A CHUNK: parse_state(daughter1=NP, 54
 daughter2=VP, node_cat=S)') 55
 (1.8380, 'PROCEDURAL', 'RULE FIRED: encode word') 56
 (1.8505, 'PROCEDURAL', 'RULE FIRED: retrieve word') 57
 (2.0028, 'retrieval', 'RETRIEVED: word(cat=Vi, form=left, 58
 pred1=pred(arity=event_plus_1, constant_name=_left_))') 59
 (2.0153, 'PROCEDURAL', 'RULE FIRED: shift and project word (not N)') 60
 (2.0153, 'imaginal', 'CREATED A CHUNK: parse_state(daughter1=left, 61
 node_cat=Vi)') 62
 (2.0403, 'PROCEDURAL', 'RULE FIRED: project and complete: VP ==> Vi') 63
 (2.0403, 'imaginal', 'CREATED A CHUNK: parse_state(daughter1=Vi, 64
 lex_head=left, mother=S, mother_of_mother=ConjS, node_cat=VP)') 65
 (2.0403, 'discourse_context', 'CREATED A CHUNK: drs(arg1=x2, 66
 discourse_status=at_issue, dref=e2, drs=d2, 67
 embedding_level=1, event_arg=e2, 68
 pred1=pred(arity=event_plus_1, constant_name=_left_))') 69
 (2.0528, 'PROCEDURAL', 'RULE FIRED: move event dref peg to e3') 70
 (2.3403, 'PROCEDURAL', 'RULE FIRED: finished: no visual input') 71

DRSs in declarative memory at the end of the simulation	72
ordered by time of (re)activation:	73
0.5761 drs(arg1=x1, discourse_status=at_issue, dref=x1, drs=d1,	74
embedding_level=0,	75
pred1=pred(arity=1, constant_name=_woman_),	76
pred2=pred(arity=1, constant_name=_female_))	77
1.2790 drs(arg1=x1, discourse_status=at_issue, dref=e1, drs=d1,	78
embedding_level=0, event_arg=e1,	79
pred1=pred(arity=event_plus_1, constant_name=_smile_))	80
1.8130 drs(arg1=x2, discourse_status=at_issue, dref=x2, drs=d2,	81
embedding_level=1,	82
pred1=pred(arity=1, constant_name=_female_))	83
1.8130 drs(arg1=x2, arg2=UNKNOWN, discourse_status=unresolved,	84
drs=d2, embedding_level=1,	85
pred1=pred(arity=2, constant_name=_equals_),	86
pred2=pred(arity=1, constant_name=_female_))	87
1.8130 drs(arg1=x1, discourse_status=at_issue, dref=x1, drs=d1,	88
embedding_level=0,	89
pred1=pred(arity=1, constant_name=_woman_),	90
pred2=pred(arity=1, constant_name=_female_))	91
1.8255 drs(arg1=x2, arg2=x1, discourse_status=presupposed,	92
drs=d2, embedding_level=1,	93
pred1=pred(arity=2, constant_name=_equals_),	94
pred2=pred(arity=1, constant_name=_female_))	95
2.3403 drs(arg1=x2, discourse_status=at_issue, dref=e2,	96
drs=d2, embedding_level=1, event_arg=e2,	97
pred1=pred(arity=event_plus_1, constant_name=_left_))	98
	99

The first point at which the temporal trace in (56) differs from the previous one in (45) is the word *and*. This word is displayed on the virtual screen at 1.06 s after the model starts reading the sentence in (55)—see lines 1–2 in (56).

The word *and* is encoded, its lexical information is retrieved and then the word is projected, i.e., a unary branching structure [*Conj*and] is created in the imaginal buffer (lines 7–8). The by-now familiar split into two cognitive sub-processes happens at this point: on one hand, a motor process to press the spacebar is started (line 9), on the other hand, we continue to process the word *and*. Specifically, the project-and-complete-*and* rule is fired (line 10), and a ternary branching structure [*Conj*S*Conj* S] is created in the imaginal buffer (lines 11–13). Furthermore, since a new clause (the second conjunct) is about to start, we update the DRS dref peg from d_1 to d_2 (line 14).

The space bar is pressed and the next word, namely the pronoun *she*, appears on the virtual screen (lines 15–16) at time 1.4165 s. We go through the usual encode-retrieve-project-spacebar sequence of rules (lines 17–25), after which the project-NP rule for pronouns is fired (line 26). At that point, three chunks are created in the imaginal, *discourse_context* and *unresolved_discourse* buffers. The parse state in the imaginal buffer (lines 27–29) encodes the unary branching [*NP*PRO] structure.

The DRS in the *discourse_context* buffer (lines 30–33) is the contribution made by the pronoun to at-issue content, and is represented in familiar DRT format as follows.

(57) At-issue DRS contributed by the pronoun *she*:

x_2
FEMALE(x_2)
[part of main DRS d_2]
[at-issue]
[embedding level: 1]

The DRS in the `unresolved_discourse` buffer (lines 34–38) is the contribution made by the pronoun to the unresolved presupposed content, and is represented in familiar DRT format as shown in (58) below. The double contribution of the pronoun to both at-issue and unresolved presupposed content follows the account of presupposition projection as anaphora resolution in van der Sandt (1992) (see also Kamp 2001a).

(58) Unresolved presupposed DRS contributed by the pronoun *she*:

$x_2 = \text{UNKNOWN}$
FEMALE(x_2)
[part of main DRS d_2]
[unresolved]
[embedding level: 1]

After the *dref* peg is updated to x_3 (line 39), we attempt to resolve the pronoun (lines 40–41), which means that a retrieval request is placed for a suitable antecedent. While we wait for the retrieval to complete, the motor module presses the space bar and reveals the final word *left* on the virtual screen (lines 42–43). At 1.8005 s, we successfully retrieve an antecedent for the pronoun (lines 44–47), namely the DRS contributed by the indefinite NP *A woman* in the first conjunct. In DRT format, the retrieved DRS is represented as follows.

(59) Potential antecedent DRS retrieved at time 1.8005 s:

x_1
WOMAN(x_1)
FEMALE(x_1)
[part of main DRS d_1]
[at-issue]
[embedding level: 0]

Since the antecedent matches in gender, we declare the pronoun successfully resolved (line 48) and add a resolved presupposition DRS to the `discourse_context` buffer at time 1.8130 s (lines 49–52). This DRS is represented in the usual DRT format as shown below.

- (60) Presupposed DRS resulting from a successful pronoun resolution, added to the `discourse_context` buffer at time 1.8130 s:

$x_2 = x_1$ $\text{FEMALE}(x_2)$ <p>[part of main DRS d_2] [presupposed] [embedding level: 1]</p>
--

After successfully resolving the pronoun, the incremental parsing process proceeds as expected. The project-S rule (line 53) is followed by:

- encoding, retrieving and projecting ('shifting') the intransitive verb *left* (lines 56–62),
- the project-VP rule and the syntactic structure and the DRS contributed by this rule (lines 63–69),
- the move-event-dref-peg rule (line 70), and finally,
- the "finished: no visual input" rule (line 71).

At the end of the simulation, we have seven DRSs in memory, listed on lines 75–99 in (56) above, together with their time stamps. The first two DRSs (lines 75–81) are part of the main DRS d_1 contributed by the first conjunct. The next two DRSs (lines 82–88) are contributed by the pronoun *she* in the second conjunct. The fifth DRS (lines 89–92) is just the first DRS (contributed by *A woman*) that is recalled to serve as the antecedent of the pronoun. The sixth DRS (lines 93–96) encodes the resolved presupposition of the pronoun, while the seventh DRS is the one contributed by the intransitive verb *left*.

Let us turn now to the example in (61) below, where the pronoun resolution fails because an antecedent with a non-matching gender is retrieved.

The only part of the temporal trace that differs from the one in (55) above is provided in (62) below. We see that the DRS contributed by the indefinite *A woman* is retrieved when we attempt to resolve the pronoun *he* (lines 5–8). We therefore declare the resolution failed because the antecedent and the pronoun do not match in gender.

As was the case in Chap. 8, the model predicts that recalling the mismatching antecedent should take slightly more time than recalling a match because the mismatching antecedent does not receive any spreading activation from the gender of the pronoun held in one of the buffers at the moment of recall.

At the end of the simulation in (62), we see that only six DRSs are stored in declarative memory (lines 14–34). These are basically the same DRSs as the ones stored after the simulation in (55), except for the seventh DRS that encoded the successfully resolved presupposition of the pronoun.

(61) A woman smiled and he left.

(62)

```

(1.6640, 'PROCEDURAL', 'RULE FIRED: attempting to resolve pronoun;\
    pronoun at embedding level 1')
(1.7765, 'manual', 'KEY PRESSED: SPACE')
****Environment: {1: {'text': 'left', 'position': (320, 180)}}
(1.8143, 'retrieval', 'RETRIEVED: drs(arg1=x1, discourse_status=at_issue,\
    dref=x1, drs=d1, embedding_level=0, event_arg=None,\
    pred1=pred(arity=1, constant_name=_woman_),\
    pred2=pred(arity=1, constant_name=_female_))')
(1.8268, 'PROCEDURAL', 'RULE FIRED: resolution of PRO failed:\
    antecedent with non-matching gender')

DRSs in declarative memory at the end of the simulation
ordered by time of (re)activation:
0.5765   drs(arg1=x1, arg2=None, discourse_status=at_issue, dref=x1,
        drs=d1, embedding_level=0, event_arg=None,
        pred1=pred(arity=1, constant_name=_woman_),
        pred2=pred(arity=1, constant_name=_female_))
1.2807   drs(arg1=x1, discourse_status=at_issue, dref=e1, drs=d1,
        embedding_level=0, event_arg=e1,
        pred1=pred(arity=event_plus_1, constant_name=_smile_))
1.8268   drs(arg1=x2, arg2=UNKNOWN, discourse_status=unresolved,
        drs=d2, embedding_level=1,
        pred1=pred(arity=2, constant_name=_equals_),
        pred2=pred(arity=1, constant_name=_male_))
1.8268   drs(arg1=x1, discourse_status=at_issue, dref=x1,
        drs=d1, embedding_level=0,
        pred1=pred(arity=1, constant_name=_woman_),
        pred2=pred(arity=1, constant_name=_female_))
1.8393   drs(arg1=x2, discourse_status=at_issue, dref=x2,
        drs=d2, embedding_level=1,
        pred1=pred(arity=1, constant_name=_male_))
2.3542   drs(arg1=x2, discourse_status=at_issue, dref=e2,
        drs=d2, embedding_level=1, event_arg=e2,
        pred1=pred(arity=event_plus_1, constant_name=_left_))

```

9.3.5 Rules for Conditionals and Cataphora Resolution

We are now ready to discuss conditionals and cataphora resolution.

The project-and-complete rule for conditionals with a sentence-final *if*-clause needed to model Experiments 1 and 2 above follows the pattern of the project-and-complete rule for *and*. As shown in (63) below, we start a new sentence S for the conditional antecedent (line 14), we advance the DRS dref peg (line 13), and we mark the embedding level of the conditional antecedent as 1 (line 23), ensuring that pronouns in a matrix clause (with embedding level 0) won't be able to access drefs introduced by expressions in the conditional antecedent.

Finally, the rule creates a $[Cp[Cif] S]$ structure in the imaginal buffer (lines 24–31) that will be Chomsky-adjoined to the previous (matrix clause) S by the next rule we will examine.

```
(63) parser.productionstring(name="project and complete: sentence-final if", 1
      string="" 2
      =g> 3
      isa parsing_goal 4
      task parsing 5
      found C 6
      parsed_word if 7
      parsed_word =w 8
      embedding_level 0 9
      ==> 10
      =g> 11
      isa parsing_goal 12
      task move_drs_peg 13
      stack1 S 14
      arg_stack1 None 15
      arg_stack2 None 16
      found None 17
      parsed_word None 18
      right_edge_stack1 S 19
      right_edge_stack2 CP 20
      right_edge_stack3 S 21
      right_edge_stack4 none 22
      embedding_level 1 23
      +imaginal> 24
      isa parse_state 25
      daughter1 C 26
      daughter2 S 27
      node_cat CP 28
      mother S 29
      mother_of_mother None 30
      lex_head =w 31
      ~discourse_context> 32
      """) 33
```

The project-and-complete rule for sentence final *if* sets the stage for a sequence of rules reanalyzing the previous (matrix) clause. We already mentioned this when we discussed Experiments 1 and 2 earlier in this chapter: a sentence-final *if*-clause triggers the reanalysis of the previous matrix clause because, until *if* is read, the incremental processor assumes the sentence is an unconditionalized assertion, so it has an embedding level of 0. When *if* is read, the previous clause has to be reanalyzed from a main clause/main assertion to a conditional consequent. Specifically, all the DRSs contributed as part of that clause should have their embedding level changed from 0 to 2.

The "start if-triggered reanalysis" rule in (64) below begins the process of recalling these DRSs for reanalysis. The task is updated to *if_reanalysis* (line 21), the CP structure created by the project-and-complete-*if* rule in (63) above is Chomsky-adjoined to the S node of the previous clause, and the structure is encoded in the imaginal buffer (lines 23–29). Most importantly, a retrieval request is placed for a DRS that has an embedding level of 0 and is indexed with the DRS dref peg of the previous sentence (lines 30–33).

```

(64) parser.productionstring(name="start if-triggered reanalysis\
      (for sentence-final if)", string="""
      =g>
      isa          parsing_goal
      task         parsing
      found        None
      parsed_word   None
      right_edge_stack2 =re2
      right_edge_stack3 =re3
      drs_peg       =drs_peg
      prev_drs_peg   =prev_drs_peg
      =imaginal>
      isa          parse_state
      daughter1     C
      node_cat      CP
      lex_head      if
      lex_head      =w
      ==>
      =g>
      isa          parsing_goal
      task         if_reanalysis
      right_edge_stack4 None
      +imaginal>
      isa          parse_state
      daughter1     S
      daughter2     =re2
      node_cat      =re3
      mother        None
      lex_head      =w
      +retrieval>
      isa          drs
      drs           =prev_drs_peg
      embedding_level 0
      """)

```

Once the first DRS is recalled for reanalysis, we want to update its embedding level. To do this, we trigger one of two rules:

- "if-triggered reanalysis (no event recalled)", provided in (65) below, or
- "if-triggered reanalysis (event recalled)", provided in (66).

We need an ‘event recalled’ version of the rule because we want to capture the Maximize Presupposition effect we observed in Experiment 2; see the discussion of *Maximize Presupposition* in Sect. 9.1.2 above.

The default version of the rule, i.e., "if-triggered reanalysis (no event recalled)" in (65), is triggered when a DRS has been successfully retrieved and is available in the retrieval buffer (lines 8–21 in (65)). The rule triggers two actions. First, an identical DRS, except with an embedding level of 2, is placed in the *discourse_context* buffer (lines 25–37). Second, a new retrieval request for a DRS that has an embedding level of 0 and is indexed with the DRS *dref_peg* of the previous sentence is placed (lines 40–43). Crucially, however, if a new DRS is to be retrieved, it should be different from the previous-sentence DRSs that have already been retrieved (lines 38–39)—we use here the FINST (‘fingers of instantiation’) feature we discussed in the previous chapter.

```

(65) parser.productionstring(name="if-triggered reanalysis\           1
      (no event recalled)", string=""                               2
      =g>                                                            3
      isa      parsing_goal                                         4
      task      if_reanalysis                                       5
      drs_peg    =drs_peg                                           6
      prev_drs_peg =prev_drs_peg                                    7
      =retrieval>                                                  8
      isa      drs                                                  9
      drs      =drs                                                10
      discourse_status =dstatus                                     11
      embedding_level 0                                           12
      dref      =dref                                              13
      pred1     =p1                                               14
      pred2     =p2                                               15
      dref      =dref                                              16
      drs      =drs                                              17
      event_arg  =ea                                              18
      event_arg  None                                           19
      arg1       =a1                                              20
      arg2       =a2                                              21
      ?retrieval>                                                  22
      state      free                                             23
      ==>                                                         24
      +discourse_context>                                         25
      isa      drs                                                26
      discourse_status =dstatus                                    27
      drs      =drs                                                28
      embedding_level 2                                           29
      dref      =dref                                              30
      pred1     =p1                                               31
      pred2     =p2                                               32
      dref      =dref                                              33
      drs      =drs                                              34
      event_arg  =ea                                              35
      arg1       =a1                                              36
      arg2       =a2                                              37
      ?retrieval>                                                  38
      recently_retrieved False                                    39
      +retrieval>                                                  40
      isa      drs                                                41
      drs      =prev_drs_peg                                       42
      embedding_level 0                                           43
      """)                                                         44

```

The "if-triggered reanalysis (event recalled)" rule in (66) below is very similar to the no-event-recalled rule in (65) above. The only difference is that when a DRS with an event dref is recalled, we keep track of its main predicate =p1 in the goal buffer: this predicate is stored as the value of an if_conseq_pred feature (line 40), i.e., it is indexed as the predicate that was contributed by the conditional consequent.

```

(66) parser.productionstring(name="if-triggered reanalysis (event recalled)", 1
      string=""                                                       2
      =g>                                                             3
      isa      parsing_goal                                         4
      task      if_reanalysis                                       5
      drs_peg    =drs_peg                                           6
      prev_drs_peg =prev_drs_peg                                    7
      =retrieval>                                                  8
      isa      drs                                                  9
      drs      =drs                                                10
      discourse_status =dstatus                                     11
      embedding_level 0                                           12
      dref      =dref                                              13
      pred1     =p1                                               14
      pred2     =p2                                               15

```

```

dref          =dref          16
drs           =drs          17
event_arg     =ea           18
event_arg     ~None         19
arg1          =a1           20
arg2          =a2           21
?retrieval>   22
state         free          23
==>          24
+discourse_context> 25
isa           drs           26
discourse_status =dstatus   27
drs           =drs          28
embedding_level 2           29
dref          =dref          30
pred1         =p1           31
pred2         =p2           32
dref          =dref          33
drs           =drs          34
event_arg     =ea           35
arg1          =a1           36
arg2          =a2           37
=g>           38
isa           parsing_goal   39
if_conseq_pred =p1          40
?retrieval>   41
recently_retrieved False    42
+retrieval>   43
isa           drs           44
drs           =prev_drs_peg  45
embedding_level 0           46
""")              47

```

The two rules for *if*-triggered reanalysis (no-event-recalled and event-recalled) run repeatedly until all the DRSs contributed by the previous sentence are recalled and reanalyzed, i.e., their embedding level gets set to 2. Once there are no more DRSs to be recalled, the *if*-triggered reanalysis is complete and the "stop if-triggered reanalysis" rule in (67) below is triggered. This rule simply resets the task to `reading_word` (line 12) and flushes the retrieval, imaginal and discourse_context buffers (lines 15–17).

```

(67) parser.productionstring(name="stop if-triggered reanalysis", string="" 1
=g> 2
isa           parsing_goal 3
task         if_reanalysis 4
?retrieval> 5
state        error 6
?manual> 7
state        free 8
==> 9
=g> 10
isa           parsing_goal 11
task         reading_word 12
found        None 13
parsed_word   None 14
~retrieval> 15
~imaginal> 16
~discourse_context> 17
""") 18

```

We are almost ready to parse the conditional & cataphora example in (7) above (*John won't eat it if a hamburger is overcooked*, Elbourne 2009), we only need to introduce two kinds of rules related to cataphoric search.

First, there are three rules that trigger a cataphoric search for an antecedent, depending on the embedding level of the potential antecedent. We only discuss the ‘embedding level 1’ rule, provided in (68) below. The other rules (linked to in Appendix 9.6.2) are similar.

```
(68) parser.productionstring(name="attempting to resolve cataphoric pronoun;\n 1
      antecedent at embedding level 1", string="" 2
      =g> 3
      isa parsing_goal 4
      task ~reading_word 5
      task ~move_dref_peg 6
      task ~move_event_peg 7
      task ~attempting_to_resolve_PRO 8
      task ~attempting_to_resolve_cataphoric_PRO 9
      task ~stop_resolution_attempt_PRO 10
      task ~if_reanalysis 11
      found None 12
      entity_cataphora True 13
      ?retrieval> 14
      state free 15
      =discourse_context> 16
      isa drs 17
      dref ~None 18
      dref =dref 19
      event_arg None 20
      pred2 ~None 21
      pred2 =p2 22
      drs =drs 23
      embedding_level 1 24
      ==> 25
      =g> 26
      isa parsing_goal 27
      task stop_resolution_attempt_PRO 28
      +unresolved_discourse> 29
      isa drs 30
      dref =dref 31
      arg2 UNKNOWN 32
      pred2 =p2 33
      drs =drs 34
      embedding_level 1 35
      discourse_status unresolved 36
      +retrieval> 37
      isa drs 38
      dref None 39
      arg1 ~None 40
      arg2 UNKNOWN 41
      pred1 ~None 42
      drs ~=drs 43
      embedding_level ~0 44
      discourse_status unresolved 45
      "", utility=5) 46
```

Just as the anaphoric search rules, the cataphoric search rules are ‘elsewhere’ rules: they have a set of negative constraints for the current task (lines 5–11 in (68)), and a high utility (line 46). The rule is triggered if the `entity_cataphora` feature is set to `True` (line 13) and the `found` feature is set to `None` (line 12). Consequently, the rule cannot immediately follow a failed anaphoric search because such a search sets the `found` feature to `no_antecedent`. Most importantly, a cataphoric search is triggered only if a suitable antecedent is available in the `discourse_context` buffer (lines 16–24).

If these conditions are met, a cataphoric search is triggered. Cataphoric searches are mirror images of anaphoric searches. For cataphora, we have a potential antecedent in place, and place a retrieval request for an unresolved presupposition DRS that could

be resolved by the antecedent. In contrast, for anaphora, we have an unresolved presupposition and we place a retrieval request for an antecedent that could resolve it.

The "attempting to resolve cataphoric pronoun" rule triggers three actions. The most important one is placing a retrieval request for an unresolved presupposition (lines 37–45 in (68)). We specify that no new drefs should be introduced in this unresolved DRS (line 39), that the `arg1` and `pred1` slots should be non-empty (lines 40 and 42), that the `arg2` slot should be set to `UNKNOWN` (line 41), that the DRS should be part of a main DRS that is different than the one that the potential antecedent belongs to (line 43), and that the embedding level of the unresolved presupposition should not be 0 (line 44) since the potential antecedent has an embedding level of 1.

The second action is to store information about the current potential antecedent in the `unresolved_discourse` buffer (lines 29–36). This is not strictly necessary since the information will be maintained in the `discourse_context` buffer, but we do it here just to show how specific buffers can be used to safeguard information that might otherwise be flushed by subsequent rules. And involving the `unresolved_discourse` buffer in the process of cataphoric search is a natural choice. We save the dref information (line 31), the gender predicate `=p2` (line 33), the DRS peg (line 34) and the embedding level of 1 (line 35).

The third and final action is to update the goal buffer (lines 26–28) so that the resolution attempt is stopped with this one retrieval request and does not enter a loop. We therefore update the `task` to `stop_resolution_attempt_PRO`.

The retrieval request, i.e., the cataphoric resolution attempt, can either succeed or fail. If the attempt fails, the same failure rules as for anaphoric attempts are triggered—see (53) and (54) above.

But if the cataphoric search succeeds, the "resolution of cataphoric PRO succeeded" rule in (69) below is triggered. This rule adds a DRS to the `discourse_context` buffer that resolves the retrieved unresolved presupposition to the currently available antecedent (lines 25–33). The DRS has a presupposed discourse status (line 33), and is otherwise identical to the DRS contributed by the "resolution of PRO succeeded" rule in (52) above.

```
(69) parser.productionstring(name="resolution of cataphoric PRO succeeded", 1
      string="" 2
      =g> 3
      isa parsing_goal 4
      task stop_resolution_attempt_PRO 5
      =retrieval> 6
      isa drs 7
      dref None 8
      arg1 ~None 9
      arg1 =a1 10
      arg2 UNKNOWN 11
      pred1 =p1 12
      pred2 =p2 13
      =unresolved_discourse> 14
      isa drs 15
      dref =dref 16
      pred2 =p2 17
      embedding_level =e1 18
      drs =drs 19
```

```

==> 20
=> 21
isa      parsing_goal 22
task     parsing      23
entity_cataphora None 24
+discourse_context> 25
isa      drs           26
arg1     =a1          27
arg2     =dref         28
pred1    =p1          29
pred2    =p2          30
drs      =drs         31
embedding_level =el    32
discourse_status presupposed 33
~unresolved_discourse> 34
~retrieval> 35
""") 36

```

We can now see how the model parses the conditional + cataphora example in (70) below, repeated from (7) above. The temporal trace is provided in (71). We omit all the output that is not directly relevant to incremental semantic interpretation.

(70) John won't eat it if a hamburger is overcooked. (Elbourne 2009)

(71) ****Environment: {1: {'text': 'John', 'position': (320, 180)}} 1
(0.2075, 'PROCEDURAL', 'RULE FIRED: project: NP ==> ProperN') 2
(0.2075, 'discourse_context', 'CREATED A CHUNK: drs(arg1=x1,\ 3
discourse_status=at_issue, dref=x1, drs=d1, embedding_level=0,\ 4
pred1=pred(arity=1, constant_name=_john_),\ 5
pred2=pred(arity=1, constant_name=_male_))') 6
****Environment: {1: {'text': 'won't', 'position': (320, 180)}} 7
(0.5645, 'PROCEDURAL', 'RULE FIRED: project and complete:\ 8
VP ==> VauxNeg VP') 9
(0.5645, 'discourse_context', 'CREATED A CHUNK: drs(drs=d1,\ 10
discourse_status=at_issue, embedding_level=0, pred1=NOT)') 11
****Environment: {1: {'text': 'eat', 'position': (320, 180)}} 12
(0.9244, 'PROCEDURAL', 'RULE FIRED: project and complete: VP ==> Vt NP') 13
(0.9244, 'discourse_context', 'CREATED A CHUNK: drs(arg1=x1, event_arg=e1,\ 14
discourse_status=at_issue, dref=e1, drs=d1, embedding_level=0,\ 15
pred1=pred(arity=event_plus_2, constant_name=_eat_))') 16
****Environment: {1: {'text': 'it', 'position': (320, 180)}} 17
(1.2893, 'PROCEDURAL', 'RULE FIRED: project and complete: NP ==> PRO') 18
(1.2893, 'discourse_context', 'CREATED A CHUNK: drs(arg1=x2,\ 19
discourse_status=at_issue, dref=x2, drs=d1, embedding_level=0,\ 20
pred1=pred(arity=1, constant_name=_nonhuman_))') 21
(1.2893, 'unresolved_discourse', 'CREATED A CHUNK: drs(embedding_level=0,\ 22
arg1=x2, arg2=UNKNOWN, discourse_status=unresolved, drs=d1,\ 23
pred1=pred(arity=2, constant_name=_equals_),\ 24
pred2=pred(arity=1, constant_name=_nonhuman_))') 25
(1.3143, 'PROCEDURAL', 'RULE FIRED: attempting to resolve pronoun;\ 26
pronoun at embedding level 0') 27
****Environment: {1: {'text': 'if', 'position': (320, 180)}} 28
(1.5168, 'retrieval', 'RETRIEVED: None') 29
(1.5293, 'PROCEDURAL', 'RULE FIRED: resolution of PRO failed:\ 30
no antecedent') 31
(1.7435, 'PROCEDURAL', 'RULE FIRED: project and complete:\ 32
sentence-final if') 33
(1.7560, 'PROCEDURAL', 'RULE FIRED:\ 34
move DRS/propositional dref peg to d2') 35
(1.7685, 'PROCEDURAL', 'RULE FIRED: start if-triggered reanalysis\ 36
(for sentence-final if)') 37
****Environment: {1: {'text': 'a', 'position': (320, 180)}} 38
(1.9053, 'retrieval', 'RETRIEVED: drs(arg1=x2, discourse_status=at_issue,\ 39
dref=x2, drs=d1, embedding_level=0,\ 40
pred1=pred(arity=1, constant_name=_nonhuman_))') 41
(1.9178, 'PROCEDURAL', 'RULE FIRED: if-triggered reanalysis\ 42
(no event recalled)') 43
(1.9178, 'discourse_context', 'CREATED A CHUNK: drs(arg1=x2,\ 44
discourse_status=at_issue, dref=x2, drs=d1, embedding_level=2,\ 45

```

pred1=pred(arity=1, constant_name=nonhuman_))' 46
(2.0643, 'retrieval', 'RETRIEVED: drs(arg1=x2, arg2=UNKNOWN,\ 47
discourse_status=unresolved, drs=d1, embedding_level=0,\ 48
pred1=pred(arity=2, constant_name=equals_),\ 49
pred2=pred(arity=1, constant_name=nonhuman_))') 50
(2.0768, 'PROCEDURAL', 'RULE FIRED: if-triggered reanalysis\ 51
(no event recalled)') 52
(2.0768, 'discourse_context', 'CREATED A CHUNK: drs(arg1=x2, arg2=UNKNOWN,\ 53
discourse_status=unresolved, drs=d1, embedding_level=2,\ 54
pred1=pred(arity=2, constant_name=equals_),\ 55
pred2=pred(arity=1, constant_name=nonhuman_))') 56
(2.2259, 'retrieval', 'RETRIEVED: drs(arg1=x1, arg2=x2, dref=e1, drs=d1,\ 57
discourse_status=at_issue, embedding_level=0, event_arg=e1,\ 58
pred1=pred(arity=event_plus_2, constant_name=eat_))') 59
(2.2384, 'PROCEDURAL', 'RULE FIRED: if-triggered reanalysis\ 60
(event recalled)') 61
(2.2384, 'discourse_context', 'CREATED A CHUNK: drs(arg1=x1, arg2=x2,\ 62
discourse_status=at_issue, dref=e1, drs=d1, embedding_level=2,\ 63
event_arg=e1,\ 64
pred1=pred(arity=event_plus_2, constant_name=eat_))') 65
(2.3894, 'retrieval', 'RETRIEVED: drs(discourse_status=at_issue, drs=d1,\ 66
embedding_level=0, pred1=NOT)') 67
(2.4019, 'PROCEDURAL', 'RULE FIRED: if-triggered reanalysis\ 68
(no event recalled)') 69
(2.4019, 'discourse_context', 'CREATED A CHUNK: drs(drs=d1,\ 70
discourse_status=at_issue, embedding_level=2, pred1=NOT)') 71
(2.5548, 'retrieval', 'RETRIEVED: drs(arg1=x1, discourse_status=at_issue,\ 72
dref=x1, drs=d1, embedding_level=0,\ 73
pred1=pred(arity=1, constant_name=john_),\ 74
pred2=pred(arity=1, constant_name=male_))') 75
(2.5673, 'PROCEDURAL', 'RULE FIRED: if-triggered reanalysis\ 76
(no event recalled)') 77
(2.5673, 'discourse_context', 'CREATED A CHUNK: drs(arg1=x1,\ 78
discourse_status=at_issue, dref=x1, drs=d1, embedding_level=2,\ 79
pred1=pred(arity=1, constant_name=john_),\ 80
pred2=pred(arity=1, constant_name=male_))') 81
(2.7698, 'retrieval', 'RETRIEVED: None') 82
(2.7823, 'PROCEDURAL', 'RULE FIRED: stop if-triggered reanalysis') 83
(2.9987, 'PROCEDURAL', 'RULE FIRED: project: NP ==> Det N') 84
(2.9987, 'discourse_context', 'CREATED A CHUNK: drs(arg1=x3,\ 85
discourse_status=at_issue, dref=x3, drs=d2, embedding_level=1)') 86
****Environment: {1: {'text': 'hamburger', 'position': (320, 180)}} 87
(3.3592, 'PROCEDURAL', 'RULE FIRED: project and complete: N') 88
(3.3717, 'PROCEDURAL', 'RULE FIRED: attempting to resolve cataphoric\ 89
pronoun; antecedent at embedding level 1') 90
(3.3717, 'unresolved_discourse', 'CREATED A CHUNK: drs(arg2=UNKNOWN,\ 91
discourse_status=unresolved, dref=x3, drs=d2, embedding_level=1,\ 92
pred2=pred(arity=1, constant_name=nonhuman_))') 93
****Environment: {1: {'text': 'is', 'position': (320, 180)}} 94
(3.5159, 'retrieval', 'RETRIEVED: drs(arg1=x2, arg2=UNKNOWN,\ 95
discourse_status=unresolved, drs=d1, embedding_level=2,\ 96
pred1=pred(arity=2, constant_name=equals_),\ 97
pred2=pred(arity=1, constant_name=nonhuman_))') 98
(3.5284, 'PROCEDURAL', 'RULE FIRED: resolution of cataphoric PRO\ 99
succeeded') 100
(3.5284, 'discourse_context', 'CREATED A CHUNK: drs(arg1=x2, arg2=x3,\ 101
discourse_status=presupposed, drs=d2, embedding_level=1,\ 102
pred1=pred(arity=2, constant_name=equals_),\ 103
pred2=pred(arity=1, constant_name=nonhuman_))') 104
****Environment: {1: {'text': 'overcooked', 'position': (320, 180)}} 105
(4.1232, 'PROCEDURAL', 'RULE FIRED: project and complete: AP ==> A') 106
(4.1232, 'discourse_context', 'CREATED A CHUNK: drs(arg1=x3,\ 107
discourse_status=at_issue, drs=d2, embedding_level=1,\ 108
pred1=pred(arity=1, constant_name=overcook_))') 109
(4.4232, 'PROCEDURAL', 'RULE FIRED: finished: no visual input') 110

```

The first word, namely the proper name *John*, contributes the first DRS to the `discourse_context` buffer—see lines 3–6 in (71) above. This DRS can be represented in the familiar DRT format as shown below.

(72) DRS contributed by the proper name *John*:

x_1
JOHN(x_1) MALE(x_1)
[part of main DRS d_1] [at-issue] [embedding level: 0]

Providing an analysis of negation is outside the scope of this model, so the negated auxiliary *won't* contributes a ‘placeholder’ DRS with no drefs or arguments and a predicate NOT that simply marks that the DRS was contributed by a form of sentential negation (lines 10–11).

The transitive verb *eat* contributes the DRS below (see lines 14–16 in (71)):

(73) DRS contributed by the transitive verb *eat*:

e_1
EAT($e_1, x_1, _$)
[part of main DRS d_1] [at-issue] [embedding level: 0]

The pronoun *it* further specifies the DRS introduced by the verb *eat* (not shown in the temporal trace) and introduces its own DRSs (line 19–25):

(74) DRS contributed by the transitive verb *eat* and updated by the pronoun *it*:

e_1
EAT(e_1, x_1, x_2)
[part of main DRS d_1] [at-issue] [embedding level: 0]

(75) At-issue DRS contributed by the pronoun *it*:

x_2
NONHUMAN(x_2)
[part of main DRS d_1]
[at-issue]
[embedding level: 0]

(76) Unresolved-presupposition DRS contributed by the pronoun *it*:

$x_2 = \text{UNKNOWN}$
NONHUMAN(x_2)
[part of main DRS d_1]
[unresolved]
[embedding level: 0]

An attempt to resolve the pronoun *it* (lines 26–27) ends in failure (lines 29–31), since there is no suitable antecedent for it. The parsing process then moves on to the complementizer *if* (lines 32–33), which triggers the reanalysis of the previous clause from a main assertion to a conditional consequent. This means recalling all the four DRSs contributed by the previous clause and creating four new DRSs with the same content, except that the embedding level is set to 2 (conditional consequent) instead of 0 (main assertion).

The first DRS that gets recalled as part of the *if*-triggered reanalysis is the at-issue DRS in (75) above contributed by the pronoun *it* (lines 39–41). The reanalysis of this DRS (which sets the embedding level to 2) creates a new DRS in the `discourse_context` buffer (lines 44–46), shown below.

(77) The first DRS contributed by *if*-triggered reanalysis:

x_2
NONHUMAN(x_2)
[part of main DRS d_1]
[at-issue]
[embedding level: 2]

The second DRS retrieved as part of *if*-reanalysis is the unresolved-presupposition DRS contributed by the pronoun (lines 47–50). We see that recency is the most important factor for DRS activation in the *if*-reanalysis process. A new, reanalyzed unresolved-presupposition DRS is added to the `discourse_context` buffer (lines 53–56), which can be represented as follows.

(78) The second DRS contributed by *if*-triggered reanalysis:

$x_2 = \text{UNKNOWN}$ $\text{NONHUMAN}(x_2)$
[part of main DRS d_1] [unresolved] [embedding level: 2]

The third DRS retrieved during the *if*-reanalysis process is the one contributed by the transitive verb *eat* (lines 57–59). Because this DRS introduces an event, it triggers the ‘event recalled’ version of the “*if*-triggered reanalysis” rule, which does do things. On one hand, it adds the verbal predicate to the goal buffer as the value of the *if_conseq_pred* feature. On the other hand, it resets the embedding level of the recalled DRS to 2 and adds the modified DRS to the *discourse_context* buffer (lines 62–65). This DRS can be represented in DRT format as follows.

(79) DRS contributed by the transitive verb *eat* and updated by the pronoun *it*:

e_1
$\text{EAT}(e_1, x_1, x_2)$
[part of main DRS d_1] [at-issue] [embedding level: 2]

After this, the ‘dummy’ negative DRS contributed by the negated auxiliary *won’t* is retrieved (lines 66–67) and reanalyzed (lines 68–71). Finally, the DRS contributed by the proper name *John* is retrieved (lines 72–75) and reanalyzed (lines 76–81). The new DRS, which is the final one contributed by the *if*-reanalysis process, can be represented as shown below.

(80) DRS contributed by the proper name *John*:

x_1
$\text{JOHN}(x_1)$ $\text{MALE}(x_1)$
[part of main DRS d_1] [at-issue] [embedding level: 2]

At this point, the reanalysis process concludes and we continue with the incremental parsing process—specifically, the interpretation of the indefinite article *a* (in ...*a hamburger is* ...). We should note here that this process is cognitively unrealistic: it predicts that we need about 1.2 s to process sentence-final *if*, because the process of retrieving all these DRSs and reanalyzing them is very time consuming. We implement it here just to show that this type of process can be modeled in our framework, and leave a more appropriate model for a future occasion.⁵

The next DRS is contributed by the indefinite NP *a hamburger* in the *if*-clause (lines 84–88), and can be represented as follows.

(81) DRS contributed by the indefinite *a hamburger*:

x_3
HAMBURGER(x_3)
[part of main DRS d_2]
[at-issue]
[embedding level: 1]

The presence of this new DRS in the `discourse_context` buffer and the fact that the `entity_cataphora` feature in the goal buffer is turned on after the unsuccessful resolution of the pronoun *it* trigger an attempt to resolve the cataphoric pronoun (lines 89–90). The rule places the relevant information about the potential antecedent *a hamburger* in the `unresolved_discourse` buffer (lines 91–93) and places a retrieval request for an unresolved presupposition contributed by a pronoun. This request is successfully completed, and as a result, the unresolved DRS contributed by *it* is available in the retrieval buffer (lines 95–98).

The resolution of the cataphoric pronoun is declared a success (line 99–100) and the resolved presupposition, provided below for convenience, is added to the `discourse_context` buffer (lines 101–104).

⁵One improvement would be to spread the reanalysis process over several words in the *if*-clause rather than eagerly complete it.

Another improvement would be to explicitly model a form of clause-final wrap-up (maybe supplemented by clause-medial wrap-ups) that integrates/merges some of the DRSs that are indexed with the same DRS dref. This would ensure that there are fewer DRSs to recall during the *if*-reanalysis process.

Yet another improvement would be to explicitly model the main DRS like we did in the previous chapter (Chap. 8), in which case we might just need to recall only one DRS—the main one—and reanalyze its embedding level only once.

A related improvement would be to maintain the distributed representation of main DRSs we use in this chapter, but somehow ‘centralize’ their embedding level and DRS dref peg encodings in a separate DRS that would act as a ‘mother-node’ DRS to all the sub-DRSs. That is, we would build tree-like structures for discourse contexts; see Vermeulen (1994) and its Incremental Dynamic Predicate Logic extension in Brasoveanu and Dotlačil (2015a, b) for one way to model discourse contexts as tree-like update histories.

(82) Presupposed DRS resolving the pronoun *it* to the indefinite *a hamburger*:

$x_2 = x_3$ NONHUMAN(x_2)
[part of main DRS d_2] [presupposed] [embedding level: 1]

The copula *is* and the adjectival participle *overcooked* contribute a final DRS (lines 107–109) to the `discourse_context` buffer:

(83) Final at-issue DRS contributed by *overcooked*:

OVERCOOK(x_3)
[part of main DRS d_2] [at-issue] [embedding level: 1]

9.4 Modeling the Interaction of Conditionals and Cataphoric Presuppositions

We are now ready to move to the somewhat more complex case of event anaphora/cataphora associated with the adverb *again*. We first introduce the rules for the syntax and semantics of *again* and for the process of presupposition resolution for event anaphora (Sect. 9.4.1). We then discuss one way of capturing the ‘maximize presupposition’ effect we saw in Experiment 2 above (Sect. 9.4.2). Finally, we discuss the results of fitting the model to part of the Experiment 2 data (Sect. 9.4.3).

9.4.1 Rules for ‘Again’ and Presupposition Resolution

The resolution of event anaphora/cataphora contributed by the adverb *again* follows the same pattern as the resolution of entity anaphora/cataphora contributed by pronouns. The main difference is in the syntax of *again*: *again* is an adverb/adjunct, and

given the optionality of adjuncts, they basically need to be parsed bottom-up.⁶ That is, the syntactic attachment of adjuncts requires a form of syntactic reanalysis.

Consider again the conditional + event cataphora example in (16d) above, repeated in (84) below:

- (84) Jeffrey will argue with Danielle again if he argued with her in the courtyard last night.

The matrix clause ends with the anaphoric adverb *again*. To parse it, we need to attach it to the VP *will argue with Danielle* that has already been completely parsed and closed by the time we encounter *again*. To build the appropriate syntactic structure, we would therefore need to recall both the VP node and the higher S node so that the adverb *again* can be attached intermediately between them. For simplicity, we will simply recall the higher S node and add the adverb *again* as a third daughter, as shown by the two rules in (85) and (86) below.

- (85) `parser.productionstring(name="recall S for adjoining adv. AGAIN",` 1
`string="")` 2
`=g>` 3
`isa` `parsing_goal` 4
`task` `parsing` 5
`stack1` `None` 6
`found` `Adv` 7
`parsed_word` `=w` 8
`embedding_level` `=el` 9
`=retrieval>` 10
`isa` `word` 11
`cat` `Adv` 12
`==>` 13
`=g>` 14
`isa` `parsing_goal` 15
`task` `recall_S` 16
`+retrieval>` 17
`isa` `parse_state` 18
`node_cat` `S` 19
`daughter1` `NP` 20
`daughter2` `VP` 21
`~imaginal>` 22
`""`) 23
- (86) `parser.productionstring(name="build S adjunction and\` 1
`recall event for AGAIN", string="")` 2
`=g>` 3
`isa` `parsing_goal` 4
`task` `recall_S` 5
`stack1` `None` 6
`found` `Adv` 7
`drs_peg` `=drs_peg` 8
`=retrieval>` 9
`isa` `parse_state` 10
`node_cat` `S` 11
`daughter1` `NP` 12
`daughter2` `VP` 13
`==>` 14
`=g>` 15
`isa` `parsing_goal` 16
`task` `recall_event` 17
`+imaginal>` 18
`isa` `parse_state` 19

⁶Left-corner, bottom-up and top-down parsing can be all unified under generalized left-corner parsing (Demers 1977). See Hale (2014) for a recent discussion.

```

node_cat      S                                20
daughter1     NP                                21
daughter2     VP                                22
daughter3     Adv                               23
+retrieval>   24
isa           drs                               25
dref          ~None                            26
event_arg     ~None                            27
drs           =drs_peg                         28
""")                                             29

```

In addition to reanalyzing the S node, rule (86) places another retrieval request for the event contributed by the verb modified by the adverb *again*. This event is needed for the semantics of *again*. The fact that we need two separate retrieval requests, one on the syntax side for the S node and one on the semantics side for the event, is an artifact of our setup for syntactic and semantic parsing.

For expository simplicity, the semantic information constructed during incremental interpretation is assembled in chunks and buffers that are separate from the chunks and buffers where syntactic information is constructed. This enabled us to import the left-corner syntax parser we introduced in Chap. 4 basically as-is, and we were able to focus on the semantic aspects of interpretation in this and the previous chapter (Chap. 8 and this chapter) without worrying about a tighter integration of the syntactic and semantic aspects of parsing.

As we investigate more complex structures and their interpretation, these simplifying assumptions are likely to come into focus and require revision. It is possible that the structures constructed during the incremental interpretation process integrate phonetics/phonology, syntax and semantics in a tighter way, for example, along the lines of the linguistic representations countenanced by HPSG or CG.

Once the event contributed by the modified verb is available in the retrieval buffer, we are able to encode the unresolved presupposition contributed by *again*. This is accomplished by the rule in (87) below. The unresolved presupposition DRS contributed by *again* on lines 22–30 is largely parallel to the unresolved presupposition DRS contributed by pronouns. There are three main differences: (i) *arg1* (not *arg2*) is marked as UNKNOWN for *again* (line 24), (ii) the first predicate is (temporally) PRECEDES (not EQUALS; line 26), and (iii) the second predicate is the verbal predicate contributed by the modified verb (not gender; line 27).

```

(87) parser.productionstring(name="encode unresolved event presupposition\ 1
      for AGAIN", string="" 2
      =g> 3
      isa parsing_goal 4
      task recall_event 5
      stack1 None 6
      found Adv 7
      drs_peg =drs 8
      =retrieval> 9
      isa drs 10
      event_arg =ea 11
      pred1 =p1 12
      drs =drs 13
      embedding_level =el 14
      discourse_status at_issue 15
      ==> 16
      =g> 17
      isa parsing_goal 18
      task parsing 19

```

```

found          None                20
parsed_word    None                21
+unresolved_discourse>          22
isa            drs                 23
arg1           UNKNOWN            24
arg2           =ea                 25
pred1          PRECEDES           26
pred2          =p1                 27
drs            =drs                28
embedding_level =el                29
discourse_status unresolved       30
~retrieval>    31
~imaginal>     32
"""           33

```

Once the *again* presupposition is encoded, we can attempt to resolve it. There are three different rules for the three different embedding levels of the presupposition. We provide only the rule for embedding level 1 in (88) below, for ease of comparison with the pronoun rule. The rule has the same structure, the main difference is that we now require the potential antecedent to have a non-empty event argument (lines 31–32).

```

(88) parser.productionstring(name="attempting to resolve event presupposition;\ 1
presupposition at embedding level 1", string="" 2
= g> 3
isa parsing_goal 4
task ~reading_word 5
task ~move_dref_peg 6
task ~move_event_peg 7
task ~attempting_to_resolve_AGAIN 8
task ~attempting_to_resolve_cataphoric_AGAIN 9
task ~stop_resolution_attempt_AGAIN 10
task ~move_event_peg_and_wait_for_retrieval 11
task ~if_reanalysis 12
found None 13
?retrieval> 14
state free 15
=unresolved_discourse> 16
isa drs 17
arg1 UNKNOWN 18
arg2 =ea 19
pred2 =p2 20
drs =drs 21
embedding_level 1 22
discourse_status unresolved 23
==> 24
= g> 25
isa parsing_goal 26
task stop_resolution_attempt_AGAIN 27
+retrieval> 28
isa drs 29
dref ~None 30
event_arg ~None 31
event_arg ~=ea 32
drs ~=drs 33
embedding_level ~2 34
discourse_status at_issue 35
""", utility=5) 36

```

If the resolution succeeds, we add the resolved presupposition to the `discourse_context` buffer with the "resolution of AGAIN succeeded" rule in (89) below.

```

(89) parser.productionstring(name="resolution of AGAIN succeeded", string="" 1
      =g> 2
      isa parsing_goal 3
      task stop_resolution_attempt_AGAIN 4
      embedding_level =e1 5
      ~retrieval> 6
      isa drs 7
      dref ~None 8
      dref =ea 9
      event_arg =ea 10
      pred1 =p2 11
      ~unresolved_discourse> 12
      isa drs 13
      arg1 UNKNOWN 14
      arg2 =ea2 15
      pred1 =p1 16
      pred2 =p2 17
      drs =drs 18
      discourse_status unresolved 19
      ==> 20
      =g> 21
      isa parsing_goal 22
      task parsing 23
      event_cataphora None 24
      +discourse_context> 25
      isa drs 26
      arg1 =ea 27
      arg2 =ea2 28
      pred1 =p1 29
      pred2 =p2 30
      drs =drs 31
      embedding_level =e1 32
      discourse_status presupposed 33
      ~retrieval> 34
      ~unresolved_discourse> 35
      """) 36

```

Just as for pronouns, the resolution of the *again* presupposition can fail, either (i) because no suitable antecedent is retrieved or (ii) because an antecedent is retrieved, but the retrieved verbal predicate is different from the verbal predicate of the unresolved *again* presupposition. These two cases are handled by the rules in (90) and (91) below, respectively.

In both cases, we turn the *event_cataphora* feature on when the retrieval of a suitable antecedent event fails. That is, we do not simply mark the presupposition resolution process as a failure and end it. Instead, we assume that the *again* presupposition is cataphoric.

```

(90) parser.productionstring(name="resolution of AGAIN failed: no antecedent", 1
      string="" 2
      =g> 3
      isa parsing_goal 4
      task stop_resolution_attempt_AGAIN 5
      ?retrieval> 6
      state error 7
      ?unresolved_discourse> 8
      buffer full 9
      ~unresolved_discourse> 10
      isa drs 11
      arg1 UNKNOWN 12
      arg2 =ea2 13
      pred1 =p1 14
      pred2 =p2 15
      drs =drs 16
      embedding_level =e1 17
      discourse_status unresolved 18

```

```

=> 19
=g> 20
isa      parsing_goal 21
task     parsing      22
event_cataphora True   23
found    no_antecedent 24
~unresolved_discourse> 25
~retrieval> 26
""") 27

(91) parser.productionstring(name="resolution of AGAIN failed:\
antecedent with non-matching verbal predicate", string="") 1
=g> 2
isa      parsing_goal 4
task     stop_resolution_attempt_AGAIN 5
?unresolved_discourse> 6
buffer    full 7
=retrieval> 8
isa      drs 9
pred1    =p2 10
=unresolved_discourse> 11
isa      drs 12
dref     None 13
pred2    ~=p2 14
discourse_status unresolved 15
=> 16
=g> 17
isa      parsing_goal 18
task     parsing      19
event_cataphora True   20
found    no_antecedent 21
~retrieval> 22
~unresolved_discourse> 23
""") 24

```

Once the event presupposition contributed by *again* is marked as cataphoric and a suitable antecedent is available in the `discourse_context` buffer, we start a cataphoric search, just as we did for pronouns. Once again, there are three rules for cataphoric search depending on the embedding level of the potential event antecedent. We provide only the embedding level 1 rule in (92) below.

```

(92) parser.productionstring(name="attempting to resolve cataphoric event\
presupposition; antecedent at embedding level 1", string="") 1
=g> 2
isa      parsing_goal 4
task     ~reading_word 5
task     ~move_dref_peg 6
task     ~move_event_peg 7
task     ~attempting_to_resolve_AGAIN 8
task     ~attempting_to_resolve_cataphoric_AGAIN 9
task     ~stop_resolution_attempt_AGAIN 10
task     ~if_reanalysis 11
found    None 12
event_cataphora True 13
=discourse_context> 14
isa      drs 15
dref     ~None 16
dref     =ea 17
arg1     ~None 18
event_arg =ea 19
pred1    ~None 20
pred1    ~None 21
pred1    =p1 22
pred2    None 23
drs      =drs 24
embedding_level 1 25
?retrieval> 26

```

```

state                free                                27
==>                 28
=>                  29
isa                  parsing_goal                        30
task                 stop_resolution_attempt_AGAIN       31
+unresolved_discourse> 32
isa                  drs                                33
dref                  =ea                                34
arg1                  UNKNOWN                           35
pred2                 =p1                                36
drs                   =drs                               37
embedding_level      1                                  38
discourse_status      unresolved                        39
+retrieval>          40
isa                  drs                                41
dref                  None                               42
arg1                  UNKNOWN                           43
arg2                  ~None                             44
pred1                 ~None                             45
drs                   ~=drs                             46
embedding_level      ~0                                 47
discourse_status      unresolved                        48
""", utility=5)                                         49

```

If the resolution of cataphoric *again* succeeds, we add the resolved presupposition to the `discourse_context` buffer with the rule in (93) below.

```

(93) parser.productionstring(name="resolution of cataphoric AGAIN succeeded", 1
      string="" 2
      => 3
      isa parsing_goal 4
      task stop_resolution_attempt_AGAIN 5
      =retrieval> 6
      isa drs 7
      dref None 8
      arg1 UNKNOWN 9
      arg2 ~None 10
      arg2 =a2 11
      pred1 =p1 12
      pred2 =p2 13
      =unresolved_discourse> 14
      isa drs 15
      dref =dref 16
      pred2 =p2 17
      embedding_level =e1 18
      drs =drs 19
      ==> 20
      => 21
      isa parsing_goal 22
      task parsing 23
      event_cataphora None 24
      +discourse_context> 25
      isa drs 26
      arg1 =dref 27
      arg2 =a2 28
      pred1 =p1 29
      pred2 =p2 30
      drs =drs 31
      embedding_level =e1 32
      discourse_status presupposed 33
      ~unresolved_discourse> 34
      ~retrieval> 35
      """) 36

```

If the resolution of cataphoric *again* fails because the antecedent does not match the verbal predicate, the rule in (94) below is triggered. If there is no suitable

antecedent, the general rule in (90) above, which applies to both anaphoric and cataphoric searches, is triggered.

```
(94) parser.productionstring(name="resolution of cataphoric AGAIN failed:\n
      antecedent with non-matching verbal predicate", string=""
      =g>
      isa          parsing_goal
      task         stop_resolution_attempt_AGAIN
      ?unresolved_discourse>
      buffer       full
      =retrieval>
      isa          drs
      pred2        =p2
      =unresolved_discourse>
      isa          drs
      dref         ~None
      pred2        ~p2
      discourse_status unresolved
      ==>
      =g>
      isa          parsing_goal
      task         parsing
      event_cataphora True
      found        no_antecedent
      ~retrieval>
      ~unresolved_discourse>
      """)
```

We are now ready to see how all these rules get deployed during the incremental interpretation of the conditional & event cataphora example in (95) below, repeated from above. We stop the parsing immediately after the preposition *with* in the *if*-clause since this is when event cataphora is resolved.

(95) Jeffrey will argue with Danielle again if he argued with [her in the courtyard last night].

```
(96) ****Environment: {1: {'text': 'Jeffrey', 'position': (320, 180)}}
      (0.2012, 'PROCEDURAL', 'RULE FIRED: project: NP ==> ProperN')
      (0.2012, 'discourse_context', 'CREATED A CHUNK: drs(dref=x1, arg1=x1,\n
      discourse_status=at_issue, drs=d1, embedding_level=0,\n
      pred1=pred(arity=1, constant_name=_jeffrey_),\n
      pred2=pred(arity=1, constant_name=_male_))')
      ****Environment: {1: {'text': 'will', 'position': (320, 180)}}
      ****Environment: {1: {'text': 'argue', 'position': (320, 180)}}
      (0.907, 'PROCEDURAL', 'RULE FIRED: project and complete: VP ==> Vt PP\n
      (no if_conseq_pred present)')
      (0.907, 'discourse_context', 'CREATED A CHUNK: drs(dref=e1, event_arg=e1,\n
      arg1=x1, discourse_status=at_issue, drs=d1, embedding_level=0,\n
      pred1=pred(arity=event_plus_2, constant_name=_argu_))')
      ****Environment: {1: {'text': 'with', 'position': (320, 180)}}
      (1.2633, 'PROCEDURAL', 'RULE FIRED: project and complete: PP ==> P NP')
      ****Environment: {1: {'text': 'Danielle', 'position': (320, 180)}}
      (1.6189, 'PROCEDURAL', 'RULE FIRED: project and complete: NP ==> ProperN')
      (1.6189, 'discourse_context', 'CREATED A CHUNK: drs(dref=x2, arg1=x2,\n
      discourse_status=at_issue, drs=d1, embedding_level=0,\n
      pred1=pred(arity=1, constant_name=_danielle_),\n
      pred2=pred(arity=1, constant_name=_female_))')
      ****Environment: {1: {'text': 'again', 'position': (320, 180)}}
      (1.9729, 'PROCEDURAL', 'RULE FIRED: recall S for adjoining adv. AGAIN')
      ****Environment: {1: {'text': 'if', 'position': (320, 180)}}
      (2.1243, 'retrieval', 'RETRIEVED: parse_state(daughter1=NP, daughter2=VP,\n
      node_cat=S)')
      (2.1353, 'PROCEDURAL', 'RULE FIRED: build S adjunction and\n
      recall event for AGAIN')
      (2.2828, 'retrieval', 'RETRIEVED: drs(dref=e1, event_arg=e1, arg1=x1,\n
      arg2=x2, discourse_status=at_issue, drs=d1, embedding_level=0,\n
```

```

pred1=pred(arity=event_plus_2, constant_name=_argu_),\
pred2=pred(arity=2, constant_name=_with_))'
(2.2938, 'PROCEDURAL', 'RULE FIRED: encode unresolved event presupposition\
for AGAIN')
(2.2938, 'unresolved_discourse', 'CREATED A CHUNK: drs(arg1=UNKNOWN,\
arg2=e1, discourse_status=unresolved, drs=d1, embedding_level=0,\
pred1=pred(arity=2, constant_name=precedes_),\
pred2=pred(arity=event_plus_2, constant_name=_argu_))'
(2.3048, 'PROCEDURAL', 'RULE FIRED: attempting to resolve event\
presupposition; presupposition at embedding level 0')
(2.4880, 'retrieval', 'RETRIEVED: None')
(2.4990, 'PROCEDURAL', 'RULE FIRED: resolution of AGAIN failed:\
no antecedent')
(2.7075, 'PROCEDURAL', 'RULE FIRED: project and complete:\
sentence-final if')
(2.7295, 'PROCEDURAL', 'RULE FIRED: start if-triggered reanalysis\
(for sentence-final if')
****Environment: {1: {'text': 'he', 'position': (320, 180)}}
(2.8768, 'discourse_context', 'CREATED A CHUNK: drs(dref=x2, arg1=x2,\
discourse_status=at_issue, drs=d1, embedding_level=2,\
pred1=pred(arity=1, constant_name=danielle_),\
pred2=pred(arity=1, constant_name=female_))'
(3.0342, 'discourse_context', 'CREATED A CHUNK: drs(arg1=UNKNOWN, arg2=e1,\
discourse_status=unresolved, drs=d1, embedding_level=2,\
pred1=pred(arity=2, constant_name=precedes_),\
pred2=pred(arity=event_plus_2, constant_name=_argu_))'
(3.1991, 'discourse_context', 'CREATED A CHUNK: drs(dref=x1, arg1=x1,\
discourse_status=at_issue, drs=d1, embedding_level=2,\
pred1=pred(arity=1, constant_name=jeffrey_),\
pred2=pred(arity=1, constant_name=male_))'
(3.3556, 'discourse_context', 'CREATED A CHUNK: drs(dref=e1,\
event_arg=e1, arg1=x1, arg2=x2,\
discourse_status=at_issue, drs=d1, embedding_level=2,\
pred1=pred(arity=event_plus_2, constant_name=_argu_),\
pred2=pred(arity=2, constant_name=_with_))'
(3.5498, 'PROCEDURAL', 'RULE FIRED: stop if-triggered reanalysis')
(3.7597, 'PROCEDURAL', 'RULE FIRED: project: NP ==> PRO')
(3.7597, 'discourse_context', 'CREATED A CHUNK: drs(dref=x3, arg1=x3,\
discourse_status=at_issue, drs=d2, embedding_level=1,\
pred1=pred(arity=1, constant_name=male_))'
(3.7597, 'unresolved_discourse', 'CREATED A CHUNK: drs(arg1=x3, drs=d2,\
arg2=UNKNOWN, discourse_status=unresolved, embedding_level=1,\
pred1=pred(arity=2, constant_name=equals_),\
pred2=pred(arity=1, constant_name=male_))'
(3.7817, 'PROCEDURAL', 'RULE FIRED: attempting to resolve pronoun;\
pronoun at embedding level 1')
(3.8984, 'retrieval', 'RETRIEVED: drs(dref=x1, arg1=x1,\
discourse_status=at_issue, drs=d1, embedding_level=0,\
pred1=pred(arity=1, constant_name=jeffrey_),\
pred2=pred(arity=1, constant_name=male_))'
****Environment: {1: {'text': 'argued', 'position': (320, 180)}}
(3.9094, 'PROCEDURAL', 'RULE FIRED: resolution of PRO succeeded')
(3.9094, 'discourse_context', 'CREATED A CHUNK: drs(arg1=x3, arg2=x1,\
discourse_status=presupposed, drs=d2, embedding_level=1,\
pred1=pred(arity=2, constant_name=equals_),\
pred2=pred(arity=1, constant_name=male_))'
(4.1306, 'PROCEDURAL', 'RULE FIRED: project and complete: VP ==> Vt PP\
(if_conseq_pred present, but also event cataphora)')
(4.1306, 'discourse_context', 'CREATED A CHUNK: drs(dref=e2, event_arg=e2,\
arg1=x3, discourse_status=at_issue, drs=d2, embedding_level=1,\
pred1=pred(arity=event_plus_2, constant_name=_argu_))'
(4.1526, 'PROCEDURAL', 'RULE FIRED: attempting to resolve cataphoric event\
presupposition; antecedent at embedding level 1')
(4.1526, 'unresolved_discourse', 'CREATED A CHUNK: drs(arg1=UNKNOWN,\
discourse_status=unresolved, dref=e2, drs=d2, embedding_level=1,\
pred2=pred(arity=event_plus_2, constant_name=_argu_))'
****Environment: {1: {'text': 'with', 'position': (320, 180)}}
(4.2762, 'retrieval', 'RETRIEVED: drs(arg1=UNKNOWN, arg2=e1,\
discourse_status=unresolved, drs=d1, embedding_level=2,\
pred1=pred(arity=2, constant_name=precedes_),\
pred2=pred(arity=event_plus_2, constant_name=_argu_))'

```



```

(4.2872, 'PROCEDURAL', 'RULE FIRED: resolution of cataphoric AGAIN\
succeeded') 102
(4.2872, 'discourse_context', 'CREATED A CHUNK: drs(arg1=e2, arg2=e1,\
discourse_status=presupposed, drs=d2, embedding_level=1,\
pred1=pred(arity=2, constant_name=_precedes_),\
pred2=pred(arity=event_plus_2, constant_name=_argu_))') 103
Time to read preposition: 0.3612000000000002 104
105
106
107
108
109

```

In (96), we only list the processing steps that are most relevant to conditionals and cataphora resolution. After the proper name *Jeffrey*, the prepositional verb (*will*) *argue with* and the proper name *Danielle* contribute DRSs to the discourse context (lines 1–21), we start parsing the adverb *again*. The event DRS contributed by (*will*) *argue with* is recalled (lines 29–32), and the *again* presupposition is encoded in the *unresolved_discourse* buffer (lines 35–38). An attempt to anaphorically resolve this presupposition fails (lines 39–43), after which the process of *if*-triggered reanalysis begins, which contributes four new DRSs to the discourse context (lines 46–66): these are the four DRSs contributed by *Jeffrey*, (*will*) *argue with*, *Danielle* and *again*, except their embedding level is set to 2 (conditional consequent) instead of 0 (main assertion).

The pronoun *he* is then parsed and correctly resolved to the dref contributed by the proper name *Jeffrey* (lines 67–86). Then, as soon as the verb *argued* is parsed (lines 87–91), a cataphoric search attempting to resolve *again* is started (lines 92–96). The search is successfully completed (lines 98–101), so the *again*-contributed presupposition is resolved (lines 102–107).

The simulation ends with the time taken to read the preposition *with*, reported on line 109. This time crucially includes the *again* cataphoric search, so it can be used to model the Experiment 2 data for this ROI. We see that the time taken to read the preposition is about 360 ms, which is reasonable. In the next (Sect. 9.4.2), we will see that this time increases under specific conditions; and in the final (Sect. 9.4.3), we will fit the predicted RTs for this region to the Experiment 2 data.

9.4.2 Rules for ‘Maximize Presupposition’

In Sect. 9.1.2 above, we noted that conditionals with matching VP meanings and no presuppositional *again*, like the one in (97) below (repeated from above), were significantly slower than conjunctions with matching meanings or conditionals with mismatching meanings.

- (97) Jeffrey will argue with Danielle if he argued with her in the courtyard last night.

We conjectured that the processing difficulty associated with these conditionals was an effect of the Maximize Presupposition principle (Heim 1991), which requires that a presupposed VP meaning should be marked as such by *again*. This penalizes conditionals with matching VP meanings, while conditionals with non-identical VP meanings and coordinations should not be affected.

While Maximize Presupposition is commonly used as an explanatory principle in the formal semantics literature, there is no received way to formalize it and no explicit conjectures about the way it could become part of a mechanistic processing model of natural language interpretation.

In this section, we propose a tentative model of Maximize Presupposition processing as noise/error correction. Specifically, we conjecture that upon encountering the verb *argued* in the *if*-clause of example (97) above, the human processor pauses to consider whether it has erroneously failed to activate the *event_cataphora* feature in the goal buffer.

That is, given that the *if*-clause could satisfy a presuppositional *again* in the matrix clause, which would not violate Maximize Presupposition, the human processor hypothesizes that such an unresolved *again* presupposition might actually be present in declarative memory, but the *event_cataphora* feature in the goal buffer has erroneously failed to encode it.

Consequently, a search for an unresolved *again* presupposition is initialized, which adds extra reading time. In sum, the processing difficulty associated with a Maximize Presupposition violation is attributed to an extra retrieval request meant to check whether a goal feature was erroneously encoded.

This account is implemented in our mechanistic processing model by means of several rules. First, when the event DRS is recalled during the *if*-reanalysis process, its verbal predicate is stored in the goal buffer as the value of the *if_conseq_pred* feature (see the rule in (66) above).

Assuming the presence of such a feature, the rule in (98) below is triggered when we parse the matching verb *argued* in the *if*-clause. Specifically, the *if_conseq_pred* features has to be non-empty (line 18), and the *event_cataphora* feature should be turned off (*~True*; line 20). If that is the case, the usual actions associated with a prepositional verb are triggered (lines 25–51). Crucially, we also place a retrieval request for an unresolved event presupposition that we might have mistakenly failed to encode in the *event_cataphora* feature because of communication noise, comprehension noise, encoding noise etc. (lines 52–61).

```
(98) parser.productionstring(name="project and complete: VP ==> Vt PP\
      (if_conseq_pred present, matching pred)", string="")
      =g>
      isa          parsing_goal
      task          parsing
      stack1        VP
      found         VtPP
      parsed_word   =w
      arg_stack1    =a1
      right_edge_stack1 VP
      right_edge_stack1 =re1
      right_edge_stack2 =re2
      right_edge_stack3 =re3
      right_edge_stack4 =re4
      event_peg     =ev_peg
      drs_peg       =drs_peg
      embedding_level =e1
      if_conseq_pred ~None
      if_conseq_pred =verbal_pred
      event_cataphora ~True
      =retrieval>
      isa          word
```

```

pred1          =verbal_pred      23
==>           24
=g>            25
isa            parsing_goal      26
task           move_event_peg_and_wait_for_retrieval 27
stack1         PP                28
found          None              29
parsed_word    None              30
right_edge_stack1 PP            31
right_edge_stack2 =re1           32
right_edge_stack3 =re2           33
right_edge_stack4 =re3           34
+imaginal>     35
isa            parse_state       36
mother         =re2              37
mother_of_mother =re3            38
node_cat       VP                39
daughter1      VtPP              40
daughter2      PP                41
lex_head       =w                42
+discourse_context> 43
isa            drs               44
dref           =ev_peg           45
event_arg      =ev_peg           46
arg1           =a1              47
pred1          =verbal_pred      48
drs            =drs_peg          49
embedding_level =el              50
discourse_status at_issue        51
+retrieval>     52
isa            drs               53
dref           None              54
arg1           UNKNOWN           55
arg2           ~None             56
pred1          ~None             57
pred2          =verbal_pred      58
drs            =drs_peg          59
embedding_level ~0               60
discourse_status unresolved      61
~imaginal>     62
~unresolved_discourse> 63
""")           64

```

In our example (97), this search for an unencoded *again* fails, and triggers the rule in (99) below.

```

(99) parser.productionstring(name="search for an unencoded AGAIN failed", 1
string="" 2
=g> 3
isa parsing_goal 4
task stop_resolution_attempt_AGAIN 5
if_conseq_pred ~None 6
?retrieval> 7
state error 8
?unresolved_discourse> 9
buffer empty 10
==> 11
=g> 12
isa parsing_goal 13
task parsing 14
~retrieval> 15
""") 16

```

We can see all these rules in action, as well as their consequences for reading times, in (101) below.

(100) Jeffrey will argue with Danielle if he argued with [her in the courtyard last night].

```

(101) ****Environment: {1: {'text': 'Jeffrey', 'position': (320, 180)}} 1
      (0.2021, 'PROCEDURAL', 'RULE FIRED: project: NP ==> ProperN') 2
      (0.2021, 'discourse_context', 'CREATED A CHUNK: drs(dref=x1, arg1=x1,\ 3
        discourse_status=at_issue, drs=d1, embedding_level=0,\ 4
        pred1=pred(arity=1, constant_name=_jeffrey_),\ 5
        pred2=pred(arity=1, constant_name=_male_))') 6
      ****Environment: {1: {'text': 'will', 'position': (320, 180)}} 7
      ****Environment: {1: {'text': 'argue', 'position': (320, 180)}} 8
      (0.9099, 'PROCEDURAL', 'RULE FIRED: project and complete: VP ==> Vt PP\ 9
        (no if_conseq_pred present))' 10
      (0.9099, 'discourse_context', 'CREATED A CHUNK: drs(dref=e1, event_arg=e1,\ 11
        arg1=x1, discourse_status=at_issue, drs=d1, embedding_level=0,\ 12
        pred1=pred(arity=event_plus_2, constant_name=_argu_))') 13
      ****Environment: {1: {'text': 'with', 'position': (320, 180)}} 14
      (1.2635, 'PROCEDURAL', 'RULE FIRED: project and complete: PP ==> P NP') 15
      ****Environment: {1: {'text': 'Danielle', 'position': (320, 180)}} 16
      (1.6187, 'PROCEDURAL', 'RULE FIRED: project and complete: NP ==> ProperN') 17
      (1.6187, 'discourse_context', 'CREATED A CHUNK: drs(dref=x2, arg1=x2,\ 18
        discourse_status=at_issue, drs=d1, embedding_level=0,\ 19
        pred1=pred(arity=1, constant_name=_danielle_),\ 20
        pred2=pred(arity=1, constant_name=_female_))') 21
      ****Environment: {1: {'text': 'if', 'position': (320, 180)}} 22
      (1.9741, 'PROCEDURAL', 'RULE FIRED: project and complete:\ 23
        sentence-final if') 24
      (1.9961, 'PROCEDURAL', 'RULE FIRED: start if-triggered reanalysis\ 25
        (for sentence-final if)') 26
      ****Environment: {1: {'text': 'he', 'position': (320, 180)}} 27
      (2.1434, 'discourse_context', 'CREATED A CHUNK: drs(dref=x2, arg1=x2,\ 28
        discourse_status=at_issue, drs=d1, embedding_level=2,\ 29
        pred1=pred(arity=1, constant_name=_danielle_),\ 30
        pred2=pred(arity=1, constant_name=_female_))') 31
      (2.3020, 'discourse_context', 'CREATED A CHUNK: drs(dref=e1, event_arg=e1,\ 32
        arg1=x1, arg2=x2,\ 33
        discourse_status=at_issue, drs=d1, embedding_level=2,\ 34
        pred1=pred(arity=event_plus_2, constant_name=_argu_),\ 35
        pred2=pred(arity=2, constant_name=_with_))') 36
      (2.4658, 'discourse_context', 'CREATED A CHUNK: drs(dref=x1, arg1=x1,\ 37
        discourse_status=at_issue, drs=d1, embedding_level=2,\ 38
        pred1=pred(arity=1, constant_name=_jeffrey_),\ 39
        pred2=pred(arity=1, constant_name=_male_))') 40
      (2.6600, 'PROCEDURAL', 'RULE FIRED: stop if-triggered reanalysis') 41
      (2.8687, 'PROCEDURAL', 'RULE FIRED: project: NP ==> PRO') 42
      (2.8687, 'discourse_context', 'CREATED A CHUNK: drs(dref=x3, arg1=x3,\ 43
        discourse_status=at_issue, drs=d2, embedding_level=1,\ 44
        pred1=pred(arity=1, constant_name=_male_))') 45
      (2.8687, 'unresolved_discourse', 'CREATED A CHUNK: drs(arg1=x3, drs=d2,\ 46
        arg2=UNKNOWN, discourse_status=unresolved, embedding_level=1,\ 47
        pred1=pred(arity=2, constant_name=_equals_),\ 48
        pred2=pred(arity=1, constant_name=_male_))') 49
      (2.8907, 'PROCEDURAL', 'RULE FIRED: attempting to resolve pronoun;\ 50
        pronoun at embedding level 1') 51
      (3.0058, 'retrieval', 'RETRIEVED: drs(dref=x1, arg1=x1,\ 52
        discourse_status=at_issue, drs=d1, embedding_level=0,\ 53
        pred1=pred(arity=1, constant_name=_jeffrey_),\ 54
        pred2=pred(arity=1, constant_name=_male_))') 55
      ****Environment: {1: {'text': 'argued', 'position': (320, 180)}} 56
      (3.0168, 'PROCEDURAL', 'RULE FIRED: resolution of PRO succeeded') 57
      (3.0168, 'discourse_context', 'CREATED A CHUNK: drs(arg1=x3, arg2=x1,\ 58
        discourse_status=presupposed, drs=d2, embedding_level=1,\ 59
        pred1=pred(arity=2, constant_name=_equals_),\ 60
        pred2=pred(arity=1, constant_name=_male_))') 61
      (3.2370, 'PROCEDURAL', 'RULE FIRED: project and complete: VP ==> Vt PP\ 62
        (if_conseq_pred present, matching pred)') 63
      (3.2370, 'discourse_context', 'CREATED A CHUNK: drs(dref=e2, event_arg=e2,\ 64
        arg1=x3, discourse_status=at_issue, drs=d2, embedding_level=1,\ 65
        pred1=pred(arity=event_plus_2, constant_name=_argu_))') 66
      ****Environment: {1: {'text': 'with', 'position': (320, 180)}} 67
      (3.4202, 'retrieval', 'RETRIEVED: None') 68
      (3.4312, 'PROCEDURAL', 'RULE FIRED: search for an unencoded AGAIN failed') 69
      Time to read preposition: 0.39780000000000015 70
      71

```

The simulation in (101) proceeds as expected until we reach the crucial point, namely the project-and-complete-VP rule on lines (62–63). This rule triggers a memory search for an unencoded *again* presupposition, which takes place while the preposition *with* is being read (line 67). The search fails (lines 68–69), but the extra time needed for such a failed search can be seen in the higher reading time associated with the preposition, which is now almost 400 ms (line 71 in (101)).

9.4.3 *Fitting the Model to the Experiment 2 Data*

We are now ready to fit the model to part of the Experiment 2 data. Specifically, we will focus on the four match conditions in ((102a)–102d) below, and the two mismatch & cataphora conditions in (102e–102f).

- (102)
- a. Jeffrey will argue with Danielle and he argued with her.
 - b. Jeffrey will argue with Danielle if he argued with her.
 - c. Jeffrey will argue with Danielle again and he argued with her.
 - d. Jeffrey will argue with Danielle again if he argued with her.
 - e. Jeffrey will argue with Danielle again and he played with her.
 - f. Jeffrey will argue with Danielle again if he played with her.

We do not attempt to model the remaining two conditions of Experiment 2 because our model does not really have to say anything about the mismatch & nothing, i.e., no-cataphora, cases. In fact, our model is not designed to capture the *and* & cataphora cases either, i.e., (102c) (match) or (102e) (mismatch), but we include them here for completeness.

Our model is set up to capture the *if*-conditions in (102b), (102d) and (102f), and we will focus on these conditions for most of our discussion in this subsection.

The mean RTs for the 6 conditions in (102) obtained in Experiment 2 (averaging over both subjects and items) are, in order: 364.05, 429.01, 390.13, 378.83, 374.28, and 387.72. The file `estimate_parser_parallel.py` (linked to in Appendix 9.6.4) lists these 6 conditions and the corresponding mean RTs, and provides the code for the Bayesian model that fits our incremental interpreter to data.

The Bayesian model is particularly simple: we only estimate the latency exponent and keep the other parameters fixed (see Appendix 9.6.1 for the exact values). The prior for the latency exponent is a half-Normal distribution—line 4 in (103) below. The ACT-R model we have introduced provides the likelihood function (lines 6–8). We sample NDRAWS (=1500) values from the posterior (lines 10–11).

(103)

```
parser_model = Model()  
with parser_model:  
    # Priors  
    latency_exponent = HalfNormal('le', sd=0.3)  
    # Likelihood  
    pyactr_rt = actrmodel_latency(latency_exponent)  
    mu_rt = Deterministic('mu_rt', pyactr_rt)  
    rt_observed = Normal('rt_observed', mu=mu_rt, sd=30, observed=RT)  
    # Compute posteriors  
    step = pm.SMC()  
    trace = sample(draws=NDRAWS, step=step, njobs=1)
```

1
2
3
4
5
6
7
8
9
10
11

The posterior estimates for the latency exponent and the 6 mean RTs are provided in (104) below and are plotted in Fig. 9.3.

(104)

	MEAN	SD	HPD _{2.5%}	HPD _{97.5%}	OBSERVED
le	0.03	0.02	0.0	0.06	N/A
μ_0 (conj-nothing-match)	352.63	0.84	351.1	354.00	364.05
μ_1 (cond-nothing-match)	385.25	12.53	366.0	408.40	429.01
μ_2 (conj-cata-match)	407.43	12.65	388.0	430.80	390.13
μ_3 (cond-cata-match)	371.55	9.23	353.5	385.70	378.83
μ_4 (conj-cata-mismatch)	407.43	12.65	388.0	430.80	374.28
μ_5 (cond-cata-mismatch)	387.72	0.16	387.5	388.00	387.72

Note that the Rhat values for this model are practically 1:

(105)

```
{'le': 0.9996690236773664,  
 'mu_rt': array([0.99966776, 0.99966795, 0.99966794,  
                 0.99972701, 0.99966794, 0.99976515])}
```

1
2
3

We see that the model captures the conditional & cataphora conditions—both match, μ_3 , and mismatch, μ_5 —very well. This is a consequence of the spreading

Fig. 9.3 Parser model: observed versus predicted RT

activation from the `discourse_context` buffer, which boosts the activation of the correct antecedent for the match condition μ_3 , but which has no effect for the mismatch condition μ_5 . The explanatory processing mechanism used here is spreading activation, i.e., the influence of the cognitive context on memory retrieval latency (and accuracy). This is the same explanatory mechanism as the one we used in the previous chapter when we captured the difference in latency between the semantic evaluation of sentences with varying fans.

The conjunction conditions are captured reasonably well, particularly the control condition μ_0 and the conjunction & cataphora & match condition μ_2 . The model does not make any distinction between the two conjunction & cataphora conditions μ_2 (match) and μ_4 (mismatch): both retrieval requests proceed and fail the same way. Unfortunately, we overestimate the time required for retrieval failure, particularly for the mismatch μ_4 condition, where the observed value is a low 374.28 ms.

It might be that, by the time the human participants in the experiment read the preposition following the second finite verb, they realize that the stimulus overall is hopeless, and they give up on deeper processing rules like attempting a cataphoric search. This would explain why the observed RT for the μ_4 condition is fairly close to the control (conjunction) condition μ_0 . One way to implement this in our model would be to have a rule that turns off the `event_cataphora` feature for overly difficult/incoherent conditions like μ_4 . Firing this extra rule would add about 10 ms relative to the control condition μ_0 , which would be almost exactly right.

Our attempt to capture the ‘maximize presupposition’ effect in the μ_1 condition provides a good qualitative fit, but quantitatively, the effect is greatly underestimated: the estimated mean is 385.25 ms, while the observed mean is 429.01 ms. Clearly, a failed attempt to retrieve an unencoded *again* from declarative memory is not sufficient to capture the processing effects of the semantic-pragmatic reasoning involved in ascertaining a failure to ‘maximize presupposition’.

But the extra retrieval request and its failure are sufficient to capture the qualitative pattern. The estimated mean for μ_1 (conditionals with matching predicates and no cataphora) is greater than the estimated mean for the control condition μ_0 (conjunctions with matching predicates and no cataphora). It is also greater than the estimated mean for μ_3 (conditionals with matching predicates and cataphora).

Similarly, we capture the qualitative pattern involving the two cataphora & match cases: the estimated mean for μ_2 (conjunctions), where the attempt to resolve the *again* cataphora fails, is higher than the estimated mean for μ_3 (conditionals), where the attempt to resolve the *again* cataphora succeeds.

However, we do not capture the difference between conditionals and conjunctions in the cataphora & mismatch cases, i.e., μ_4 versus μ_5 . We predict that conjunctions (estimated mean: 407.43 ms) take longer than conditionals (estimated mean: 387.72 ms). The observed values exhibit the opposite pattern: 374.28 and 387.72 ms, respectively. Once again, the observation we made above about conjunction & cataphora & mismatch cases could resolve this issue: if these conditions are too hard and human participants never even start a cataphoric search process, we expect the predicted pattern to be reversed.

We leave further developments of this model for future work. But given the relatively poor data fit exhibited by our complex model, it is reasonable to ask if going for a simpler, but less explanatory model, for example, a linear model of some sort, is not a better way to proceed.

We believe that the independently-motivated commitments we made to specific (i) formal semantics representations and theories, (ii) cognitive-architectural organization principles and constraints and (iii) language processing theories and models should not be abandoned in future iterations of this modeling endeavor. As Neal (1996) puts it:

“Sometimes a simple model will outperform a more complex model [...] [But] deliberately limiting the complexity of the model is not fruitful when the problem is evidently complex. Instead, if a simple model is found that outperforms some particular complex model, the appropriate response is to define a different complex model that captures whatever aspect of the problem led to the simple model performing well.” (Neal 1996, 103–104)

9.5 Conclusion

In this chapter, we built the first (to our knowledge) formally and computationally explicit mechanistic model of active anaphoric/cataphoric search for an antecedent. This model integrates rich semantic representations (independently motivated in the formal semantics literature) and processing mechanisms (independently motivated in the psycholinguistics literature) into a wide-coverage cognitive architecture (ACT-R).

In the spirit of van der Sandt (1992), Kamp (2001a,b), the model analyzes anaphora and presupposition as fundamentally processing-level phenomena that guide and constrain the cognitive process of integration, or linking, of new and old semantic information. Anaphora and presupposition have semantic effects, but they are not exclusively, or even primarily, semantics.

There are many open questions left for future research about (i) the exact nature of the semantic representations deployed in these models, (ii) the fine details of the processing mechanisms, (iii) how exactly these representations and processes should be integrated into a general, independently-constrained cognitive architecture, and (iv) the exact division of labor between semantics and processing for the analysis of anaphora and presupposition. We hope the modeling endeavor pursued in this chapter provides a framework for formulating these questions in a precise way, and for mounting a systematic search for answers.

We think that the most important lesson to be drawn from the extensive and detailed modeling attempt in this chapter is methodological. Computationally explicit mechanistic processing models that can be fit to experimental data are crucial when working at the interface between theoretical linguistics and experimental psycholinguistics in general, and at the semantics-psycholinguistics interface in particular.

Our argument for this is as follows. We started the chapter with a general question about the processing of semantic representations (is it incremental and predictive?).

To shed light on this question, we collected a reasonably rich amount of real-time experimental data, we analyzed the data with standard methods (mixed-effects linear models) and we informally stated an account that linked the theoretical question and the experimental data in a reasonably adequate way.

However, our ACT-R model, which explicitly formalized the proposed account, was able to quantitatively capture some, but crucially not all the data. This partial quantitative failure of our detailed, computationally-explicit cognitive model is enlightening: it opens up a variety of specific questions for future research that would have been missed had we stayed at the informal level of our initial account.

Specifically, it turns out that for the informal account to really work, we need auxiliary hypotheses that are both crucial for and completely glossed over by that informal account. Looking carefully only at the experimental data (or the results of standard statistical methods applied to that data), or only at the formal semantic representations, or even at both, but separately, is not enough. We need to be formally and computationally explicit about how we link them via mechanistically-specified processing models.

This lesson is not new by any means. In fact, it is very familiar to generative linguists when it comes to formulating competence-level theories:

“Precisely constructed models for linguistic structure can play an important role, both negative and positive, in the process of discovery itself. By pushing a precise but inadequate formulation to an unacceptable conclusion, we can often expose the exact source of this inadequacy and, consequently, gain a deeper understanding of the linguistic data. More positively, a formalized theory may automatically provide solutions for many problems other than those for which it was explicitly designed. Obscure and intuition-bound notions can neither lead to absurd conclusions nor provide new and correct ones, and hence they fail to be useful in two important respects. [We need] to recognize the productive potential in the method of rigorously stating a proposed theory and applying it strictly to linguistic material with no attempt to avoid unacceptable conclusions by *ad hoc* adjustments or loose formulation.” (Chomsky 1957, p. 5)

Computational modeling of cognitive phenomena has become increasingly central to cognitive science for fundamentally the same reason. As Lewandowsky and Farrell (2010, p. 9) put it: “[e]ven intuitively attractive notions may fail to provide the desired explanation for behavior once subjected to the rigorous analysis required by a computational model.”

We take the explicit focus on *computationally-specified mechanistic processing models* that are both (i) theoretically informed and (ii) quantitatively fit to experimental data in a statistically informed and thoughtful way to be a distinguishing feature of research at the *semantics-psycholinguistics interface*. We distinguish this kind of work from experimentally-informed semantics, as well as from semantically-informed psycholinguistics.

In our view, work in experimentally-informed semantics engages primarily with semantic theories using empirical investigation methodologies (mostly offline: forced choice, acceptability etc.) that have become standard in psycholinguistics. But the semantic theories are connected to the experimental measurements of linguistic behavior only implicitly and/or informally, hence weakly. In addition, designing

properly powered experiments in semantics and pragmatics, where the effects are usually subtle and difficult to detect, is non-trivial,⁷ which makes the presumed links between theory and experimental data even more tenuous.

Similarly, work in semantically-informed psycholinguistics engages only in informal ways with formal semantics theories. While insightful, this work falls short of the standard of systematicity and formalization that permeates work in formal semantics, and does not engage in substantial ways with formal semantics frameworks and systems as a whole.

In sum, taking real-time experimental data and explicit computational modeling seriously opens up exciting new directions of research in formal semantics, and new ways of (re)connecting formal semantics and the broader field of cognitive science. This is the reason for the copious amounts of code and behavioral data introduced in this chapter, and in the book overall.

The specifics of the code and models we included in the book will likely become obsolete in the near future, just as many of our other auxiliary assumptions will. That is perfectly fine: their main purpose is to get the larger project off the ground and demonstrate its feasibility. Ultimately, our intention was to argue for a new range of theoretical and empirical goals for semantics, introduce an appropriate research workflow, and help semanticists and psycholinguists start using it.

9.6 Appendix: The Complete Syntax/Semantics Parser

All the code discussed in this chapter is available on GitHub as part of the repository <https://github.com/abrsvn/pyactr-book>. If you want to inspect it and run it, install `pyactr` (see Chap. 1), download the files and run them the same way as any other Python script.

9.6.1 *File ch9/parser_dm.py*

https://github.com/abrsvn/pyactr-book/blob/master/book-code/ch9/parser_dm.py.

9.6.2 *File ch9/parser_rules.py*

https://github.com/abrsvn/pyactr-book/blob/master/book-code/ch9/parser_rules.py.

⁷See Kruschke (2011), Vasisht and Nicenboim (2016), Nicenboim and Vasisht (2016) among others for detailed discussions of power and related statistical issues.

9.6.3 File *ch9/run_parser.py*

https://github.com/abrsvn/pyactr-book/blob/master/book-code/ch9/run_parser.py.

9.6.4 File *ch9/estimate_parser_parallel.py*

https://github.com/abrsvn/pyactr-book/blob/master/book-code/ch9/estimate_parser_parallel.py.

Open Access This chapter is licensed under the terms of the Creative Commons Attribution 4.0 International License (<http://creativecommons.org/licenses/by/4.0/>), which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons license and indicate if changes were made.

The images or other third party material in this chapter are included in the chapter's Creative Commons license, unless indicated otherwise in a credit line to the material. If material is not included in the chapter's Creative Commons license and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder.