1 From derivation trees to normalised dependency trees

(preamble: there are two terms which are **not** interchangeable. A DETERMINER is a word with specific syntactic properties. A SPECIFIER is an item with a specific semantic function. Most determiners function as specifiers most of the time, but they are not the same notion (I may not be as careful about using these terms as I ought to be)).

The grammar I am using is an HPSG/CG compromise. It's HPSG-like, but there are only two rules of combination: heads combine with their arguments, modifiers combine with their targets. The second of these two conflates two things – simple adjuncts, which copy a lot of features from the target to the result, and specifiers, which change the value of specified. **Some things do both**, which actually makes this quite a nice rule. What I'm going to do is to say that actually everything does both, but sometime the modifier bit is null, and sometimes the specifier bit is null. So I have one rule for everything, which is nice; but I have to do some tidying up afterwards to remve null bits, which is less nice. Swings vs. roundabouts.

We start with a simple example:

(1) He eats the ripe peaches.

Doing this gives us Fig. 1

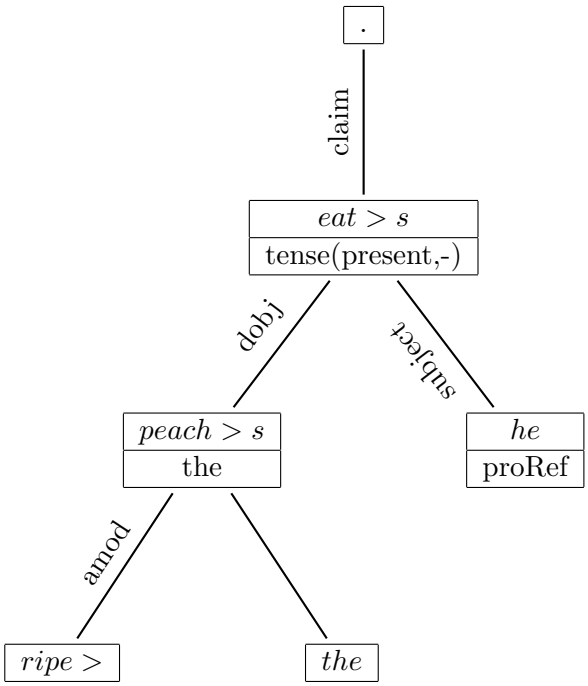

Figure 1: '*he eats the ripe peaches .*'

There are two things to note about this tree before we proceed:

- As noted above, some nodes carry a specifier, either because some lexical item provided it ('*the*' in '*the ripe peaches*') or because some sublexical item provided it ('*-s*' in '*eats*') or implicitly ('*he*' is specified because it's a full NP by itself). Specifiers are marked in the second line of the box.
- As also noted above, I don't really distinguish between specifiers and modifiers, because some things can be both, and some can sometimes be one and sometimes both. So I say that **every** specifier carries the potential for including a modifier. However, some specifiers, however, don't actually realise this potential. In such cases, the label that **would have** marked this item as a modifier is unlabelled. This leads to Fig. 2.

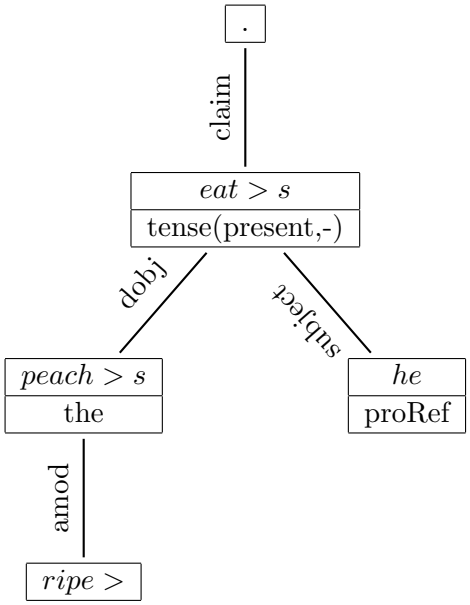

Figure 2: '*he eats the ripe peaches .*', null modifier deleted

The notion that trees may contain nodes that don't actually contribute anything may be extended to other phenomena. The auxiliary in (2) contributes a specifier, but it doesn't add anything to the propositional content. We therefore leave it without a label to its daughter (Fig. 3(a)), which means that it is not doing anything and hence can be deleted (Fig. 3(b)).

(2) he was eating the ripe peaches.

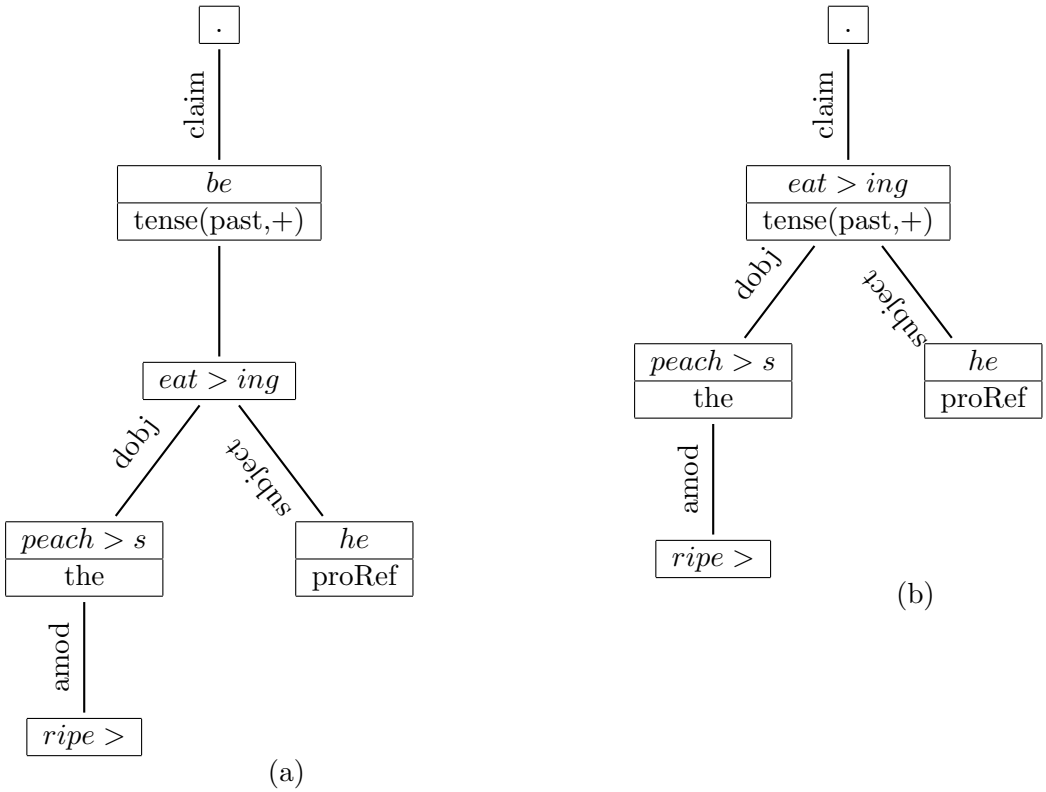

Figure 3: '*he was eating the ripe peaches .*'

We will **not** do this with modals. Modals have similar syntactic properties to auxiliaries, and they also contribute a specifier, but they do make a contribution to the propositional content. (3a) and (3b) do not mean the same thing.

(3) a. He might eat the unripe peaches .

b. He can eat the unripe peaches .

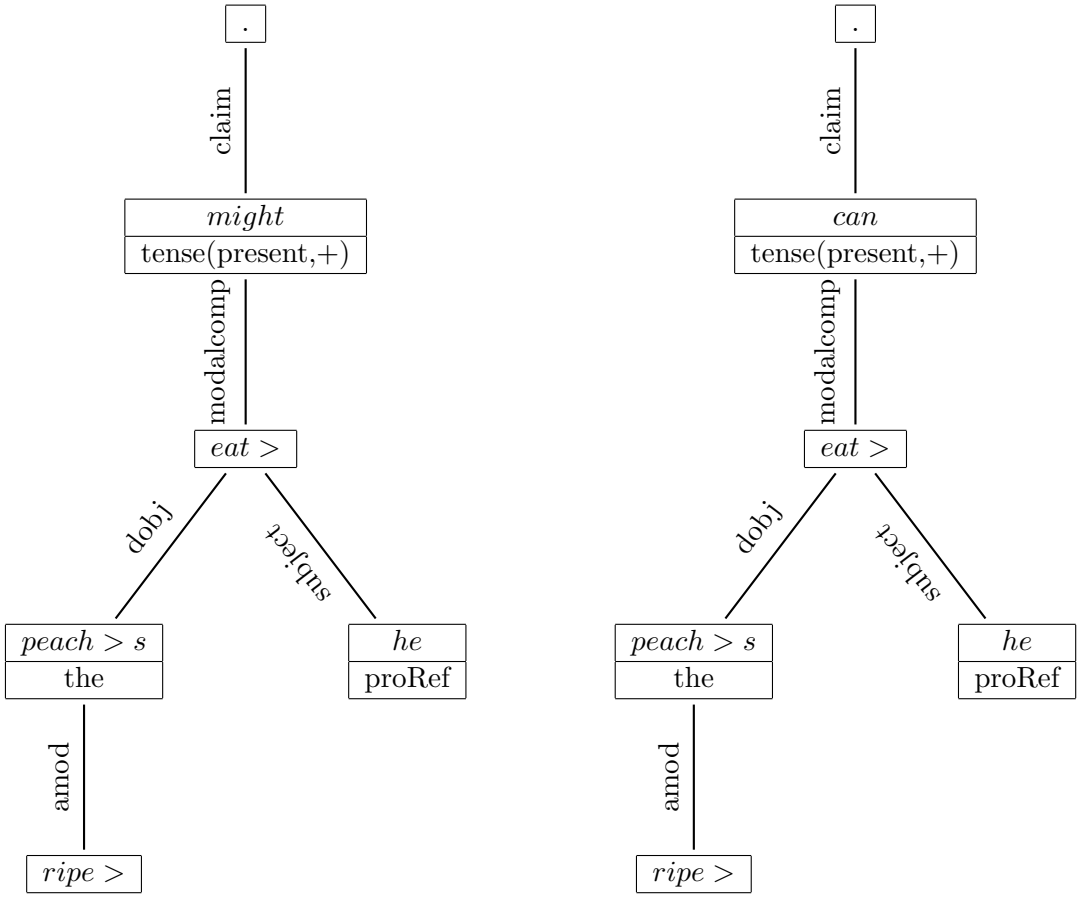

Figure 4: '*he might/can eat the ripe peaches .*'

We now return to determiners. The first complication arises when we look at numbers. We start with a simple case:

(4) He ate six ripe peaches.

'*six*' is a modifier which specifies the cardinality of the set of peaches, and also a specifier which marks the whole NP as indefinite. The outcome seems reasonable enough, even if the way I'm getting there is a bit unorthodox, since I'm treating '*six*' as a modifier **and** a specifier (Fig. 5).

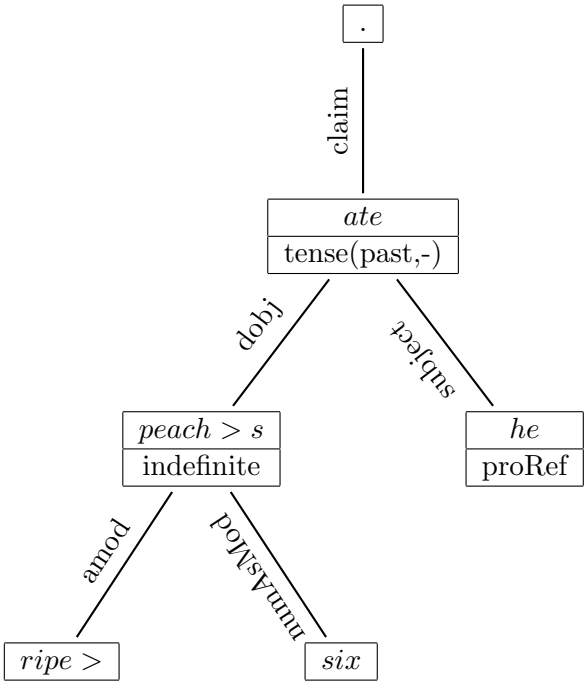

Figure 5: '*he ate six ripe peaches .*'

But we can combine other determiners with '*six*':

(5) a. he ate the six peaches.

b. he ate some six peaches.[1]

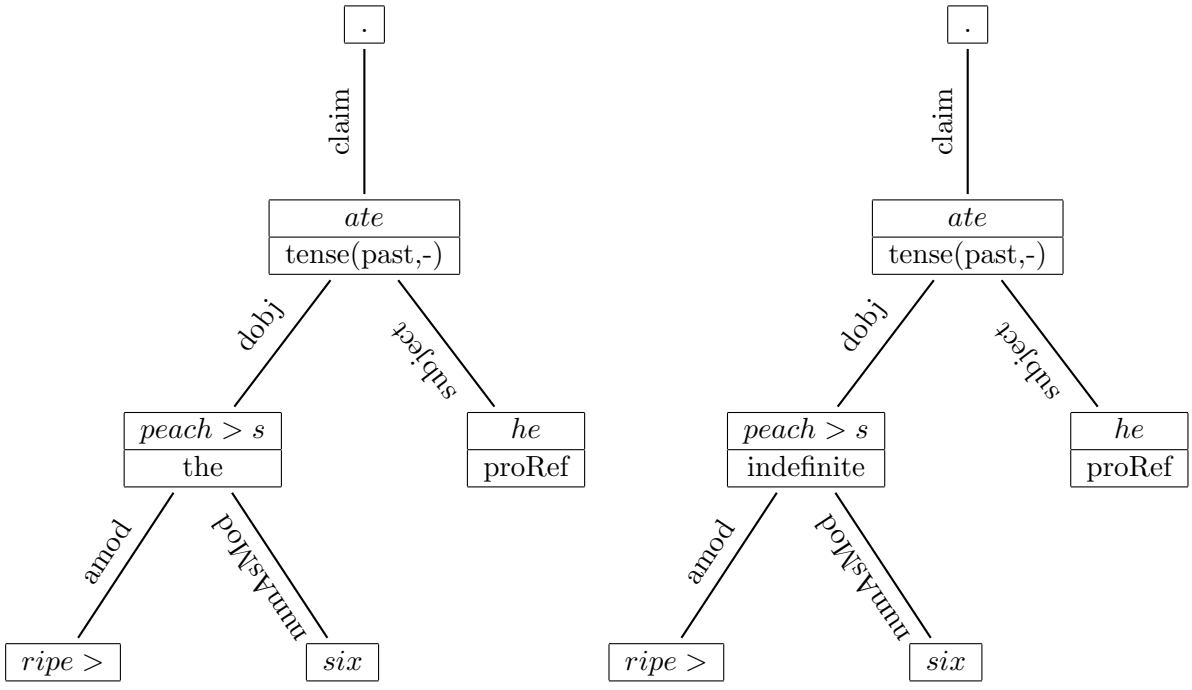

Figure 6: '*he ate the/some six ripe peaches .*'

This time '*six*' is just a modifier, fixing the cardinality of the set of peaches, and '*the*'/'*some*' are the specifiers. '*six*' isn't a specifier because '*six ripe peaches*' isn't `+specified`: a subtree only get to be specified if the external context demands it, and in the current examples this is not the case.

[1]This one is slightly mannered, but definitely sayable: '*Petipa studied and taught fur some six years in Madrid before returning*', `BNC/A/A1/A12`: around 400 examples in the BNC, often but not overwhelmingly with temporal PPs as here.

We also have to allow NPs with no determiner:

(6) a. He was eating ripe peaches .

b. He does not like unripe ones.

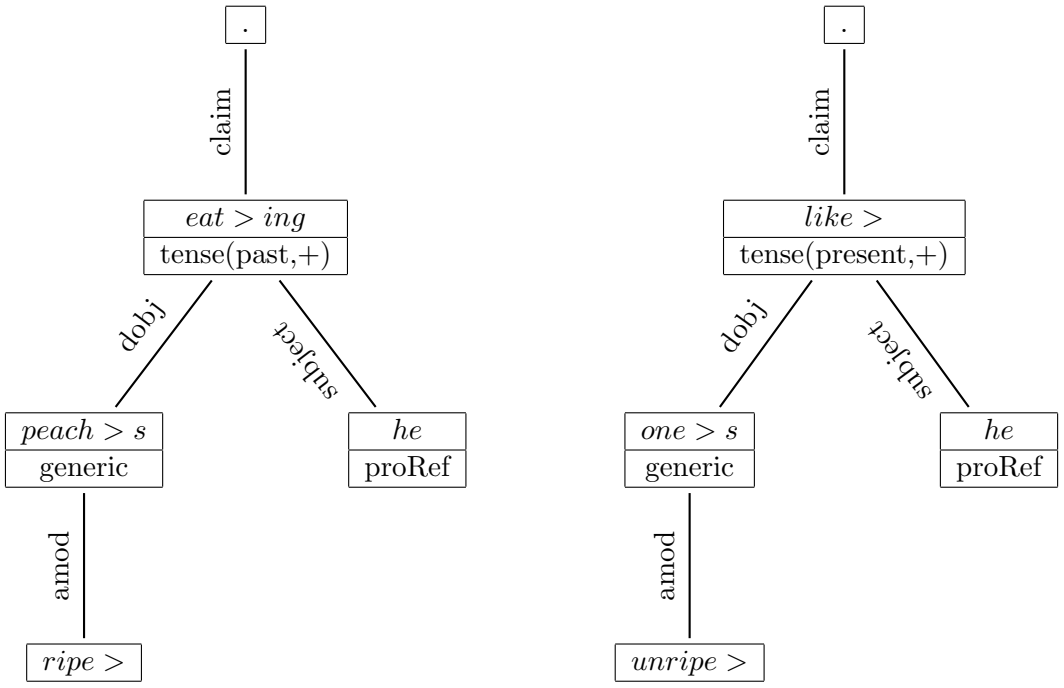

Figure 7: '*he does not like unripe ones .*'

I have not yet seen a treatment of bare plurals that I like. My preferred interpretation remains the one that I gave in (Ramsay 1992). For now I will just introduce a specifier called `generic`, and work out what it means later.

And then we get determiners that don't seem to need a noun to specify:

(7) a. He ate some.

b. He did not eat any.

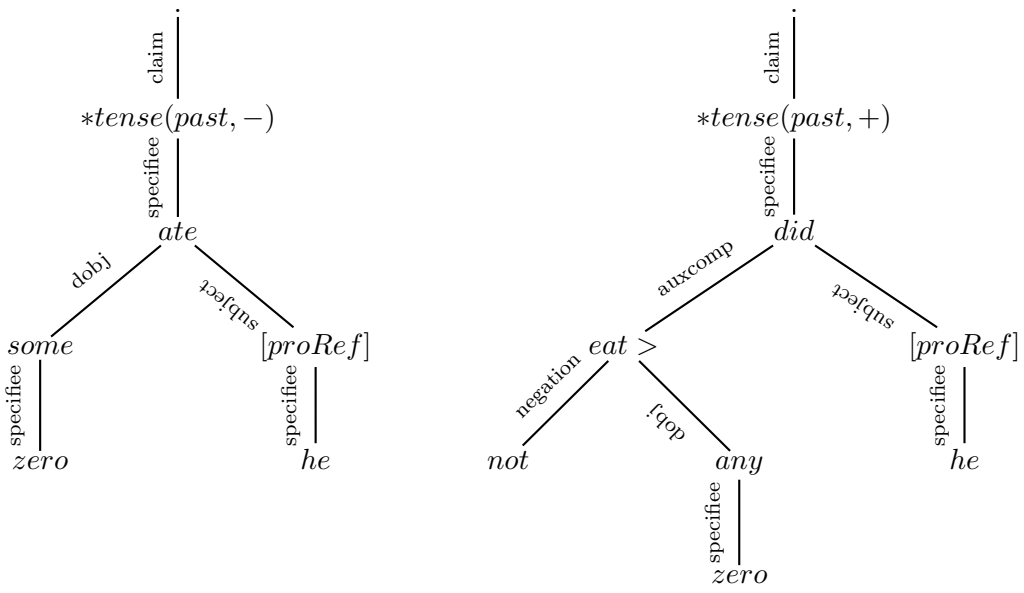

Figure 8: he ate some/he did not eat any .

This is a fairly common phenomenon – lots of determiners can take an elliptical head noun. Even more striking, they can take an elliptical head noun that has a post-modifier. Examples from the BNC – (a)-(c) have postmodifying PPs, (d)-(f) have postmodifying relative clauses:

(8) a. its peoples are very different from the Utopian fantasies of many in the developed world who profess a passionate concern for Amazonia

b. there are many in the world who still recognize it as their own

c. some in Germany may be tempted to seek reunification on the basis

d. not without some opposition from some who thought they were losing several days of their life

e. there can not have been many who doubted West Indies would win

f. they are vulnerable to the ordinary wear and tear of any who seek to serve among the wayward and weak

We just allow this, at least for some determiners Fig. 9 shows the analyses for a pair of sentences where one has '*which she had given him*' as a relative clause modifying '*peaches*' and one has it as a relative clause with no target noun.

(9) a. He was eating some peaches which she had given him.

b. He was eating some which she had given him.

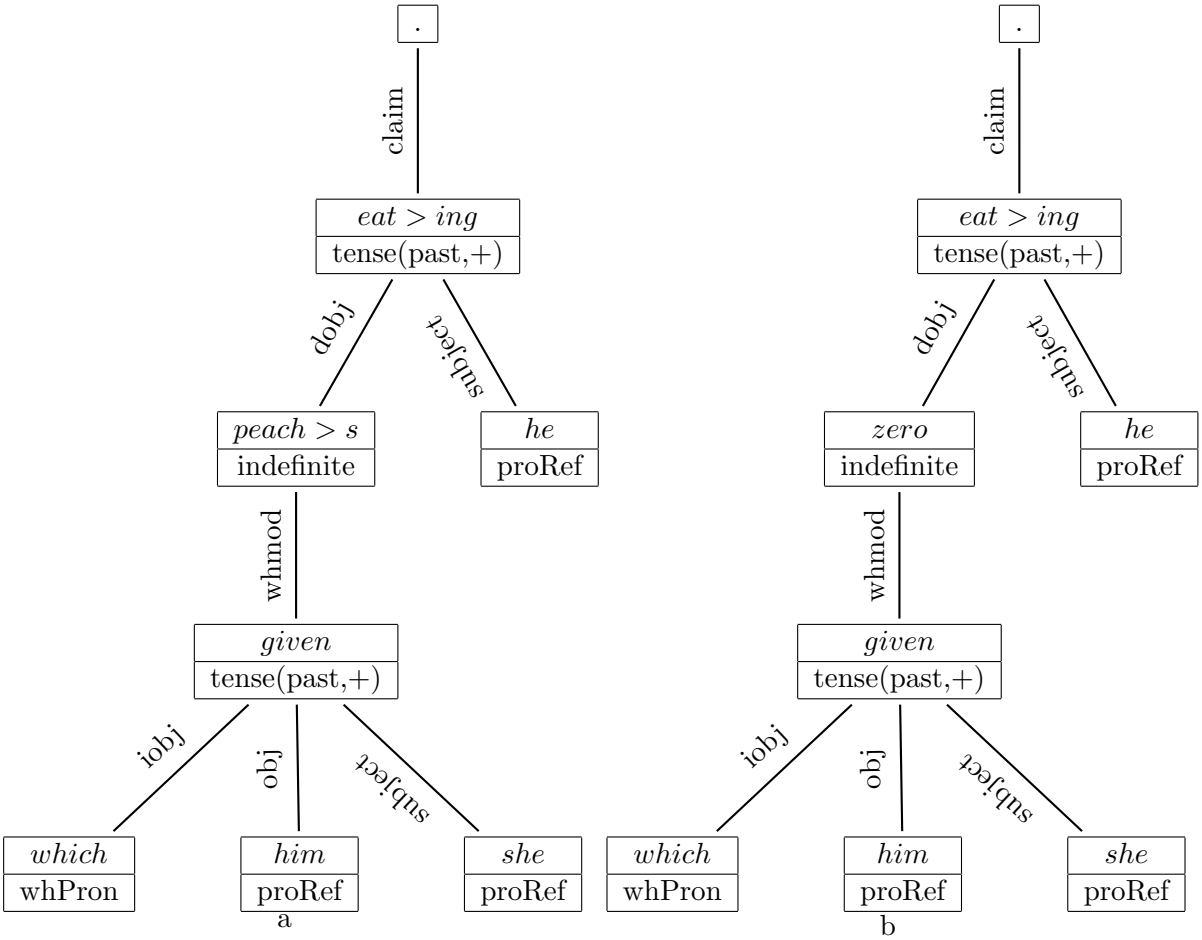

Figure 9: '*he was eating some which she had given him .*'

'*some*' should be a specifier: in (9)(b) there is no noun for it to specify. '*which she had given him*' should be a relative clause: in (9)(b) there is no noun for it to modify. There is no noun at the heart of the NP '*some which she had given him*'. In such cases, the noun is elliptical and would normally have be to determined with referece to the context – in the context of the examples so far, for instance, the zero noun in (9)(b) is almost certainly '*peaches*'!

The examples in (8) and (9) show that we have to, and can, allow for NPs with a determiner and a PP but no head noun. In particular, there are a big pile of determiners that can take '*of the ...*' (well actually '*of NP[+def]*'). Since we have had to allow for PPs with zero heads to in order to cope with (8)(a–f), we will simply use the same mechanism to allow for (10).

(10) a. He ate some of the ripe peaches.

b. He ate some of them.

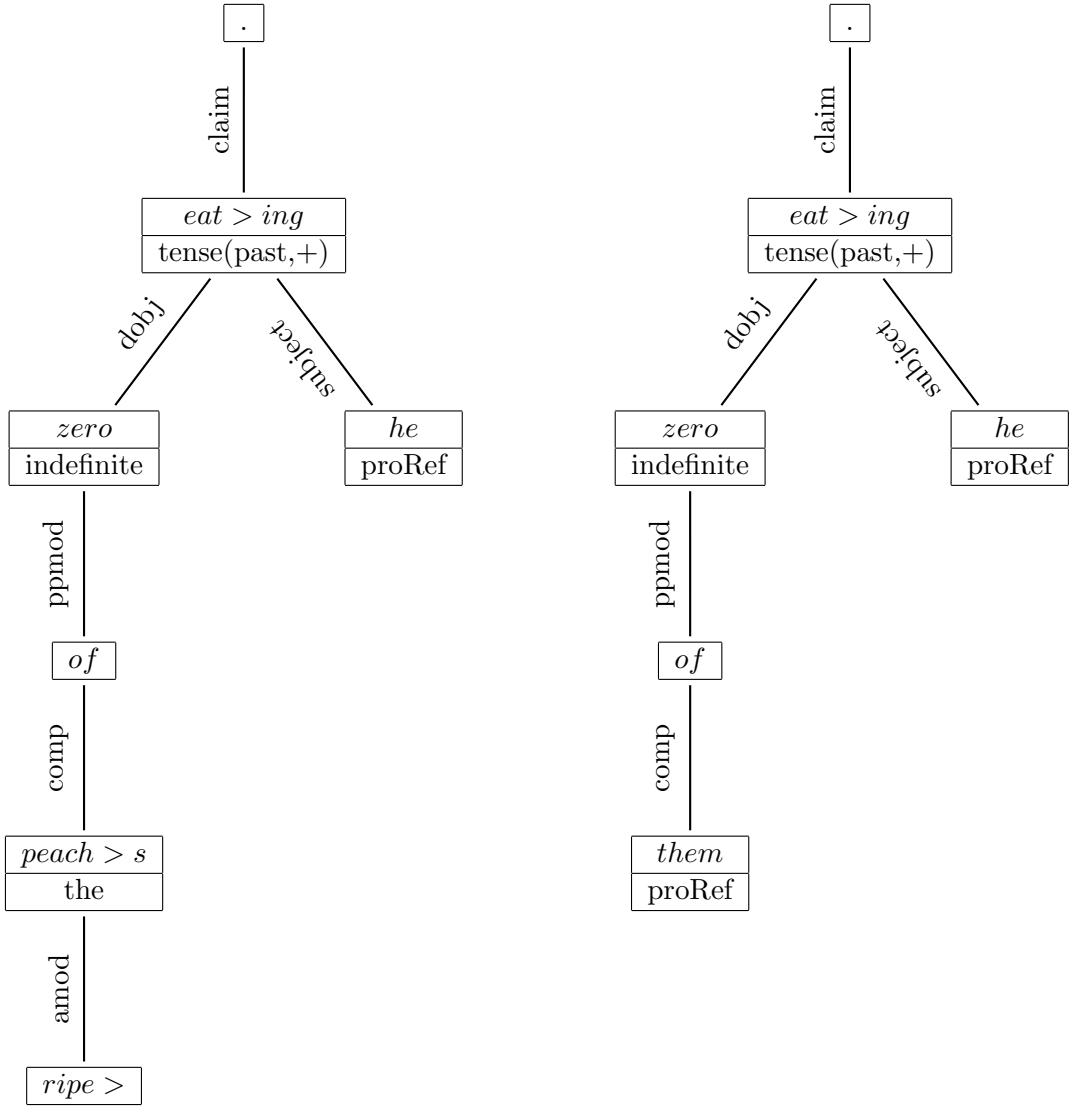

Figure 10: '*he was eating some of them .*'

As before, we have a zero noun, but in these cases this noun is some word like '*group*' or '*set*':

(11) a. He ate a bunch of the peaches.

b. He ate some of the peaches.

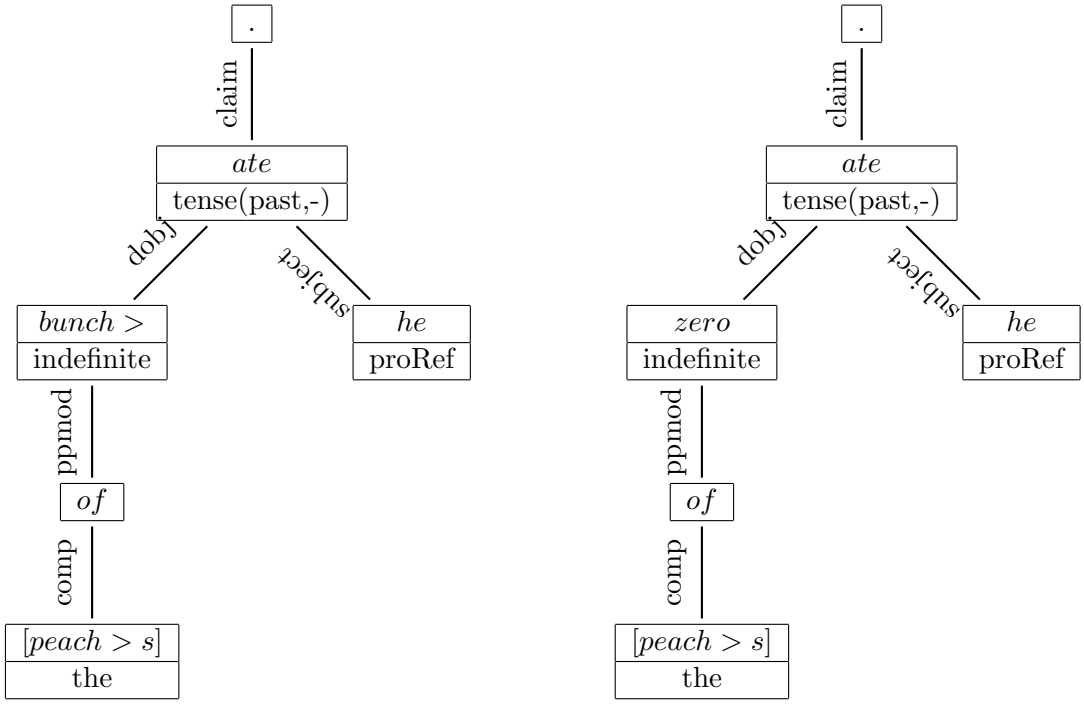

Figure 11: '*he ate some of the peaches .*'

'*all*' introduces a complication for which I do not have a solution. At first sight, it seems to behave like '*some*' – Fig. 12 assigns exactly parallel analyses to (12):

(12) a. He ate some of the peaches.

b. He ate all of the peaches.

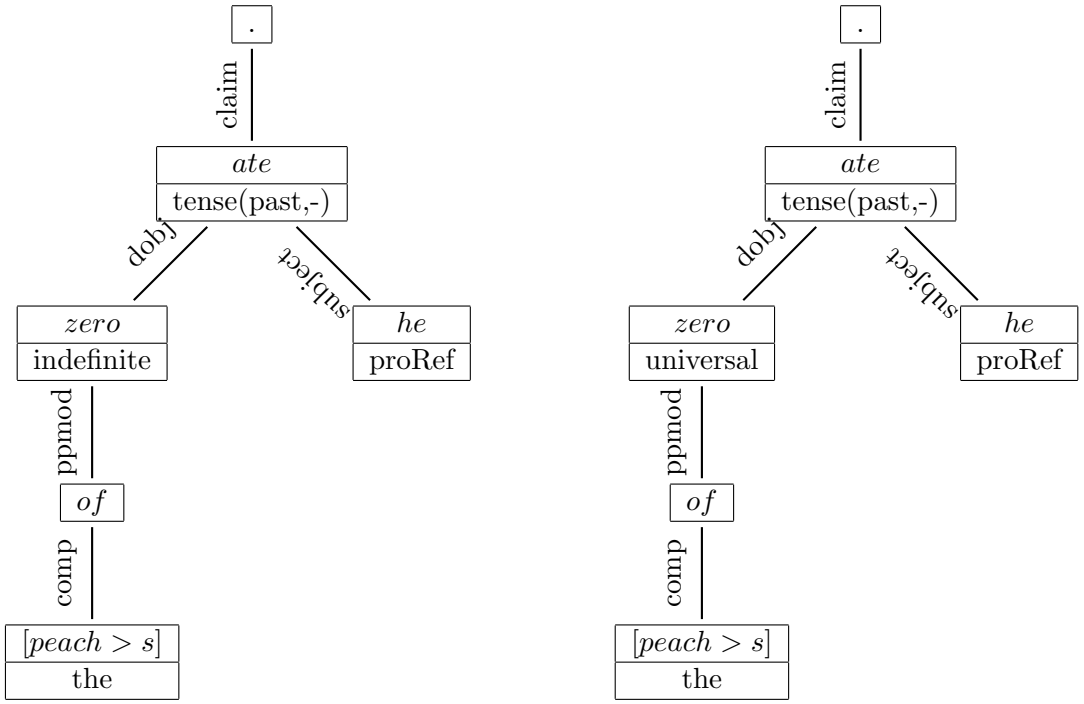

Figure 12: '*he ate all of the peaches .*'

But '*all*' takes part in constructions where (i) the preposition '*of*' can be omitted and (ii) '*all*' can float to a position after the definite NP (Appendix A contains examples of these constructions from the BNC). All the examples in (13) seem to mean the same thing, and it is very tempting to think that they all have the same underlying structure. I can't make that happen: I can give (13)(b) and (13)(c) the same structure, but I can't make that the same as the structure of (13)(b). the trees in Fig. 13 are the raw parse trees **before** we remove items with no function: the big

problem that I have (might seem like a small one, but it isn't), is that I cannot retain the specifier `*the` in the trees for (13)(b) and (13)(c) . We will return to this: right now it's a puzzle.

(13) a. All of the men were eating peaches.

b. All the men were eating peaches.

c. The men were all eating peaches.

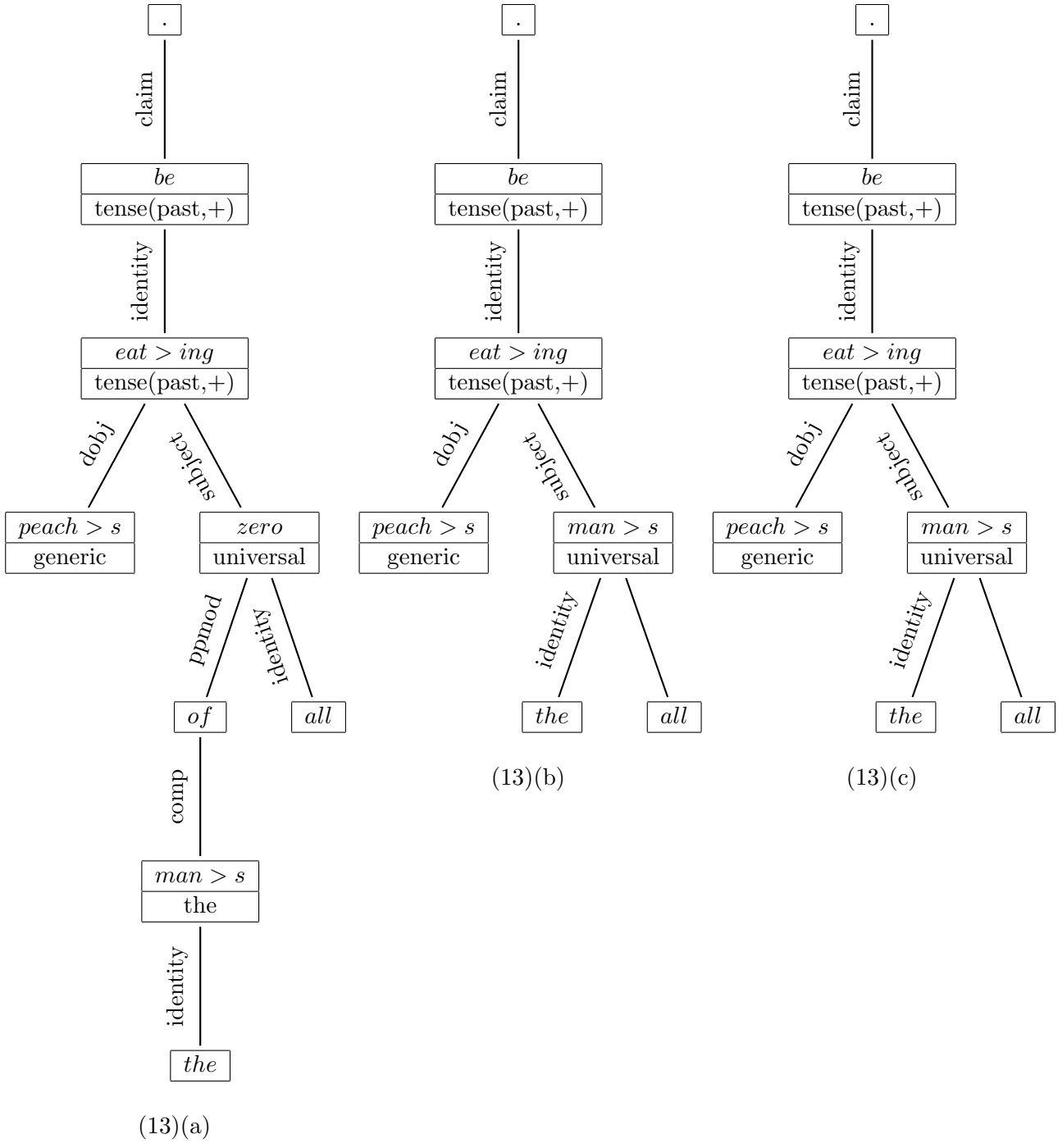

Figure 13: '*he ate all of the peaches .*'

We can now deal with a reasonably wide range of structures involving determiners, including ones with no determiner, ones with no noun, and ones with complex sequences of determiners such as (14):

(14) he ate all six of the peaches .

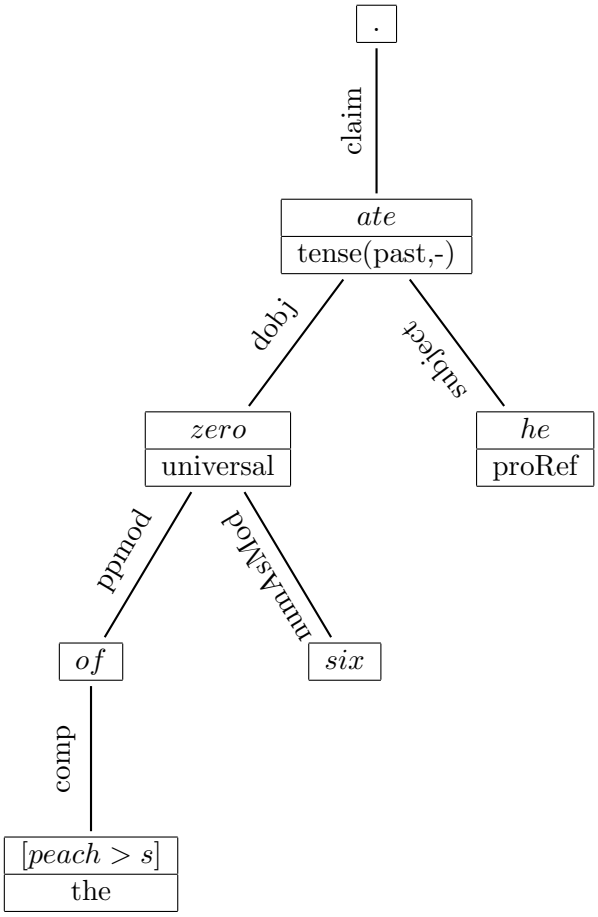

Figure 14: '*he ate all six of the peaches .*'

All these constructions are attested in the BNC, the parse trees we get for all of them seem sensible, and in most cases we can construct a normalised tree which does the right thing with the specifiers. We will return to the cases where the normalised tree is wrong later, but it now seems sensible to consider the interpretation of these trees.

2 What do specifiers do, anyway?

Why am I doing any of this? Because I want to do inference. I want to be able to do things like

(15)

a.

Some accountants are bookkeepers
All bookkeepers are crooks
Some accountants are crooks

b.

Most swans are white
Bruce is a swan
Bruce is probably white

c.

All accountants are bookkeepers
All bookkeepers are crooks
All crooks should go to jail
Bruce is an accountant
Bruce should go to jail

d. ...

2.1 Syllogistic reasoning

One way to proceed would be to construct a set of syllogistic patterns, and then try to string these together to make inference chains. Suppose we decided that (16) was an example of a sound pattern of inference. We could turn that into a rule by abstracting away the shared open class words.

(16) a.

All bookkeepers are crooks
Simon is a bookeeper
Simon is a crook

The trees for (16) are as in Fig. 15.

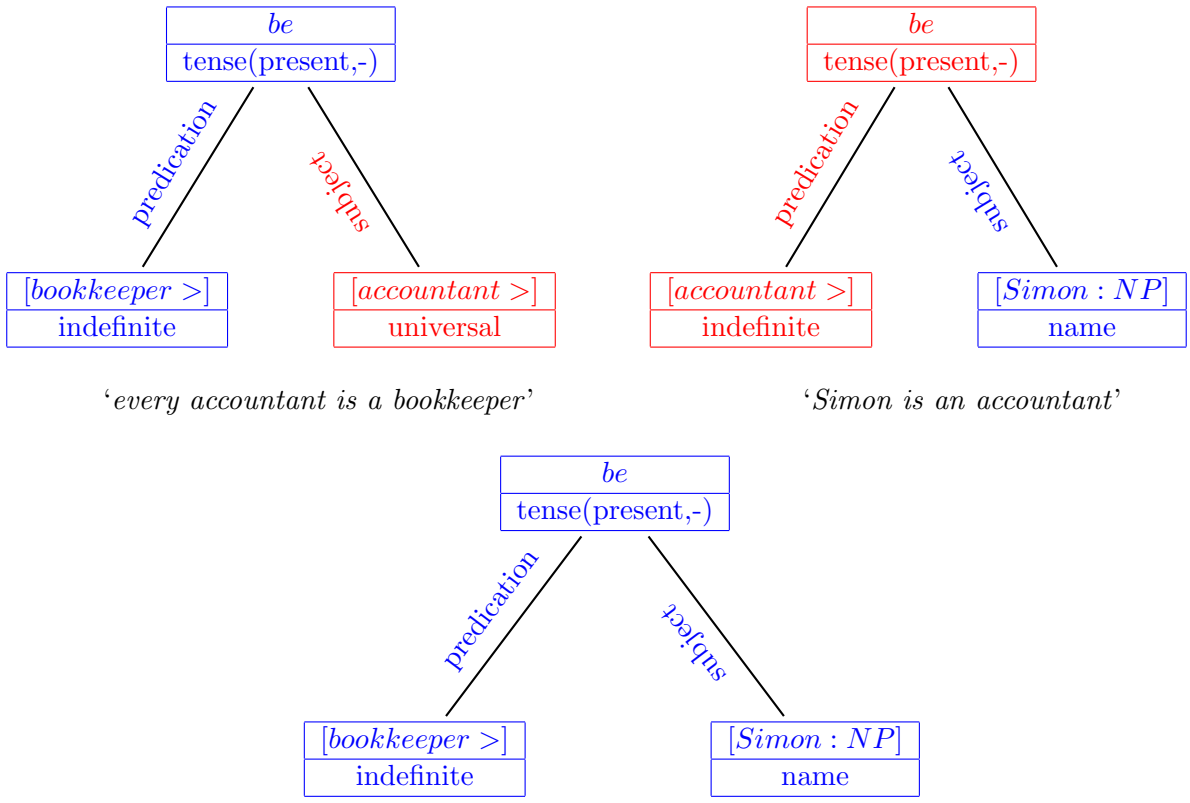

Figure 15: Trees for (16)

The red bits get cut, leaving the blue. Abstracting away the shared elements, this turns into

```
([[be, A], arg(subject, *(universal), [B, modifier(every)])]
  & ([[be, arg(D, E, [B, modifier(a)])], F]
      => [[be, A], F]))
```

Figure 16: Abstraction from Fig. 15: if every B is A and F is a B then F is A

This would be an interesting way to go. You could do this for all the syllogisms (including interesting ones like {'*Most bookkeepers are crooks*', '*Simon is a bookkeeper*' ⊢ '*Simon is probably a crook*'}), and then you could chain through applications of these patterns to solve a reasonably wide set of problems. Obtaining the patterns is easy – just parse the hypothesis and conclusion of a typical example and then abstract away the shared elements. Writing the engine is also easy – if you have sentences whose trees match the antecedent of some rule then you can infer its conclusion. I would be tempted to run them forwards whenever I could, because it feels as though that would be more effective, but I'd have to do some experimentation to see.

But I think it would be fairly inefficient, and I think it would feel rather *ad hoc*. So I'm going to go down a slightly different route. But I do think it would be an interesting thing to follow up. Maybe later.

2.2 Facts and rules

Most theorem provers for ordinary logic expect you to apply a series of NORMAL FORM rules which convert from (comparatively) readable formulae like

```
exists(A,
       woman(A)
       & forall(B, man(B)
                   => exists(C, event(C, love)
                                & theta(C, object, A)
                                & theta(C, agent, B)
                                & aspect(now, simple, C))))
```

to flatter forms (e.g. quantifier-free form, clausal form, ...) like

```
woman(#1)
man(B) => event(#2(B), love)
man(B) => theta(#2(B), object, #1)
man(B) => theta(#2(B), agent, B)
man(B) => aspect(now, simple, #2(B))
```

The flatter forms make it easier for the inference engine to match facts and rules: in the original standard form, the fact that this rule will let you conclude that there was a loving event is buried deep in the tree, whereas in the flattened form it is immediately visible and can be used for indexing and matching the rule. Typically the flatter forms introduce SKOLEM FUNCTIONS (including functions with no arguments, i.e. constants), i.e. new names for things that would otherwise be anonymous.

To do construct normal forms from dependency treees, I will start by splitting specifiers into SPECIFIC and GENERAL, and I will further split specific specifiers into INDEFINITE and REFERENTIAL. Things marked by specific specifiers are fact-like, and will contain terms; things marked by general ones are rule-like, and will contain variables and some kind of implicational marker.

We will start with a very simple example:

(17) Some accountants are crooks.

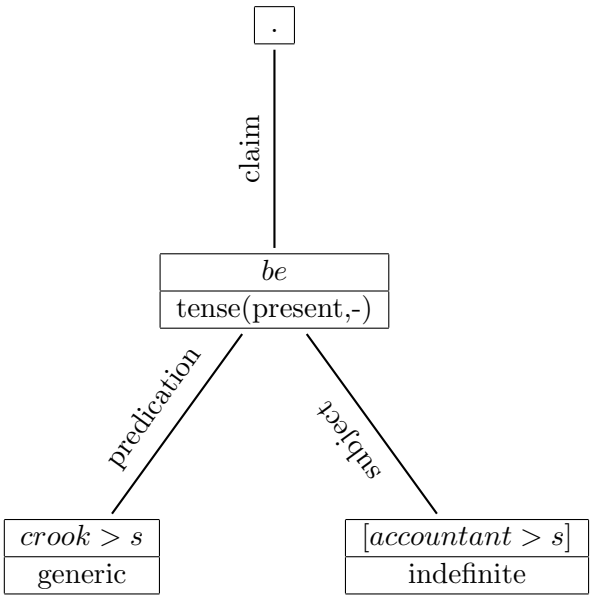

Figure 17: '*some accountants are crooks .*'

Even this very simple example raises two problems: what does the `*generic` quantifier do, and what does '*be*' mean? We will postpone consideration of generics for a while and consider an even simpler case:

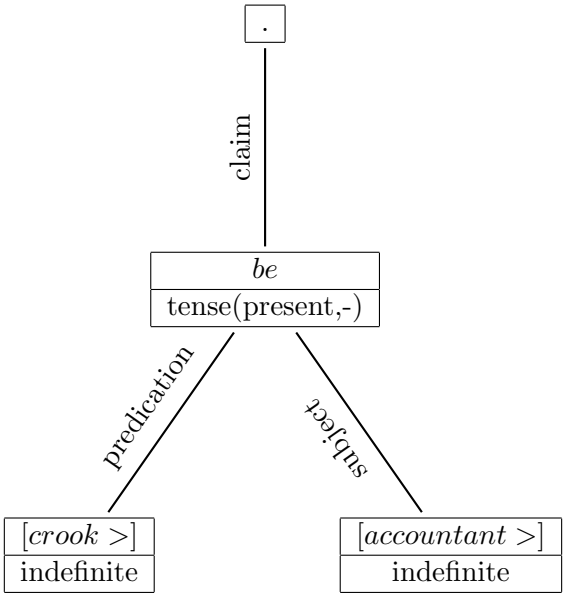

Figure 18: '*some accountant is a crook .*'

(18) Some accountant is a crook.

To get to where we want to be I'm going to do something that looks a lot like Skolemisation. OK, it **is** Skolemisation. To get to there we go through several stages. We start by replacing specifiers by IN-SITU QUANTIFIERS (Milward and Cooper 1994) (see (Cooper 1983; Keller 1987; Vestre 1991) for similar notions), allowing the tense marker to introduce the time at which the event in question occurred and a reification of the event (that's quite a lot for it do, but it seems fair enough. What the tense marker does is to introduce the time at which the event happened, i.e. introduces and a time and an event and relates them. We will later on want to add the aspect marker, but this will suffice for now): Fig. 19. We will refer to trees containing in-situ quantifiers as QUASI-LOGICAL FORMS (QLFs) (van Eijck and Alshawi 1992).

```
claim(qq(tense(-)::present,A,
         at(A,
            qq(indefinite::B,
               ([be,                    qq(indefinite::[crook],C,                    predi
```

Figure 19: QLF for '*some accountant is a crook .*'

(note that the in-situ quantifiers in Fig. 19 include a mixture of simple quantifiers (`indefinite::B`) and restricted quantifiers (`qq(indefinite::{[crook],C}`). There's nothing very mysterious about these)

We then extract the quantifiers, leaving the quantified variable in place, and apply them in order of precedence (e.g. giving '*any*' very wide scope): Fig. 20

```
exists(A::present,A,
       exists(B,
              exists(C::[accountant],C,
                     exists(D::[crook],D,
                            claim(at(A,
                                     ([be,                                    predicat
```

Figure 20: '*some accountant is a crook .*'

That's not too bad, but scope of the quantifiers seems to be too wide: Fig. 20 says that there is a present time, and there is an accountant, and there is a crook, and that at this time the accountant and the crook are in some relationship. It would seem better if the existence of the time and the crook and the accountant were all **inside** the claim, and indeed probably we would want the existence of the crook and the accountant to be fixed with respect to the time. We therefore allow

BLOCKING CONTEXTS, which restrict the extraction of quantifiers. Later one we will give weights to blocking contexts to specify which quantifiers can escape, but for now we will just set them to block all quantifiers. That gives us Fig. 21.

```
claim(exists(A::present,A,
             at(A,
                exists(B,
                       exists(C::[accountant],C,
                              exists(D::[crook],D,
                                     ([be,                                    predicat
```

Figure 21: '*some accountant is a crook .*'

We can then convert to quantifier-free form (QFF) (and do a little bit of tidying-up along the way, including Currying (I think) some combinations of terms and variables):Fig. 22

```
claim((present(#0)
        & at(#0,
             (accountant(#2)
               & (crook(#3)
                   & (be(#1)
                       & (predication(#1, #3, xbar(v(-), n(+)))
                           & subject(#1, #2))))))))
```

Figure 22: QFF for '*some accountant is a crook .*'

One last move and then we have this one sorted. '*Some accountant is a crook*' will be true if at the present time there is some entity which is an artist and some (other) entity which is a crook and these two are in fact the same thing.

This may feel oddish, but I think there are good reasons for taking it as the right way to go. Consider, firstly, copula sentences where both arguments are definite NPs.

(19) a. Allan is the one with an earring

b. The one with an earring is Allan

Both examples in (19) pick out two individuals and say they are the same. The only reasonable way to interpret these two is as something like Fig. 23 – there is something, `A`, which is called Allan, and there is something, `B`, which has an earring, and these two things are in fact the same thing. Seems about as reasonable as anything.

```
claim((now=#0
        & at(#0,
             (earring(#2)
               & ref(A,name(A,Allan))
                 = ref(B, (one(B) & modifier(B, ppmod, [with, comp(#2)])))))))
```

Figure 23: '*Allan is the one with an earring .*'

So for copula sentences where both arguments are definite NPs, we just want to say that the two things in question are the same. And, as we have seen in numerous examples, indefinite NPs are usually treated as involving something like existential quantification. If we maintain the view that copula sentences with NPs as their objects assert identity, and that indefinite NPs introduce entities into the conversation, then for

(20) John is a fool.

we get

```
claim((now=#0
        & at(#0, fool(#2)&ref(A,name(A,John),[])=#2)))
```

Figure 24: '*John is a fool .*'

What are the consequences of Fig. 24? Suppose we had a rule `fool(X) => P(X)` for some property `P`. Then if the person who satisfies `ref(A, name(A, John))` is `#36` then `P(#36)` will hold. That is exactly what we want – to be a fool is to satisfy all the properties that fools satisfy, and that is what we will get from Fig. 24. In other words, we can get the right consequences of copula sentences with NP predicates by saying that such sentences say that their subject and predicate are identical, without forcing a non-standard interpretation of indefinite NPs. That means that when I come to do inference I will need to think about equality, but that's not going to be the most challenging thing that I face.

Returning to our syllogism, we obtain Fig. 25 for (18).

```
claim((#0=now & at(#0, accountant(#2)&crook(#3)&(#3=#2))))
```

Figure 25: '*some accountant is a crook .*'

OK, '*Some accountant is a crook.*' means '*I am claiming that at a time which happens to be* `now` *there is something which we will call* `#2` *which is an accountant and something which we will call* `#3` *which is a crook and that these two things are actually identical*'. That's not an incorrect paraphrase.

Similarly, '*every accountant is a bookkeeper*' comes out as Fig. 26. So if we can deal with equality we'll be in business.

```
claim((#0=now & at(#0, accountant(A)=>(bookkeeper(#2(A))&(#2(A)=A)))))
```

Figure 26: '*every accountant is a bookkeeper .*'

'*I am claiming that at the moment if you show me an accountant I will find you a bookeeper who is in fact the same person*'

What shall we do about equality? In the antecedent of a rule, it means that we have to be able to prove that two possibly distinct terms denote the same thing, which could be tricky if they are complex terms – if you know that `#1` and `#2` are the same entity and that `#3(X)` and `#4(X)` are the same functions, you're still going to have to do some work to determine that `#3(#1)` and `#4(#2)` are different names for the same thing. But we can deal with equality in the consequent of a rule, or in a fact, simply by merging the two names. Fig. 25 and Fig. 27(a) will be true under exactly the same circumstances, and likewise Fig. 26 and Fig. 27(b). So we can eliminate equality in positive contexts just by merging the two terms on either side of the equality.

```
claim(at(now, accountant(#2)&crook(#2)))
```

(a) some accountant is a crook

```
claim(at(now, accountant(A)=>bookkeeper(A)))
```

(b) every accountant is a bookkeeper

Figure 27: Eliminating equality in positive contexts

2.3 Queries and Proofs

So far we have looked at simple declarative sentences. We also need to handle queries. This is fairly straightforward: where a declarative sentence has `claim(...)` wrapped round it by the full stop, a question as `query(...)` wrapped round it by the question mark. The QLF for a query is thus very similar to that for the corresponding statement:

(21) Is some accountant a crook?

```
query(opaque(qq(tense(-)::present,A,
                at(A,
                   opaque(qq(indefinite::B,
                             ([be,                                    qq(indefinite::[crook],C
```

Figure 28: QLF for '*is some accountant a crook ?*'

When we convert this to quantifier-free form, we have to note that questions place their propositional contents inside negative contexts, so we have to be carefull when removing the quantifiers. What we want as the QFF for (21) has the indefinite specifiers replaced by place-holder variables:

```
query(at(now, accountant(A)&crook(A)))
```

Figure 29: QFF for '*is some accountant a crook ?*'

Fig. 29 says that I would like to know whether there is something which is currently both an accountant and a crook. Seems like a fair enough interpretation of '*Is some accountant a crook?*'.

3 Inference

We now have representations of sentences expressing facts, rules and questions. We want to be able to answer the questions by using the facts and rules, in other words we want an inference engine that will operate over these representations. There is a wide variety of mechanisms for carrying out inference – resolution, tableaux, ... Because we are going to want to do intensional reasoning (reasoning in a framework that allows quantification over properties and propositions) we want something which we are confident can be adapted for such tasks: we will therefore use Ramsay (2001)'s version of the SATCHMO inference engine (Manthey and Bry 1988; Loveland et al. 1995) which we have already used for reasoning with a fully intensional logic, namely property theory (Ramsay 1995; Cryan and Ramsay 1997; Turner 1987).

We start with a simple constructive version of SATCHMO (Fig. 30) which embodies the standard natural deduction operators for constructive logic. We assume that `not(P)` is a shorthand for `P => absurd`: there are no rules explicitly about negation below because anything that we want to say about negation will be captured in the rules relating to `=>`. There are a large number of optimisations (Loveland 1991; Loveland et al. 1995; Ramsay 1991) that can be added to this skeletal version of the algorithm, but we omit these here for clarity.

```
prove(P, LABEL) :-
    fact(P).
```

```
%% &-elimination
prove(P, LABEL) :-
    prove(P & Q, LABEL)

%% or-elimination
prove(P, LABEL) :-
    prove(P or Q, label),
    prove(Q => absurd).

%% =>-elimination, modus ponens
prove(P, LABEL) :-
    A => P,
    prove(A, LABEL).
```

```
%% &-introduction
prove(P & Q, LABEL) :-
    prove(P, LABEL), prove(Q, LABEL).

%% or-introduction
prove(P or Q, LABEL) :-
    prove(P, LABEL); prove(Q, LABEL).

%% =>-introduction (conditional proof)
prove(P => Q, LABEL) :-
    assert(P),
    (prove(Q, LABEL) ->
        retract(P);
        (retract(P), fail)).
```

Figure 30: Basic implementation of constructive SATCHMO

Why use SATCHMO? We are going to want do a variety of rather complicated things – reasoning with defaults, with epistemic & temporal contexts, with fine-grained intensionality, with asymmetric approximate unification, ... So we will accrete extra complexity and complication to whatever engine we start with. Starting with something very simple will make it easier to add all this extra material without becoming swamped by it. The basic implementation of SATCHMO is startlingly simple – Fig. 30 shows the whole thing. This makes it an attractive starting point for adding non-standard facilities to.

3.1 Unification or subsumption?

The version of SATCHMO presented in Fig. 30 used unification, as most theorem provers do, for matching goals to the consequents of rules. There are, however, circumstances under which other matching algorithms may be useful.

- It may be useful to exploit subsumption relations between terms: if we have a hierarchy of terms that includes, for instance, that `man` $\subseteq$`human` (e.g. via WordNet) then it may be better to note that `man(X)` $\subseteq$`human(X)` when matching rules than to have a rule that says `forall(X, man(X) => human(X))`
- Given that the terms in our representations are dependency trees, it may be appropriate to use a partial matching algorithm which allows, for instance, modifiers to be ignored so that the term corresponding to '*a man with a telescope*' in '*A man with a telescope saw me*' can be matched with the term for '*a man*' in '*A man saw me*'.

We therefore replace the original presentation of SATCHMO with Fig. 31. Using subsumption instead of unification does make it more difficult to index rules, since it is no longer clear when a fact or rule will be useful in a proof – given the fact `man(John)` and the rule `human(X) => mother(#1(X), X)`, it is not immediately evident that this rule will let you infer that John has a mother. We therefore have to be careful about how we store rules. Nonetheless, using subsumption rather than unification has the potential to unlock resources such as WordNet, and also to add extra facilities such as partial matching of subtrees.

```
                        prove(P, LABEL) :-
                            fact(P'), P' ⊆ P.

%% &-elimination                        %% &-introduction
prove(P, LABEL) :-                     prove(P & Q, LABEL) :-
    prove(P' & Q, LABEL), P' ⊆ P.          prove(P', LABEL), P' ⊆ P,
                                           prove(Q', LABEL), Q' ⊆ Q.

%% or-elimination                       %% or-introduction
prove(P, LABEL) :-                     prove(P or Q, LABEL) :-
    prove(P' or Q, label), P' ⊆ P.         (prove(P', LABEL), P' ⊆ P);
    prove(Q => absurd).                    (prove(Q', LABEL), Q' ⊆ Q).


%% =>-elimination, modus ponens        %% =>-introduction (conditional proof)
prove(P, LABEL) :-                     prove(P => Q, LABEL) :-
    A => P', P' ⊆ P.                       assert(P),
    prove(A, LABEL).                       (prove(Q, LABEL) ->
                                               retract(P);
                                               (retract(P), fail)).
```

Figure 31: Constructive SATCHMO with subsumption

3.2 Backwards or forwards?

You can run an inference engine backwards (to prove `P`, find a rule `P1 & P2 & ...& Pn => P` and try to prove each `Pi`) or forwards (if you have a rule `P1 & P2 & ...& Pn => P` and you know that each of the `Pi` holds then you can add `P` to what you know). Theorem provers are run largely backwards, but it is reasonable to suggest that people actually do a degree of forward inference. Consider the examples in (22):

(22) a. My father was born on the moon.

b. All cats have wings.

c. I have an infinite number of friends.

All these sentences are false. What is more, they are patently, obviously, blatantly false, and anyone reading them will immediately realise this.

This suggests that you do not just passively listen to what you are told and store it away for future reference, to be used if it would help you solve some problem (e.g. answer a question). You actively assimilate it. There are good reasons for this. When people tell you things, it is usually because they think that you will do something with what they have told you. If they shout '*Fire!*', they expect you to leave your current location and go somewhere safer. If they say '*I could murder a cup of tea*', they expect you to go and make them a cup of tea. Whatever it is they expect you to do, they generally expect you to do it now. So you should think about what you have just been told, and about how it fits into the current situation, because the odds are that you should do something about it.

The activity of assimilating what you have just been told is often referred to as MODEL BUILDING (Bos 2001). I am going to talk about it in terms of forward inference – technically there is little difference between the two (indeed, the standard presentation of SATCHMO, like many other theorem provers, proceeds by attempting to build a model of the premises and the negation of the goal and hoping that this will turn out to be impossible).

Forward inference is, in general, a very open-ended task. Given the premises {`human(J)`, `forall(X :: {human(X)}, exists(Y, human(Y) & mother(Y, X)))`} there are an infinite number of (non-tautologous) conclusions that can be drawn, the effect that J's mother is human, and J's mother's mother is human, and J's mother's mother's mother is human, and ... Assimilating what you have just been told involves a bit of forward inference, but not an unbounded amount.

You could, perhaps, do some kind of unconvincing Johnson-Laird-ish kinds of pyschological experiments to try to determine just how much; or you could inpose some aribtrary restrictions, safe in the knowledge that you can always fall back on doing backward inference to solve complex problems as and when you need to. For simplicity, I will apply the following strategy:

- Add any ground facts that can be added using Horn clauses without introducing Skolem functions (the only functions we have are Skolem functions, so this blocks cases like the one above about J's mother and J's mother's mother). It may be that in certain situations we should block Skolem functions that are nested to a specified depth, rather than blocking all such functions.
- Add anything that can be added by application of a rule with an intensional consequent. The key here is that it is almost impossible to use rules with intensional consequents backwards, because the consequent will be at best vaguely described and hence it will be difficult to work out whether it is relevant to a specific goal. To take a simple case, consider (23):

  (23) She managed to finish the book she was writing.

  One simple consequence of (23) is that she did indeed finish writing it – '*manage*' is a factive verb which asserts, among other things, the truth of the embedded clause. The obvious way to capture this is with a rule like Fig. 32.

  ```
  manage(X) & xcomp(X, Y) & subject(X, Z) => (Y:Z)
  ```

  Figure 32: If Z managed to Y then Z did Y

  Allowing this rule to be used backwards would mean that you would try to use it every time you wanted to prove anything. Using it forwards means that it will be triggered only when you are trying to assimilate a sentence about someone managing to do something (Ramsay 2001).
- Add anything that can be added by application of a default rule, so long as there is no **current** reason not to. The evidence for this lies in the double-take that is induced by sentences like (24)(b)

  (24) a. If a farmer owns a donkey he beats it.
  b. If a farmer owns a donkey she looks after it very nicely.

  It is perfectly possible for farmers to be women. Indeed, in 2012 45% of farmers in Arizona were women[2]. Nonetheless,(24)(b) seems odd because most people's preconceptions (= default rules) suggest that farmers are men, so that by the time you have assimilated '*a farmer owns a donkey*' you will already have visualised a male farmer, and hence have difficulty dereferencing '*she*'.
- If you can prove the antecedent of a rule with a disjunctive head and all but one of the disjuncts is provably false then add the remaining one (subject to the same restriction on Skolem functions as above).

These rules will suffice to produce a set of immediate ground consequences/partial model of an utterance given a context and a set of background rules; and, given suitable background knowledge, to make it possible to detect the impossibility of examples like (22) as soon as they are produced.

3.3 Contexts

We now extend our representations to enable us to talk about CONTEXTS – about, for instance, temporal contexts such as the fact that our interpretation '*Some accountant is a bookkeeper*' says

[2] `https://www.agcensus.usda.gov/Publications/2012/Online_Resources/Highlights/Women_Farmers/Highlights_Women_Fa`

that `accountant(#2) & crook(#2)` holds at 'now'. We will deal with this and other similar relations by adding to every formula a specification of the (possibly nested) context(s) in which it is available. We will do this by adding to every formula a LABEL (Gabbay 1989, 1996), i.e. a bundle of information about that formula[3]. We will return to this in Section **??**. The key notion for now is that we allow a formula to be annotated with a specification of the context(s) in which it is available, writing `P∈C` to mean that `P` is available in the context `C`. Contexts can be temporal (the instant at which something is true, the interval during which it is true) or epistemic (a proposition may be available in someone's belief set or in the set of things they know) or ... They do **not** in general distribute over logical connectives – you cannot assume that `(P & Q)∈C` = `((P∈C) & (Q∈C))` or `(P or Q)∈C` = `((P∈C) or Q∈C)`. We will however assume that they nest – that `(P∈C1)∈C2` = `P∈(C1+C2)`, i.e. if the fact that `P` is available in `C1` is available in `C2` then `P` is available in `C1+C2`, and we will always use this rule to flatten nested contexts.

We use this notion to turn Fig. 29 into Fig. 33(a) and likewise Fig. 27(b) becomes Fig. 33(b).

```
(accountant(A)∈[now,believes(hearer)] & crook(A)∈[now,believes(hearer)])
```

(a) is some accountant a crook?

```
(accountant(A)=>bookkeeper(A))∈[now,believes(speaker)]
```

(b) every accountant is a bookkeeper.

Figure 33: Contexts added as labels: '*is some accountant a crook?*'/'*every accountant is a bookkeeper .*'

Fig. 33(a) asks whether the information available to the hearer (who is the person who is being asked the question, and hence is the person who should attempt to use their beliefs to find an answer) supports a proof that there is someone who is currently an accountant and is also currently a crook; Fig. 33(b) says that the speaker, who is the person making the claim, believes that they have access to a rule that says that is currently the case that if someone is an accountant then they are also a bookkeeper (note the label applies to the entire rule).

We now consider what happens in a dialogue with two participants, S and H:

(25) S: every accountant is a bookkeeper.

H: OK

S: every bookkeeper is a crook.

H: OK

S: some man is an accountant.

H: OK

S: is some man a crook?

H: Hmmm, don't ask me, haven't a clue, how on earth would I know?

After each of S's statements, H says '*OK*'. Why does she do this? Because all that the statements say is that S believes something, and there is no necessary requirement for H to believe something just because S does. There are Grice-ish reasons to expect that H will believe what S says, but unless she says that she has accepted S's statements then S cannot be sure that she has. So she says '*OK*' to indicate that she has, at least tentatively, accepted what she has been told. We therefore construct a model of H's beliefs at this point that looks like Fig. 34
The context says that is currently true that H believes that it is currently true that there is someone, who might as well call `#2`, who is a man and is also an accountant, and likewise for the two rules.

So by the time S asks his final question, H should have the information she needs to answer it, and (again by some Grice-ish rules) she should try to do so (well actually by a proper Grice-ish set of rules she should say '*Well obviously, why are you asking me?*'): Fig. 35

[3]carried in the second argument of `prove` in Fig. 30: this will eventually turn out to include all sorts of useful things.

```
man(#2)∈[now,bel(hearer),now]
accountant(#2)∈[now,bel(hearer),now]
accountant(A)=>bookkeeper(A)∈[now,bel(hearer),now]
bookkeeper(A)=>crook(A)∈[now,bel(hearer),now]
```

Figure 34: H's beliefs after the first part of (25)

```
Trying to prove man(_158190)&crook(_158190) in [now,bel(hearer),now]
  Trying to prove man(_158190) in [now,bel(hearer),now]
  Found man(#2) in [now,bel(hearer),now]
  Trying to prove crook(#2) in [now,bel(hearer),now]
  Using bookkeeper(#2)=>crook(#2) to prove crook(#2) in [now,bel(hearer),now]
    Trying to prove bookkeeper(#2) in [now,bel(hearer),now]
    Using accountant(#2)=>bookkeeper(#2) to prove bookkeeper(#2) in [now,bel(hearer),now]
      Trying to prove accountant(#2) in [now,bel(hearer),now]
      Found accountant(#2) in [now,bel(hearer),now]
```

Figure 35: H can prove there is a crook

Straightforward enough: just a backward chaining proof with a label that includes the context where things can be proved, with the same context for the facts that H can use and the query she is trying to answer. Various things will get more complicated from here on, but that's our basic engine. If someone tells you something, then if you're prepared to accept it you should say '*OK*' and add it to your own belief set; if they ask you a question you should try to answer it. There are much more complicated things that speakers can do (tell jokes, be sarcastic, lie, ...) and much more complicated things that hearers can do (believe what they are told, say they don't believe it, pretend to believe it, ...) which we have discussed elsewhere (Ramsay and Field 2006a, b, 2008, 2009); but this simple model will suffice for now.

Given that our representations now include contexts, we now need to modify the specification of SATCHMO to allow for this. We will write `C1 << C2` to mean that anything that is available in C2 is available in C1. Thus, since anything which anyone knows in a context must be true in that context we can say `C << [know(#1) | C]` for an arbitrary context `C`. This version of SATCHMO is given in Fig. 36. A number of the rules now depend on properties of the context, since not all contexts support all the standard introduction and elimination rules (e.g. consider `A` and `B` where `B = not(B)`: then `poss(A) & poss(B)` $\not\vdash$ `poss(A & B)`, `nec(A or B)` $\not\vdash$ `nec(A) or nec(B)`). We therefore have to attach side-conditions to the various rules which indicate the kinds of contexts where they apply.

The first three rules of Fig. 36 just say that you can prove something in a context `C` if you can prove it in `C'` where anything which is available in `C'` is available in `C`. The fourth just unwraps `not(P)` in the standard manner of constructive logic. The last rule says that you can prove `P => Q` in `C` by assuming `P` is available in `C` and trying to prove `Q` in the same context. This is not, however, always legitimate – the argument above showed that in general `(P => Q)∈C` entails `P∈Q => Q∈C`. We do not, however, in general have the converse. We will refer to contexts in which `P∈Q => Q∈C` entails `(P => Q)∈C` as CLOSED contexts. Conditional proofs of the kind embodied by the final rule in Fig. 36 can only be carried out in contexts which are known to be closed.

```
prove(P ∈ C, LABEL) :-
    fact(P' ∈ C'), P' ⊆ P, C << C'.
```

```
%% &-elimination
prove(P ∈ C, LABEL) :-
    prove((P' & Q) ∈ C, LABEL),
    P' ⊆ P, C << C'.

%% or-elimination
prove(P ∈ C, LABEL) :-
    prove((P' or Q) ∈ C', label),
    C << C', P' ⊆ P,
    prove((Q => absurd) ∈ C''),
    C << C''.

%% =>-elimination, modus ponens
prove(P ∈ C, LABEL) :-
    (A => P') ∈ C',
    P' ⊆ P, C << C',
    prove(A ∈ C'', LABEL),
    C << C''.
```

```
%% &-introduction
prove((P & Q) ∈ C, LABEL) :-
    prove(P' ∈ C', LABEL), P' ⊆ P, C << C',
    prove(Q' ∈ C'', LABEL), Q' ⊆ Q, C << C''.

%% or-introduction
prove((P or Q) ∈ P, LABEL) :-
    (prove(P' ∈ C', LABEL), P' ⊆ P, C << C');
    (prove(Q', LABEL ∈ C''), Q' ⊆ Q, C << C'').

%% =>-introduction (conditional proof)
prove((P => Q) ∈ C, LABEL) :-
    assert(P ∈ C),
    (prove(Q ∈ C, LABEL) ->
        retract(P ∈ C);
        (retract(P ∈ C), fail)).
```

Figure 36: SATCHMO with contexts

References

Bos, J., 2001. Model building for natural language understanding. In: in: Proceedings of ICoS-4. pp. 25–26.

Cooper, R., 1983. Quantification and Syntactic Theory. D. Reidel, Dordrecht.

Cryan, M., Ramsay, A. M., 1997. Constructing a normal form for Property Theory. In: Proceedings of the 14th International Conference on Automated Deduction (CADE-14). Vol. 1249 of Lecture Notes in Artificial Intelligence. Springer-Verlag, Berlin, pp. 237–251.
URL http://link.springer.com/chapter/10.1007%2F3-540-63104-6_22

Gabbay, D. M., 1989. Labelled deductive systems. Tech. rep., Dept. of Computing, Imperial College.

Gabbay, D. M., 1996. Labelled Deductive Systems. Oxford University Press, Oxford.

Keller, W. R., 1987. Nested Cooper storage: the proper treatment of quantification in ordinary noun phrases. Tech. Rep. CSRP. 73, University of Sussex.

Loveland, D. W., 1991. Near-horn Prolog and beyond. Journal of Automated Reasoning 7, 1–26.

Loveland, D. W., Reed, D. W., Wilson, D. S., 1995. Satchmore: Satchmo with relevancy. Journal of Automated Reasoning 14 (2), 325–351.
URL http://dx.doi.org/10.1007/BF00881861

Manthey, R., Bry, F., 1988. Satchmo: a theorem prover in Prolog. In: Lusk, R., Overbeek, R. (Eds.), Proceedings of the 9th International Conference on Automated Deduction (CADE-9). Vol. 310 of Lecture Notes in Artificial Intelligence. Springer-Verlag, Berlin, pp. 415–434.

Milward, D., Cooper, R., 1994. Incremental interpretation: Applications, theory and relationship to dynamic semantics. In: Wilks, Y. (Ed.), Proceedings of the 15th International Conference on Computational Linguistics (COLING-94). Kyoto, pp. 88748–754.

Ramsay, A. M., 1991. Generating relevant models. Journal of Automated Reasoning 7, 359–368.
URL http://link.springer.com/article/10.1007/BF00249019

Ramsay, A. M., 1992. Generic plural NPs and habitual VPs. In: Proceedings of the 14th International Conference on Computational Linguistics (COLING-92). Nantes, pp. 226–231.
URL https://aclweb.org/anthology/C/C92/C92-1037.pdf

Ramsay, A. M., 1995. Theorem proving for intensional logic. Journal of Automated Reasoning 14, 237–255.
URL http://link.springer.com/article/10.1007/BF00881857

Ramsay, A. M., 2001. Theorem proving for untyped constructive λ-calculus: implementation and application. Logic Journal of the Interest Group in Pure and Applied Logics 9 (1), 89–106.
URL http://www3.oup.co.uk/igpl

Ramsay, A. M., Field, D. G., 2006a. How to change a person's mind: understanding the difference between the effects and consequences of speech acts. In: 5th International Conference on Inference in Computational Semantics (ICoS 06). Buxton, pp. 27–36.
URL http://www.aclweb.org/anthology/W06-3903

Ramsay, A. M., Field, D. G., 2006b. Planning ramifications: when ramifications are the norm, not the problem. In: 11th International Workshop on Non-monotonic Reasoning (NMR-06). AAAI, Ambleside, pp. 344—351.
URL http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.111.5227

Ramsay, A. M., Field, D. G., 2008. Speech acts, epistemic planning and Grice's maxims. Logic and Computation 18, 431–457.
URL http://logcom.oxfordjournals.org/content/18/3/431.full.pdf

Ramsay, A. M., Field, D. G., 2009. 'Sorry' is the hardest word. In: Computational Approaches to Creativity (NAACL HLT Workshop). University of Colorado, Boulder, pp. 94–101.
URL http://www.aclweb.org/anthology/W09-2013

Turner, R., 1987. A theory of properties. Journal of Symbolic Logic 52(2), 455–472.

van Eijck, J., Alshawi, H., 1992. Logical forms. In: Alshawi, H. (Ed.), The Core Language Engine. Bradford Books/MIT Press, Cambridge, Mass., pp. 11–40.

Vestre, E., 1991. An algorithm for generating non-redundant quantifier scopings. In: Fifth Conference of the European Chapter of the Assoc. for Computational Linguistics. Berlin, pp. 251–256.

A *'all of the . . .', 'all the . . .', 'the . . . were all . . .'*

Examples of constructions involving '*all*' from the BNC.

(26)
a. who had been hostile , in print , to all of the participants in the historical events which supplied part of his
b. anything about , but would be willing to dismiss along all of the social sciences , I was in danger of being irrevocably
c. This mixed condition he shares with many others , not all of them writers ; it is a condition we are entitled to
d. organs of the body may be affected , and when all of these factors are combined the results can be severe , with
e. , ' said Thomas , ' which master announced to all of them was faulty , and that was why Rover slept there
f. description ' which most alarmed them ; for as almost all of them wistfully pointed out : ' it shows exactly how we
g. with the sights , sounds , smells and tastes –all of them clamorous and variegated and , not least , the girls

(27)
a. coalesced , inevitably , in the act of painting when all the discrete , scattered moments , followed up , caught on
b. compiler of a catalogue raisonné will have seen and compared all the works listed , or will scruple to state if some
c. ' tenderised ' – a word Roth likes , for all the awkwardness it imparts to the operations to which it refers
d. I would like to thank you and your Team for all the effort and resources you have put into providing a home
e. the other people who figure in his books , wrote all the time about himself , both in autobiographical and in fictional
f. a taxable income at equal to the gross amount of all these payments, as the Gift Aid payment being contemplated ,
g. the form requires the donor to state that he satisfies all the conditions relating to Gift Aid (as which , see

(28)
a. a hillside over Lake Garda , the lakeside promenade and the beach are all within easy walking distance , even it 's an uphill
b. , St John Ambulance men , special constables , and the like were all used to define them as being somewhat ' unreal '
c. a beard , the diadem , the upturned eye and the hairstyle are all derived from that of the great conqueror Alexander the Great
d. 's announcement that its development-area status is being removed , the staff are all cowering under their desks while the population throw bricks through